Signal Processing Blockset™ 6
Getting Started Guide

MATLAB®
&SIMULINK®

The MathWorks™
Accelerating the pace of engineering and science
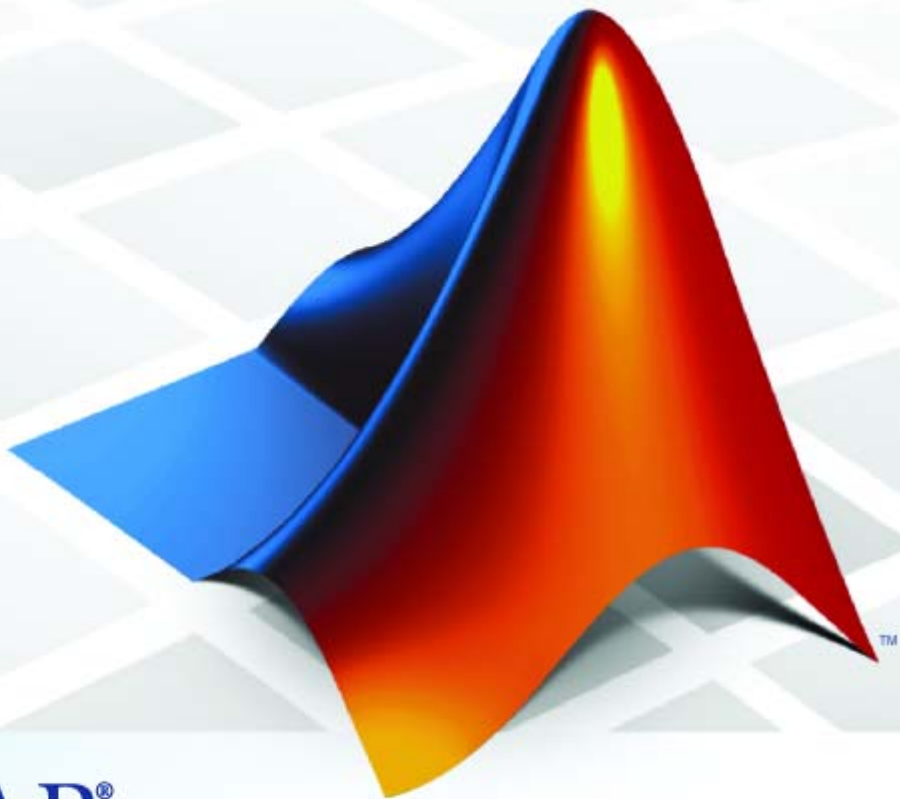
How to Contact The MathWorks

	www.mathworks.com	Web
	comp.soft-sys.matlab	Newsgroup
	www.mathworks.com/contact_TS.html	Technical Support
@	suggest@mathworks.com	Product enhancement suggestions
	bugs@mathworks.com	Bug reports
	doc@mathworks.com	Documentation error reports
	service@mathworks.com	Order status, license renewals, passcodes
	info@mathworks.com	Sales, pricing, and general information

☎ 508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Signal Processing Blockset™ Getting Started Guide

Trademarks

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

Patents

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

Revision History

Contents

Signal Processing Models

3

Filters

4

Code Generation

5

Frequency Domain Signals

6

Index

1

Introduction

- "Product Overview" on page 1-2
- "System Setup" on page 1-3
- "Product Demos" on page 1-5
- "Working with the Documentation" on page 1-9

Product Overview

Signal Processing Blockset™ software provides tools for digital signal processing (DSP) algorithm simulation and code generation. It adds frame-based processing to the Simulink® environment. The blockset is made up of block libraries containing signal processing, linear algebra, and matrix math blocks. All of the blocks support double- and single-precision floating-point data types. Most blocks also support fixed-point and integer data types when you also have the Simulink® Fixed Point™ product. You can interconnect Signal Processing Blockset blocks to create sophisticated models capable of simulating operations such as speech and audio processing, wireless digital communications, radar/sonar, and medical electronics.

The Signal Processing Blockset product requires Simulink, a tool for simulating dynamic systems. Simulink is a *model definition* environment. Use Simulink blocks to create a block diagram that represents the computations of your system or application. Simulink is also a *model simulation* environment. Run the block diagram to see how your system behaves. All Signal Processing Blockset blocks are designed for use with the blocks in the Simulink libraries. If you are new to Simulink, read *Simulink Getting Started Guide* to better understand its functionality.

You can use Signal Processing Blockset software and Simulink to develop signal processing concepts and to efficiently revise and test these concepts until your design is production ready. You can also use the blockset in conjunction with the Real-Time Workshop® product to automatically generate code for real-time execution on DSP hardware.

System Setup

In this section...
"Installation" on page 1-3
"Required Products" on page 1-3
"Related Products" on page 1-4

Installation

Before you begin working, you need to install the product on your computer.

Installing the Signal Processing Blockset Software

The Signal Processing Blockset software follows the same installation procedure as the MATLAB® toolboxes. See the MATLAB installation documentation for your platform.

Installing Online Documentation

Installing the documentation is part of the installation process:

- Installation from a DVD — Start the MathWorks™ installer. When prompted, select the **Product** check boxes for the products you want to install. The documentation is installed along with the products.

- Installation from a Web download — If you update the Signal Processing Blockset software using a Web download and you want to view the documentation with the MATLAB Help browser, you must install the documentation on your hard drive.

  Download the files from the Web. Then, start the installer, and select the **Product** check boxes for the products you want to install. The documentation is installed along with the products.

Required Products

The Signal Processing Blockset product is part of a family of products from The MathWorks™. You need to install the following products to use the blockset:

- MATLAB — You use MATLAB to open model files and view Signal Processing Blockset demos. You can import signal values from the MATLAB workspace into signal processing models and export signal values from signal processing models to the MATLAB workspace.

- Simulink — Simulink provides an environment that enables you to create a block diagram to model your physical system. You can create these block diagrams by connecting blocks and using graphical user interfaces (GUIs) to edit block parameters.

- Signal Processing Toolbox™ — The Signal Processing Toolbox product provides basic filter capabilities. You can design and implement filters using the Filter Design and Analysis Tool (FDATool) and use them in your signal processing models.

Related Products

The MathWorks provides several products that are relevant to the kinds of tasks you can perform with Signal Processing Blockset software.

For more information about any of these products, see either

- The online documentation for that product if it is installed on your system

- The MathWorks Web site, at `http://www.mathworks.com/products/sigprocblockset/related.jsp`

Product Demos

In this section...
"Demos in the Help Browser" on page 1-5
"Demos on the Web" on page 1-8
"Demos on MATLAB Central" on page 1-8

Demos in the Help Browser

You can find interactive Signal Processing Blockset demos in the MATLAB Help browser. This example shows you how to locate and open a typical demo:

1 To open the Help browser, type doc at the MATLAB command line.

2 To see a list of demo categories, expand the **Signal Processing Blockset** node in the Help browser, then the **Demos** node:
