System Identification Toolkit

June 2008, 372458A–01

The LabVIEW System Identification Toolkit assists you in identifying large, multivariable models of high-order systems from large amounts of data. The System Identification Toolkit provides two tools, an assistant and a library of VIs, for identifying these linear systems. Both tools enable you to complete the entire system identification process, from analyzing raw data to validating the identified model.

This help file contains:

- Concepts—An overview of how to use the System Identification Toolkit.
- Reference—Detailed information about the System Identification VIs.

To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

Related Documentation (System Identification Toolkit)

The following documents contain information that you might find helpful as you use the LabVIEW System Identification Toolkit.

You must install the PDFs to access them from this help file. You must have Adobe Reader 6.0.1 or later installed to view or search the PDF versions of these manuals. Refer to the Adobe Systems Incorporated Web site to download Adobe Reader. Refer to the National Instruments Product Manuals Library for updated documentation resources.

- LabVIEW System Identification Toolkit Algorithm References—Use this manual to learn about the algorithms and function references that the System Identification VIs use.
- LabVIEW System Identification Toolkit Readme—Use this file to learn important last-minute information, including system requirements, installation instructions, known issues, and so on. Open this readme by selecting **Start»All Programs»National Instruments»LabVIEW»Readme** and opening readme_SI.html or by navigating to the labview\readme directory and opening readme_SI.html.
- LabVIEW System Identification Toolkit Example VIs—Refer to the labview\examples\System Identification directory for example VIs that demonstrate common tasks using the System Identification Toolkit. You also can access these VIs by selecting **Help»Find Examples** from the pull-down menu and selecting **Toolkits and Modules»System Identification Toolkit** in the NI Example Finder window.
- The LabVIEW Control Design and Simulation Module documentation
- Additional LabVIEW documentation

System Identification Concepts (System Identification Toolkit)

Use this book to learn about concepts in the LabVIEW System Identification Toolkit.

To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

Introduction to System Identification (System Identification Toolkit)

System identification, the first step in the model-based control design process, involves building mathematical models of a dynamic system based on a set of measured stimulus and response data samples. You can use system identification in a wide range of applications, including mechanical engineering, biology, physiology, meteorology, economics, and model-based control design. For example, engineers use a system model of the relationship between the fuel flow and the shaft speed of a turbojet engine to optimize the efficiency and operational stability of the engine. Biologists and physiologists use system identification techniques in areas such as eye pupil response and heart rate control. Meteorologists and economists build mathematical models based on historical data for use in forecasting.

The LabVIEW System Identification Toolkit provides the following tools.

System Identification VIs

The System Identification Toolkit provides VIs that you can use to preprocess raw data from a dynamic system and develop a model that reflects the behavior of that system. The Data Preprocessing VIs enable you to analyze the response of a plant or dynamic system to a certain stimulus. After analyzing the data, you can use the Parametric Model Estimation, Nonparametric Model Estimation, Partially Known Model Estimation, Recursive Model Estimation, and/or Frequency-Domain Model Estimation VIs to estimate a model for the plant or dynamic system. Finally, you can use the Model Validation or Model Analysis VIs to determine whether the model accurately describes the dynamics of the identified system.

The System Identification VIs enable you to customize a LabVIEW block diagram to achieve specific goals. You also can use other LabVIEW VIs and functions to enhance the functionality of the application. Creating a LabVIEW application using the System Identification VIs requires basic knowledge about programming in LabVIEW. Refer to the LabVIEW Fundamentals and Getting Started with LabVIEW manuals for more information about the LabVIEW programming environment.

The following case studies demonstrate how to use the System Identification VIs to estimate different model representations by using time-domain or frequency domain data.

- System Identification Case Study—Guides you through the entire system identification process. This case study demonstrates how to preprocess time-domain data from a dynamic system, estimate an ARX and state-space model by using the time-domain data, and validate the models to ensure they accurately reflect the dynamic system.
- Partially Known Model Estimation Case Study—Demonstrates how to estimate a state-space model by using prior knowledge about the system you want to define.
- Frequency-Domain Model Estimation Case Study—Demonstrates how to estimate and validate a state-space and transfer function model by using frequency-domain data from a dynamic system.

System Identification Assistant

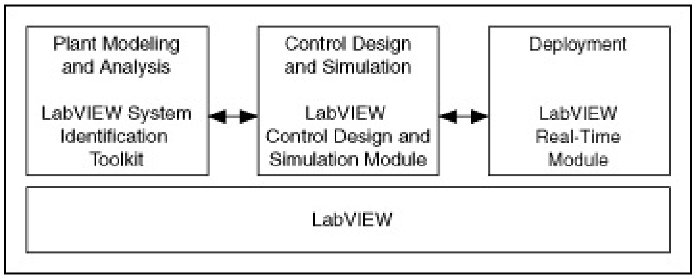
If you do not have prior knowledge about programming in LabVIEW, you can use the NI System Identification Assistant to develop a model that reflects the behavior of a certain dynamic system. You access the System Identification Assistant by launching LabVIEW and selecting **Tools»Control Design and Simulation»Launch System Identification Assistant**.

Using the System Identification Assistant, you can create a project that encompasses the whole system identification process. In a single project, you can load or acquire raw data into the System Identification Assistant, preprocess the data, estimate a model that describes the system, and then validate the accuracy of the model. SignalExpress provides windows in which you can see the raw data, the response data, the estimated model, the validation results, and the mathematical equations that describe the model.

After creating a project in SignalExpress, you can convert the project to a LabVIEW block diagram and customize the block diagram in LabVIEW. This conversion enables you to enhance the capabilities of the application. Refer to the *LabVIEW SignalExpress Help*, available in the LabVIEW SignalExpress environment by selecting **Help»LabVIEW SignalExpress Help**, for more information about using the assistant to develop models.

Model-Based Control Design Process (System Identification Toolkit)

The model-based control design process involves building a model of a plant, analyzing and synthesizing a controller for the plant, simulating the plant and controller, and deploying the controller. National Instruments provides a complete solution for identifying and validating a dynamic system model, designing a controller for the model, analyzing the controller, and prototyping and deploying the control system. The following figure shows this process, which is based on LabVIEW and National Instruments Real-Time (RT) Series hardware.

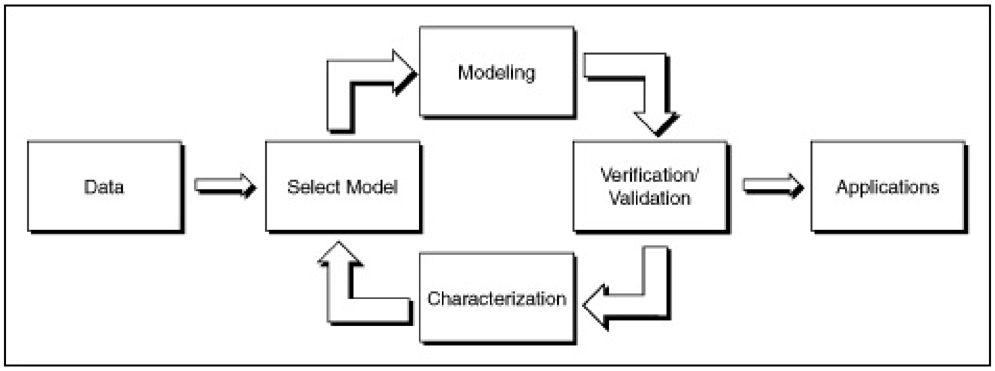

Plant Modeling and Analysis
LabVIEW System Identification Toolkit
Control Design and Simulation
LabVIEW Control Design and Simulation Module
Deployment
LabVIEW Real-Time Module
LabVIEW

Analyzing Data and Creating a Dynamic System Model

In the initial phase of the design process, you must obtain a mathematical model of the plant you want to control. One way to obtain a model is by using a numerical process known as system identification. This process involves acquiring data from a plant and then numerically analyzing stimulus and response data to estimate the parameters of the plant.

National Instruments provides DAQ and modular instrumentation software and hardware that you can use to stimulate and measure the response of the plant. You then can use the System Identification Toolkit to estimate and create accurate mathematical models of the plant.

System identification is a process that includes acquiring and preprocessing raw data from a real-world system and identifying mathematical models based on the raw data. You then validate that the resulting model accurately describes the observed system behavior. If the results are unsatisfactory, you revise the parameters and iterate through the process. The following flowchart shows a typical system identification process.



A real-world system seldom has one model that perfectly describes all the observed behaviors of the system. Because system identification involves many variables—such as sampling frequency, type of mathematical model, model order, and so on—you usually have a number of models you can use. Each model describes the behavior of the system to some extent or in a particular mode of operation.

Furthermore, multiple applicable algorithms might be available for the same model. The algorithms you select depend on the model structure, stochastic assumptions, and numerical properties of the algorithm. The System Identification Toolkit includes different adaptive techniques for recursive system identification and different algorithms for model estimation.

Designing a Controller

In the second phase of the design process, you synthesize and analyze a controller. The LabVIEW Control Design and Simulation Module provides a set of VIs for classical and modern linear control analysis and design techniques. With these VIs you can design, implement, and deploy linear time-invariant (LTI) system models.

You can use the Control Design and Simulation Module to analyze the plant model you identified with the System Identification Toolkit. The Control Design and Simulation VIs help you determine an appropriate controller structure. You then can synthesize a controller to achieve the desired performance criteria of the system based on the dynamic behavior of the plant and/or control system. Finally, you can analyze the overall system by combining the controller with the identified plant model.

Simulating the Dynamic System

In the third phase of the design process, you simulate the dynamic system. You can simulate dynamic systems in LabVIEW by using the Control Design and Simulation Module. You can use this software to perform offline simulations of a linear or nonlinear dynamic system model. You can use this simulation to investigate the time and frequency responses of the dynamic system due to complex, time-varying inputs. If you install the LabVIEW Real-Time Module, you also can perform rapid control prototyping (RCP), and hardware-in-the-loop (HIL) simulations by using NI RT Series hardware.

Deploying the Controller

The last stage of the design process is to deploy the controller to an RT target, such as a PXI or CompactRIO controller. LabVIEW, the Control Design and Simulation Module, the RT Module, and NI RT Series hardware provide a common platform that you can use to design, prototype, and deploy the embedded control system. You also can use the Control Design and Simulation Module and the Real-Time Module as the platform for implementing the control system.

National Instruments also provides products for I/O and signal conditioning, such as NI DAQ and SCXI devices, that you can use to gather and process data. Using these tools, which are built on the LabVIEW platform, you can experiment with different approaches at each stage in the design process and quickly identify the optimal design solution for an embedded control system.

Refer to the National Instruments Web site at ni.com for more information about these National Instruments products.

Model Types and Representations (System Identification Toolkit)

You can represent a dynamic system using several types of dynamic system models.

Model Types

You base the type of a dynamic system model on the properties of the dynamic system that the model represents.

Linear versus Nonlinear Models

Dynamic system models are either linear or nonlinear. A linear model obeys the principles of superposition and homogeneity. The following equations are true for linear models.

$y_1 = f(u_1)$

$y_2 = f(u_2)$

$f(u_1 + u_2) = f(u_1) + f(u_2) = y_1 + y_2$

$f(a_1 u_1) = a_1 f(u_1) = a_1 y_1$

where u_1 and u_2 are the system inputs and y_1 and y_2 are the system outputs.

Conversely, nonlinear models do not obey the principles of superposition or homogeneity. Nonlinear effects in real-world systems include saturation, dead-zone, friction, backlash, and quantization effects; relays; switches; and rate limiters. Many real-world systems are nonlinear, but you can simulate most real-world systems with linear models to simplify a design or analysis procedure.

Time-Variant versus Time-Invariant Models

Dynamic system models are either time-variant or time-invariant. The parameters of a time-variant model change with time. For example, you can use a time-variant model to describe an automobile. As fuel burns, the mass of the vehicle changes with time.

Conversely, the parameters of a time-invariant model do not change with time. For an example of a time-invariant model, consider a simple robot. Generally, the dynamic characteristics of robots do not change over short periods of time.

Continuous versus Discrete Models

Dynamic system models are either continuous or discrete. Both continuous and discrete system models can be linear or nonlinear and time-invariant or time-variant. Continuous models describe how the behavior of a system varies continuously with time, which means you can obtain the properties of a system at any certain moment from the continuous model. Discrete models describe the behavior of a system at separate time instants, which means you cannot obtain the behavior of the system between every two sampling points.

Continuous system models are analog. You derive continuous models of a physical system from differential equations of the system. The coefficients of continuous models have clear physical meanings. For example, you can derive the continuous transfer function of an RC circuit if you know the details of the circuit. The coefficients of the continuous transfer function are the functions of *R* and *C* in the circuit. You use continuous models if you need to match the coefficients of a model to some physical components in the system.

Discrete system models are digital. You derive discrete models of a physical system from difference equations or by converting continuous models to discrete models. In computer-based applications, signals and operations are digital. Thus, you can use discrete models to implement a digital controller or to simulate the behavior of a physical system at discrete instants. You also can use discrete models in the accurate model-based design of a discrete controller for a plant.

Model Representations

You can use the System Identification Toolkit to represent each model type in any of the following representations:

- Transfer function models
- Zero-pole-gain models
- State-space models

If the model is discrete, you also can use a general-linear polynomial model.

Acquiring and Preprocessing Data (System Identification Toolkit)

The first step in identifying an unknown system is acquiring data. You can acquire plant data by using NI DAQ hardware and software or you can use data from a file on disk. You can acquire data in the time domain and/or the frequency domain.

For verification and validation reasons, you must acquire two sets of input-output data samples or split the data into two sets. You use one set of samples to estimate the mathematical model of the system. You use the second set of samples to validate the resulting model. If the resulting model does not meet the predefined specifications, such as the mean square error (MSE), modify the settings and re-verify the resulting model with the data sets.

After acquiring the data, you must preprocess the raw data samples. Preprocessing involves steps such as removing trends, filtering noise, and so on. You can use the Data Preprocessing VIs to analyze the raw data and determine whether that data accurately reflects the response of the system you want to identify. To identify a system model in the frequency domain by using time-domain data, you must preprocess the time-domain data by estimating the frequency response function (FRF). You can use the SI Estimate FRF VI to estimate the FRF from time-domain data.

Acquiring Data from a System (System Identification Toolkit)

Identifying a system involves a number of choices with regard to the system output signals you want to measure and the input signals you want to manipulate. The choices you make about how to manipulate system inputs, types of signal conditioning, signal ranges, and sampling behavior affect the validity of the model you obtain. You can use different modeling techniques on the same experimental data set. However, if the data set does not reflect the behavior of interest, you must acquire a more descriptive data set.

Because the system identification process is often an experimental process, the entire process often is time consuming and possibly costly. Therefore, you must think about the design of process prior to experimenting with various identification techniques.

Accounting for Factors that Influence a System (System Identification Toolkit)

The key to the system identification process is having some knowledge of the system for which you want to identify a model. This knowledge provides the basis for determining which signals are outputs, which in turn determines sensor placement, and which signals are inputs that you can use to excite the system. Simple tests might be necessary to determine influences, coupling, time delays, and time constants to aid in the modeling effort.

You also must consider signals that are not directly capable of being manipulated but still affect the system. You must include those signals as inputs to the system model. For example, consider the effect of wind gusts on the pitch dynamics of an airplane. The airplane responds in pitch to the elevator angle as a direct input. A wind gust affects the pitch of an airplane, which in turn influences the dynamics of the airplane, but the wind gust is not directly adjustable. To create an accurate model of the airplane, you might want to include wind gusts as an input variable.

Choosing a Stimulus Signal (System Identification Toolkit)

The choice of stimulus signals has an important role in the system behavior and the accuracy of the estimated model. These signals determine the operating points of the system. While the system under test often limits the choice of signals, the input signal must exhibit certain characteristics. These characteristics must produce a response that provides the information you need for developing an accurate model. The following list summarizes these characteristics.

- To obtain meaningful dynamic behavior, you must test the system under conditions similar to the actual operating conditions. When you complete experiments in these conditions, you identify the system in the same conditions under which you will implement the resulting model. This criterion is extremely important for nonlinear systems.
- You want the inputs to the system under test to excite the system. Exciting the system is dependent on the spectrum of the input signal. Specifically, you must excite the system with an input frequency similar to the frequency at which such inputs change during normal operations.
- You want the amplitude of the step input to cover a wide range of variations. Therefore, in the data you use for model estimation, you must cover the normal operation range of system inputs, especially when you use the calculated model for model-based control. To cover the normal operation range, you can combine the positive and negative step changes of different magnitudes in the system inputs.
- You want the input signal to deliver as much input power to the system as possible. However, in the real-world, you must ensure that this input power stays within the limits of the physical system. The crest factor C_f, defined by the following equation, describes this property.

$$C_f^2 \equiv \frac{\max_t u^2(t)}{\lim_{N\to\infty} \frac{1}{N} \sum_{t=1}^{N} u^2(t)}$$

The smaller the crest factor, the better the signal excitation. A better signal excitation results in larger total energy delivery and enhanced signal-to-noise ratio. The theoretical lower bound for the crest factor is 1.

Common Stimulus Signal Types

The system response data is dependent on the physics of the system you want to study. Some systems tend to respond faster than others. Other systems have large time constants and delays. For these reasons, defining a stimulus signal that provides enough excitation to the system is important. The system response must capture the important features of the system dynamics.

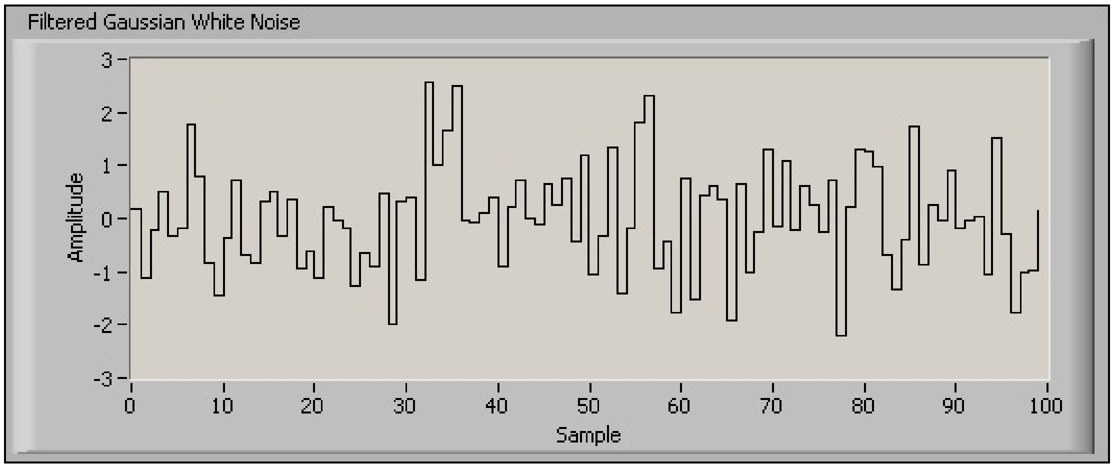
You can use the following types of stimulus signals for exciting the system under test.

- Filter Gaussian White Noise
- Random Binary Signal
- Pseudo-Random Binary Sequence
- Chirp Waveform

Filtered Gaussian White Noise (System Identification Toolkit)

Filtered Gaussian white noise is a simple signal that can generate virtually any signal spectra in conjunction with the proper linear filtering. The theoretical crest factor C_f for a Gaussian is infinite, but clipping the Gaussian amplitude to the input signal limit reduces the crest factor. Doing so minimally affects the generated spectrum.

The following figure shows an example of Filtered Gaussian white noise.



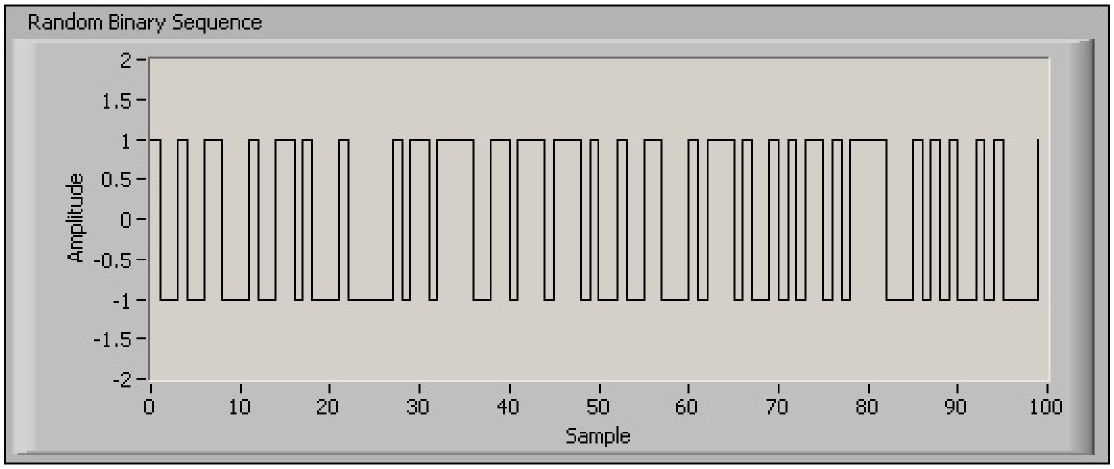
Random Binary Signal (System Identification Toolkit)

A random binary signal is a random process that can assume one of two possible values at any time. A simple method of generating a random binary signal is to take Gaussian white noise, filter it for the desired spectra and then convert it to a binary signal by taking the sign of the filtered signal. The desired spectra is a function of the system time constraints. The appropriate scaling must provide a meaningful response to the system, well above the noise level.

You can scale the signal to any desired amplitude. The resulting signal has a minimum crest factor C_f of 1. You can expect some differences in the resulting spectra. Therefore, you must perform offline analysis of the signal.

Binary signals are useful for identifying linear systems. However, the dual-level signal does not allow for validation against nonlinearities. If a system is nonlinear, you can use an input interval corresponding to the desired operating point. You might need to work with more than two input levels in these cases. You can combine multiple binary signals of different levels to form the stimulus signal.

The following figure shows an example of a random binary signal.

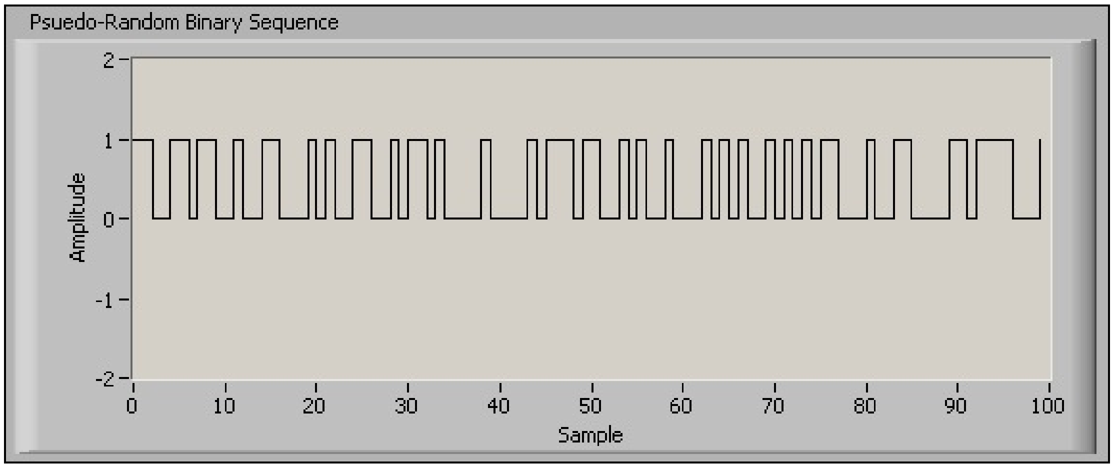


Pseudo-Random Binary Sequence (System Identification Toolkit)

A Pseudo-Random Binary Sequence, also known as Maximal Length Sequence (MLS), is a periodic, deterministic signal with properties similar to white noise. You often generate a pseudo-random binary sequence using an *n*-bit shift register with feedback through an exclusive or (XOR) function. While appearing random, the sequence actually repeats every 2*n*–1 values.

When using a whole period, the pseudo-random binary sequence has special mathematical advantages that make it attractive as a stimulus signal. In particular, you can attribute variations in response signals between two periods of the stimulus to noise due to the periodic nature of the signal. Also, like the white random binary noise, the pseudo-random binary sequence has a low crest factor C_f. You can use the SI Generate Pseudo-Random Binary Sequence VI to generate a Pseudo-Random Binary Sequence.

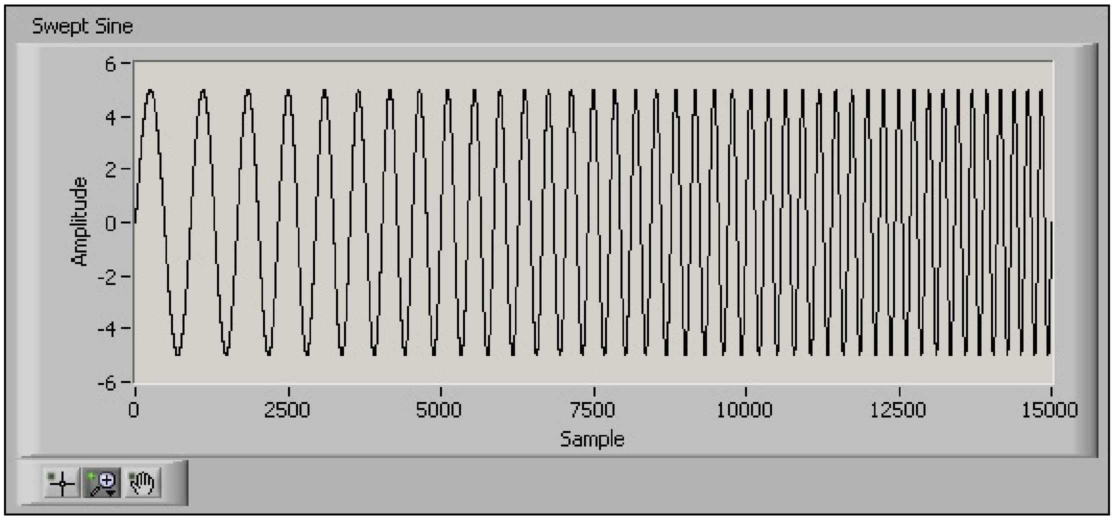
The following figure shows an example of a pseudo-random binary sequence.



Chirp Waveform (System Identification Toolkit)

The chirp waveform, also known as a swept sine wave, is a sinusoid waveform with a frequency that varies continuously over a certain range of values $\omega_1 \le \omega \le \omega_2$ for a specific period of time $0 \le t \le T$. The resulting signal has a crest factor C_f of $\sqrt{2}$. You can modify the signal to excite specific signal spectra.

In comparison to other stimulus signals, such as white noise, a chirp waveform is easier to generate and control. The following figure shows an example of a chirp waveform.



Selecting a Sampling Rate (System Identification Toolkit)

The time constants of a system influence the selection of a sampling rate. Sampling at rates substantially greater than the system bandwidth leads to data redundancy, numerical issues, and modeling of high frequency artifacts likely due to noise. Sampling at rates slower than system dynamics leads to difficulties determining an accurate system model and problems introduced by aliasing. You can use an anti-aliasing filter to counter the effects of aliasing.

A common rule of thumb is to sample signals at 10 times the bandwidth of the system or the bandwidth of interest for the model. If uncertainty exists in the system bandwidth and a fast data acquisition environment is available, you can sample as fast as possible, then use a digital filter and decimation to reduce the sampling rate to the desired value. Decimation is a form of downsampling the data set.

Applying an Anti-Aliasing Filter (System Identification Toolkit)

According to the Nyquist sampling theorem, the sampling rate must be greater than twice the maximum frequency component of the signal of interest. In other words, the maximum frequency of the input signal must be greater than half the sampling rate.

This criterion, in practice, is often difficult to ensure. Even if you are sure that the measured signal has an upper limit on its frequency, external factors, such as signals from the powerline interference or radio stations, can contain frequencies higher than the Nyquist frequency. These frequencies might then alias into the frequency range of interest and give you inaccurate results.

To ensure that you limit the frequency content of the input signal, you can add a lowpass filter before the sampler and the analog to digital converter (ADC). A lowpass filter passes low frequencies and attenuates high frequencies. This filter is an anti-aliasing filter because by attenuating the frequencies greater than the Nyquist frequency, the filter prevents the sampling of aliased components. When you use a filter before the sampler and ADC, the anti-aliasing filter is an analog filter. Using an analog filter satisfies the Nyquist sampling theorem.

Similarly, you can use a digital filter to remove frequency content above the system bandwidth and then downsample the data to the desired sampling rate.

Preprocessing Data from a System (System Identification Toolkit)

You can use a number of preprocessing techniques to ensure that the incoming data samples are free from external noise, scaling problems, outliers, and other corruptions. These preprocessing techniques include the following methods:

- Visually inspecting data
- Removing offsets and trends
- Filtering and downsampling
- Data Scaling

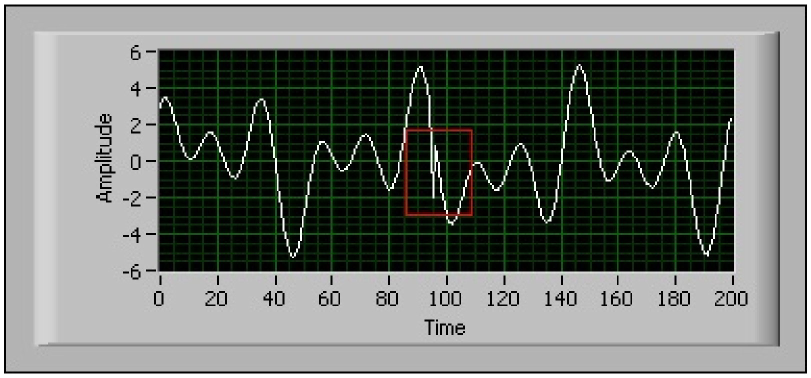
Note To identify a system model in the frequency domain by using time-domain data, you must preprocess the time-domain data by estimating the frequency response function (FRF).

Validating the quality of the data at each step in the preprocessing procedure is important in ensuring that you accurately identify a model in the later steps of the system identification process.

Visually Inspecting the Data (System Identification Toolkit)

Various unexpected events, such as an abnormal pulse, a temporary sensor failure, or transmitter failure, can corrupt the raw data samples. These disturbances can result in outliers, clipped saturation, and/or quantization effects that severely distort the resulting model estimation. Visually inspecting the data is the best way to detect these disturbances.

In the following figure, you can recognize outliers by visually inspecting the data.



In the previous figure, notice that the data acquired between 85–100 seconds is abnormal. When preprocessing data, you want to remove all outliers in the data set. You must remove the outliers manually.

Note You also can plot the data waveform and the spectral density function of the data to discover periodic disturbances.

Traditionally, you examine data samples either in the time domain or the frequency domain. An effective approach is to display the data in the joint time-frequency domain, which provides a better understanding about the measured signals. Refer to the *Time Frequency Analysis Tools User Manual*, available at ni.com/manuals, for more information about joint time-frequency domain techniques for data processing.

Removing Offsets and Trends (System Identification Toolkit)

You can remove offsets and trends from the raw data set by using the SI Remove Trend VI. You can specify whether to remove offsets or trends by using the **trend type** input of this VI.

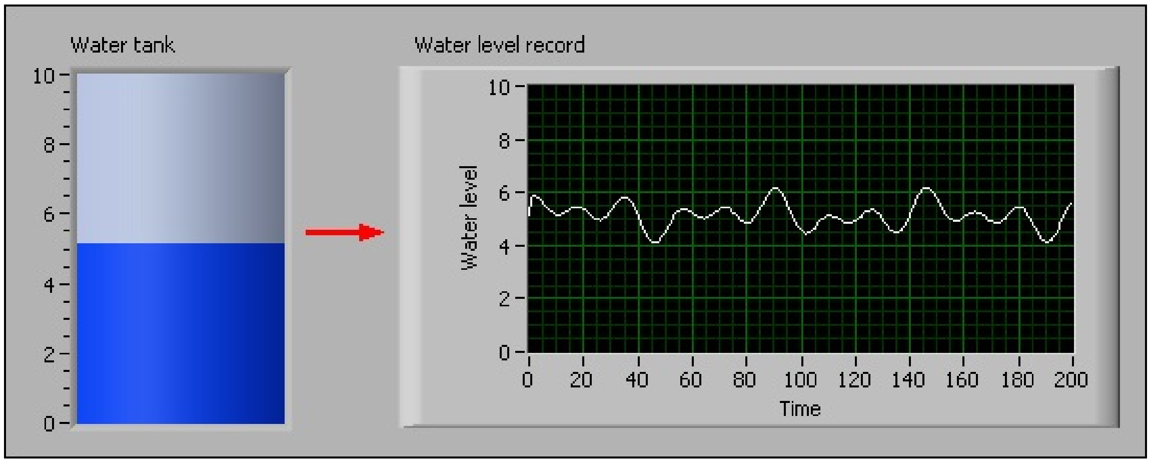
Refer to the Remove Trend VI in the labview\examples\System Identification\Getting Started\General.llb for an example that demonstrates how to remove the offset or trend from a signal.

Open example Browse related examples

Removing Offsets

An estimated system model is a linearized version of the plant around the operating point. You must subtract the operating points from the raw data samples because linearization is done with respect to the signal values relative to the operating point, which is the offset level of the signal.

The following figure shows an example of removing the offset level of a signal. The goal of the water tank is to keep the water level at six meters. The **Water level record** graph shows that the water level changes in the vicinity of the operating point of six meters. If you use the water level record for system identification, you must remove the six meter operating point value.



The SI Remove Trend VI enables you to remove the offset from the raw data set. You must set the **trend type** to **mean** to use this preprocessing technique.

Removing Trends

External influences might add some low frequency or periodic components to the data. These additional components are not relevant to the specific modeling problem. Examples of external influences include variations due to the 24-hour day cycle in power plants, seasonal influences in biological and economical systems, thermal expansion in rolling mills, 50 or 60 Hz interference in power lines, and so on. The amplitude of these trends can be large and can corrupt the results of signal analysis and parametric identification algorithms.

The SI Remove Trend VI provides a way for you to remove these external influences, or trends, from the raw data set. You must set the **trend type** to **linear** to use this preprocessing technique.

Filtering and Downsampling (System Identification Toolkit)

You might be interested in only a specific frequency range of the frequency response for a model. You can filter and enhance the data in the frequency range to improve the fit in the regions of interest. If the sampling frequency is much higher than the bandwidth of the system, the sampling frequency might substantially increase the computation burden for complicated identification algorithms. You can decrease the sampling frequency by taking every n^{th} sample to construct a new downsampled data set. Applying an anti-alias filter on the data before downsampling prevents corruption of the downsampled data set.

You can use the SI Lowpass Filter VI or the SI Bandpass Filter VI to apply a lowpass or bandpass filter, respectively, to the data from the system. You then can use the SI Down Sampling VI to reduce the number of samples in the data set.

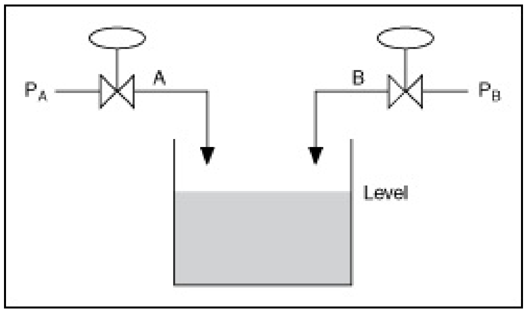
Refer to the Down Sampling VI in the labview\examples\System Identification\Getting Started\General.llb for an example that demonstrates how to use the SI Down Sampling VI to reduce the sampling rate of a signal.

▪ Open example ▪ Browse related examples

After preprocessing the data you acquired from a dynamic system, the result is a data set that you can use to estimate a model that reflects the system dynamics.

Data Scaling (System Identification Toolkit)

Multiple-input multiple-output (MIMO) systems commonly have inputs and outputs of different amplitude ranges. This diversity can result in an ill-conditioned model estimation, which reduces the accuracy of the model. For example, consider the valves A and B in the following figure.



Valves A and B operate between 0–100% and 50–60%, respectively. The pressure in the respective stream lines are P_A and P_B. If you assume that P_B can be much larger than P_A, you might need to normalize the range of operation of valve *B* for numerical robustness. You can use the following relationship to normalize the range of operation.

$$\frac{\Delta Level}{\Delta \% A} = \frac{\Delta Level}{(\Delta \% B - 50) \cdot 10}$$

The SI Normalize VI ensures that all stimulus and response signals have a zero mean and unit variance over the sample data range used for model estimation. This process standardizes the range of the equation for all signals considered for model estimation. This data preprocessing step considers all inputs and outputs equally important from the numerical calculation viewpoint.

Estimating Models (System Identification Toolkit)

After acquiring and preprocessing the data from a linear time-invariant system, the next step in the system identification process is estimating the model. You can use the LabVIEW System Identification Toolkit to estimate models by using any of the following methods:

- Nonparametric
- Parametric
- Partially Known
- Closed-Loop Systems
- Recursive
- Frequency-Domain

Nonparametric Model Estimation Methods (System Identification Toolkit)

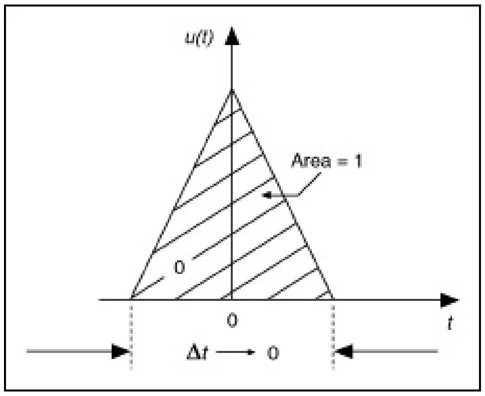
You can describe linear time-invariant models with transfer functions or by using the impulse response or frequency response of the system. The impulse response and frequency response are two ways of estimating a nonparametric model. The impulse response reveals the time-domain properties of the system, such as time delay and damping. The frequency response reveals the frequency-domain properties, such as the natural frequency of the system.

Nonparametric model estimation is more efficient, but often less accurate, than parametric estimation. However, you can use a nonparametric model estimation method to obtain useful information about a system before applying parametric model estimation. For example, you can use nonparametric model estimation to determine whether the system requires preconditioning, what the time delay of the system is, what model order to select, and so on. You also can use nonparametric model estimation to verify parametric models. For example, you can compare the Bode plot of a parametric model with the frequency response of the nonparametric model.

You can use the least squares and correlation analysis methods to estimate the impulse response of a dynamic system. You can use the spectral analysis method to estimate the frequency response of a dynamic system.

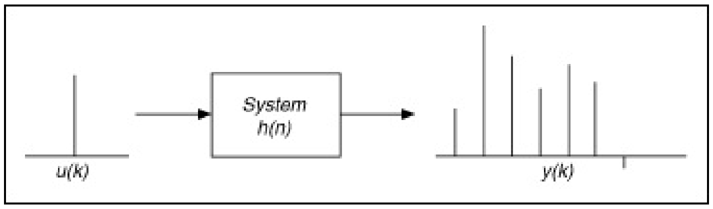
Impulse Response (System Identification Toolkit)

An impulse input, as shown in the following figure, to a dynamic system is defined differently depending on whether the system is discrete or continuous. For a continuous dynamic system, an impulse input, also known as the Dirac delta function, is a unit-area signal with an infinite amplitude and infinitely small duration occurring at a specified time. At all other times, the input signal value is zero. For a discrete system, an impulse is a physical pulse that has unit amplitude at the first sample period and zero amplitude for all other times.



Because the impulse signal excites all frequencies and the duration of this signal is infinitely small, you can see the natural response of the system.

The following figure shows that the impulse response of a linear time-invariant system is equal to the output *y*(*k*) of the system when you apply an impulse signal to the input *u*(*k*) of the system. The impulse response provides the complete characteristic information of a system.



If you know the impulse response *h*(*n*) and the input signal *u*(*k*) of a system, then you can compute the output *y*(*k*) of the system by using the following equation.

$$y(k) = \sum_{n=-\infty}^{\infty} u(k-n)h(n) + e(k)$$

where $e(k)$ is the disturbance of the system.

According to impulse response theory, when you apply a Dirac delta function to a system, the output of the system is the impulse response. You can think of the Dirac delta function $\delta(x)$ as a function that has the value of infinity for $x = 0$, the value zero elsewhere, and a total integral of one. However, generating an ideal Dirac delta function is unrealistic.

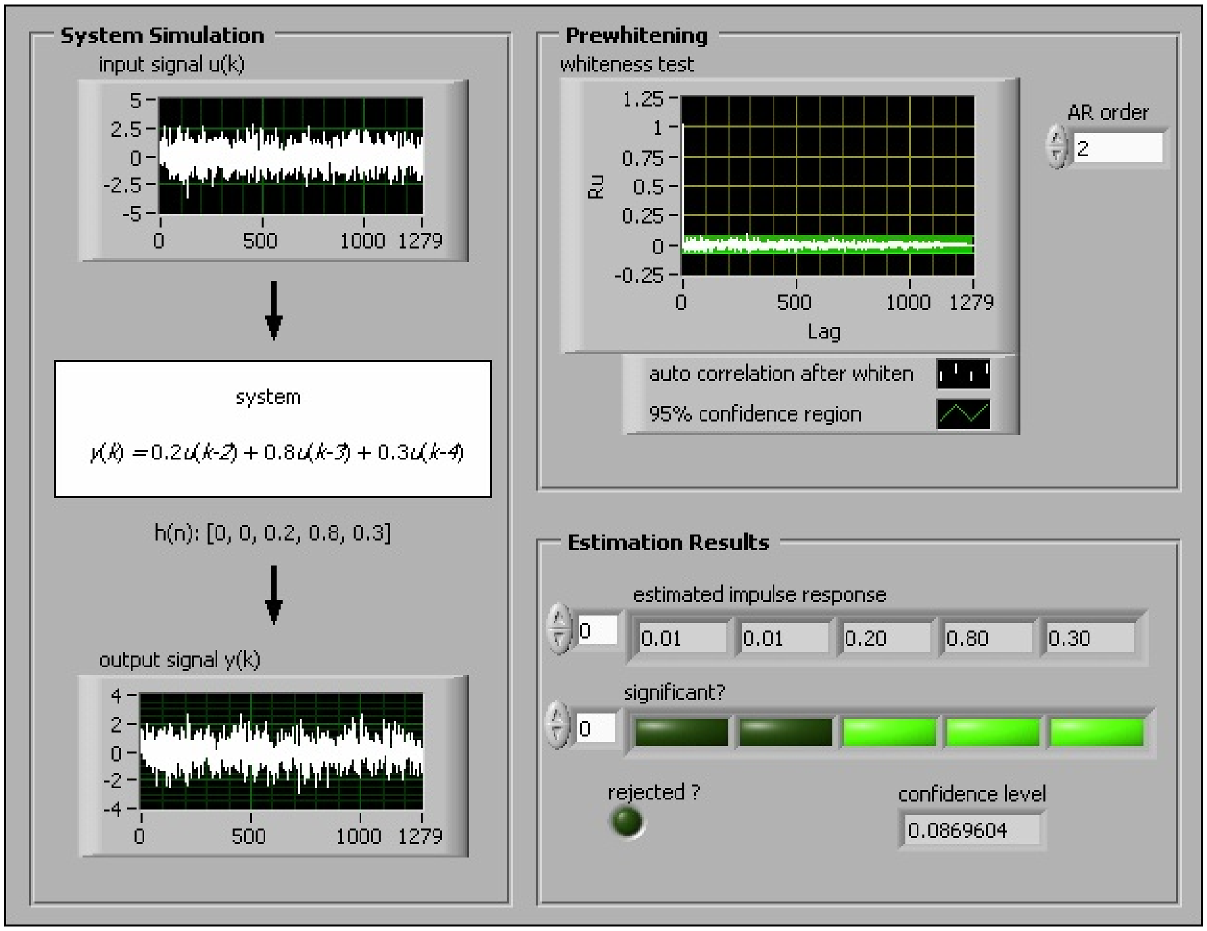
If you apply an approximate impulse with a small duration to the input of a system, the output of the system is the approximation of the impulse response of the system. The smaller the duration of the impulse, the closer the output of the system is to the true impulse response. However, an impulse carries little energy and might not excite the system, and noise might corrupt the output of the system. An impulse with a large amplitude and duration can improve the signal-to-noise ratio of the output signal. However, a large amplitude impulse can damage the hardware of the system, and a long-duration impulse leads to inaccuracy. For these reasons, you can use the least squares and correlation analysis methods to estimate the impulse response.

Applications of the Impulse Response

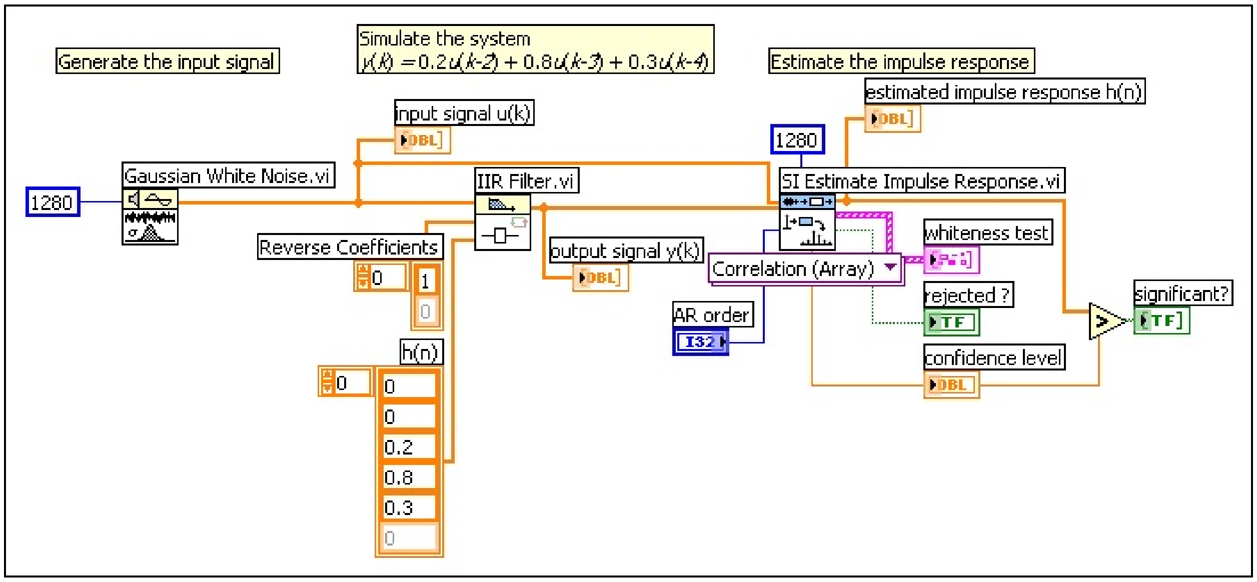
The impulse response not only indicates the stability and causality of the system if feedback exists in the system, but also provides information on properties such as the damping, dominating time constant, and time delay. Some of this information, such as the time delay, is useful for parametric model estimation. Therefore, you can use nonparametric impulse response estimation before parametric model estimation to help estimate the parameters. You can use the SI Estimate Impulse Response VI to estimate the impulse response and determine the time delay of a system by using the correlation analysis method.

The following figure shows the front panel of a VI that simulates a system defined by the following equation.

$$y(k) = 0.2u(k-2) + 0.8u(k-3) + 0.3u(k-4)$$



The following figure shows the block diagram of this VI.



In the previous figure, the two initial values of the **estimated impulse response** are smaller than the **confidence level**. You can have 99.0% confidence that values less than the **confidence level** are insignificant, and you can consider those values to be equal to 0. Therefore, you can conclude that the time delay of the system is 2 because the beginning of the first two values of the impulse response are zero.

Another common application of the impulse response is to detect feedback in systems by using the least squares method. If feedback exists in a system, the impulse response of the system becomes significantly large at negative lags and the correlation between the input signal and disturbance *e*(*k*) is nonzero. The correlation analysis method assumes the input signal and the disturbance *e*(*k*) are independent from each other. Thus, this method cannot estimate accurately the impulse response of the system that contains feedback. Only the least squares method can provide reliable results. You can use the SI Detect Feedback VI to estimate the impulse response of a system and determine whether feedback exists in the system.

Refer to the Feedback Detection VI in the labview\examples\System Identification\Getting Started\General.llb for an example that demonstrates how to use the SI Detect Feedback VI to detect feedback in a system.

Open example   Browse related examples

Least Squares Method (System Identification Toolkit)

If both the input signal *u*(*k*) and output signal *y*(*k*) of a system are available, you can obtain the value of the impulse response *h*(*k*). This method does not require a Dirac delta function as the input signal of the system. Instead, you can use a common stimulus signal and the corresponding response signal from the system to compute the impulse response mathematically. You can obtain the impulse response for both positive and negative lags.

The Least Squares instance of the SI Estimate Impulse Response VI implements the least squares method to obtain the value of *h*(*k*). Refer to the LabVIEW System Identification Toolkit Algorithm References manual for more information about the least squares method.

Correlation Analysis Method (System Identification Toolkit)

The correlation analysis method uses the cross correlation between the input and output signals as an estimation of the impulse response, as shown by the following equation:

$$y(k) = \sum_{n=0}^{\infty} u(k-n)h(n) + e(k)$$

The input signal must be zero-mean white noise with a spectral density that is equally distributed across the whole frequency range. The SI Estimate Impulse Response VI can prewhiten input signals that are not white noise.

Assuming the input $u(k)$ of the system is a stationary, stochastic process and statistically independent of the disturbance $e(k)$, the following equation is true.

$$R_{uy}(\tau) = \sum_{k=0}^{\infty} R_{uu}(k-\tau)h(k)$$

R_{uy} represents the cross-correlation function between the stimulus signal $u(k)$ and the response signal $y(k)$, as defined by the following equation.

$$R_{uy}(\tau) = \frac{1}{N} \sum_{k=min(\tau,0)}^{N-max(\tau,0)-1} y(k+\tau)u(k)$$

R_{uu} represents the autocorrelation of the stimulus signal $u(k)$, as defined by the following equation.

$$R_{uu}(\tau) = \frac{1}{N} \sum_{k=0}^{N-\tau-1} u(k+\tau)u(k)$$

N is the number of data points. If the stimulus signal is a zero-mean white noise signal, the autocorrelation function reduces to the following equation.

$$R_{uu}(\tau) = \sigma_u^2 \delta(\tau)$$

where σ_u is the standard deviation of the stimulus white noise and $\delta(\tau)$ is the Dirac function. Substituting $R_{uu}(\tau)$ into the cross-correlation function between the stimulus signal $u(k)$ and the response signal $y(k)$ yields the following equation.

$$R_{uy}(\tau) = \sigma_u^2 \sum_{k=0}^{\infty} \delta(k - \tau)h(k) = \sigma_u^2 h(\tau)$$

You can rearrange the terms of this equation to obtain the following equation defining the impulse response *h*(*k*).

$$h(k) = \frac{R_{uy}(k)}{\sigma_u^2}$$

Prewhitening

The correlation analysis method that estimates the impulse response is useful only when the input signal *u*(*k*) is a zero-mean white noise signal. However, the input signal is not white noise in most real-world applications. Therefore, you must precondition the input *u*(*k*) and output *y*(*k*) signals before you apply the correlation analysis method.

Prewhitening is a preconditioning technique for the correlation analysis method. Prewhitening involves applying a filter to the input signal *u*(*k*) and the output signal *y*(*k*) to obtain a prewhitened input signal *u'*(*k*) and a prewhitened output signal *y'*(*k*). If the filter is well designed such that *u'*(*k*) is white noise, you can perform a correlation analysis on *u'*(*k*) and *y'*(*k*) to estimate the impulse response. The impulse response that you estimate with *u'*(*k*) and *y'*(*k*) is equivalent to the impulse response that you estimate with *u*(*k*) and *y*(*k*) because the following equation remains true.

$$y'(k) = \sum_{n=0}^{\infty} u'(k-n)h(n) + e(k)$$

You now must design the prewhitening filter so that *u*'(*k*) is white noise. The SI Estimate Impulse Response VI uses an AR model for this purpose.

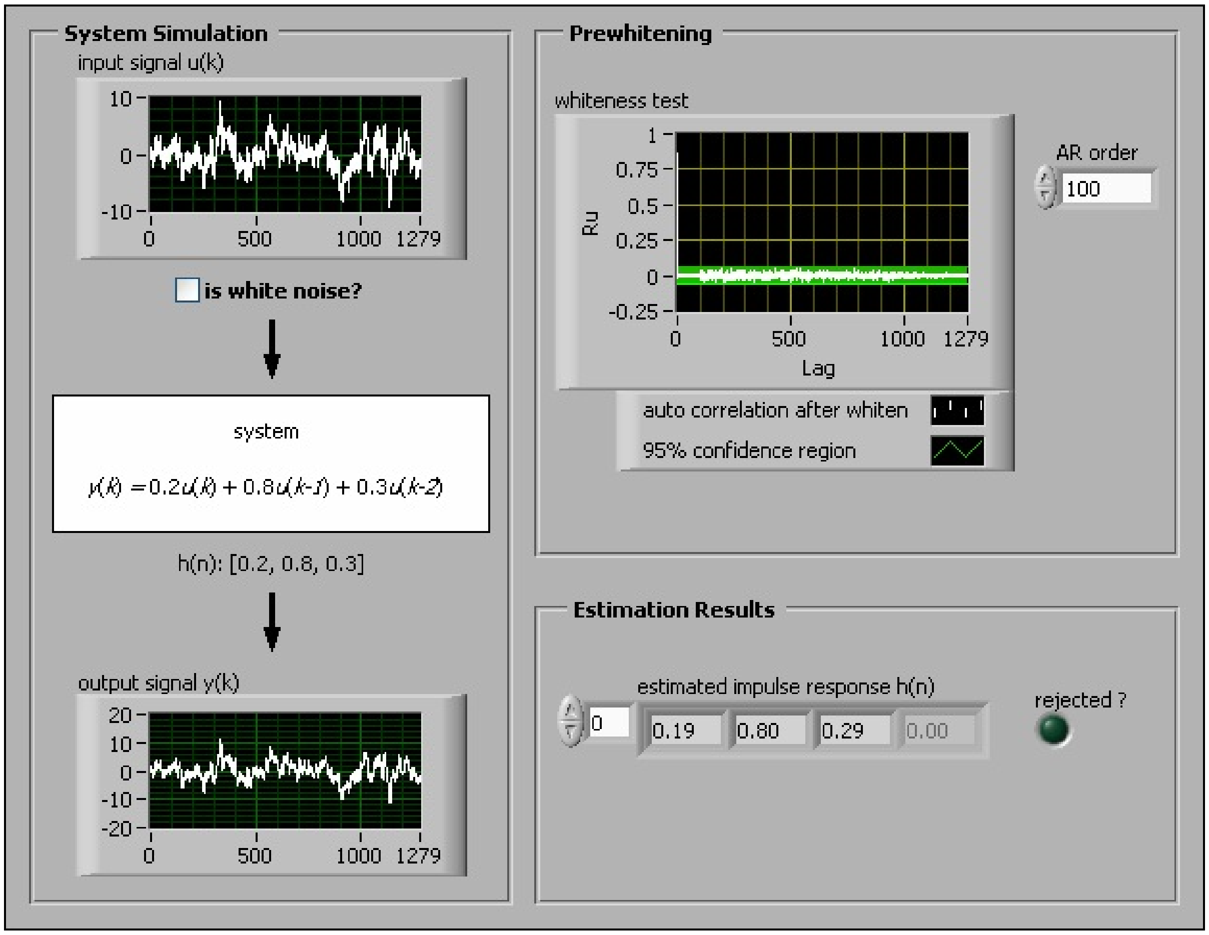
Accuracy of the Impulse Response

The accuracy of the impulse response estimation using the correlation analysis method depends on the performance of the prewhitening filter, specifically whether the filter produces a white noise result $u'(k)$ for $u(k)$. The performance of the filter depends on the signal and the AR order of the filter. The rule of thumb for selecting the AR order is trial-and-error. If $u'(k)$ is not white enough, the result from the correlation method is not reliable. You can increase the AR order to improve the accuracy of the impulse response.

The SI Estimate Impulse Response VI provides the outputs **whiteness test** and **rejected?** to indicate whether you have properly set the AR order and consequently whether the impulse response estimation is reliable. The following example shows how the whiteness property of the input signal affects the correlation analysis method and how to use the outputs **whiteness test** and **rejected?** to justify the impulse response estimation.

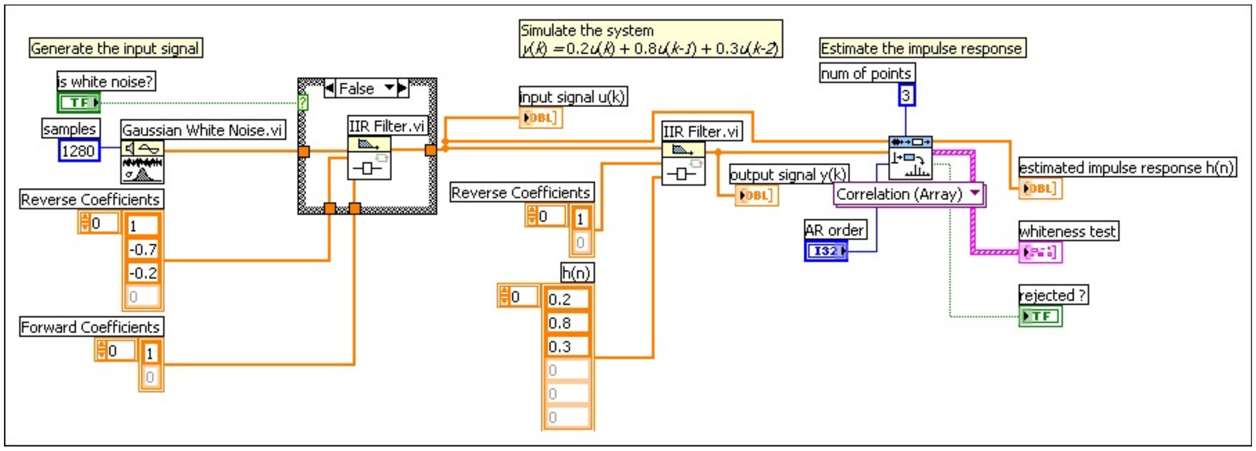
The following figure shows the front panel of a VI that simulates a system defined by the following equation.

$$y(k) = 0.2u(k) + 0.8u(k - 1) + 0.3u(k - 2)$$



The following figure shows the block diagram of this VI. This example VI demonstrates the accuracy of the impulse response estimation in the following circumstances:

- Zero-mean, pseudo-white noise input signal without prewhitening
- Zero-mean, pseudo-white noise input signal with prewhitening
- Non-zero-mean, white noise input signal without prewhitening
- Non-zero-mean white noise input signal with prewhitening



In this example VI, the **is white noise?** checkbox determines whether the SI Estimate Impulse Response VI generates zero-mean white noise as an input to the system. When you place a checkmark in the **is white noise?** checkbox and run the VI, the generated input signal is zero-mean white noise, and the estimated impulse response closely approximates the true impulse response. When you do not place a checkmark in the **is white noise?** checkbox, the generated input signal is not zero-mean white noise. As a result, the estimated impulse response is different from the true impulse response. These results indicate that the correlation analysis method is accurate and reliable when the input signal is zero-mean white noise.

The **AR order** box determines the level of prewhitening. When **AR order** equals 0, the SI Estimate Impulse Response VI does not apply prewhitening to the system. When **AR order** is small and you do not place a checkmark in the **is white noise?** checkbox, the variance of the impulse response is large because the input signal is not always white noise. The greater the value of **AR order**, the better the VI whitens the signal, but the more computation time and memory the VI requires.

The **whiteness test** indicator of this VI shows whether the input is zero-mean white noise. This indicator displays the autocorrelation of the stimulus signal after whitening. If most of the autocorrelation is within the confidence region, the input signal is well prewhitened, and the estimation of the impulse response is reliable. If the autocorrelation is outside of the confidence region, the estimation is unreliable. When the estimation is unreliable, **rejected?** is TRUE and indicates a 5% risk of rejecting an impulse response estimation that might be reliable.

If you apply proper prewhitening, the correlation analysis method is accurate and reliable for any input signal. To obtain the best prewhitening settings, start with a small **AR order** value such as 2 and observe the **whiteness test** and **rejected?** outputs of the SI Estimate Impulse Response VI. If necessary, increase the value of **AR order**. Generally, the smaller the bandwidth of the input signal, the larger the **AR order** you need. However, avoid setting the value of **AR order** greater than 500.

Selecting an Impulse Response Length (System Identification Toolkit)

Theoretically, the length of the impulse response might be infinite. For some systems, the impulse response quickly reaches zero, and the number of nonzero points is finite. For other systems, the impulse response never reaches zero. Realistically, you can obtain only the first *N* points of the impulse response due to limited signal length and limited memory size. Therefore, the SI Estimate Impulse Response VI has inputs to specify how many points of the impulse response to observe. With the least squares method, you must ensure the sum of **num of points (t<0)** and **num of points (t>=0)** is no larger than the signal length. With the correlation analysis method, you can set **num of points** to be as large as the signal length.

Frequency Response (System Identification Toolkit)

Theoretically, the results from impulse response estimation and the results from frequency response estimation are equivalent. For example, the Fourier transform of the impulse response $h(n)$, which you can compute using impulse response estimation, equals the frequency response $G(e^{j\omega})$. However, this equivalence does not hold in most real-world applications because of different preprocessing schemes in impulse response estimation and frequency response estimation.

The frequency response provides the complete frequency-domain characteristics of the system, including the passband and the natural frequency of the system. A sinusoidal input signal has the following general form:

$$u(t) = \sin(\omega^0 t)$$

The response of a linear time-invariant system to a sinusoidal input also is a sinusoidal signal. However, the response to the sinusoidal input might have a different magnitude and phase than the sinusoidal signal, as shown in the following equation.

$$y(t) = b\sin(\omega_0 t + \theta)$$

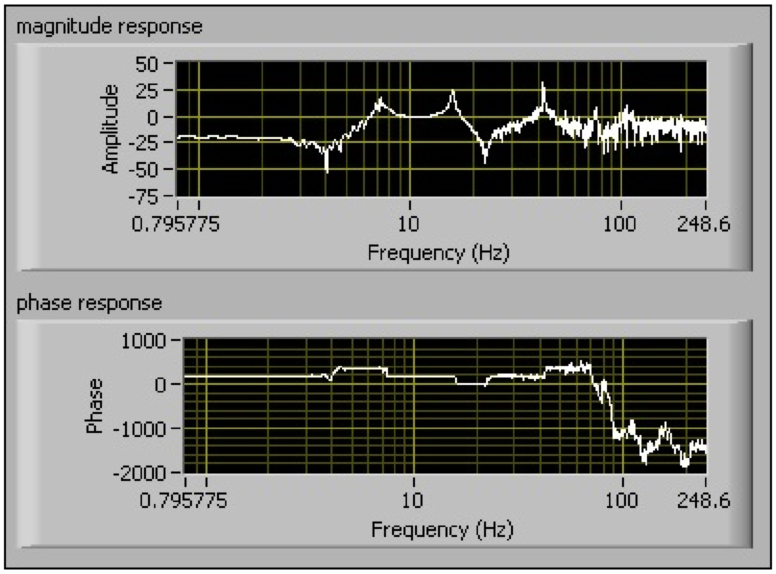
where b and θ are the magnitude and phase, respectively, of the frequency response of the system to an input sinusoidal frequency ω_o. If you apply input signals with a number of sinusoids at different frequencies, you can obtain an estimate of the frequency response $G(\omega)$ of the system at those frequencies. The frequency response is a complex-valued sequence. The magnitude of $G(\omega)$ is the magnitude response of the system and the phase of $G(\omega)$ is the phase response of the system. This method of obtaining the frequency response is straightforward but takes a long time to complete and is sensitive to noise. For these reasons, the SI Estimate Frequency Response VI uses the spectral analysis method to estimate the frequency response function (FRF).

Note You can use the SI Estimate FRF VI to estimate the FRF if you have only time-domain data available.

Applications of the Frequency Response

The frequency response gives the characteristics of the system in the frequency domain. You can use the frequency response to obtain useful information before applying parametric estimation. For example, you can use the frequency response to determine whether you must pre-filter the signals or what the model order of the system is. You also can use nonparametric frequency response to verify parametric model estimation results by comparing the frequency response of the parametric model with the nonparametric frequency response.

One example of a real-world application of the frequency response is with the flexible arm, as shown in the following figure. The input of this system is the reaction torque of the structure on the ground. This input is a multi-sine wave with 200 frequency points equally spaced over the frequency band from 0.122 Hz to 24.4 Hz. The output of this system is the acceleration of the flexible arm. The frequency response of this system is not significant outside of the range of interest, which is the frequency band of the input signal, or 0.122 Hz to 24.4 Hz. However, notice that the **magnitude response** has a peak around 42 Hz. The peak around 42 Hz may be the result of noise, or nonlinearity, or another input source. You can use lowpass filtering to remove the 42 Hz peak before applying parametric estimation.



Spectral Analysis Method (System Identification Toolkit)

You can use the spectral analysis method with any input signal. However, the frequency bandwidth of the input signal must cover the range of interest.

Because the frequency response is the Fourier transform of the impulse response, applying the Fourier transform to both sides of the cross-correlation function yields the following equation.

$$\Phi_{uy}(e^{j\omega}) = \Phi_{uu}(e^{j\omega})G(e^{j\omega})$$

$G(e^{j\omega})$ is the frequency response of the system. $\Phi_{uu}(e^{j\omega})$ is the auto-spectral density of the stimulus signal. $\Phi_{uy}(e^{j\omega})$ is the cross-spectral density between the stimulus signal $u(k)$ and the response signal $y(k)$.

You then can use the following equation to compute the frequency response $G(e^{j\omega})$.

$$G(e^{j\omega}) = \frac{\Phi_{uy}(e^{j\omega})}{\Phi_{uu}(e^{j\omega})}$$

You can compute $\Phi_{uu}(e^{j\omega})$ and $\Phi_{uy}(e^{j\omega})$ by applying a fast Fourier transform (FFT) to the autocorrelation function R_{uu} and the cross-correlation function R_{uy}, respectively. The number of data points you need to compute the autocorrelation function R_{uu} and the cross-correlation function R_{uy} decreases as the lag τ increases. Therefore, R_{uu} and R_{uy} become inaccurate for a large lag τ. In this situation, you can apply a lag window to counter the effects of a large lag τ and improve the accuracy of the frequency response estimation.

Applying a Lag Window (System Identification Toolkit)

When computing $\Phi_{uu}(e^{j\omega})$ and $\Phi_{uy}(e^{j\omega})$ to obtain the frequency response $G(e^{j\omega})$, you can apply a lag window ω(τ) to the autocorrelation function R_{uu} and the cross-correlation function R_{uy} before performing the FFT operation. Applying a lag window improves the accuracy of the frequency response estimation, according to the following equations.

$$\Phi_{uu}(e^{j\omega}) = \sum_{\tau=-N}^{N} R_{uu}(\tau)w_m(\tau)e^{-j\omega\tau}$$

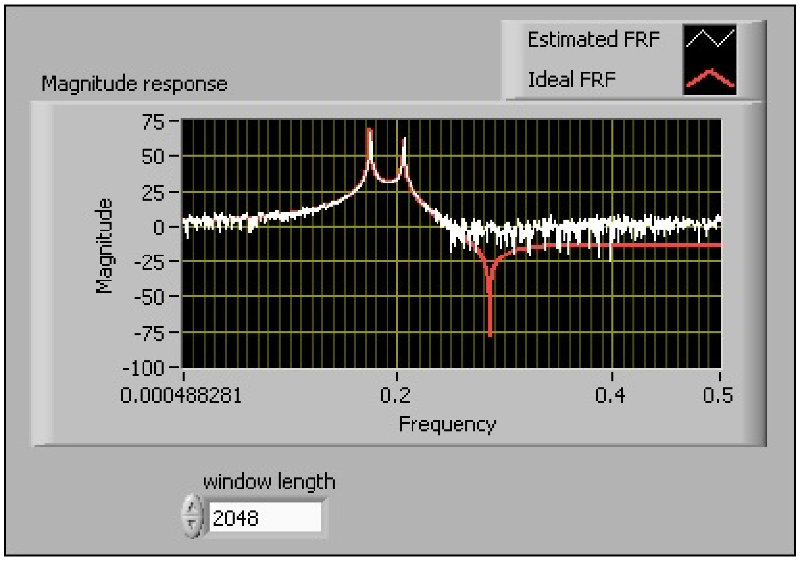
$$\Phi_{uy}(e^{j\omega}) = \sum_{\tau=-N}^{N} R_{uy}(\tau)w_m(\tau)e^{-j\omega\tau}$$

The lag window approaches zero when the lag τ is large. The window weighs out the points of R_{uu} and R_{uy} with large lag τ, thereby improving the accuracy of the frequency response estimation. The SI Estimate Frequency Response VI uses a Hanning window as the lag window.

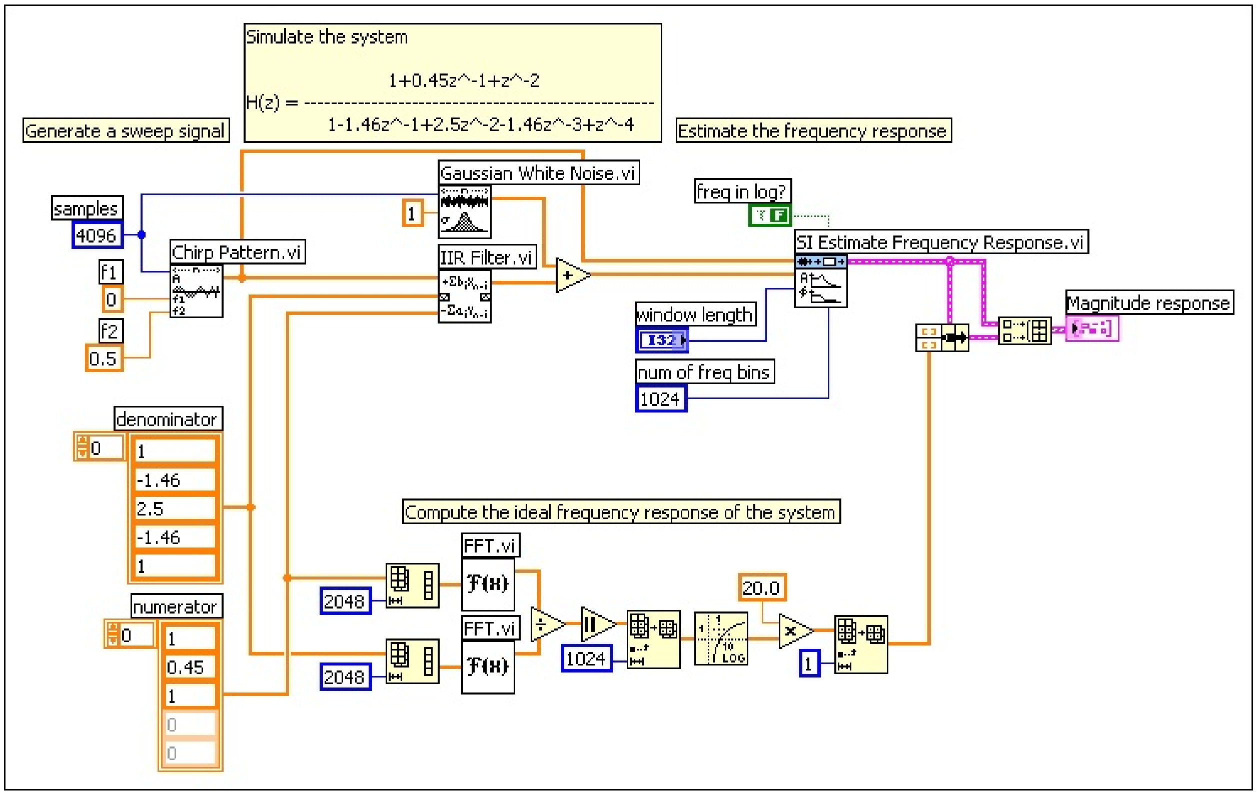
The frequency response with the lag window, $G'(e^{j\omega})$, is equivalent to the moving average version of the frequency response without the lag window, $G(e^{j\omega})$. The average smooths the frequency response, but the smooth frequency response also can deviate from the true frequency response. Adjusting the length of the lag window can balance the trade-off between variance and bias of the frequency response estimation. The larger the length of the lag window, the fewer points of $G(e^{j\omega})$ the SI Estimate Frequency Response VI averages to compute $G'(e^{j\omega})$, and hence the larger the variance and the smaller the bias of the frequency estimation.

The following example demonstrates how the length of the lag window affects the frequency response estimation. The following figure shows the front panel of a VI that simulates a system defined by the following equation.

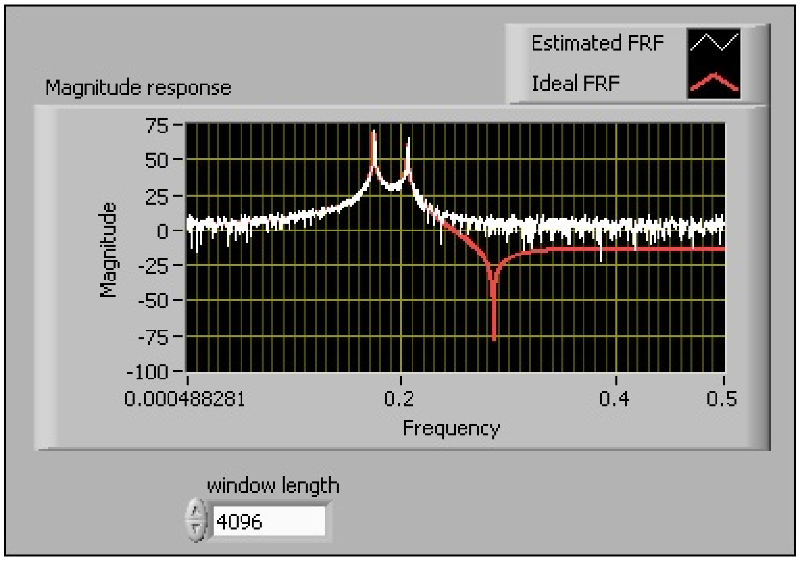
$y(k) - 1.46y(k-1) + 2.5y(k-2) - 1.46y(k-3) + yk - 4 = u(k) + 0.45u(k-1) + u(k-2)$

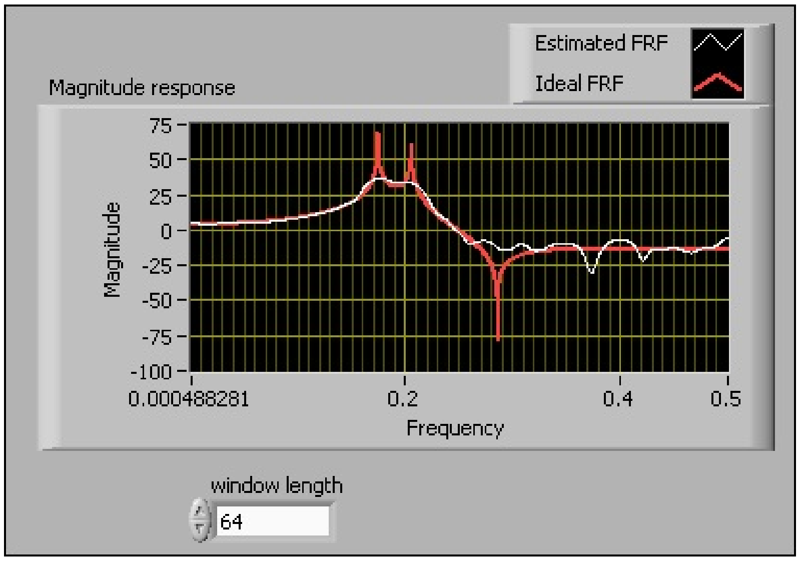


The following figure shows the block diagram of this VI.



In this example VI, the input signal *u*(*k*) is a swept sine wave whose normalized frequency is from 0 to 0.5 Hz. The number of data points in the input signal is 4096. The length of the lag window therefore must be less than or equal to 4096. The following figures show the resulting frequency responses when the window length is 4096 and 64, respectively.





The frequency response curve is smoother and the variance is smaller when the length of the lag window is small. However, when the length of the lag window is too small, you cannot distinguish between the two close peaks in the frequency response, as shown in the previous figure with window length equal to 64. When the length of the lag window is large, the SI Estimate Frequency Response VI accurately estimates the peaks, as shown in the previous figure with window length equal to 4096. The bias is small with a large lag window, but the variance of the estimated frequency response is large.

Setting the length of the lag window to 5–10% of the number of data points when estimating the frequency response often results in a good trade-off between the bias and variance. The length also depends on the signals, the properties of the system, and the purpose of application. For example, to identify the passband of a system, use a smaller lag window. To identify the dynamic properties of a system, such as its natural frequency, use a larger lag window.

Parametric Model Estimation Methods (System Identification Toolkit)

Parametric models describe systems in terms of difference or differential equations, depending on whether a system is represented by a discrete or continuous model. Compared to nonparametric models, parametric models might provide a more accurate estimation if you have prior knowledge about the system dynamics to determine model orders, time delays, and so on.

The following table lists the representations of parametric models you can develop by using the LabVIEW System Identification Toolkit. Each representation supports one or more input-output configurations: single-input single-output (SISO), multiple-input single-output (MISO), and/or multiple-input multiple-output (MIMO).

	SISO	**MISO**	**MIMO**
General-Linear	X	X	
Autoregressive (AR)	X		
Autoregressive with exogenous terms (ARX)	X	X	X
Autoregressive moving average with exogenous terms (ARMAX)	X	X	
Box-Jenkins	X	X	
Output-Error	X	X	
Transfer Function	X	X	
Zero-Pole-Gain	X	X	
State-Space	X	X	X

General-Linear Model Definitions (System Identification Toolkit)

Generally, you can describe a discrete system by using the general-linear polynomial model. This model provides flexibility for both system dynamics and stochastic dynamics.

Use the SI Estimate General Linear Model VI to estimate general-linear polynomial models. The following equation describes this model.

$y(k) = z^{-n} G(z^{-1}, \theta)u(k) + H(z^{-1}, \theta)e(k)$

where $u(k)$ and $y(k)$ are the input and output of the system, respectively

$e(k)$ is the disturbance of the system which usually is zero-mean white noise

$G(z^{-1}, \theta)$ is the transfer function of the deterministic part of the system

$H(z^{-1}, \theta)$ is the transfer function of the stochastic part of the system

The deterministic transfer function specifies the relationship between the output and the input signal. The stochastic transfer function specifies how the random disturbance affects the output signal. Often the deterministic and stochastic parts of a system are referred to as system dynamics and stochastic dynamics, respectively.

The term z^{-1} is the backward shift operator, which is defined by the following equations:

$z^{-1} x(k) = x(k - 1)$

$z^{-2} x(k) = x(k - 2)$

...

$z^{-n} x(k) = x(k - n)$

z^{-n} defines the number of delay samples between the input and the output.

$G(z^{-1}, \theta)u(k)$ and $H(z^{-1}, \theta)e(k)$ are rational polynomials as defined by the following equations:

$$G(z^{-1},\theta) = \frac{B(z,\theta)}{A(z,\theta)F(z,\theta)}$$

$$H(z^{-1},\theta) = \frac{C(z,\theta)}{A(z,\theta)D(z,\theta)}$$

The vector θ is the set of model parameters. The following equations do not display θ to make the equations easier to read.

The following equation shows the form of the general-linear model.

$$A(z)y(k) = \frac{B(z)}{F(z)}u(k-n) + \frac{C(z)}{D(z)}e(k)$$

where *y*(*k*) is the system outputs
u(*k*) is the system inputs
n is the system delay
e(*k*) is the system disturbance

A(*z*), *B*(*z*), *C*(*z*), *D*(*z*), and *F*(*z*) are polynomial with respect to the backward shift operator z^{-1} and defined by the following equations.

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

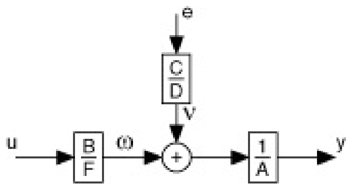
$$B(z) = b_0 + b_1 z^{-1} + \ldots + b_{k_b-1} z^{-(k_b-1)}$$

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

$$D(z) = 1 + d_1 z^{-1} + \ldots + d_{k_d} z^{-k_d}$$

$$F(z) = 1 + f_1 z^{-1} + \ldots + f_{k_f} z^{-k_f}$$

The following figure depicts the signal flow of a general-linear model.



where *u* is the system inputs
e is the system disturbance
y is the system outputs

Setting one or more of *A*(*z*), *C*(*z*), *D*(*z*), and *F*(*z*) equal to 1 can create simpler models such as autoregressive with exogenous terms (ARX), autoregressive-moving average with exogenous terms (ARMAX), output-error, and Box-Jenkins models, which you commonly use in real-world applications.

SISO

The following are the time domain equations for the general-linear SISO model.

$$w(k) + f_1 w(k-1) + \ldots + f_{k_f} w(k-k_f)$$
$$= b_0 u(k-n) + b_1 u(k-n-1) + \ldots + b_{k_b-1} u(k-n-k_b+1)$$

$$v(k) + d_1 v(k-1) + \ldots + d_{k_d} v(k-k_d)$$
$$= e(k) + c_1 e(k-1) + \ldots + c_{k_c} e(k-k_c)$$

$$y(k) + a_1 y(k-1) + \ldots + a_{k_a} y(k-k_a) = w(k) + v(k)$$

where k_f is the *F* order

k_b is the *B* order

k_c is the *C* order

k_d is the *D* order

k_a is the *A* order

$u(k)$ is the system inputs

n is the system delay

$e(k)$ is the system disturbance

Refer to the Estimate Polynomial Models VI in the labview\examples\System Identification\Getting Started\Parametric Estimation.llb for an example that demonstrates how to estimate general-linear polynomial models for an unknown system.

Open example Browse related examples

AR Model Definitions (System Identification Toolkit)

The autoregressive (AR) model does not include the dynamics between the system input and output. Therefore, the AR model is more suitable for representing signals rather than a system because a system generally has an input and an output. Time series analysis methods, such as power spectrum envelope estimation, prewhitening, and linear prediction coding, commonly use the AR model. Refer to the *Time Series Analysis Tools User Manual* at ni.com/manuals for more information about time series analysis methods.

Use the SI Estimate AR Model VI to estimate AR system models. The following equation shows the form of the AR model.

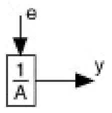
$A(z)y(k) = e(k)$

where	$y(k)$ is the system outputs
	$e(k)$ is the system disturbance

$A(z)$ is polynomial with respect to the backward shift operator z^{-1} and defined by the following equation.

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

The following figure depicts the signal flow of an AR system model.



where	e is the system disturbance
	y is the system outputs

If you consider $A(z)$ to be a filter, $A(z)y(k)$ is the filtering of $A(z)$ on the signal $y(k)$. The result of the filtering is white noise $e(k)$, as shown in the AR model equation. Hence, the filter $A(z)$ also is known as the prewhitening filter. From the frequency-domain standpoint, the prewhitening filter $A(z)$ suppresses the spectrum at frequencies where the magnitude of the spectrum is large. Suppressing the high-magnitude frequencies results in a flat spectrum.

As shown in the AR model equation, if you know the AR coefficients $A(z)$

and the noise $e(k)$, you can reconstruct the signal $y(k)$. $A(z)$ and $e(k)$ completely characterize a signal. $A(z)$ normally has a small number of coefficients. $e(k)$ has a small dynamic range and requires a smaller number of bits for encoding. Therefore, you can use the AR model for compression purposes in a process known as linear prediction coding (LPC). Speech and vibration signal processing methods, such as compression and pattern recognition, commonly use LPC. You also can use $A(z)$ and $e(k)$ to estimate the power spectrum of the signal $y(k)$.

ARX Model Definitions (System Identification Toolkit)

When $C(z)$, $D(z)$, and $F(z)$ equal 1, the general-linear polynomial model reduces to an autoregressive with exogenous terms (ARX) model. This model is the simplest model that incorporates the stimulus signal. However, the ARX model captures some of the stochastic dynamics as part of the system dynamics. In this model, the transfer function of the deterministic part $G(z^{-1}, \theta)$ of the system and the transfer function of the stochastic part $H(z^{-1}, \theta)$ of the system have the same set of poles. This coupling can be unrealistic. The system dynamics and stochastic dynamics of a system do not share the same set of poles all the time. You can reduce this disadvantage if the signal-to-noise ratio is high.

When the disturbance $e(k)$ of a system is not white noise, the coupling between the deterministic and stochastic dynamics can bias the estimation of the ARX model. You can set the model order higher than the actual model order to minimize the estimation error, especially when the signal-to-noise ratio is low. However, increasing the model order can change some dynamic characteristics of the model, such as the stability of the model.

Use the SI Estimate ARX Model VI to estimate ARX models. The identification method for the ARX model is the least squares method, which is a special case of the prediction error method. The least squares method is the most efficient polynomial estimation method because this method solves linear regression equations in analytic form. Moreover, the solution is unique. Refer to the LabVIEW System Identification Toolkit Algorithm References manual for more information about the least squares and prediction error methods.

The following equation shows the form of the ARX model.

$$A(z)y(k) = B(z)u(k - n) + e(k)$$

where	$u(k)$ is the system inputs
	$y(k)$ is the system outputs
	n is the system delay
	$e(k)$ is the system disturbance

$A(z)$ and $B(z)$ are polynomial with respect to the backward shift operator z

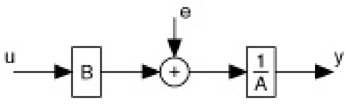
$^{-1}$ and defined by the following equations.

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

$$B(z) = b_0 + b_1 z^{-1} + \ldots + b_{k_b - 1} z^{-(k_b - 1)}$$

Note The backward shift operator makes $z^{-n} u(k) = u(k - n)$.

The following figure depicts the signal flow of an ARX model.



where u is the system inputs

e is the system disturbance

y is the system outputs

SISO

The following is the time domain equation for the ARX SISO model.

$$y(k) + a_1 y(k-1) + \ldots + a_{k_a} y(k-k_a)$$
$$= b_0 u(k-n) + b_1 u(k-n-1) + \ldots + b_{k_b-1} u(k-n-k_b+1) + e(k)$$

where k_A order

k_b is the *B* order

n is the system delay

e(*k*) is the system disturbance

Refer to the Estimate Polynomial Models VI in the labview\examples\System Identification\Getting Started\Parametric Estimation.llb for an example that demonstrates how to estimate ARX models for an unknown system.

■ Open example ■ Browse related examples

ARMAX Model Definitions (System Identification Toolkit)

When $D(z)$ and $F(z)$ equal 1, the general-linear polynomial model reduces to an autoregressive-moving average with exogenous terms (ARMAX) model. Unlike the autoregressive with exogenous terms (ARX) model, the system structure of an ARMAX model includes the stochastic dynamics. ARMAX models are useful when you have dominating disturbances that enter early in the process, such as at the input. For example, a wind gust affecting an aircraft is a dominating disturbance early in the process. The ARMAX model has more flexibility than the ARX model in handling models that contain disturbances.

Use the SI Estimate ARMAX Model VI to estimate ARMAX models. This VI uses the Gauss-Newton method to optimize the mean square value of the prediction error when searching for the optimal ARMAX model. This searching process is iterative and might converge to a local minimum. Therefore, you must validate the estimated model. If the estimated model passes the validation test, you can use this model even if the SI Estimate ARMAX Model VI might locate only a local minimum.

The following equation shows the form of the ARMAX model.

$A(z)y(k) = B(z)u(k - n) + C(z)e(k)$

where $y(k)$ is the system outputs

$u(k)$ is the system inputs

n is the system delay

$e(k)$ is the system disturbance

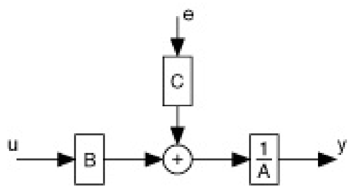
$A(z)$, $B(z)$, and $C(z)$ are polynomial with respect to the backward shift operator z^{-1} and defined by the following equations.

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

$$B(z) = b_0 + b_1 z^{-1} + \ldots + b_{k_b - 1} z^{-(k_b - 1)}$$

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

The following figure depicts the signal flow of an ARMAX model.



where *u* is the system inputs

e is the system disturbance

y is the system outputs

SISO

The following is the time domain equation for the ARMAX SISO model.

$$\begin{aligned}&y(k) + a_1 y(k-1) + \ldots + a_{k_a} y(k-k_a)\\&= b_0 u(k-n) + b_1 u(k-n-1) + \ldots + b_{k_b-1} u(k-n-k_b+1)\\&+ e(k) + c_1 e(k-1) + \ldots + c_{k_c} e(k-k_c)\end{aligned}$$

where k_a is the *A* order

k_b is the *B* order

k_c is the *C* order

n is the system delay

e(*k*) is the system disturbance

Refer to the Estimate Polynomial Models VI in the labview\examples\System Identification\Getting Started\Parametric Estimation.llb for an example that demonstrates how to estimate ARMAX models for an unknown system.

Open example Browse related examples

Box-Jenkins Model Definitions (System Identification Toolkit)

When *A*(*z*) equals 1, the general-linear polynomial model reduces to the Box-Jenkins model. This model provides a complete model of a system because this model represents disturbance properties separately from system dynamics. This model is useful when you have disturbances that enter late in the process, such as measurement noise on the output.

Use the SI Estimate BJ Model VI to estimate Box-Jenkins models. The identification method of the Box-Jenkins model is the prediction error method, which is the same as that of the ARMAX model.

The following equation shows the form of the Box-Jenkins model.

$$y(k) = \frac{B(z)}{F(z)}u(k-n) + \frac{C(z)}{D(z)}e(k)$$

where *y*(*k*) is the system outputs

u(*k*) is the system inputs

n is the system delay

e(*k*) is the system disturbance

B(*z*), *C*(*z*), *D*(*z*), and *F*(*z*) are polynomial with respect to the backward shift operator z^{-1} and defined by the following equations.

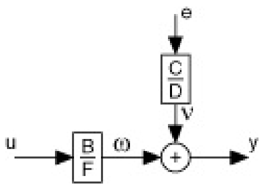
$$B(z) = b_0 + b_1 z^{-1} + \dots + b_{k_b-1} z^{-(k_b-1)}$$

$$C(z) = 1 + c_1 z^{-1} + \dots + c_{k_c} z^{-k_c}$$

$$D(z) = 1 + d_1 z^{-1} + \dots + d_{k_d} z^{-k_d}$$

$$F(z) = 1 + f_1 z^{-1} + \dots + f_{k_f} z^{-k_f}$$

The following figure depicts the signal flow of a Box-Jenkins model.



where *u* is the system inputs

e is the system disturbance

y is the system outputs

SISO

The following are the time domain equations for the Box-Jenkins SISO model.

$$w(k) + f_1 w(k-1) + \ldots + f_{k_f} w(k-k_f)$$
$$= b_0 u(k-n) + b_1 u(k-n-1) + \ldots + b_{k_b-1} u(k-n-k_b+1)$$

$$v(k) + d_1 v(k-1) + \ldots + d_{k_d} v(k-k_d)$$
$$= e(k) + c_1 e(k-1) + \ldots + c_{k_c} e(k-k_c)$$

$$y(k) = w(k) + v(k)$$

where k_f is the *F* order

k_b is the *B* order

k_c is the *C* order

k_d is the *D* order

u(*k*) is the system input

n is the system delay

e(*k*) is the system disturbance

Refer to the Estimate Polynomial Models VI in the labview\examples\System Identification\Getting Started\Parametric Estimation.llb for an example that demonstrates how to estimate Box-Jenkins models for an unknown system.

Open example Browse related examples

Output-Error Model Definitions (System Identification Toolkit)

When $A(z)$, $C(z)$, and $D(z)$ equal 1, the general-linear polynomial model reduces to the output-error model. This model describes the system dynamics separately from the stochastic dynamics. The output-error model does not use any parameters for simulating the disturbance characteristics.

Use the SI Estimate OE Model VI to estimate output-error models. The identification method of the output-error model is the prediction error method, which is the same as that of the ARMAX model. If the disturbance $e(k)$ is white noise, all minima are global. However, a local minimum can exist if the disturbance is not white noise.

The following equation shows the form of the output-error model.

$$y(k) = \frac{B(z)}{F(z)}u(k-n) + e(k)$$

where $y(k)$ is the system outputs

$u(k)$ is the system inputs

n is the system delay

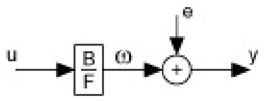
$e(k)$ is the system disturbance

$B(z)$ and $F(z)$ are polynomials with respect to the backward shift operator z^{-1} and defined by the following equations.

$$B(z) = b_0 + b_1 z^{-1} + \ldots + b_{k_b-1} z^{-(k_b-1)}$$

$$F(z) = 1 + f_1 z^{-1} + \ldots + f_{k_f} z^{-k_f}$$

The following figure depicts the signal flow of an output-error model.



where u is the system inputs

e is the system disturbance

y is the system outputs

SISO

The following are the time domain equations for the output-error SISO model.

$$w(k) + f_1 w(k-1) + \ldots + f_{k_f} w(k-k_f)$$
$$= b_0 u(k-n) + b_1 u(k-n-1) + \ldots + b_{k_b-1} u(k-n-k_b+1)$$

$$y(k) = w(k) + e(k)$$

where k_f is the *F* order

k_b is the *B* order

n is the system delay

e(*k*) is the system disturbance

Refer to the Estimate Polynomial Models VI in the labview\examples\System Identification\Getting Started\Parametric Estimation.llb for an example that demonstrates how to estimate Output-Error models for an unknown system.

▪ Open example ▪ Browse related examples

Transfer Function Model Definitions (System Identification Toolkit)

You can use a transfer function to define either a continuous system or a discrete system. The following equations describe a continuous system and a discrete system, respectively, from which the transfer function is derived.

$y(t) = G(s)u(t) + e(t)$

$y(k) = G(z)u(k) + e(k)$

where $y(t)$ and $y(k)$ are the system outputs

$G(s)$ and $G(z)$ is the transfer function between the stimulus and the response

$u(t)$ and $u(k)$ are the system inputs

$e(t)$ and $e(k)$ are the system disturbance

Note Continuous models use the *s* variable to define time whereas discrete models use the *z* variable.

SISO

The following is the equation for the continuous transfer function SISO model.

$$G(s) = \frac{b_0 + b_1 s + \ldots + b_{m-1} s^{m-1} + b_m s^m}{a_0 + a_1 s + \ldots + a_{n-1} s^{n-1} + a_n s^n} e^{-sT_d}$$

Note For transfer function SISO models based on frequency-domain data, T_d equals zero.

The following is the equation for the discrete-time transfer function SISO model.

$$G(z) = \frac{b_0 + b_1 z + \ldots + b_{m-1} z^{m-1} + b_m z^m}{a_0 + a_1 z + \ldots + a_{n-1} z^{n-1} + a_n z^n}$$

MISO

The following is the equation for the continuous transfer function MISO model.

$$y_i = \sum_{j=1}^{n} G_{ij}(s)u_j$$

The following is the equation for the discrete-time transfer function MISO model.

$$y_i = \sum_{j=1}^{n} G_{ij}(z)u_j$$

You can use the SI Estimate Transfer Function Model VI to estimate both continuous and discrete models. For discrete models, this VI implements the prediction error method. For continuous models, this VI internally performs the following three consecutive steps to estimate the model:

1. Calculates a discrete model with the prediction error method.
2. Applies the Zero-Order-Hold method to convert the discrete model to a continuous model.
3. Uses the Gauss-Newton method to optimize the continuous model this VI converted in step 2.

You can use the SI Estimate Transfer Function Model from FRF VI to estimate both continuous and discrete SISO models in the frequency domain.

Transfer function models describe only the deterministic part of the system. For stochastic control, general-linear polynomial models commonly are used because these models separately describe the deterministic and stochastic parts of a system. However, in classical control engineering, the deterministic part of the system is more important than the stochastic part. Therefore, you can take advantage of the relationship between input and output signals of the transfer function model to describe the deterministic part of the system.

Zero-Pole-Gain Model Definitions (System Identification Toolkit)

If you rewrite the equations for the transfer function model to show the locations of the zeroes and poles of the dynamic system, you obtain the zero-pole-gain model.

SISO

The following is the equation for the continuous zero-pole-gain SISO model.

$$G(s) = \frac{k(s - Z_1)(s - Z_2)\ldots(s - Z_m)}{(s - P_1)(s - P_2)\ldots(s - P_n)}$$

The following is the equation for the discrete-time zero-pole-gain SISO model.

$$G(z) = \frac{k(z - Z_1)(z - Z_2)\ldots(z - Z_m)}{(z - P_1)(z - P_2)\ldots(z - P_n)}$$

where *k* is the transfer function gain, Z_i are the zeroes, and P_j are the poles. When *s* or *z* equals 0, you can calculate the static gain from the two equations.

$$\text{static gain} = (-1)^{m-n} \frac{kz_1z_2\ldots z_m}{P_1P_2\ldots P_n}$$

MISO

The following is the equation for the continuous zero-pole-gain MISO model.

$$y_i = \sum_{j=1}^{n} G_{ij}(s)u_j$$

The following is the equation for the discrete-time zero-pole-gain MISO model.

$$y_i = \sum_{j=1}^{n} G_{ij}(z)u_j$$

The LabVIEW System Identification Toolkit does not provide a VI to estimate zero-pole-gain models directly because you can use the SI Model Conversion VI to convert another model representation to a zero-pole-gain model.

State-Space Model Definitions (System Identification Toolkit)

The state-space model is the most convenient model for describing multiple-input multiple-output (MIMO) systems. State-space models often are preferable to polynomial models, especially in modern control applications that focus on multivariable systems. You can estimate both continuous and discrete state-space models.

Continuous

Use partially known model estimation methods to estimate continuous state-space models. You must provide an initial guess for each parameter before conducting estimation. The following equations show the form of the continuous state-space model.

$$\dot{x} = \boldsymbol{A}x + \boldsymbol{B}u + \boldsymbol{K}e$$
$$y = \boldsymbol{C}x + \boldsymbol{D}u + e$$

Discrete

Use the SI Estimate State-Space Model and SI Estimate State-Space Model from FRF VIs to estimate discrete state-space models. The SI Estimate State-Space Model VI supports the following two estimation methods:

- **Deterministic-stochastic subspace method**—This method uses principal component analysis to estimate parameters. This method uses both stimulus and response signals to estimate state-space models. This method includes the stochastic parts of the system in the model structure.
- **Realization method**—This method uses the impulse response to estimate only the deterministic state-space model. This method does not include stochastic parts of the system in the model structure.

Refer to the LabVIEW System Identification Toolkit Algorithm References manual for more information about the deterministic-stochastic subspace method and the realization method.

The following equations show the form of the discrete state-space model.

$$x(k+1) = \boldsymbol{A}x(k) + \boldsymbol{B}u(k) + \boldsymbol{K}e(k)$$

$$y(k) = \boldsymbol{C}x(k) + \boldsymbol{D}u(k) + e(k)$$

Note The equations for the discrete state-space model based on frequency-domain data do not contain $\boldsymbol{K}e(k)$ and $e(k)$.

where $\boldsymbol{A}$ is an $n \times n$ state matrix of the given system

$\boldsymbol{B}$ is an $n \times m$ input matrix of the given system

$\boldsymbol{C}$ is an $r \times n$ output matrix of the given system

$\boldsymbol{D}$ is an $r \times m$ direct transmission matrix of the given system

$\boldsymbol{K}$ is the Kalman gain matrix

k is the model sampling time multiplied by the discrete time step, where the discrete time step equals 0, 1, 2, …

n is the number of model states

m is the number of model inputs

r is the number of model outputs

x is the model state vector

u is the model input vector

y is the model output vector

e(*k*) is the system disturbance

The state-space transfer matrices ***A***, ***B***, ***C***, and ***D*** often reflect physical characteristics of a system. The dimension of the state vector *x* is the only setting you must provide for the state-space model.

User Defined Model Definitions (System Identification Toolkit)

If a general-linear polynomial, transfer function, zero-pole-gain, or state-space models model cannot represent the model you want to estimate, you can define a model by revising a template VI. You can find template VIs in the \vi.lib\addons\System Identification\User-Defined Model Templates.llb. Then you can estimate the model you define using the SI Estimate User-Defined Model VI. This VI enables you to estimate some other model representations in addition to the general-linear, transfer function, zero-pole-gain, and state-space models that the LabVIEW System Identification Toolkit directly supports. For example, you can use this VI to estimate nonlinear models. With this VI, you also can estimate linear models that you define early.

Comparing Polynomial and State-Space Models (System Identification Toolkit)

Selecting the correct model type and model order is crucial for successfully estimating a parametric model. In general, state-space models provide a more complete representation of the system, especially for multiple-input multiple-output (MIMO) systems, than polynomial models because state-space models are similar to first principle models that can provide more degree of freedom in describing MIMO systems.

The identification procedure for state-space models does not involve nonlinear optimization so the estimation reaches a solution regardless of the initial guess. Moreover, the parameter settings for the state-space model are simpler. You need to select only the order, or the number of states, of the model. The order can come from prior knowledge of the system. You also can determine the order by analyzing the singular values of the information matrix. However, the states that the state-space identification procedure identifies might not reflect the physical characteristics of a system accurately. Using a similarity transformation, you can identify equivalent models with states that better represent the system. Similarity transformations enable you to transform the states without misrepresenting the input-output behavior of the system.

When model order is high, state-space models are preferable to polynomial models. Polynomial models with high order might encounter numerical problems in computation.

Determining Parameters for the Prediction Error Method (System Identification Toolkit)

The identification method for most of the polynomial models is the prediction error method. Determining the delay and model order for the prediction error method is typically a trial-and-error process. The following steps can help you obtain a suitable model. These steps are not the only methods you can use, nor are these steps a comprehensive procedure.

1. Obtain useful information about the model order by observing the number of resonance peaks in the nonparametric frequency response function. Normally, the number of peaks in the magnitude response equals half the order of $A(z)F(z)$.
2. Obtain a reasonable estimate of the delay by observing the impulse response or by testing reasonable values in a medium-sized ARX model. Choose the delay that provides the best model fit based on prediction errors or another criterion.
3. Test various ARX model orders with this delay, choosing those orders that provide the best fit.
4. Reduce the model order by plotting the poles and zeros with confidence intervals and looking for potential cancellations of pole-zero pairs. The resulting model might be unnecessarily high in order because the ARX model describes both the system dynamics and noise properties using the same set of poles. The ARMAX, output-error, and Box-Jenkins models use the resulting orders of the poles and zeros as the *B* and *F* model parameters and the first- or second-order models for the noise characteristics.
5. Determine if additional signals influence the output if you cannot obtain a suitable model at this point. You can incorporate measurements of these signals as extra input signals.

If you still cannot obtain a suitable model, additional physical insight into the problem might be necessary. Compensating for nonlinear sensors or actuators and handling important physical nonlinearities often are necessary in addition to using a ready-made model.

From the prediction error standpoint, the higher the order of the model is, the better the model fits the data because the model has more degrees of freedom. However, you need more computation time and memory for higher orders. The parsimony principle says to choose the model with the smallest degree of freedom, or number of parameters, if all the models fit the data well and pass the verification test. The criteria to assess the model order therefore not only must rely on the prediction error but also must incorporate a penalty when the order increases. Akaike's Information Criterion (AIC), Final Prediction Error Criterion (FPE), and the Minimum Description Length Criterion (MDL) are criteria you can use to estimate the model order. The SI Estimate Orders of System Model VI implements the AIC, FPE, and MDL methods to search for the optimal model order in the range of interest. You also can plot the prediction error as a function of the model dimension and then visually find the minimum in the curve or apply an F-test to obtain an appropriate estimation of the model order.

Akaike's Information Criterion

The Akaike's Information Criterion (AIC) is a weighted estimation error based on the unexplained variation of a given time series with a penalty term when exceeding the optimal number of parameters to represent the system. For the AIC, an optimal model is the one that minimizes the following equation:

$$AIC = V_n\left(1 + \frac{2p}{N}\right)$$

N is the number of data points, V_n is an index related to the prediction error, or the residual sum of squares, and *p* defines the number of parameters in the model.

Final Prediction Error Criterion

The Final Prediction Error Criterion (FPE) estimates the model-fitting error when you use the model to predict new outputs. For the FPE, an optimal model is the one that minimizes the following equation:

$$FPE = V_n\left(1 + \frac{2p}{N - p}\right)$$

You want to choose a model that minimizes the FPE, which represents a balance between the number of parameters and the explained variation.

Minimum Description Length Criterion

The Minimal Description Length Criterion (MDL) is based on V_n plus a penalty for the number of terms used. For the MDL, an optimal model is the one that minimizes the following equation:

$$MDL = V_n\left(1 + \frac{p \ln N}{N}\right)$$

You want to choose a model that minimizes the MDL, which allows the shortest description of data you measure.

Partially Known Model Estimation Methods (System Identification Toolkit)

The nonparametric and parametric model estimation methods, also known as black-box estimation methods, assume that systems are unknown. Because these estimation methods do not take prior system knowledge into account, you must use either an algorithm or trial-and-error to vary model parameters until the behavior of the model matches the measured input-output data. Although you can use the estimated parameters to reproduce the response of the system accurately, these parameters might not have any physical meanings.

However, in practice, many systems are partially known because you have information about the underlying dynamics or some of the physical parameters. You can use partially known model estimation methods, also known as grey-box estimation methods, to estimate models when you have this information.

Black-box methods also assume that all model parameters are adjustable. However, in many real-world applications, you cannot adjust all the parameters arbitrarily, because the parameters might have constraints. For example, in some chemical processes, water must flow only in one direction. When estimating the flow rate of water, you know that the flow rate cannot be negative. Thus, the constraint is that the flow rate must be a positive value. You must consider this constraint and any other constraints when you estimate the flow rate of water in this process. Such constraints usually follow one of the following guidelines:

- A parameter must be as close to a value as possible.
- A parameter must be between two values.
- Two or more parameters must correlate to each other.

These constraints reflect the knowledge you have of the physical system. This knowledge can result in a more realistic parameter estimation. Parameter constraints increase the possibility of the System Identification VIs locating the optimal parameters that describe the real-world model. Parameter constraints also improve the accuracy of locating these optimal parameters.

You can set parameter constraints when using grey-box estimation methods, whereas you cannot set parameter constraints when using black-box estimation methods. When using these methods, you can set only the model order that specifies the number of parameters to calculate.

Defining and Estimating Partially Known Models (System Identification Toolkit)

Before estimating partially known models, you first must define those models. Using prior knowledge, you choose a model for the plant in a system and set parameter constraints for the model. You then can estimate the model to represent the real-world plant. The LabVIEW System Identification Toolkit provides two VIs you can use to define partially known models—the SI Create Partially Known State-Space Model VI and the SI Create Partially Known Continuous Transfer Function Model VI.

Use the SI Create Partially Known State-Space Model VI to define partially known continuous or discrete state-space models.

You can use the SI Create Partially Known State-Space Model VI, for example, to define a state-space model that represents an RLC circuit consisting of a resistor *R*, an inductor *L*, and a capacitor *C*. Using prior knowledge, you describe the relationship of *R*, *L*, and *C* with the following equations:

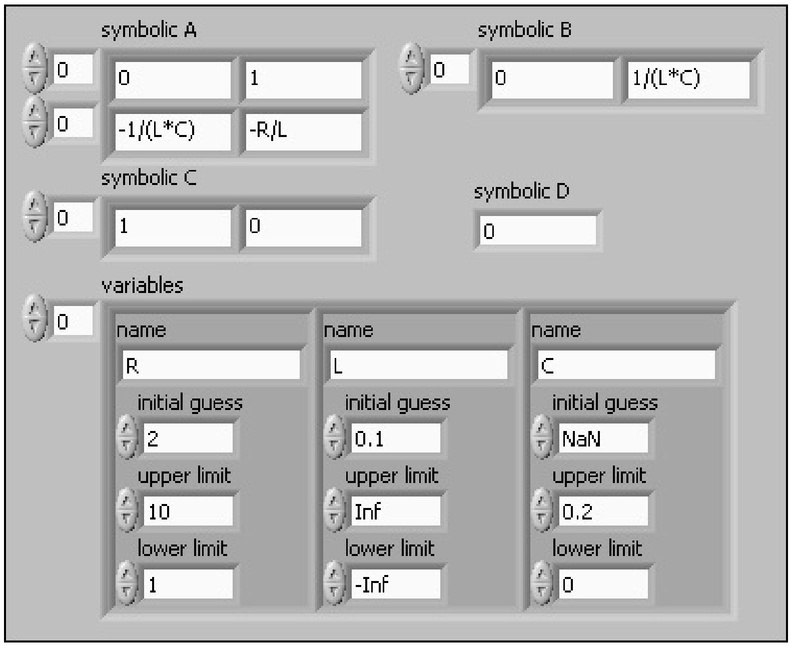
$$A = \begin{bmatrix} 0 & 1 \\ -1/(L \times C) & -R/L \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 1/(L \times C) \end{bmatrix}$$

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

$D = 0$

You also can use prior knowledge to define the initial guesses and upper and lower limits of *R*, *L*, and *C*. The SI Create Partially Known State-Space Model VI uses variables rather than numerical values to construct a symbolic model. As the following figure shows, you use variable names, such as *R*, *L*, and *C*, in the **symbolic A**, **symbolic B**, **symbolic C**, and **symbolic D** inputs to define the RLC circuit. Then you specify values for *R*, *L*, and *C* in the **variables** input.



symbolic A

0	0	1
0	-1/(L*C)	-R/L

symbolic B

0	0	1/(L*C)

symbolic C

0	1	0

symbolic D

0

variables

0

name	name	name
R	L	C
initial guess	initial guess	initial guess
2	0.1	NaN
upper limit	upper limit	upper limit
10	Inf	0.2
lower limit	lower limit	lower limit
1	-Inf	0

Use the SI Create Partially Known Continuous Transfer Function Model VI to define partially known continuous transfer function models. The following equation represents a continuous transfer function model.

$$\frac{K_p\left(1+a_1 s+a_2 s^2\ldots\right)}{\prod_i\left(1+T_p^i s\right)\prod_j\left(1+2r^j s/w^j+\left(s/w^j\right)^2\right)}e^{-sT_d}$$

where K_p is the transfer function gain

T_d is the delay

T_p is the first-order time constant

w is the natural frequency

r is the damping ratio

s represents the time

You can apply the prior knowledge you have about the parameters K, T_d, T_p, w, and r to the **static gain**, **delay(s)**, **T$_\mathrm{p}$(s)**, **natural freq (rad/s)**, and **damping ratio** inputs, respectively, of the SI Create Partially Known Continuous Transfer Function Model VI by defining the initial guesses and upper and lower limits.

With the System Identification Toolkit and a partially known model, you can set constraints on each parameter of a state-space or continuous transfer function model in two ways—with an upper and lower limit or with an initial guess.

Setting Parameter Constraints with a Range (System Identification Toolkit)

If you have prior knowledge of a parameter, you can set constraints by providing upper and lower limits for the parameter. With the limit range, the SI Estimate Partially Known State-Space Model VI randomly selects a value within the range as an initial guess of the parameter. From this initial value, the VI then performs optimization to minimize the difference between the estimated output and the measured real output. The goal of constraint optimization is to find a global optimum, or the smallest difference between the estimated output and the real output, with parameters of physical meaning. Successfully finding the global optimum depends on the limit range you set and the random initial value the SI Estimate Partially Known State-Space Model VI selects.

To increase the possibility of finding the global optimum, complete the following steps:

1. Use prior knowledge to set the range as narrow as possible.
2. Perform multiple estimates with the range you set. You might get different optimization results because the SI Estimate Partially Known State-Space Model VI randomly selects an initial value within the range each time you run the VI. If you repeatedly obtain the same result, this result might be the optimum you want to find. If you obtain inconsistent results, either choose the result that best meets the system requirements, or continue with step 3 to adjust the limit range.
3. Select one of the previous results you got in step 2 according to the prior knowledge you have of the system. Narrow the range in which the result falls. Run the SI Estimate Partially Known State-Space Model VI multiple times. A consistent result you get might be the optimum you want to find. Otherwise, repeat this step until you find a consistent result.

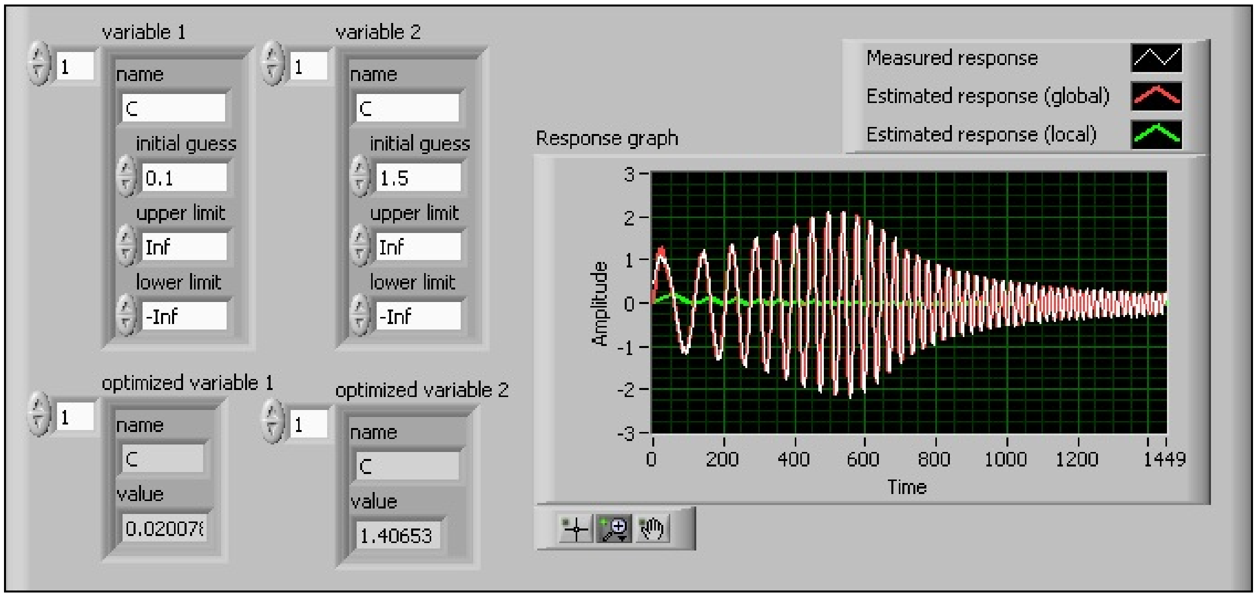
You set limits in the SI Estimate Partially Known Continuous Transfer Function Model VI the same way you do in the SI Estimate Partially Known State-Space Model VI.

Setting Parameter Constraints with an Initial Guess (System Identification Toolkit)

If you have information about a certain parameter and can estimate a value for that parameter, you can refine estimation by using that value as an initial guess.

The SI Estimate Partially Known State-Space Model VI and the SI Estimate Partially Known Continuous Transfer Function Model VI perform optimization using the initial guess you provide. These two VIs then use the upper and lower limit settings you specify as boundary constraints during the optimization process.

The initial guess you provide greatly affects the performance of any optimization technique. Whether an optimization process reaches a global optimum depends on the initial guess. With some initial guesses, optimization processes might locate only a local optimum, which is the smallest difference between the estimated output and the real output within a certain smaller range rather than in the whole range of interest. Therefore, to decrease the risk of locating a local optimum instead of the global optimum, try different initial guesses. The following figure shows an example of different estimations resulting from different initial guesses and illustrates the importance of setting different initial guesses to find the global optimum.



variable 1: 1
name: C
initial guess: 0.1
upper limit: Inf
lower limit: -Inf

variable 2: 1
name: C
initial guess: 1.5
upper limit: Inf
lower limit: -Inf

optimized variable 1: 1
name: C
value: 0.02007(

optimized variable 2: 1
name: C
value: 1.40653

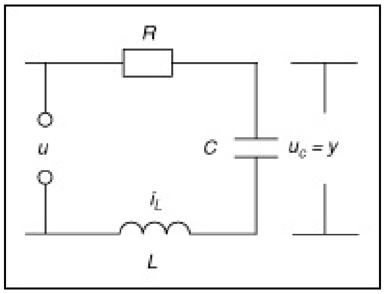
As the previous figure shows, if you set *C* to an initial guess of 0.1, you obtain an optimized value of 0.02. You can see the **Estimated response**

(global) plot and the **Measured response** plot match in the **Response graph**. This response from the estimated model is close to the real-world model response. However, if you set the initial guess of *C* to 1.5, you get an optimized value of 1.41. The **Estimated response (local)** plot does not match the **Measured response** plot in the **Response graph**. Thus, with this initial guess, the estimated model response does not represent the real-world model response accurately.

Partially Known Model Estimation Case Study (System Identification Toolkit)

This case study gives an example that uses the prior knowledge you have about a system to define and estimate state-space models. You use the same procedure when estimating continuous transfer function models. However, you apply different methods to define continuous transfer function models.

The following figure shows an RLC circuit, where *u* is the input voltage, *y* is the output voltage, i_L is the current, and u_C is the capacitor voltage. In this example, *y* equals the capacitor voltage u_C.



Suppose *R* is 1.5 Ω and *L* and *C* are unknown. You can complete the following steps to identify the values of *L* and *C*.

1. Apply a wide-band voltage to *u* and measure the output *y* simultaneously. The Continuous State-Space Model of an RLC Circuit example VI uses a chirp signal from 0.5 Hz to 6 Hz as the stimulus signal. The response to the chirp signal is the response signal. This example VI then preprocesses the stimulus and response signals to remove the offset level in these signals.
2. Define a model for this circuit. Because you have information about the approximate values of *L* and *C*, you can build a partially known state-space model or a partially known transfer function model.
3. Estimate the model you defined in step 2 and then estimate *L* and *C*.

The Continuous State-Space Model of an RLC Circuit example VI guides you through defining and estimating a state-space model for the RLC circuit.

Refer to the Continuous State-Space Model of an RLC Circuit VI in the labview\examples\System Identification\Getting Started\Grey-Box Model.llb to access the VI in this case study.

Open example   Browse related examples

Refer to the Continuous Transfer Function Model of a DC Motor with Known Gain VI in the labview\examples\System Identification\Getting Started\Grey-Box Model.llb for an example that demonstrates how to use a partially known transfer function model to estimate the RLC circuit.

Open example   Browse related examples

You can use the following first-order differential equation to represent the relationship between the capacitor voltage and the current of this RLC circuit.

$$\dot{u}_C = \frac{1}{C} i_L$$

You can use the following first-order differential equation to represent the voltage relationship in this RLC circuit.

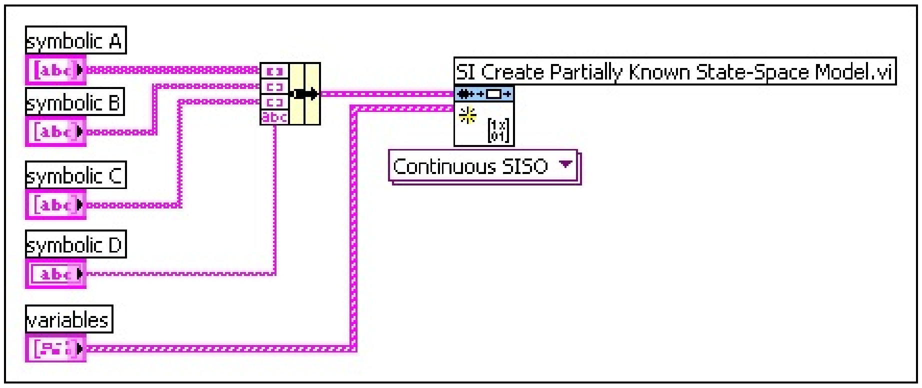
$$Ri_L + u_C + L\dot{i}_L = u$$

By manipulating the previous two equations, you can deduce the continuous state-space model for this RLC circuit using the following two equations:

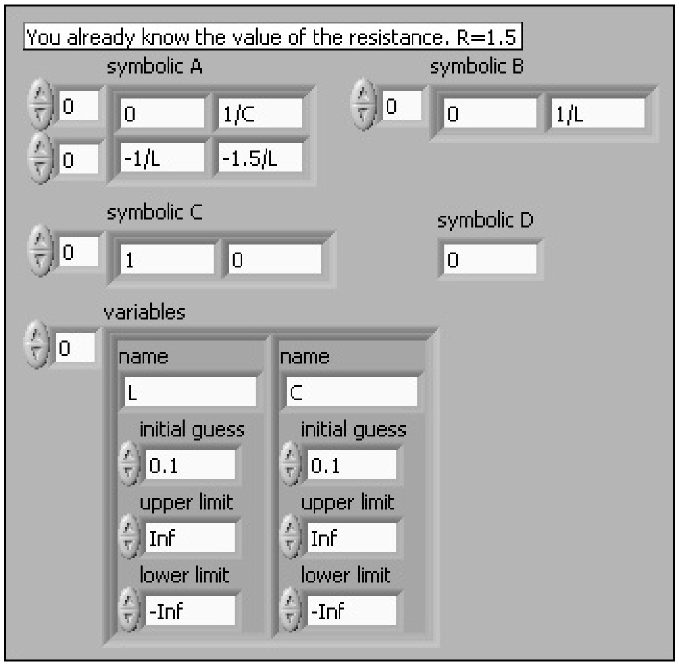
$$\begin{bmatrix} \dot{u}_C \\ \dot{i}_L \end{bmatrix} = \begin{bmatrix} 0 & 1/C \\ -1/L & -R/L \end{bmatrix} \begin{bmatrix} u_C \\ i_L \end{bmatrix} + \begin{bmatrix} 0 \\ 1/L \end{bmatrix} u$$

$$y = u_C = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} u_C \\ i_L \end{bmatrix}$$

The LabVIEW System Identification Toolkit provides the SI Create Partially Known State-Space Model VI with which you can build the symbolic state-space model for this circuit, as shown in the following figure.



You specify the symbolic state-space model using formula strings, such as 1/C, -1/L, and -1.5/L, with *L* and *C* as variables. Then you define *L* and *C* with the **variables** input, as shown in the following figure. Using prior knowledge, you know that *L* is a positive value around the initial value of 0.1 H, and *C* is a value between 0 F and 0.3 F.



You already know the value of the resistance. R=1.5

symbolic A

0	1/C
-1/L	-1.5/L

symbolic B

0	1/L

symbolic C

1	0

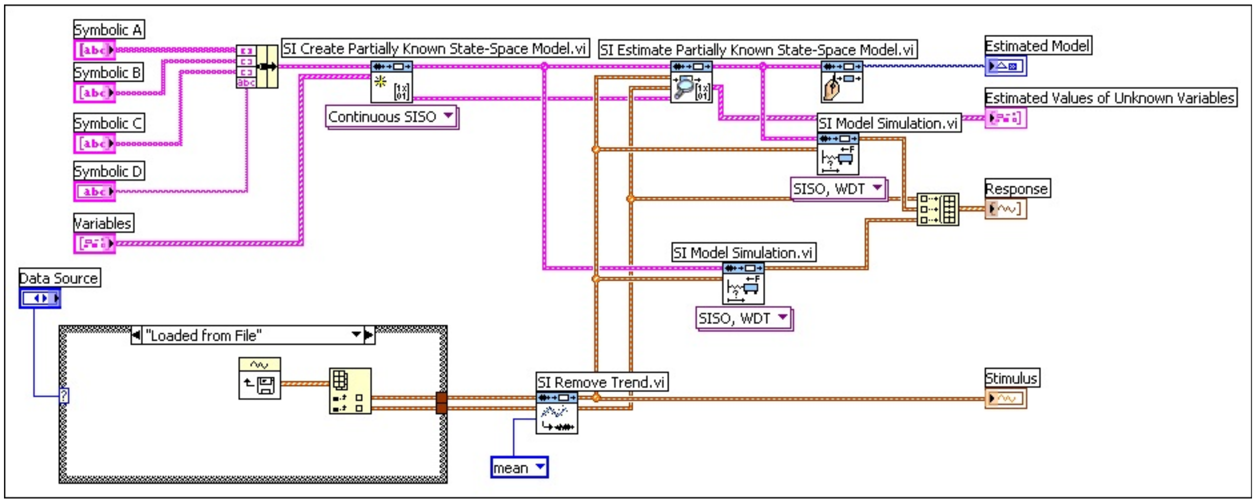
symbolic D

0

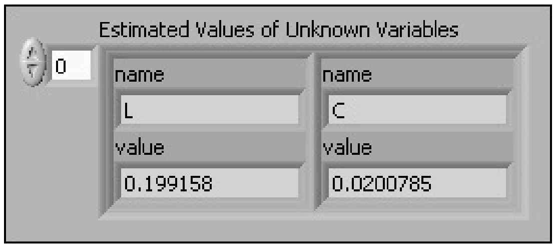
variables

name	L	C
initial guess	0.1	0.1
upper limit	Inf	Inf
lower limit	-Inf	-Inf

Next, you can estimate the state-space model with the SI Estimate Partially Known State-Space Model VI, as shown in the following figure.



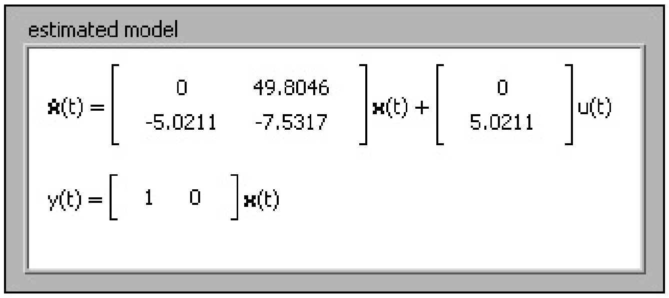
The SI Estimate Partially Known State-Space Model VI estimates each parameter of the model. You obtain the estimated model and optimized variables of the model after this VI performs an optimization. In this example, you obtain the values 0.20 H for *L* and 0.02 F for *C*, as shown in the following figure.


Estimated Values of Unknown Variables

0

name	name
L	C
value	value
0.199158	0.0200785

The Continuous State-Space Model of an RLC Circuit example VI uses the SI Draw Model VI and the values of *L* and *C* you obtain to display the estimated model in a picture indicator, as shown in the following figure.


estimated model

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 49.8046 \\ -5.0211 & -7.5317 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ 5.0211 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x}(t)$$

You then can determine how accurately this model simulates the real-world plant by validating the model. Refer to the System Identification Case Study book for an example of validating a model.

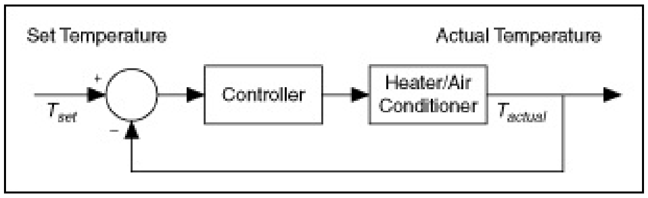
Closed-Loop Systems Model Estimation Methods (System Identification Toolkit)

Systems in many real-world applications contain feedback. Feedback is a process in which the output signal of a plant is passed, or fed back, to the input to regulate the next output. Systems without feedback are open-loop systems. Systems with feedback are closed-loop systems.

In an open-loop system, the stimulus signal and the output noise do not correlate with each other. In a closed-loop system, the stimulus signal correlates to the output noise. Though you can apply many open-loop model estimation methods to closed-loop data, not all open-loop model estimation methods handle the correlation between the stimulus signal and output noise well.

Feedback in Systems

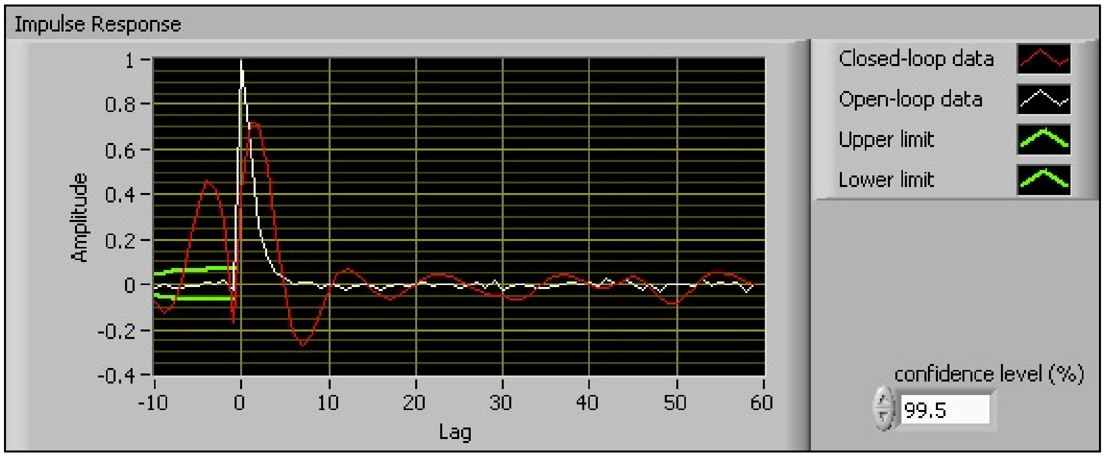
Feedback is common in control systems. With feedback, the system output corresponds to a reference input. Feedback also reduces the effect of input disturbances. One example of a closed-loop system is a system that regulates room temperature, as shown in the following figure. In this example, the reference input is the temperature T_{set} at which you want the room to stay. The thermostat senses the actual temperature, T_{actual}, of the room. Based on the difference between T_{actual} and T_{set}, the thermostat activates the heater or the air conditioner. The thermostat returns T_{actual} as the feedback to compare again with T_{set}. Then the thermostat uses the difference between T_{actual} and T_{set} to regulate the temperature at the next moment.



You must verify if feedback exists before choosing a model estimation method because not all open-loop model estimation methods work correctly with closed-loop data.

Note You must know whether the data you collect is from an open-loop system or a closed-loop system according to the real-world system configuration. If you do not have such information, you can determine if feedback exists by using the SI Detect Feedback VI or by obtaining the impulse responses of a plant. You can use the Least Squares instances of the SI Estimate Impulse Response VI to estimate the impulse response of a plant.

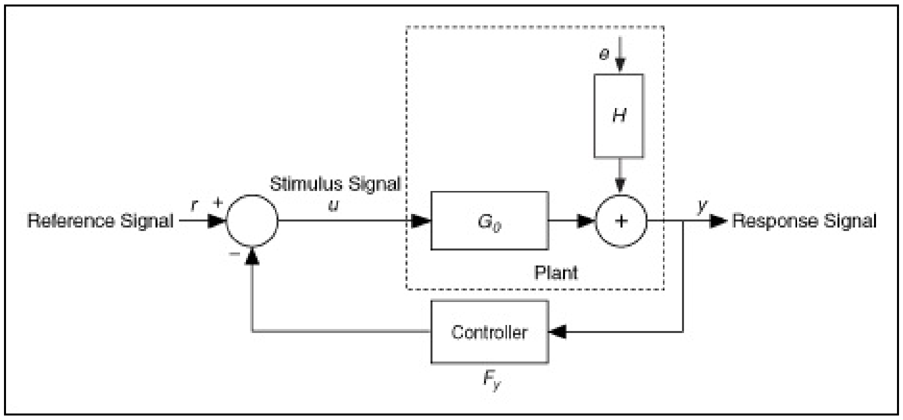
The following figure shows a comparison of the impulse responses of the plant in a closed-loop system and an open-loop system.



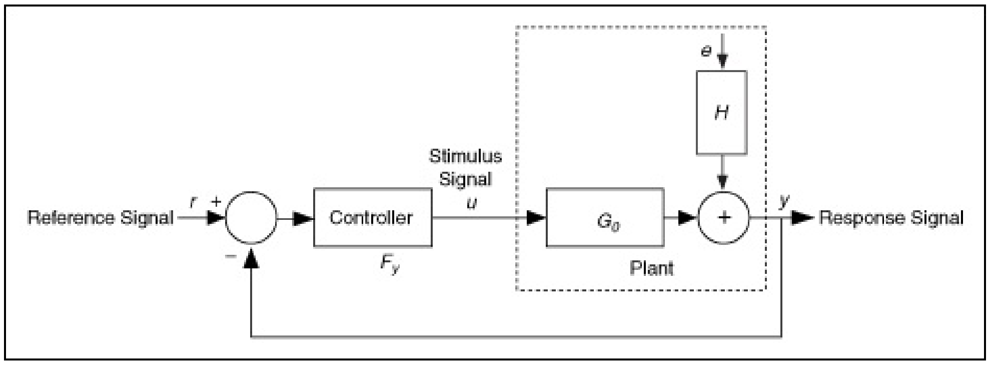
The values outside the **upper limit** and **lower limit** range at the negative lag, which appears between –10 and 0 on the x-axis, are considered significant values. Significant values in the impulse response at negative lags imply feedback in data. As shown in the following figure, significant values exist in the **Closed-loop data** plot. Therefore, feedback exists in the closed-loop system. No significant impulse response values exist in the **Open-loop data** plot. Thus, feedback does not exist in the open-loop system.

Understanding Closed-Loop Model Estimation Methods

Closed-loop model estimation methods use data from a closed-loop system to build a model for a plant that a controller regulates. The following figure shows a system that consists of a plant and a controller. In this system, G_0 is the plant, F_y is the controller, *H* is the stochastic part of the plant, *u* is the stimulus signal, *y* is the response signal, *r* is the reference signal that is an external signal, and *e* is the output noise. In control engineering, this system is known as a feedback-path closed-loop system, which is a typical closed-loop system.



In some cases, the controller comes before the plant in a closed-loop system. This system is known as a feedforward-path closed-loop system, as shown in the following figure.



Depending on the amount of prior knowledge you have about the feedback, the controller, and the reference signal of a system, you can categorize closed-loop model estimation approaches into the following three groups:

- Direct identification—Uses the stimulus and response signals to identify the plant model as if the plant is in an open-loop system.

You can apply the direct identification approach to compute all types of models except state-space models by using the LabVIEW System Identification Toolkit.

- Indirect identification—Identifies a closed-loop system by using the reference signal and the response signal and then determines the plant model based on the known controller of the closed-loop system. You can apply the indirect identification approach to compute transfer function models.
- Joint input-output identification—Considers the stimulus signal and the response signal as outputs of a cascaded system. The reference signal and the noise jointly perturb the system, and the plant model is identified from this joint input-output system. You can apply the joint input-output identification approach to compute transfer function models.

You can choose a suitable model identification approach according to the information you have about the closed-loop system. The following table summarizes the information you must have to use each identification approach.

	Stimulus Signal	**Response Signal**	**Reference Signal**	**Controller Information**
Direct	X	X	—	—
Indirect	—	X	X	X
Joint Input-Output	X	X	X	—

With the LabVIEW System Identification Toolkit, you can choose to use the direct, indirect, or joint input-output identification approaches for different types of closed-loop systems. The direct identification approach supports single-input single-output (SISO), multiple-input single-output (MISO), and multiple-input multiple-output (MIMO) systems. The indirect and joint input-output identification approaches support SISO systems only.

Direct Identification (System Identification Toolkit)

If the stimulus and response signals of a closed-loop system are available but you do not have any other information about the system, you can use only the techniques developed for open-loop models to estimate the closed-loop system. However, you cannot apply all open-loop identification methods to estimate the model of a plant in a closed-loop system. Some open-loop model identification methods assume zero correlation between the stimulus signal and output noise. In closed-loop systems, this correlation is nonzero. Thus, if you use certain open-loop model estimation methods, such as the instrument variable (IV) method and the correlation analysis methods, with closed-loop data, you might estimate a model incorrectly. You can use the prediction error method to identify the plant in a closed-loop system.

The direct identification approach is used commonly in real-world applications. This approach is convenient because you do not need to have additional information about a closed-loop system, such as the reference signal or the controller. However, the estimation might not be accurate if the model type you select for a plant does not describe the output noise of the system accurately. For example, if the output noise of a plant is color noise and you select an output-error (OE) model, which assumes the output noise is white noise, the estimation for the OE model might be biased when you use direct identification. The bias might be small, though, if the signal-to-noise ratio (SNR) of the system is high.

Indirect Identification (System Identification Toolkit)

The indirect identification approach, which estimates the transfer function model of a plant in a closed-loop system, first identifies the transfer function model of the closed-loop system based on the reference signal and the response signal. This approach then retrieves the transfer function model of the plant from the identified closed-loop system. The indirect identification approach can identify the transfer function of the plant accurately even when the signal-to-noise ratio (SNR) of the system is low and no matter whether the output noise is white noise or color noise. However, this approach requires prior knowledge about the controller of the system and the reference signal also must be available. In addition, any inaccuracy or nonlinearity of the controller in the system might affect estimating the model of the plant.

With indirect identification, you can use the following two equations to describe the feedback-path closed-loop system.

$y(k) = G_0(z)u(k) + e(k)$

$u(k) = r(k) - F_y(z)y(k)$

where $G_0(z)$ is the open-loop transfer function of the plant

$F_y(z)$is the transfer function of a linear, time-invariant (LTI) controller

$u(k)$ is the stimulus signal of the system

$y(k)$ is the response signal of the system

$r(k)$ is the reference signal of the system

$e(k)$ is the output noise of the system

By combining the previous two equations, you can represent the closed-loop relationship with the following equation:

$$y(k) = \frac{G_0(z)}{1 + G_0(z)F_y(z)} r(k) + \frac{1}{1 + G_0(z)F_y(z)} e(k)$$

If you define G_{cl} as the closed-loop transfer function between the reference signal and the response signal, and let G_{cl} satisfy the following equation:

$$G_{cl} = \frac{G_0(z)}{1 + G_0(z)F_y(z)}$$

you can estimate G_{cl} with $r(k)$ as the input and $y(k)$ as the output using an open loop method, because $r(k)$ and $e(k)$ are uncorrelated. You then can calculate G_0 after you calculate G_{cl}, as the following equation shows:

$$G_0 = \frac{G_{cl}}{1 - G_{cl}F_y}$$

For feedforward-path closed-loop systems, you use the following two equations to describe the systems.

$$y(k) = G_0(z)u(k) + e(k)$$

$$u(k) = [r(k) – y(k)]F_y(z)$$

By combining the previous two equations, you can represent the feedforward-path closed-loop relationship with the following equation:

$$y(k) = \frac{F_y(z)G_0(z)}{1 + G_0(z)F_y(z)}r(k) + \frac{1}{1 + G_0(z)F_y(z)}e(k)$$

If you define G_{cl} as the feedforward-path closed-loop transfer function and let G_{cl} satisfy the following equation:

$$G_{cl} = \frac{F_y(z)G_0(z)}{1 + G_0(z)F_y(z)}$$

you can estimate G_{cl} with $r(k)$ as the input and $y(k)$ as the output using an open loop method, because $r(k)$ and $e(k)$ are uncorrelated. You then can calculate G_0 after you calculate G_{cl}, as the following equation shows:

$$G_0 = \frac{G_{cl}}{(1 - G_{cl})F_y}$$

With indirect identification, you calculate G_{cl} by performing polynomial operations on G_o and F_y. Because of the limitations of polynomial operations, the orders of the numerator and denominator might change after manipulation. Thus, the SI Estimate Transfer Function Model VI or the SI Transfer Function Estimation ExpressVI, which you can use with the indirect identification approach, might return an error regarding the mismatch between the order you set and the order of the estimated model. In this case, you must adjust the tolerance setting of these two

VIs so that the numerator and denominator orders match the orders you set. A larger tolerance facilitates zero-pole cancellations, which reduce the numerator and denominator polynomial orders.

Joint Input-Output Identification (System Identification Toolkit)

If you do not have any knowledge about the controller structure but the stimulus, response, and reference signals are all available, you can use the joint input-output identification approach to estimate the transfer function model of a plant in a closed-loop system. This approach uses the transfer functions from different input-output signal pairs to estimate a closed-loop system. The LabVIEW System Identification Toolkit implements the following two-stage method for the joint input-output approach.

1. Let $T_0(z)$ satisfy the following equation:

$$T_0(z) = \frac{1}{1 + G_0(z)F_y(z)}$$

By manipulating two equations describing the feedback-path closed-loop system, you can rewrite $u(k)$ as follows:

$$u(k) = T_0(z)r(k) - F_y(z)T_0(z)e(k)$$

Any open-loop model estimation method then can estimate $T_0(z)$ because $r(k)$ and $e(k)$ are uncorrelated signals. After you obtain the value of $T_0(z)$, you can compute $\hat{u}(k) = T_0(z)r(k)$. You then can represent $u(k)$ as follows:

$$u(k) = \hat{u}(k) - F_y(z)T_0(z)e(k)$$

Using the previous equation, you obtain an input signal $\hat{u}(k)$, which is constructed from $r(k)$ and is uncorrelated with the measurement noise.

2. By manipulating the equation $y(k) = G_0(z)u(k) + e(k)$, you can rewrite $y(k)$ as follows:

$$y(k) = G_0(z)\hat{u}(k) + T_0(z)e(k)$$

Because $\hat{u}(k)$ is uncorrelated with $e(k)$, the original closed-loop model estimation problem between $u(k)$ and $y(k)$ becomes an open-loop problem between $\hat{u}(k)$ and $y(k)$.

You use the same methodology to compute $y(k)$ for a feedforward-path closed-loop system, where

$$T_0(z) = \frac{F_y(z)}{1 + G_0(z)F_y(z)}$$

You rewrite *y*(*k*) as follows:

$$y(k) = \hat{u}(k)G_0(z) + [1 - T_0(z)G_0(z)]e(k)$$

The two-stage method does not require you to know anything about the feedback or the controller structure and controller parameters. Also, you treat the closed-loop model estimation as an open-loop model estimation within each of the two steps. Therefore, you can use any method that works with open-loop models. Whether the real-world output noise is white noise or color noise, the two-stage method provides reliable estimations.

Using System Identification VIs for Model Estimation (System Identification Toolkit)

To apply the direct identification approach, you can use the LabVIEW System Identification Toolkit to estimate a plant in a closed-loop system with general-linear polynomial, transfer function, and zero-pole-gain models. To apply the indirect or joint input-output approach to identify a plant, you can use this toolkit with transfer function models. Select the System Identification VIs using the following guidelines:

- Use the Polynomial Model Estimation VIs or the SI Model Estimation Express VI to estimate ARX, ARMAX, output-error, Box-Jenkins, and general-linear models. For ARX models, the System Identification Toolkit uses the least squares method, which is a special case of the prediction error method. For all other models, this toolkit uses the prediction error method. This method can accurately identify a plant model in a closed-loop system. Hence, you can use the Polynomial Model Estimation VIs to estimate the model of a plant in a closed-loop system.
- Use the SI Estimate Transfer Function VI or the SI Transfer Function Estimation Express VI to estimate a transfer function model of the plant in a closed-loop system. You can apply direct, indirect, and joint input-output identification to compute transfer function models.
- To identify zero-pole-gain models for a plant, you first must identify the plant using other model representations. You then can convert other model representations to zero-pole-gain models using the Model Conversion VIs.

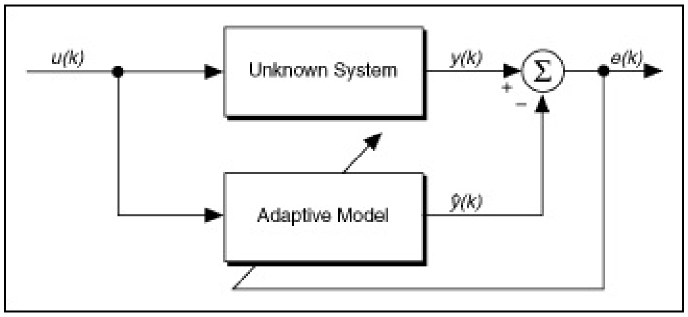
Refer to the LabVIEW System Identification Toolkit Algorithm References manual for more information about the prediction error method, the deterministic-stochastic subspace method, and the realization method.

Recursive Model Estimation Methods (System Identification Toolkit)

Recursive model estimation is a system identification technique that enables you to develop a model that adjusts based on real-time data coming from the system. Recursive model estimation processes the measured input-output data recursively as the data becomes available. This technique is helpful because you obtain the mathematical model of the system in real time. In many real-world applications such as adaptive control and adaptive prediction, having a model of the system update while the system is running is necessary or helpful.

By comparison, the nonparametric, parametric, partially known, and closed-loop systems model estimation methods use nonrecursive methods to estimate a model of the plant in a system. These nonrecursive methods identify a model for a plant based on input-output data gathered at a time prior to the current time.

The following figure represents a general recursive system identification application. A system identification application consists of an unknown system that has an input signal, or stimulus signal $u(k)$ and an output signal, or response signal $y(k)$.



The stimulus signal $u(k)$ is the input to both the unknown system and the recursive model. The response of the system $y(k)$ and the predicted response of the adaptive model $\hat{y}(k)$ are combined to determine the error of the system. The error of the system is defined by the following equation.

$$e(k) = y(k) - \hat{y}(k)$$

The adaptive model generates the predicted response $\hat{y}(k+1)$ based on $u(k + 1)$ after adjusting the parametric vector $\vec{w}(k)$ based on the error $e(k)$.

The previous figure shows how the error information $e(k)$ is sent back to the adaptive model, which adjusts the parametric vector $\vec{w}(k)$ to account for the error. You iterate on this process until you minimize the magnitude of the least mean square error *e*(*k*).

Before you apply the recursive model estimation, you must first select the parametric model structure that determines the parametric vector $\vec{w}(k)$. Then, you must select the method that automatically adjusts the parametric vector such that the error *e*(*k*) goes to the minimum.

The LabVIEW System Identification Toolkit provides Recursive Model Estimation VIs, which you can use to estimate the following parametric model representations:

- ARX
- ARMAX
- Output-Error
- Box-Jenkins
- General-Linear

Refer to the Online Model Estimation VI in the labview\examples\System Identification\Getting Started\Recursive Estimation.llb for an example that demonstrates how to use the SI Recursively Model Estimation VIs to estimate the linear models for an unknown system recursively.

Open example Browse related examples

The Recursive Model Estimation VIs have a **recursive method** parameter that enables you to specify which recursive estimation method to use. The adaptive method you use affects the performance of recursive system identification application. You can choose from the following four methods:

- Least mean squares (LMS)
- Normalized least mean squares (NLMS)
- Recursive least squares (RLS)
- Kalman filter (KF)

The goal of each method is to adjust the parametric vector $\vec{w}(k)$ until you minimize the cost function *J*(*k*). The following equation defines the cost function *J*(*k*).

$J(k) = E[e^{2}(k)]$

where *E* is the expectation of the enclosed term(s).

When the cost function $J(k)$ is sufficiently small, the parametric vector $\vec{w}(k)$ is considered optimal for the estimation of the actual system.

Least Mean Squares (System Identification Toolkit)

The least mean squares (LMS) method uses the following equations to define the cost function $J(k) = E[e^{2}(k)]$.

The parametric vector $\hat{y}(k)$ updates according to the following equation.

$$\vec{w}(k+1) = \vec{w}(k) + \mu e(k)\vec{\varphi}(k)$$

k is the number of iterations, μ is step-size, which is a positive constant, and $\vec{\varphi}(k)$ is the data vector from the past input data $u(k)$ and output data $y(k)$. $\vec{\varphi}(k)$ is defined by the following equation.

$$\varphi(k) = \left[-y(t-1)\ldots-y(t-k)u(t-1)\ldots u(t-m)\right]^{T}$$

The following procedure describes how to implement the LMS algorithm.

1. Initialize the step-size μ.
2. Initialize the parametric vector $\vec{w}(k)$ using a small positive number ε.

   $$\vec{w}(0) = [\varepsilon, \varepsilon, \ldots, \varepsilon]^{T}$$

3. Initialize the data vector $\vec{\varphi}(k) = [u(k)y(k)]$.

   $$\vec{\varphi}(0) = [0, 0, \ldots, 0]^{T}$$

4. For $k = 1$, update the data vector $\vec{\varphi}(k)$ based on $\vec{\varphi}(k-1)$ and the current input data $u(k)$ and output data $y(k)$.
5. Compute the predicted response $\hat{y}(k)$ using the following equation.

   $$\hat{y}(k) = \vec{\varphi}^{T}(k)\vec{w}(k)$$

6. Compute the error $e(k)$ by solving the following equation.

   $$e(k) = y(k) - \hat{y}(k)$$

7. Update the parameter vector $\vec{w}(k)$.

   $$\vec{w}(k+1) = \vec{w}(k) + \mu e(k)\vec{\varphi}(k)$$

8. Stop if the error is small enough, else set $k = k + 1$ and repeat steps 4–8.

The LMS algorithm is one of the most widely used and understood adaptive algorithms. Selecting the step-size μ is important with the LMS algorithm, because the selection of the step-size μ directly affects the rate of convergence and the stability of the algorithm. The convergence rate of the LMS algorithm is usually proportional to the step-size μ. The larger the step-size μ, the faster the convergence rate. However, a large step-size μ can cause the LMS algorithm to become unstable. The following equation describes the range of the step-size μ.

$$0 < \mu < \mu_{max}$$

μ_{max} is the maximum step-size that maintains stability in the LMS algorithm. μ_{max} is related to the statistical property of the stimulus signal. A uniformly optimized step-size μ that achieves a fast convergence speed while maintaining the stability in the system does not exist, regardless of the statistical property of the stimulus signal. For better performance, use a self-adjustable step-size μ and the normalized least mean squares (NLMS) algorithm.

Normalized Least Mean Squares (System Identification Toolkit)

The following equation defines a popular self-adjustable step-size μ(*k*) that you use in the normalized least mean squares (NLMS) algorithm.

$$\mu(k) = \frac{\mu}{\varepsilon + \|\vec{\varphi}(k)\|^2}$$

$\vec{\varphi}(k)$ represents the data vector. ε is a very small positive number that prevents the denominator from equaling zero when $\|\vec{\varphi}(k)\|^2$ approaches zero.

The step-size μ(*k*) is time-varying because the step-size changes with the time index *k*.

Substituting μ(*k*) into the parametric vector $\vec{w}(k)$ equation yields the following equation.

$$\vec{w}(k+1) = \vec{w}(k) + \mu(k)e(k)\vec{\varphi}(k)$$

Compared to the least mean squares (LMS) algorithm, the NLMS algorithm is always stable if the step-size μ(*k*) is between zero and two, regardless of the statistical property of the stimulus signal *u*(*k*).

The procedure of the NLMS algorithm is the same as the LMS algorithm except for the estimation of the time-varying step-size μ(*k*).

Recursive Least Squares (System Identification Toolkit)

The recursive least squares (RLS) algorithm and Kalman filter algorithm use the following equations to modify the cost function $J(k) = E[e^2(k)]$.

$$J(k) = E\left[e^2(k)\right] \cong \frac{1}{N}\sum_{i=0}^{N-1} e^2(k-i)$$

Compare this modified cost function, which uses the previous N error terms, to the cost function, $J(k) = E[e^2(k)]$, which uses only the current error information $e(k)$. The modified cost function $J(k)$ is more robust. The corresponding convergence rate in the RLS algorithm is faster, but the implementation is more complex than that of LMS-based algorithms.

The following procedure describes how to implement the RLS algorithm.

1. Initialize the parametric vector $\vec{w}(k)$ using a small positive number ε.

   $$\vec{w}(0) = [\varepsilon, \varepsilon, \ldots, \varepsilon]^T$$

2. Initialize the data vector $\vec{\varphi}(k)$.

   $$\vec{\varphi}(0) = [0, 0, \ldots, 0]^T$$

3. Initialize the $k \times k$ matrix $\boldsymbol{P}(0)$.

   $$\boldsymbol{P}(0) = \begin{bmatrix} \varepsilon & 0 & 0 & 0 \\ 0 & \varepsilon & 0 & 0 \\ 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & \varepsilon \end{bmatrix}$$

4. For $k = 1$, update the data vector $\vec{\varphi}(k)$ based on $\vec{\varphi}(k-1)$ and the current input data $u(k)$ and output data $y(k)$.
5. Compute the predicted response $\hat{y}(k)$ by using the following equation.

   $$\hat{y}(k) = \vec{\varphi}^T(k) \cdot \vec{w}(k)$$

6. Compute the error $e(k)$ by solving the following equation.

   $$e(k) = y(k) - \hat{y}(k)$$

7. Update the gain vector $\vec{K}(k)$ defined by the following equation.

$$\vec{K}(k) = \frac{\boldsymbol{P}(k) \cdot \vec{\varphi}(k)}{\lambda + \vec{\varphi}^T(k) \cdot \boldsymbol{P}(k) \cdot \vec{\varphi}(k)}$$

   The properties of a system might vary with time, so you must ensure that the algorithm tracks the variations. You can use the forgetting factor λ, which is an adjustable parameter, to track these variations. The smaller the forgetting factor λ, the less previous information this algorithm uses. When you use small forgetting factors, the adaptive filter is able to track time-varying systems that vary rapidly. The range of the forgetting factor λ is between zero and one, typically 0.98 < λ < 1.

   P(*k*) is a *k* × *k* matrix whose initial value is defined by ***P***(0) in step 3.
8. Update the parametric vector $\vec{w}(k+1)$.

$$\vec{w}(k+1) = \vec{w}(k) + e(k) \cdot \vec{K}(k)$$

9. Update the ***P***(*k*) matrix.

$$\boldsymbol{P}(k+1) = \lambda^{-1}\boldsymbol{P}(k) - \lambda^{-1}\vec{K}(k) \cdot \vec{\varphi}^T(k) \cdot \boldsymbol{P}(k)$$

10. Stop if the error is small enough, else set *k* = *k* + 1 and repeat steps 4–10.

Kalman Filter (System Identification Toolkit)

The Kalman filter is a linear optimum filter that minimizes the mean of the squared error recursively. The convergence rate of the Kalman filter is relatively fast, but the implementation is more complex than that of LMS-based algorithms.

Recall that the equation $J(k) = E[e^{2}(k)]$ defines the cost function. The following procedure lists the steps of the Kalman filter algorithm.

1. Initialize the parametric vector $\vec{w}(k+1) = \vec{w}(k) + e(k) \cdot \vec{K}(k)$ using a small positive number ε.

   $$\vec{w}(0) = [\varepsilon, \varepsilon, \ldots, \varepsilon]^{T}$$

2. Initialize the data vector $\vec{\varphi}(k)$.

   $$\vec{\varphi}(0) = [0, 0, \ldots, 0]^{T}$$

3. Initialize the $k \times k$ matrix $\boldsymbol{P}(0)$.

   $$\boldsymbol{P}(0) = \begin{bmatrix} \varepsilon & 0 & 0 & 0 \\ 0 & \varepsilon & 0 & 0 \\ 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & \varepsilon \end{bmatrix}$$

4. For $k = 1$, update the data vector $\vec{\varphi}(k)$ based on $\vec{\varphi}(k-1)$ and the current input data $u(k)$ and output data $y(k)$.
5. Compute the predicted response $\hat{y}(k)$ by solving the following equation.

   $$\hat{y}(k) = \vec{\varphi}^{T}(k) \cdot \vec{w}(k)$$

6. Compute the error $e(k)$ by solving the following equation.

   $$e(k) = y(k) - \hat{y}(k)$$

7. Update the Kalman gain vector $\vec{K}(k)$ defined by the following equation.

   $$\vec{K}(k) = \frac{\boldsymbol{P}(k) \cdot \vec{\varphi}(k)}{\boldsymbol{Q}_{M} + \vec{\varphi}^{T}(k) \cdot \boldsymbol{P}(k) \cdot \vec{\varphi}(k)}$$

$\boldsymbol{Q}_M$ is the measurement noise and $\boldsymbol{P}(k)$ is a $k \times k$ matrix whose initial value is defined by $\boldsymbol{P}(0)$ in step 3.

8. Update the parametric vector $\vec{w}(k)$.

   $$\vec{w}(k+1) = \vec{w}(k) + e(k) \cdot \vec{K}(k)$$

9. Update the $\boldsymbol{P}(k)$ matrix.

   $$\boldsymbol{P}(k+1) = \boldsymbol{P}(k) - \vec{K}(k) \cdot \vec{\varphi}^{\,T}(k) \cdot \boldsymbol{P}(k) + \boldsymbol{Q}_P$$

   $\boldsymbol{Q}_P$ is the correlation matrix of the process noise.

10. Stop if the error is small enough, else set $k = k + 1$ and repeat steps 4–10.

Frequency-Domain Model Estimation Methods (System Identification Toolkit)

Frequency-domain model estimation involves identifying a model of a plant by using the frequency-domain representation of the plant data. You acquire frequency-domain data by using a frequency analyzer. If you acquire time-domain data, you also can convert this time-domain data to frequency-domain data by estimating the frequency response function (FRF) of a plant. The FRF represents the frequency-domain relationship between the inputs and outputs of a plant.

The LabVIEW System Identification Toolkit provides tools you can use to obtain the FRF and estimate two categories of parametric models—transfer function and state-space. Use the SI Estimate Transfer Function Model from FRF VI to estimate continuous and discrete single-input single-output (SISO) transfer function models. Use the SI Estimate State-Space Model from FRF VI to estimate discrete SISO and multiple-input multiple-output (MIMO) state-space models.

Advantages of Frequency-Domain System Identification

Compared to time-domain model estimation methods, frequency-domain model estimation methods have the following advantages:

- You can reduce a large number of time-domain data samples to a finite number of frequency-domain data samples.
- You can reduce the effects of noise by averaging the FRF from multiple time-domain data measurements.
- You can focus on frequency bands of interest by directly weighting the frequency-domain data.

Estimating the Frequency Response Function (System Identification Toolkit)

The frequency response function (FRF) represents the frequency-domain relationship between the inputs and outputs of a plant. The FRF contains the magnitude, phase, and frequency information of the plant data. You estimate the FRF when you have time-domain data, but you want to identify a system model in the frequency domain. If you acquire frequency-domain data, this data already contains the FRF.

When you estimate the FRF, the following factors can affect the FRF negatively.

- Noise
- Spectral leakage
- Nonlinear distortion

Use the SI Estimate FRF VI to estimate the FRF and to minimize the effects of these factors. This VI supports windowing, which helps to minimize spectral leakage. This VI also supports averaging, which helps to minimize noise and nonlinear distortion.

Minimizing Spectral Leakage

To minimize the effects of spectral leakage, you can apply a window to the time-domain data. The SI Estimate FRF VI supports several types of windows for different types of signals. The type of window you choose depends on the characteristics of the signal. For example, use a Hanning window for random excitation signals. For impact excitation signals, use an Exponential window.

Minimizing Noise and Nonlinear Distortion

You can average multiple FRF measurements to minimize the effects of nonlinear distortion and reduce the effects of noise in the data measurements. Averaging the data smooths the frequency response by reducing fluctuations that exist in the data.

The SI Estimate FRF VI supports both RMS averaging and vector averaging to average plant data.

Note The multiple-input multiple-output (MIMO) instances of the SI Estimate FRF VI support the RMS averaging mode only.

Frequency-Domain Model Estimation Case Study (System Identification Toolkit)

This case study demonstrates how to use frequency-domain data from a flexible robotic arm to estimate and validate a transfer function model and state-space model of the robotic arm. This case study estimates the models directly from frequency-domain data.

Note If the system you want to estimate contains time-domain data, you first must estimate the FRF by using the SI Estimate FRF VI.

Refer to the Flexible Arm (Frequency Domain) VI in the labview\examples\System Identification\Industry Applications\Mechanical Systems.llb to access the VI in this case study.

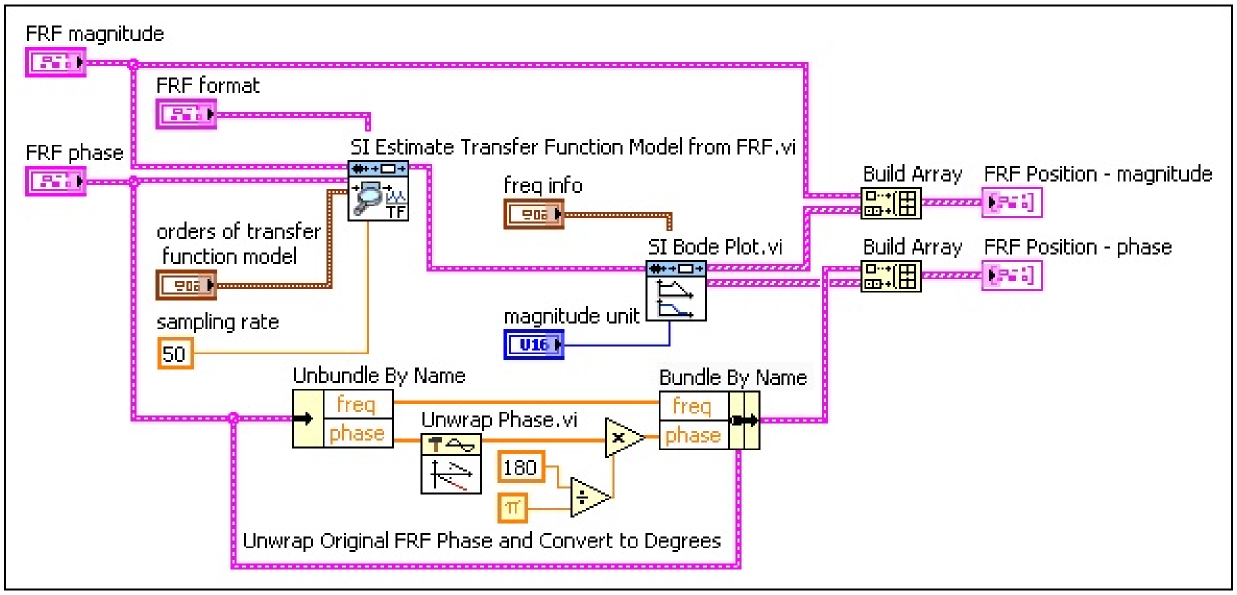
Open example Browse related examples

In this case study, the input is voltage and the outputs are strain and position. Strain is proportional to the acceleration of the flexible arm. Position is the radial position of the arm.

Because this system contains one input and two outputs, two single-input single-output (SISO) FRFs or one multiple-input multiple-output (MIMO) FRF define this system. One FRF defines the relationship between the voltage input and the strain output. The other FRF defines the relationship between the voltage input and the position output. Each FRF contains information about the magnitude and phase of the frequency response and the scale of the frequency data. This information is specific to each input-output pair. To measure the FRFs in this case study, sine waves at different frequencies excite the flexible arm. This data set has a sampling time of 0.02 seconds, which is equivalent to a sampling rate of 50 Hz.

Estimating and Validating a Transfer Function Model

Use the SI Estimate Transfer Function Model from FRF VI to estimate a transfer function model from a SISO FRF. Because this case study involves one input and two outputs, this VI must estimate and validate each SISO transfer function model separately. The following figure shows the block diagram for estimating a transfer function model and validating the resulting transfer function model against the original FRF data. This block diagram uses the FRF describing the relationship between the voltage input and the position output to estimate the transfer function model.



The **FRF magnitude** and the **FRF phase** inputs of the SI Estimate Transfer Function Model from FRF VI require that the data input is in polar form.

Note If the data input is in complex form, you can use the Re/Im To Polar Function to convert the complex data into polar form to separate the magnitude, phase, and frequency components of the complex representation. After converting the complex data into polar form, bundle the magnitude and frequency components into one cluster and bundle the phase and frequency components into another cluster. Each cluster then contains the data you wire to the **FRF magnitude** and the **FRF phase** inputs of the SI Estimate Transfer Function Model from FRF VI.

In this case study, the **FRF magnitude** and the **FRF phase** inputs contain data in polar form. Therefore, the **FRF magnitude** and the **FRF phase** inputs wire directly into the SI Estimate Transfer Function Model from FRF VI.

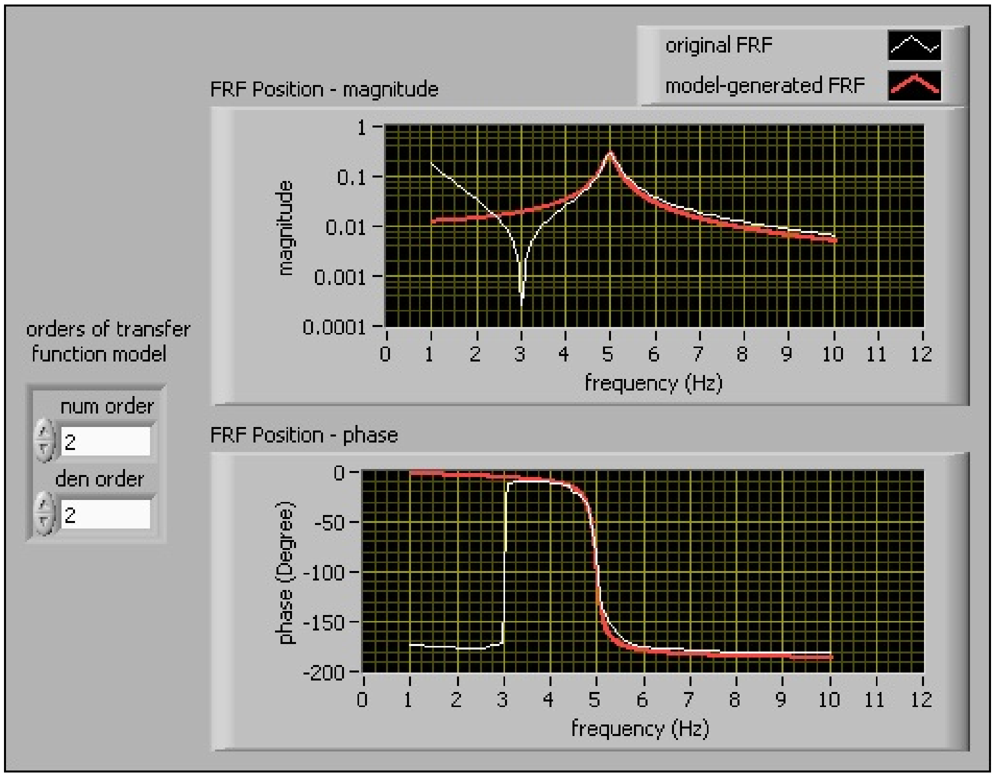
Note The **FRF format** input of this VI specifies whether the original FRF magnitude is in decibels or in a linear scale and whether the original FRF phase is wrapped and in radians or in degrees. In this case study, the original FRF data uses the default values of **FRF format**. The original FRF magnitude is not in decibels. The original FRF phase is wrapped and in radians.

Because a transfer function model is a type of parametric model representation, you must specify the model parameters before estimating the model. For a transfer function model, you must specify the orders of the numerator and denominator polynomial functions. If you do not have prior knowledge about the system, you must use trial and error to determine the optimal orders of the model.

One way to identify the optimal orders is to observe the peaks in the magnitude response of the original FRF. The optimal order is at least twice the number of peaks. You can plot and observe the magnitude by using the SI Bode Plot VI. You can increase the accuracy of the model by modifying the initial guess after analyzing the model. In the previous block diagram, 2 is the initial guess for both the numerator order and the denominator order.

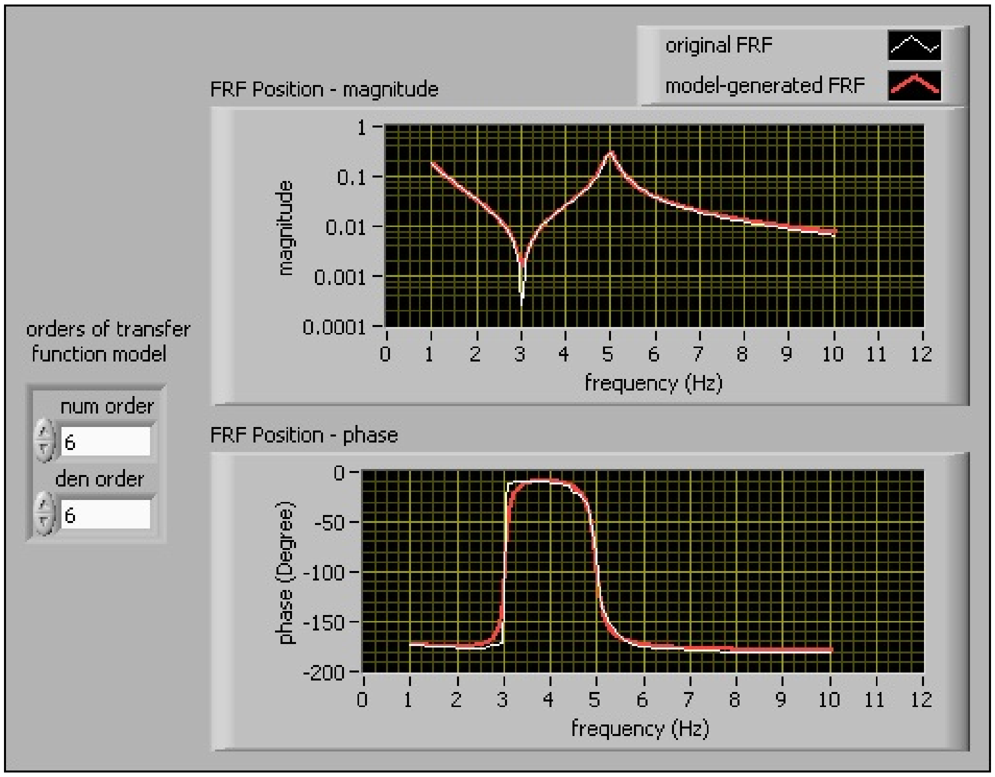
You can validate the resulting model by comparing the original FRF and the FRF that the model generates. Compare the two FRFs on an XY graph by using the SI Bode Plot VI. By default, the SI Bode Plot VI unwraps the phase information and converts the units of the resulting phase to degrees. To compare the original FRF and the FRF that the SI Bode Plot VI displays more accurately, use the Unwrap Phase VI to unwrap the phase in the original FRF. This case study also uses the Mathematics VIs to convert the original FRF from radians to degrees.

The following figure shows the Bode plot of a 2nd-order transfer function model.



Visually inspect the alignment of the **model-generated FRF** and the **original FRF** to determine whether the estimated transfer function model accurately describes the plant. Notice that the **original FRF** and the **model-generated FRF** do not align at frequencies lower than 4 Hz. Therefore, 2 is not the most appropriate order value for the transfer function model.

You can increase the accuracy of the model by specifying different values for the **orders of transfer function model** control. The following figure shows the Bode plots of a 6th-order transfer function model.



In this case study, setting the value of the **orders of transfer function model** to 6 provides a closer alignment of the **model-generated FRF** and the **original FRF**. Therefore, a 6th-order transfer function model describes the voltage-position FRF of the plant more accurately than a 2nd-order model.

Identify a second transfer function model by using the FRF of the voltage input and strain output. You must apply the same method when optimizing the model order.

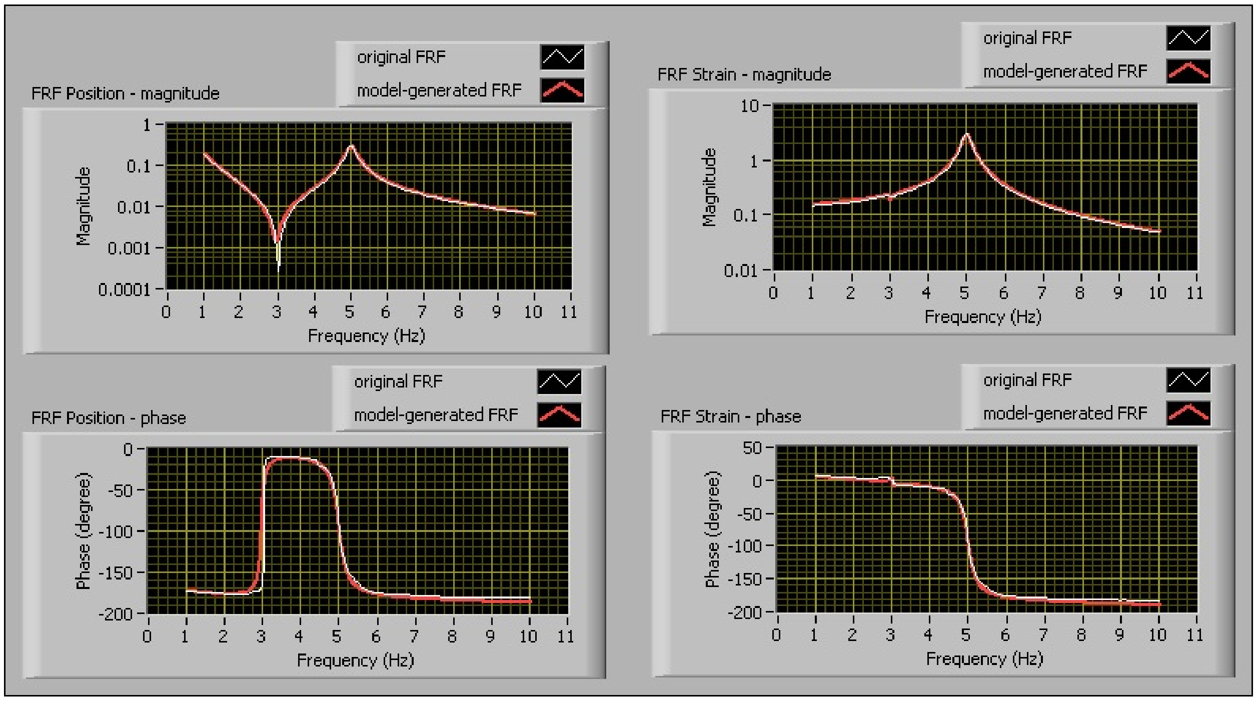
Estimating and Validating a State-Space Model

In this case study, a single MIMO FRF also can represent the frequency-domain relationship of the input and outputs of the flexible arm system. You can use this MIMO FRF to estimate state-space models of a plant. The SI Estimate State-Space Model from FRF VI estimates a state-space model for SISO and MIMO systems. This VI requires you to specify the **number of states** of the system you want to estimate. This value comes from prior knowledge about the system.

You also can provide an initial guess if you do not have prior knowledge. A guideline for identifying the number of states of a system is the same as the guideline for identifying the orders of a transfer function model. The number of states is at least twice the number of peaks in the magnitude of the FRF. Because a MIMO FRF is composed of more than one SISO FRF, you must observe the magnitude of the SISO FRF with the maximum number of peaks.

In this case study, the SISO FRFs that comprise the MIMO FRF both have one peak. Therefore, the initial guess at the number of states is 2. By increasing the number of states and plotting the model-generated MIMO FRF along with the original MIMO FRF on a Bode plot, you can observe that entering eight states produces an acceptable estimation for the system.

The following figure shows the Bode Plots of the original MIMO FRF and the MIMO FRF that the state-space model generates.



Compare the **FRF Position** graphs in the previous figure with the 6th-order transfer function **FRF Position** graphs. Both model-generated FRFs are close to the original FRF in the **FRF Position** graphs. The proximity of these FRFs indicates that the 6th-order transfer function model and the state-space model with eight states are close approximations for estimating system models in this case study.

Analyzing, Validating, and Converting Models (System Identification Toolkit)

After estimating a model of a plant, you can observe model characteristics by analyzing the model. You also can verify that the model represents the real-world plant by validating the model. The LabVIEW System Identification Toolkit provides tools that enable you to analyze and validate models.

According to linear system theory, you can represent a linear system with different models. Each model representation has benefits and drawbacks for characterizing a dynamic system. Certain model representations are more suitable for certain analysis techniques. You can use the System Identification Toolkit to convert models from one representation to another to identify the best-fit model for the system.

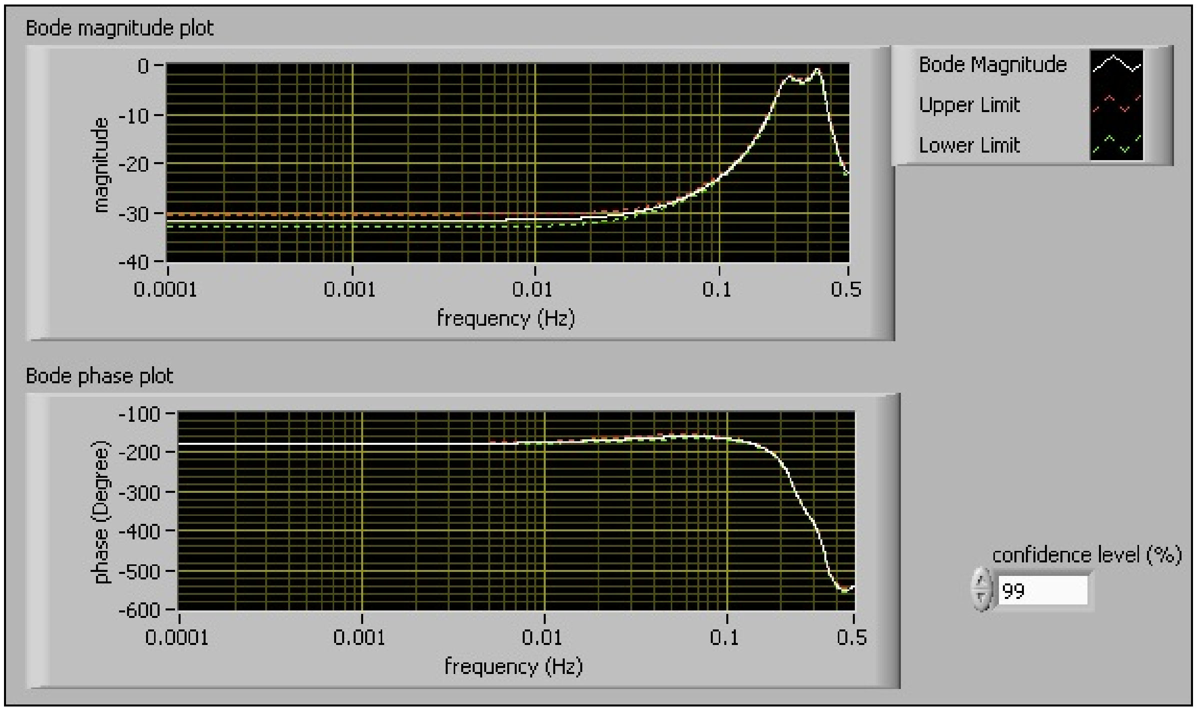
Analyzing Models (System Identification Toolkit)

You can use model analysis to observe some characteristics, such as frequency response, stability, and order, of the model. Use the LabVIEW System Identification Toolkit to investigate model estimation results and present these results in Bode plot, the Nyquist plot, and the pole-zero plot graphs.

Refer to the Model Presentation VI in the labview\examples\System Identification\Getting Started\General.llb for an example that demonstrates how to use the SI Bode Plot, SI Nyquist Plot, and SI Pole-Zero Plot VIs to compute the Bode, Nyquist, and pole-zero plots, respectively, of an system model.

Bode Plot

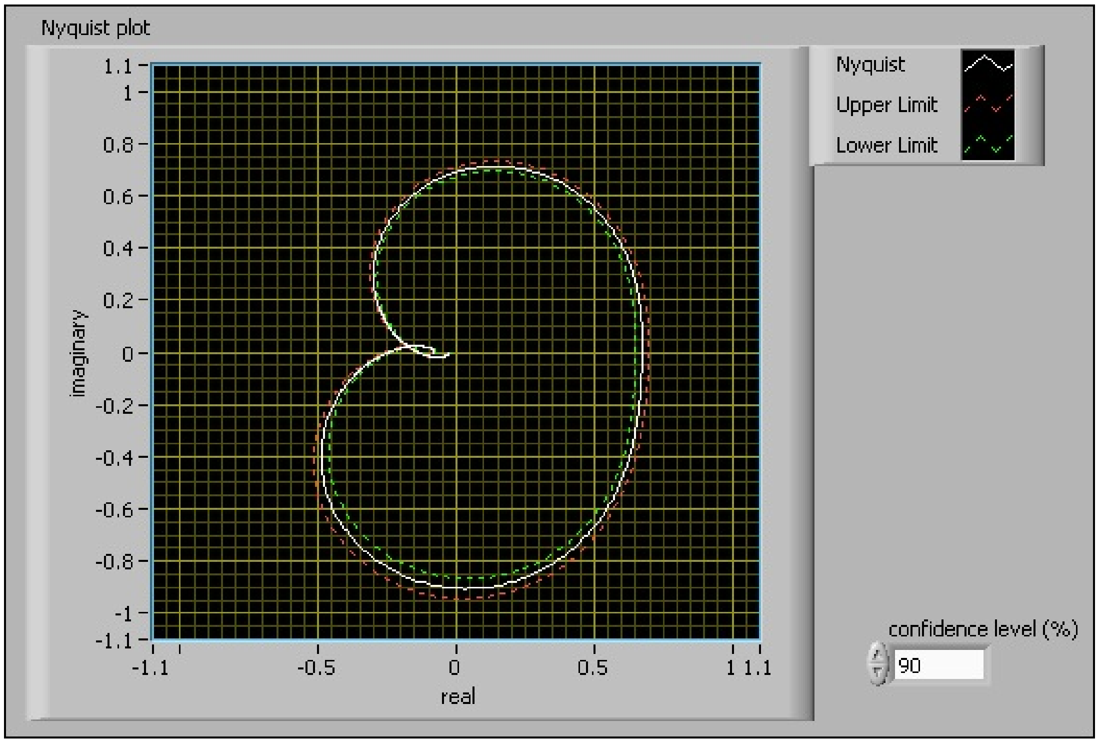
A Bode plot contains a Bode magnitude plot and a Bode phase plot. These two plots together describe the frequency response of the plant model you estimate. A Bode magnitude plot describes magnitude versus frequency. A Bode phase plot describes phase versus frequency. The following figure shows an example of a Bode plot.



The SI Bode Plot VI calculates the upper and lower limits according to the confidence level you set. You can obtain information, such as the gain of the system and the cutoff frequency, by evaluating the Bode plot. You can use the SI Bode Plot VI to produce the Bode magnitude and Bode phase plots. You then can display the Bode magnitude and phase using the SI Bode Plot Indicator.

Nyquist Plot

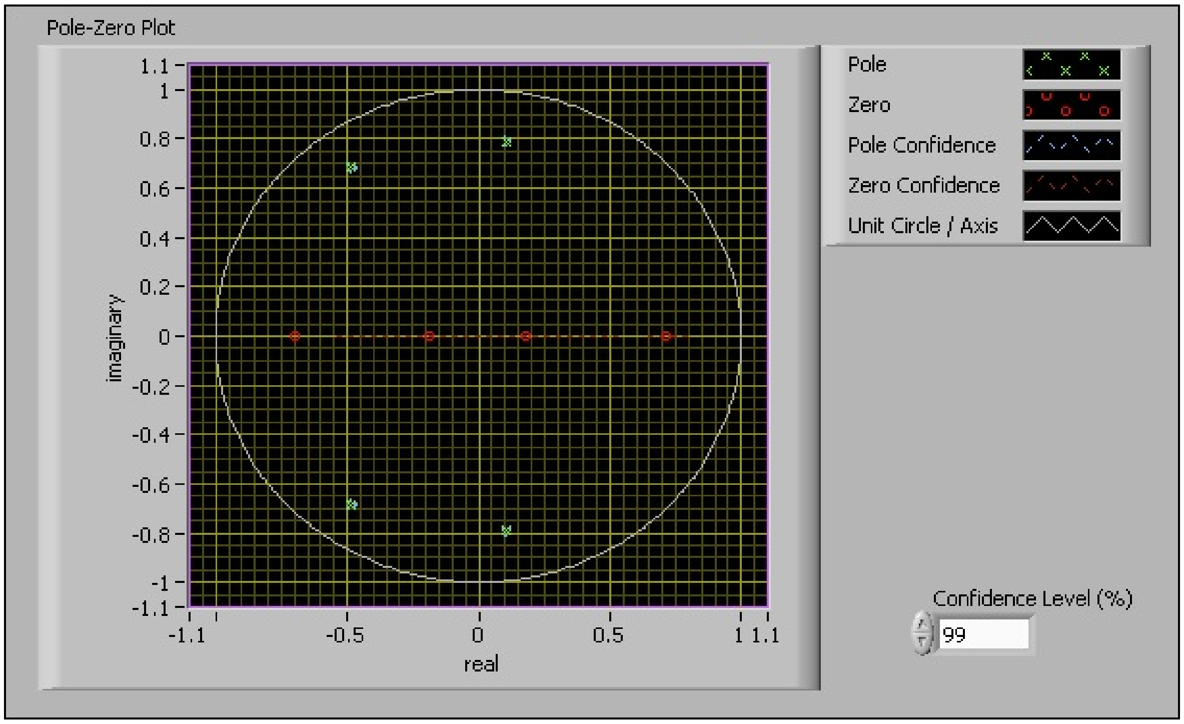
A Nyquist plot describes the gain and phase of a frequency response in polar coordinates by plotting the imaginary part of the complex frequency response versus the real part. You can use the Nyquist plot to predict the stability of a system. In polar coordinates, a Nyquist plot shows the phase as the angle and the magnitude as the distance from the origin, as shown in the following figure.



The SI Nyquist Plot VI calculates the upper and lower limits according to the confidence level you set. You can use the SI Nyquist Plot VI to generate the Nyquist plot and display this plot by using the SI Nyquist Plot Indicator.

Pole-Zero Plot

The pole-zero plot displays the poles and zeros of a system. By observing the locations of the poles and zeros, you can conclude whether the system is stable. In a stable system, all poles are within the unit circle. The following figure shows a pole-zero plot of a stable model.



You can use the SI Pole-Zero Plot VI to generate a pole-zero plot and display the plot by using the SI Pole-Zero Plot Indicator.

You also can use the pole-zero plot to determine if you can reduce model orders. By observing the pole-zero placements, you can determine if any pole-zero pairs have overlapping confidence intervals. A confidence interval is a region the SI Pole-Zero Plot VI calculates from the confidence level you set. The existence of overlapping confidence intervals implies that pole-zero cancellations exist and that the model order might be unnecessarily high. The pole-zero plot shown in the previous figure is an optimal model with the appropriate order because the pole-zero pairs do not have overlapping confidence intervals.

If the model order is too high, you can try reducing the model order. You then can use the F-test criterion to assess if the reduction in model order leads to a significant increase in the prediction error. If the reduction in model order leads to a significant increase in the prediction error, do not reduce the model order.

Validating Models (System Identification Toolkit)

Model estimation determines the best model of the system within the chosen model structure. Model estimation does not determine if the model provides the most accurate description of the system. After you obtain a model, you must validate the model to determine how well the behavior of the model corresponds to the data you measured, to any prior knowledge of the system, and to the purpose for which you use the model. Model validation also determines if the model is flexible enough to describe the system. If the model is inadequate, you must revise the system identification process or consider using another method.

Note When validating the model you obtain, you must use a set of data that is different from the data you used to estimate the model.

The best way to validate a model is to experiment with the model under real-world conditions. If the model works as you expect, the model estimation is successful. However, experimenting with the model under real-world conditions might be dangerous. For example, introducing arbitrary perturbations to the input of a chemical plant might lead to a harmful explosion. Therefore, before you incorporate the model into real-world applications, validate the model by using plots and common sense or by using statistical tests on the prediction error.

The LabVIEW System Identification Toolkit provides three of the most common validation methods—model simulation, model prediction, and model residual analysis. You can use any or all of these methods to validate the model. The method(s) you use depend on the purpose for which you created the model.

Model Simulation

Use model simulation to understand the underlying dynamic relationship between the model inputs and outputs. The SI Model Simulation VI determines the outputs of a system for given inputs. After you build a model for the system using the input and output data you measured, you can use the model to simulate the response of the system by using the model equations. You then can evaluate the behavior of the system. You also can use simulation to validate the model by comparing the simulated response with the measured response.

Refer to the Model Simulation VI in the labview\examples\System Identification\Getting Started\General.llb for an example that demonstrates how to simulate the response of an unknown system with the estimated model.

▪ Open example ▪ Browse related examples

Model Prediction

Use model prediction to test the ability of the model to predict the response of the system using past input and output data. The SI Model Prediction VI determines the response of a system at time *t* based on the output information available at time *t* – *k* and all the inputs applied from time *t* – *k* to time *t*. *k* represents the size of the prediction window. Therefore, model prediction can determine how useful a model is in estimating future responses of the system, given all information at time *t* and an expected input profile in the future. Some control techniques take advantage of model prediction to improve control performance. For example, model predictive control uses some of the prediction properties of a model to determine if a particular limitation or constraint is active in the future. This method allows the controller to take preventive actions before such constraints become active.

If you have the measured input and output of a system, you also can validate the model of the system by comparing the predicted output and the measured output. If the prediction error is small, the model is acceptable.

Refer to the Model Prediction VI in the labview\examples\System Identification\Getting Started\General.llb for an example that demonstrates how to use the *k*-step ahead prediction to verify the model for an unknown system.

▪ Open example ▪ Browse related examples

Model Residual Analysis

Use model residual analysis to analyze the prediction error, which is the difference between the response that an estimated model predicts and the actual response from the system. The following equation defines the prediction error, also known as the residual $e(k)$.

$$e(k) = y(k) - y'(k)$$

$y(k)$ is the measured output and $y'(k)$ is the output from the one-step-ahead prediction. If the model is capable of describing the plant, the residual is zero-mean white noise and independent of the input signal. You can use autocorrelation analysis to test if the residual is zero-mean white noise. You can use cross-correlation analysis to test if the residual is independent of the input signal. The SI Model Residual Analysis VI calculates both the autocorrelation and the cross-correlation values.

Autocorrelation

The following equation defines the autocorrelation of the residuals.

$$R_e^N(\tau) = \frac{1}{N}\sum_{k=1}^{N} e(k)e(k-\tau)$$

Ideally, the residual is white noise, and therefore the autocorrelation function $R_e^N(\tau)$ is zero when τ is nonzero. A large autocorrelation when τ is nonzero indicates that the residual is not zero-mean white noise and also implies that the model structure is not relevant to the system or that you might need to increase the model order.

In real-world applications, the autocorrelation function $R_e^N(\tau)$ cannot be zero when τ is nonzero because of the limited length of data points. However, the SI Model Residual Analysis VI assesses if the autocorrelation value is sufficiently small to be ignored. If the value of autocorrelation falls within the confidence range, the autocorrelation value is insignificant and you can consider this value to be equal to zero.

Cross Correlation

The following equation defines the cross correlation between residuals and past inputs.

$$R_{eu}^N(\tau) = \frac{1}{N}\sum_{k=1}^{N} e(k)u(k-\tau)$$

If the residual is independent of the input, the cross correlation is zero for all τ. If the residual correlates with the input, the cross correlation is nonzero, suggesting that the model did not capture all deterministic variations from the data. Therefore, you must revise the model variation.

The SI Model Residual Analysis VI assesses if the value of cross correlation is sufficiently small. If the value of cross correlation falls within the confidence range, the value is insignificant and you can consider this value to be equal to zero.

Refer to the Residual Analysis VI in the labview\examples\System Identification\Getting Started\General.llb for an example that demonstrates how to analyze the residuals of the model estimation for an unknown system.

■ Open example ■ Browse related examples

Converting Models (System Identification Toolkit)

With the LabVIEW System Identification Toolkit, you can use the Model Conversion VIs to convert system models from one representation to another, from continuous to discrete models, and from discrete to continuous models. You can convert models you created in this toolkit to models you can use in another toolkit. You also can convert a model after you estimate the model or after you analyze or validate the model.

You can use the Model Conversion VIs to switch between different model types and representations. For example, when estimating a digital system, you can convert an existing continuous model to a discrete model to approximate the real-time behavior of the system. You do not need to create a new discrete model for the digital system. Using the Model Conversion VIs, you also can convert models you create in the System Identification Toolkit into transfer function, zero-pole-gain, or state-space models that you then can use with the LabVIEW Control Design and Simulation Module. This model conversion process enables you to identify a model for an unknown system with the System Identification Toolkit and then design a controller for this system using the Control Design and Simulation Module.

Refer to the National Instruments Web site at ni.com for more information about the Control Design and Simulation Module.

System Identification Case Study (System Identification Toolkit)

This case study guides you through the system identification process. The case study uses sample data that the LabVIEW System Identification Toolkit provides in the SI Data Samples VI. The SI Data Samples VI includes data sets for a DC motor, a flexible robot arm, a ball and beam apparatus, an RC circuit, and so on. This case study uses the flexible arm data to demonstrate the system identification process and to compare different estimation methods.

The flexible arm is a nonlinear dynamic system. The System Identification Toolkit enables you to build models for systems linearly. This case study guides you through obtaining a linear representation of a nonlinear system.

Refer to the labview\examples\system identification\SICaseStudy1.llb to access the VIs in this case study.

▪ Open example ▪ Browse related examples

In addition to this case study, you can find the following system identification case studies in this help file.

- Partially Known Model Estimation Case Study
- Frequency-Domain Model Estimation Case Study

The partially known model estimation case study guides you through estimating a state-space model by using prior knowledge about the system you want to define. The frequency-domain model estimation case study guides you through estimating a state-space and transfer function model by using frequency-domain data.

Preprocessing the Data (System Identification Toolkit)

After you gather data, the next step in the system identification process is to preprocess the data. The input to the system in this case study is the reaction torque of the structure on the ground. This input is a swept sine wave with 200 frequency points equally spaced over the frequency band from 0.122 Hz to 24.4 Hz.

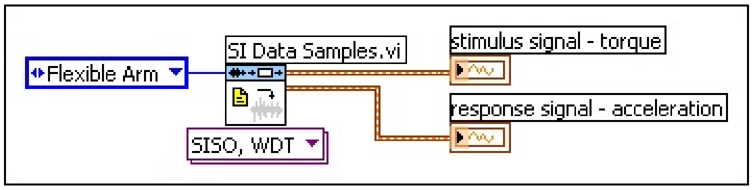
The output of this system is the acceleration of the flexible arm. The acceleration contains information about the flexible resonances and anti-resonances.

The data set contains 4096 samples at a sampling rate of 500 Hz or sampling time of 0.002 seconds. Thus the total time of the response is 8.192 seconds.

You can preprocess the raw data by examining the time and frequency responses of the system. Based on those analyses, you can filter and downsample the data set to reduce the amount of data in the raw data set for simpler identification.

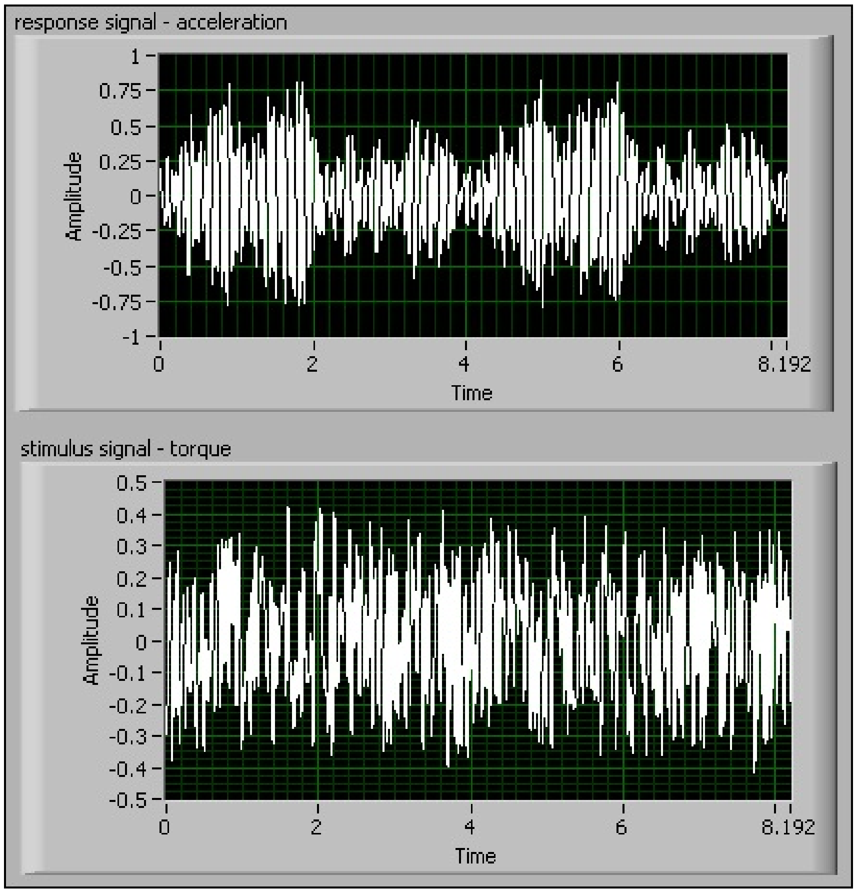
Examining the Time Response Data

Using the data in the SI Data Samples VI for the flexible robotic arm, you can view the input and output data, as shown in the following figure.



The **stimulus signal – torque** output corresponds to the input data, or the torque, and the **response signal – acceleration** output corresponds to the output data, or the acceleration.

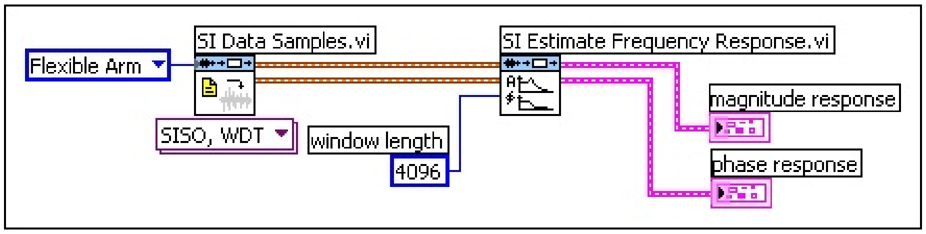
The following figure shows the input and output data on graphs during the length of the response. By looking at the graphs, you can inspect the data for outliers, clipped saturation, or quantization effects that you can remove because they are not representative of the system behavior.



The previous figure shows no obvious nominal, trend, or outlier values in the input or output time waveforms.

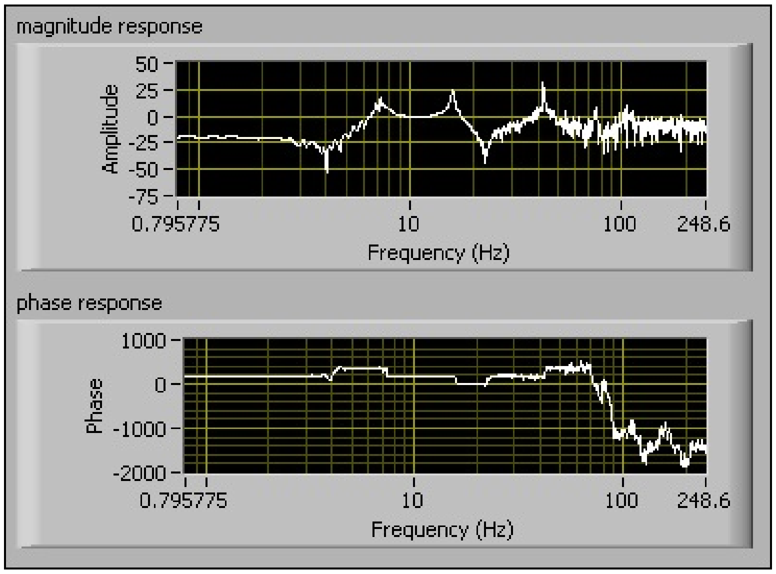
Examining the Frequency Response Data

In addition to examining the time response data, you also want to examine the frequency response data. You can use the SI Estimate Frequency Response VI to view the frequency response of the measured output signal, as shown in the following figure.



The input data is periodic over 4096 samples, which is the signal length. Notice that in the previous figure the **window length**, 4096, is the same as the signal length so as to obtain a smaller bias in the frequency response estimation.

The following figure shows the magnitude and phase responses of the measured output signal. The **magnitude response** graph shows three resonances and two anti-resonances in the frequency domain. Resonances are vibrations of large amplitude in a system caused by exciting the system at its natural frequency.



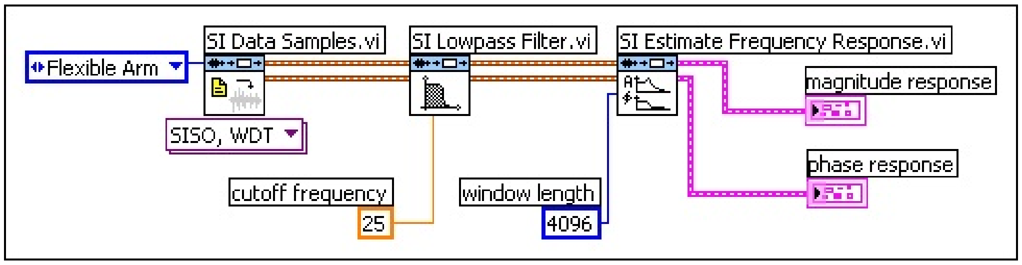
Notice the resonance at approximately 42 Hz. You can deduce that this resonance is caused by noise or nonlinear system behavior because the 42 Hz falls outside the frequency range of the input data, 0.122–24.4 Hz. At 42 Hz, there is no input energy, thus implying that the response at

42 Hz is not a result of the input.

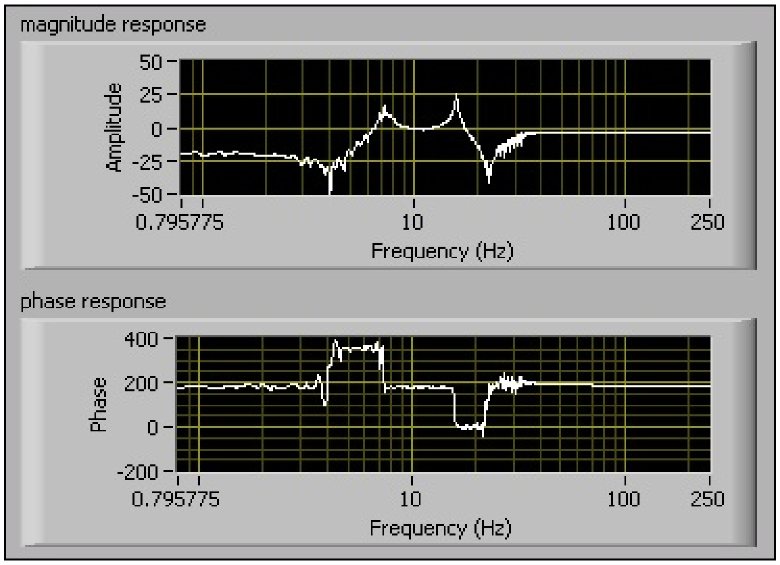
By examining the frequency response data, you see that filtering is necessary to remove this resonance peak at 42 Hz. You can use the LabVIEW System Identification Toolkit to apply a filter to the flexible arm data.

Applying a Filter to the Raw Data

To eliminate the resonance peak at 42 Hz, you can apply a filter to the raw data. By first applying a lowpass filter with a cutoff frequency of 25 Hz, you eliminate the high-frequency noise from the raw data set. The following figure shows how to use SI Lowpass Filter to apply a lowpass filter to the raw data set.



You can see the effects of the lowpass filter by comparing the frequency response of the filtered data set in the following figure to the frequency response of the non-filtered data set. By using a lowpass filter, you can see that the resonance at approximately 42 Hz is no longer part of the data set you will use to estimate the model.



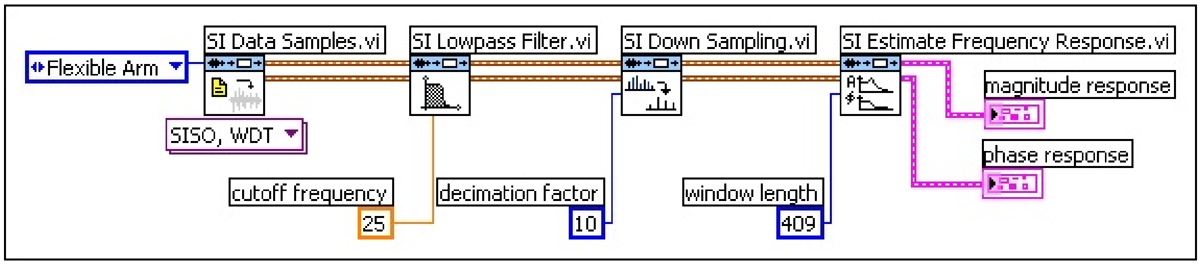
Downsampling the Raw Data

Sampling theory, in conjunction with the Nyquist criterion, enables you to reduce the sampling rate from 500 Hz to 50 Hz. Applying a filter and downsampling the data set reduces the number of samples in and the computational complexity of the data set. The goal is to use as few samples as possible to evaluate the behavior of the system.

Sampling theory enables you to downsample, or decimate, the data set. Downsampling reduces the sampling rate, 500 Hz, by a factor of 10. Thus downsampling enables you to acquire the data at a sampling rate of 50 Hz. The Nyquist criterion states that you must sample the signal at a minimum of twice the highest frequency in the system.

Recall that the input data is equally spaced over the frequency band 0.122–24.4 Hz. Therefore, according to the Nyquist criterion, you must sample at a minimum of 50 Hz to avoid any antialiasing. The benefit of sampling at 50 Hz is that you still acquire all the data in the frequency band, yet you eliminate the resonance peak at 42 Hz.

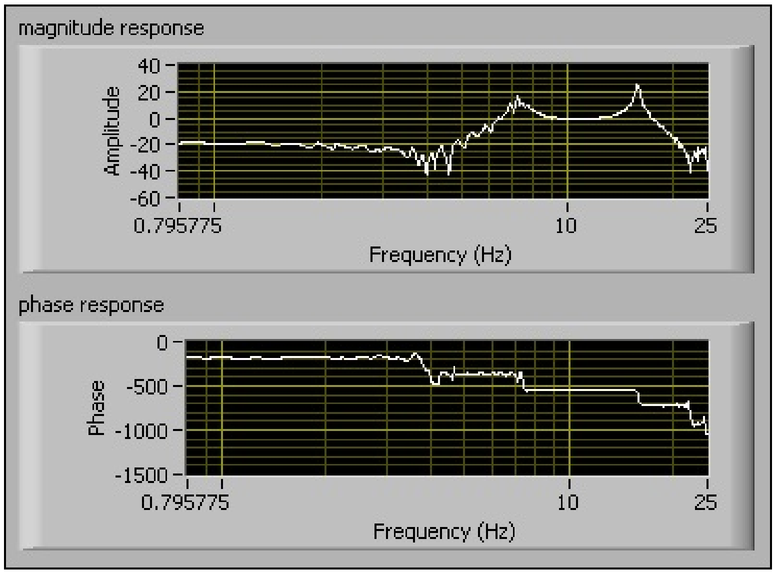
Therefore, in the following figure, the SI Lowpass Filter VI sets the **cutoff frequency** to 25. In addition to applying a lowpass filter to the data, you must downsample the reduced data set. The SI Down Sampling VI in the following figure uses a **decimation factor** of 10.



The SI Lowpass Filter VI applies a lowpass filter before downsampling the data set to avoid aliasing at the 42 Hz resonance. Together, the lowpass filter and downsampling remove the high frequency disturbance and make the process faster and more efficient.

Notice that the **window length** parameter of the SI Estimate Frequency Response VI in the previous figure is around 400 instead of 4096, as shown in the previous figure. You can reduce the window length by a factor of 10 because the number of samples in the reduced data set is one tenth of the number of samples in the raw data set.

The following figure shows the frequency response after applying a filter to and downsampling the raw data set.



Filtering and downsampling are beneficial because they eliminate the nonrealistic parts of the frequency response and reduce the amount of work required in the model estimation process.

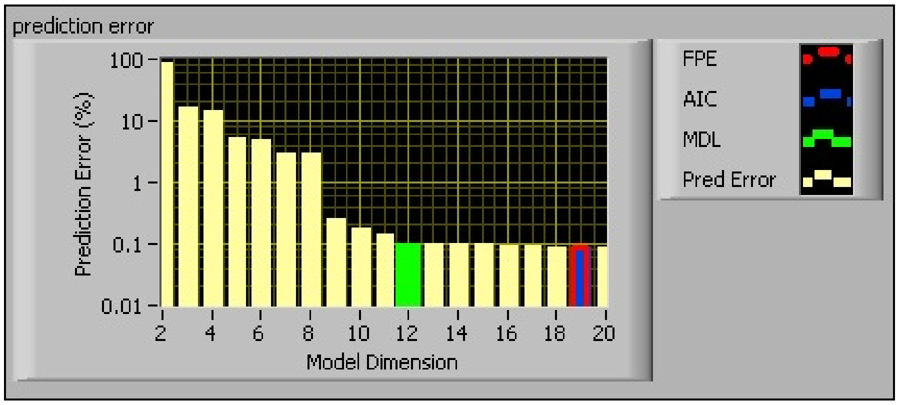
Estimating an ARX Model (System Identification Toolkit)

One of the biggest challenges in model estimation is selecting the correct model and the order of the model. The LabVIEW System Identification Toolkit supports three different criteria to aid in the estimation of the order of a model.

- Final Prediction Error (FPE) Criterion
- Akaike's Information Criterion (AIC)
- Minimum Description Length (MDL) Criterion

Sometimes the results you obtain with these three criteria might be inconsistent. You can use a pole-zero plot for further investigation and to verify the results of the order estimation.

The following figure shows a prediction error plot generated by the SI Estimate Orders of System Model VI for an ARX model. The y-axis is the prediction error and the x-axis is the model dimension. The three different color bars on the chart represent the FPE, AIC, and MDL criteria.

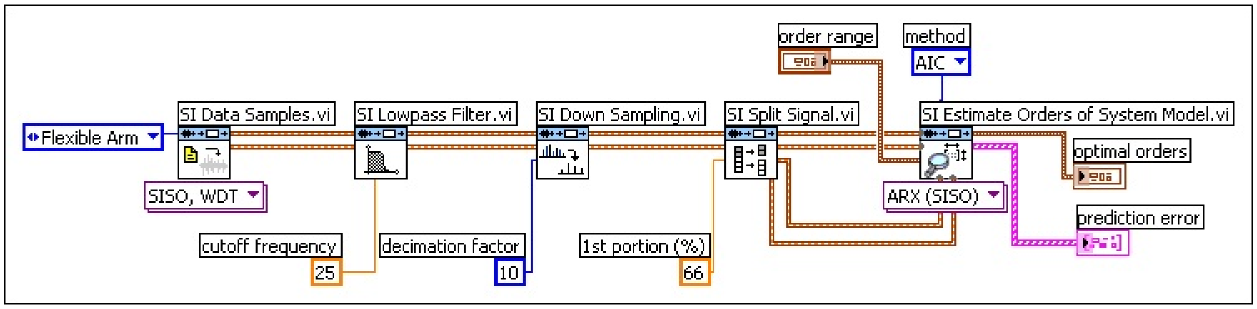


You can use the AIC, MDL, and a user-defined criterion to determine the A and B orders of the ARX model.

Akaike's Information Criterion

The following block diagram uses the SI Estimate Orders of System Model VI for order estimation. To estimate the orders of a model, the SI Estimate Orders of System Model VI requires two data sets—one for estimation and one for validation. You do not need to acquire two data sets from a system, rather, you can partition one data set into two using the SI Split Signals VI. The SI Split Signals VI divides the preprocessed data samples into a portion for model estimation and a portion for model validation.

In the following figure, the **1st portion (%)** is 66, which means the SI Estimate Orders of System Model VI will use 66% of the data samples for estimation and the remainder of the data samples for validation.



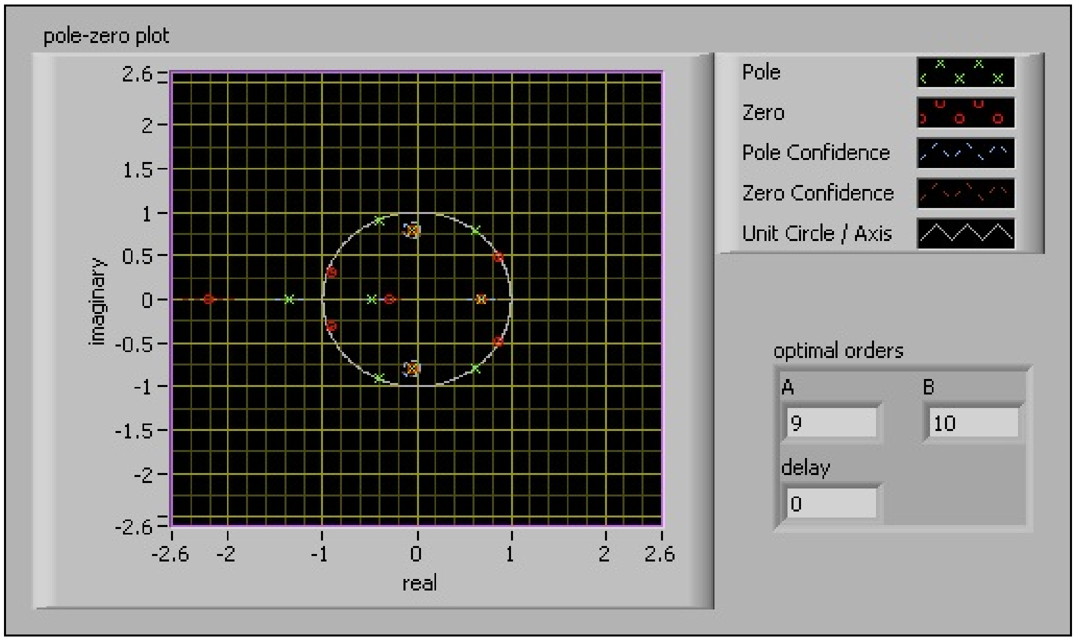
The SI Estimate Orders of System Model VI generates the **prediction error** plot for the ARX model and the optimal A order and B order based on the AIC criterion. By using the AIC criterion, the lowest prediction error corresponds to a model dimension of 19, as shown in the **prediction error** plot. For an ARX model, the model dimension is equal to the sum of the A order, B order, and delay values. The SI Estimate Orders of System Model VI returns the following optimal orders:

- A order = 9
- B order = 10
- delay = 0

Verifying the Results

After determining the orders of the model, you want to verify the results to ensure the model accurately describes the system. One method is to plot a pole-zero map and visually inspect the plot to determine whether there is any redundancy in the data. If a pole and a zero overlap, the pole and zero cancel out each other, which indicates the estimated optimal order is too high.

The **pole-zero plot** graph in the following figure shows a pole-zero plot with three overlapping pole-zero pairs. Due to numerical error, it is unlikely that a zero and a pole perfectly overlap. You can use the confidence region to justify whether the pole and the zero cancel out each other.



optimal orders

A: 9

B: 10

delay: 0

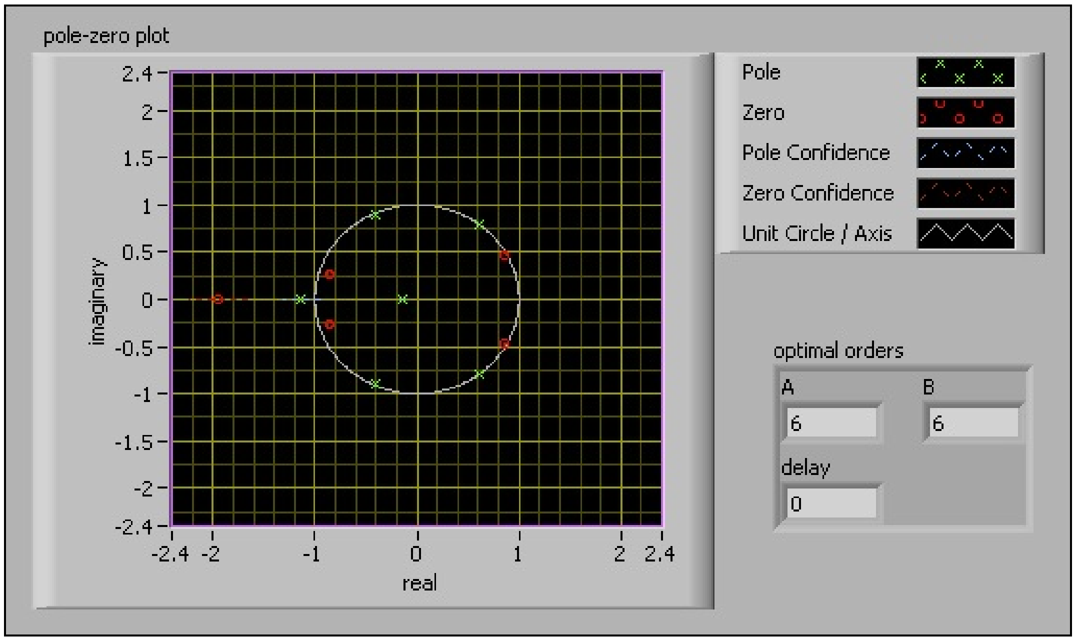
Because there are three pole-zero pairs, you can conclude that the AIC criterion does not produce the most optimal orders.

Minimum Description Length Criterion

Because the AIC criterion produced a model with non-optimal orders, you can try estimating the model orders with the MDL criterion. By using the MDL criterion, the lowest prediction error corresponds to a model dimension of 12, as shown in the **prediction error** plot. The SI Estimate Orders of System Model VI returns the following optimal orders:

- A order = 6
- B order = 6
- delay = 0

The following figure shows a pole-zero plot of a model with a model dimension of 12.



optimal orders

A: 6

B: 6

delay: 0

Compare the previous figure, which uses the MDL criterion and the figure which uses the AIC criterion. Because there are no overlapping pole-zero pairs in the previous figure, you can conclude that the MDL criterion fits better than the AIC criterion in this particular example.

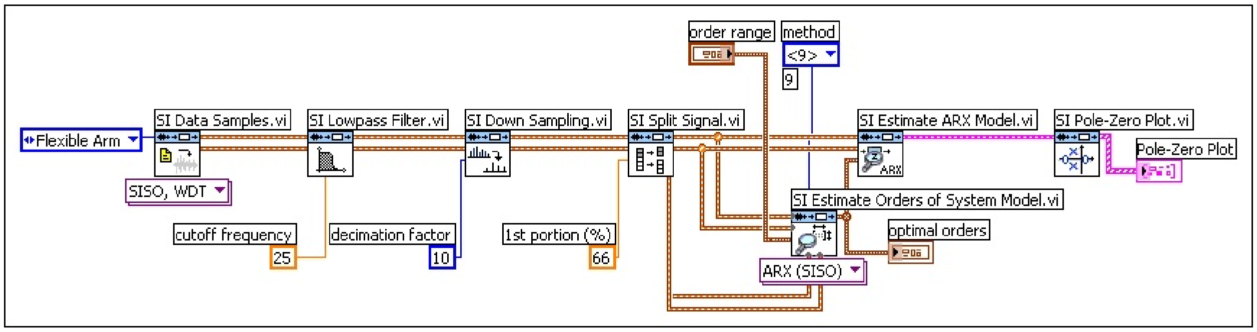
In addition to examining redundancy, you also can use the pole-zero plot for other purposes. For example, both the previous two figures show poles outside the unit circle. Having poles outside the unit circle implies that this model is not optimal because the ARX system based on the AIC or MDL criteria is unstable. One way to stabilize the system is to change the order.

In addition to the FPE, AIC, and MDL criteria, you can set user-defined orders in the SI Estimate Orders of System Model VI.

User-Defined Criterion

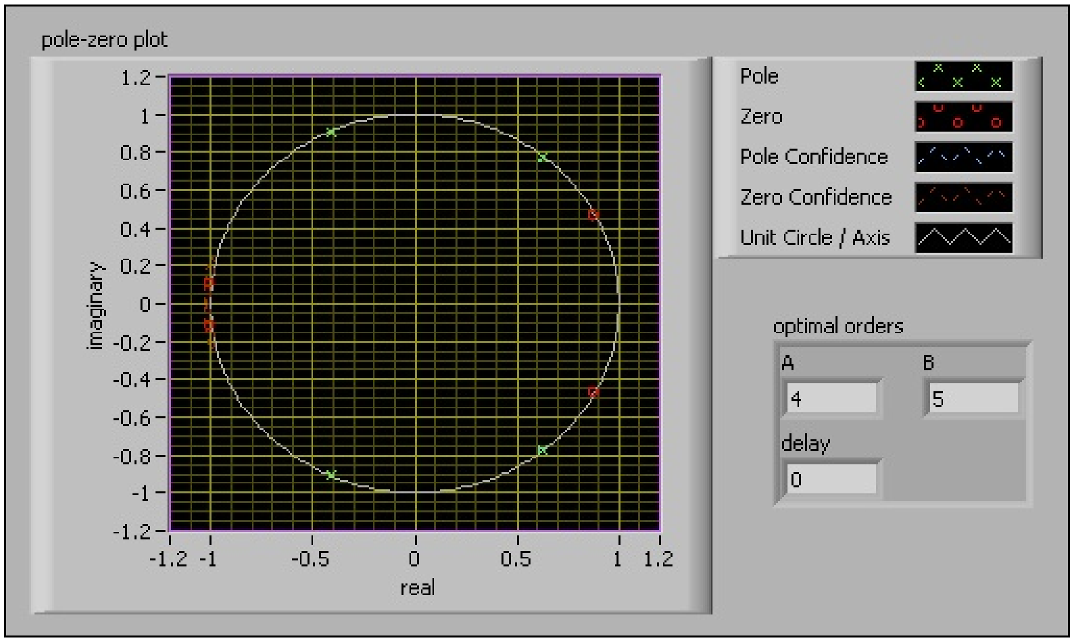
If you know nothing about the system, you might have to rely on trial and error to determine the optimal orders of the model. However, if you have some knowledge about a system, you can customize the estimation to find a model that fits a certain model dimension. For this model, assume you know that the system is stable; therefore, no poles exist outside the unit circle. Because both the AIC and MDL criterion did not produce stable models, the model orders do not describe the system accurately.

On the following block diagram, you can customize the **method** parameter. Instead of **AIC** or **MDL**, you can select **<Other>** and enter the desired model dimension in the textbox. Assume you know that the model dimension is nine.



The following figure shows the corresponding **pole-zero plot** graph with a model dimension of nine, which corresponds to the following optimal orders:

- A order = 4
- B order = 5
- delay = 0



optimal orders

A: 4

B: 5

delay: 0

Compare the pole-zero plot in the previous figure with the previous two figures which uses the MDL criterion and the AIC criterion. The figure which uses the user-defined criterion has no overlapping pole-zero pairs and all the poles are within the unit circle. By visually inspecting the pole-zero plot, you can see that this model is stable and not redundant. Using these model orders, you now can estimate and verify the system model.

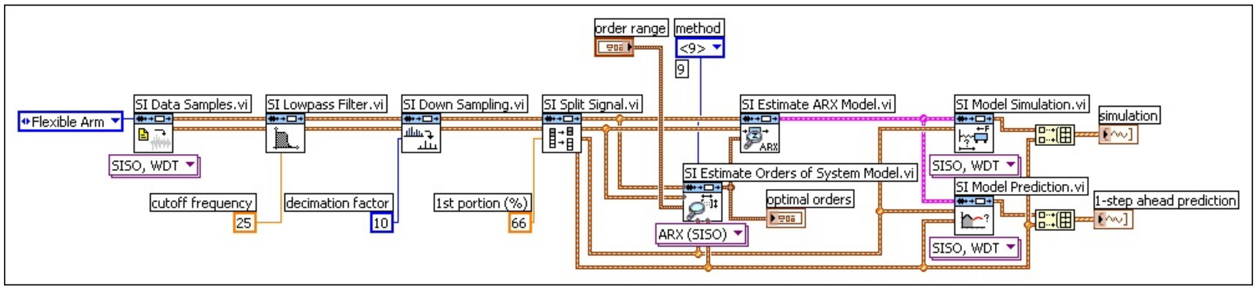
Validating the ARX Model (System Identification Toolkit)

The goal of model validation is to determine whether or not the estimated model accurately reflects the actual system. Using the model orders found in the User-Defined Criterion section, you can simulate and predict the response of the system. You can compare these responses to the actual response and determine the accuracy of the estimated model. You also can analyze the residuals to determine the accuracy of the estimated model.

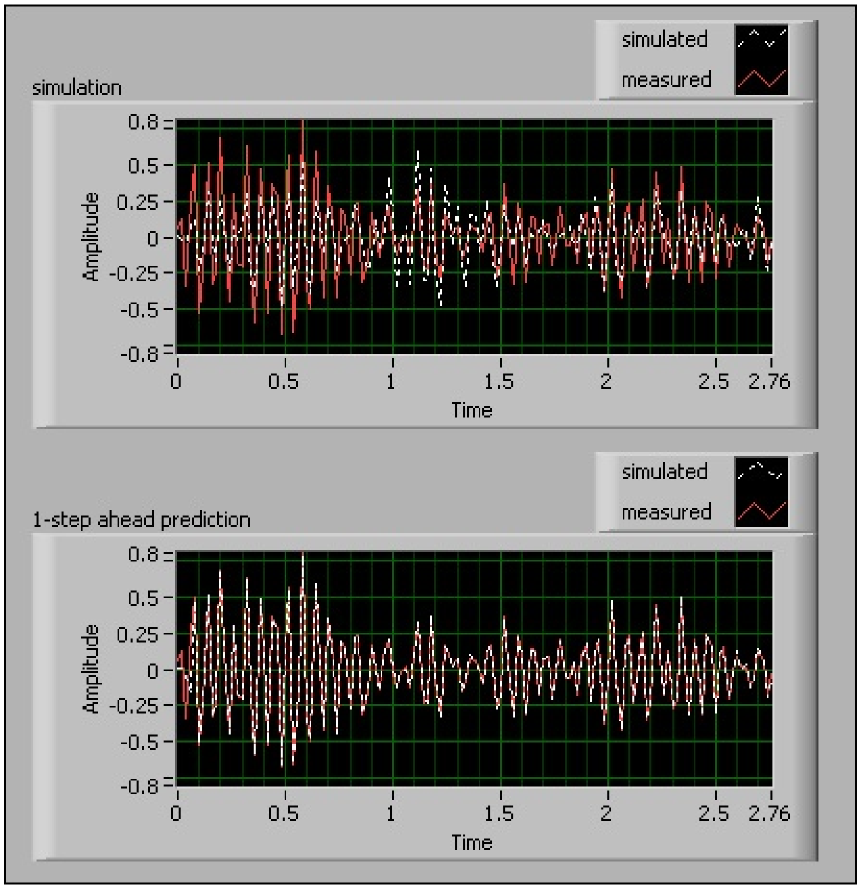
Simulation and Prediction

You can use the SI Model Simulation VI and SI Model Prediction VI to determine the accuracy of the estimated model. The SI Model Simulation VI simulates the system model and the SI Model Prediction VI performs a prediction of the system model. The results of the SI Model Prediction VI might differ from the SI Model Simulation VI because the SI Model Prediction VI periodically makes corrections to the estimated response based on the actual response of the system.

The following figure shows how you use these VIs to verify the ARX model created in the User-Defined Criterion section.



The **simulation** and **1-step ahead prediction** graphs enable you to visually determine how accurate the model is. The following figure shows the results of the simulation and prediction as well as the actual response of the system.

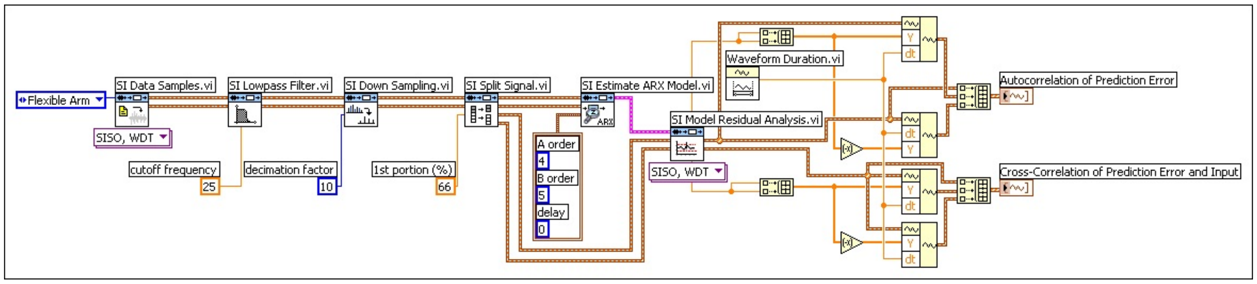


Notice how the actual response, or the **measured response**, is different from the **simulated response** in the **simulation** graph. The SI Model Simulation VI simulates the response of the system without considering the actual response of and the noise dynamics in the system.

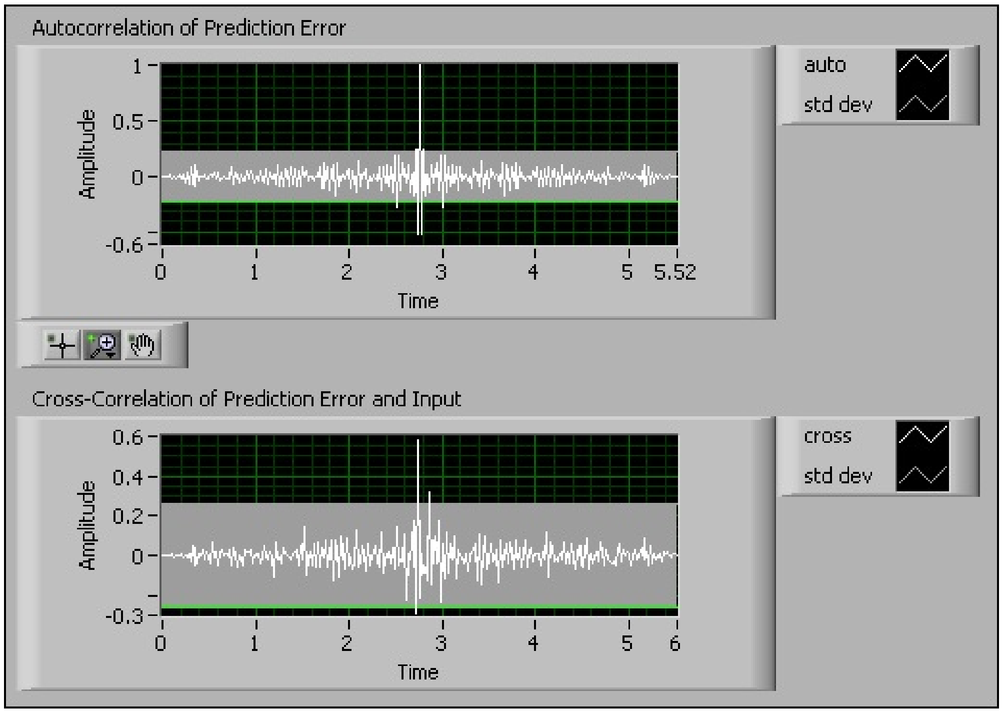
Residual Analysis

In addition to simulation and prediction, you can perform a residual analysis to validate the system model. Residual analysis tests whether the prediction error correlates to the stimulus signal. Prediction errors are usually uncorrelated with all stimulus signals in an open-loop system.

The following block diagram shows how you can use the SI Model Residual Analysis VI with the ARX model identified in the User-Defined Criterion section to analyze the residuals.

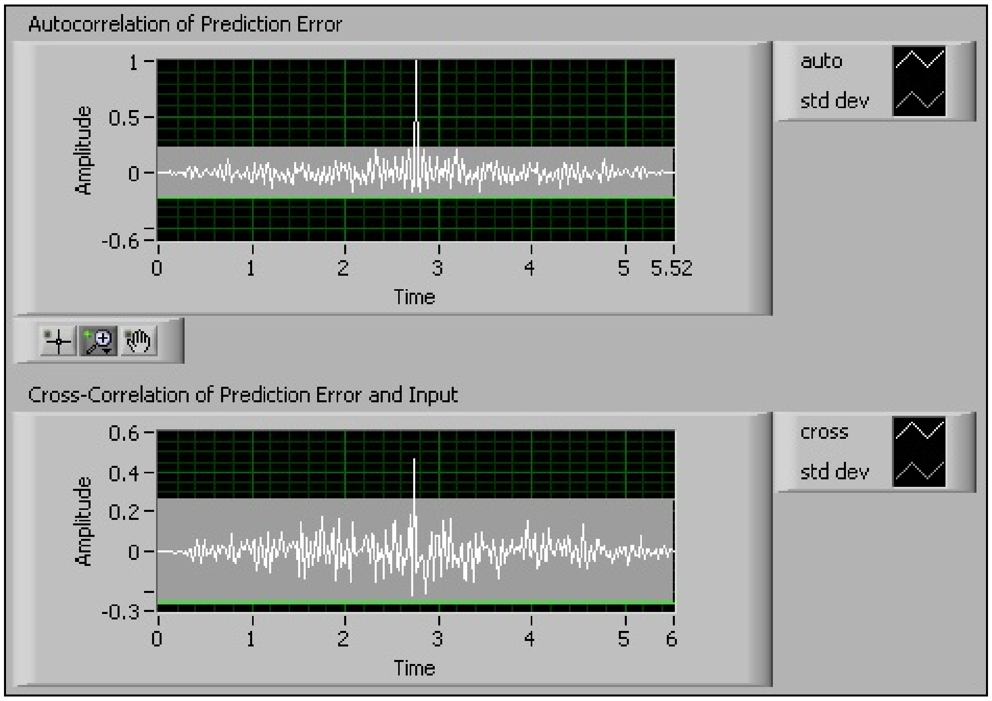


The following figure shows an example of ideal results where both autocorrelation and cross correlation are inside the confidence region except those in the vicinity of τ = 0. This result indicates that the estimated model accurately describes the system.



When you verify and validate the identified model, you must use multiple analysis techniques to determine if the estimated model accurately represents the system. Some analysis techniques can be misleading. For example, if you performed a residual analysis on the model identified in the Minimum Description Length Criterion section, you might conclude that this model is an accurate representation of the system. The following figure shows the autocorrelation and cross-correlation residual analysis for the model in the Minimum Description Length Criterion section. Recall that this model has the following orders:

- A order = 6
- B order = 6
- delay = 0



The previous figure shows that both the autocorrelation and cross correlation are inside the confidence region. Therefore, without performing any other analyses, you might conclude that this model is an accurate representation of the system. However, the pole-zero analysis in the Minimum Description Length Criterion section showed poles outside of the unit circle. So you already determined that this model is unstable. Thus, despite acceptable autocorrelation and cross-correlation values, concluding that this model is accurate is incorrect.

Thus, if you only performed a residual analysis, you might not discover that this model is actually unstable. When validating a model, perform multiple analyses to ensure the accuracy of the model.

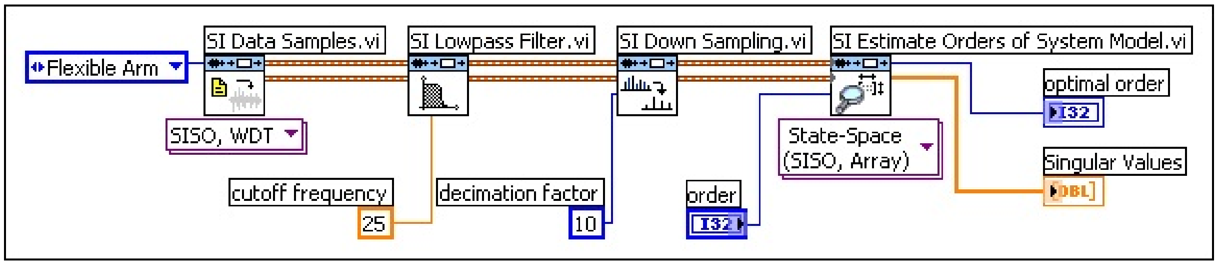
Estimating and Validating a State-Space Model (System Identification Toolkit)

For a state-space model, order estimation is equivalent to estimating the number of significant singular values, which correspond to the number of states in the model. After identifying a state-space model that represents the system, you can use the same validation and verification technique used in the Simulation and Prediction section and the Residual Analysis section.

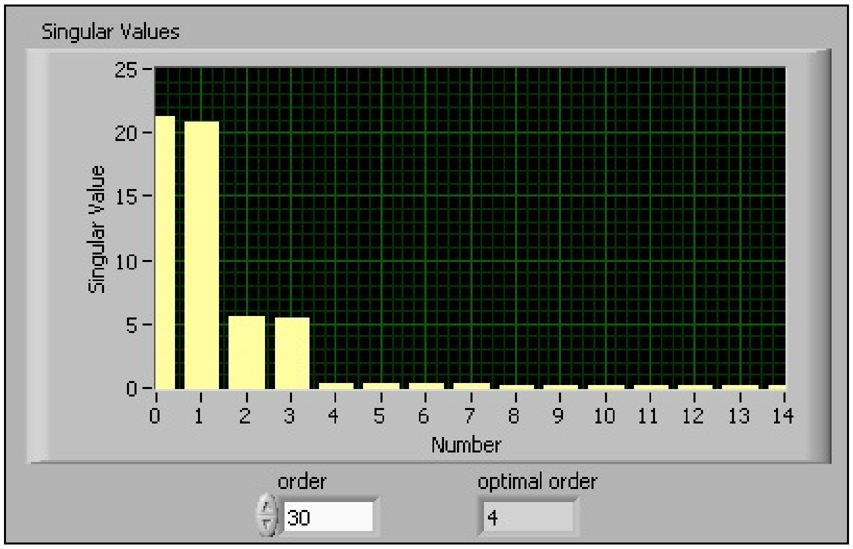
The examples in this topic use the same flexible robotic arm data and the same preprocessing techniques.

Finding the Singular Values

The following block diagram shows how to use the SI Estimate Orders of System Model VI to find the optimal order and the number of significant singular values.



The **Singular Values** graph in the following figure shows a singular value plot with four leading singular values.



order: 30

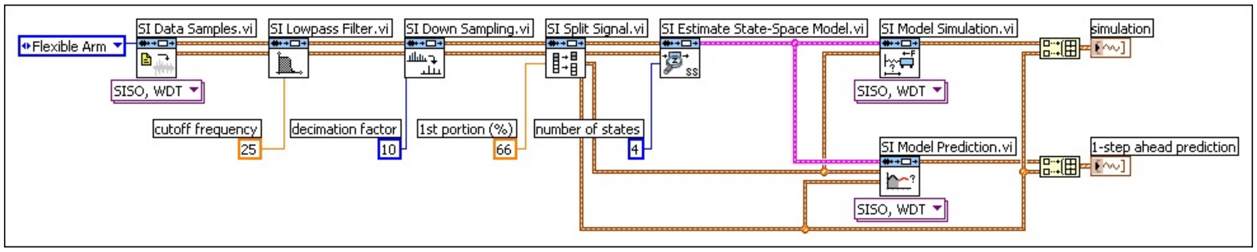
optimal order: 4

By looking both at the **Singular Values** graph and the **optimal order**, you can see that there are four states in this state-space model.

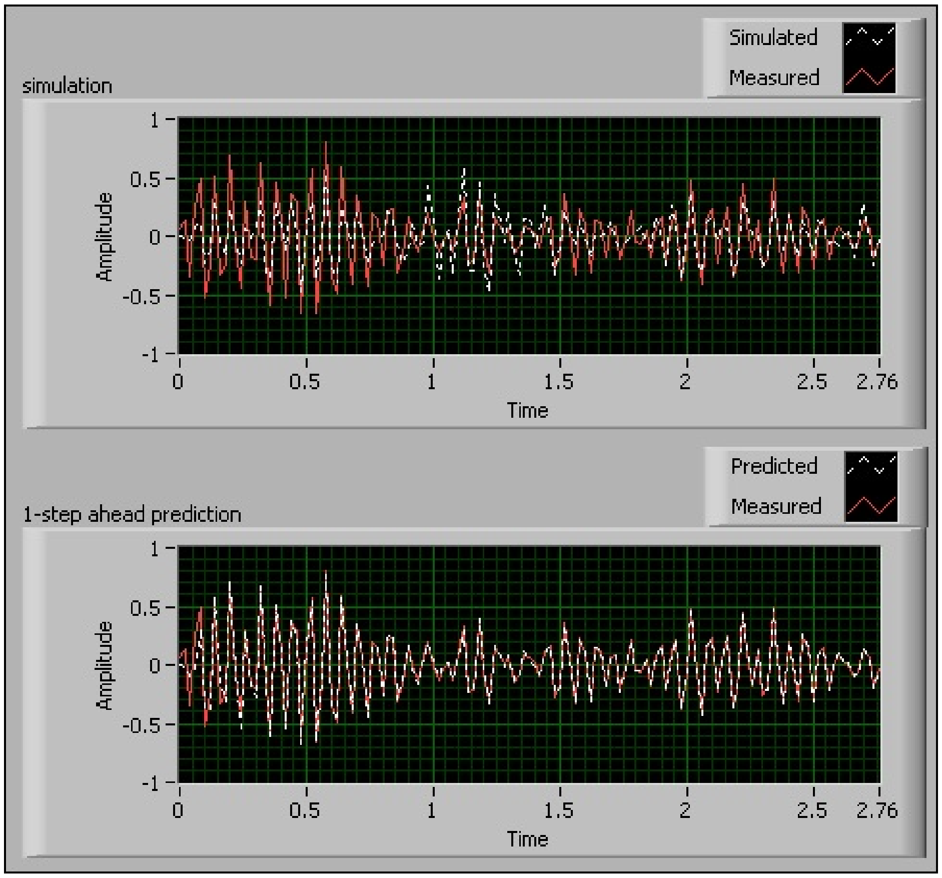
Validating the Estimated State-Space Model

You can validate the state-space model in the same way that you validated the ARX model. You use the SI Model Simulation VI and the SI Model Prediction VI to determine the accuracy of the state-space model.

The following figure shows the complete process, from estimating the state-space model to simulating and predicting the response of the model.



The **simulation** and **1-step ahead prediction** graphs in the previous figure show simulation and prediction plots for a state-space model.


simulation
Simulated
Measured
Amplitude
Time
1-step ahead prediction
Predicted
Measured
Amplitude
Time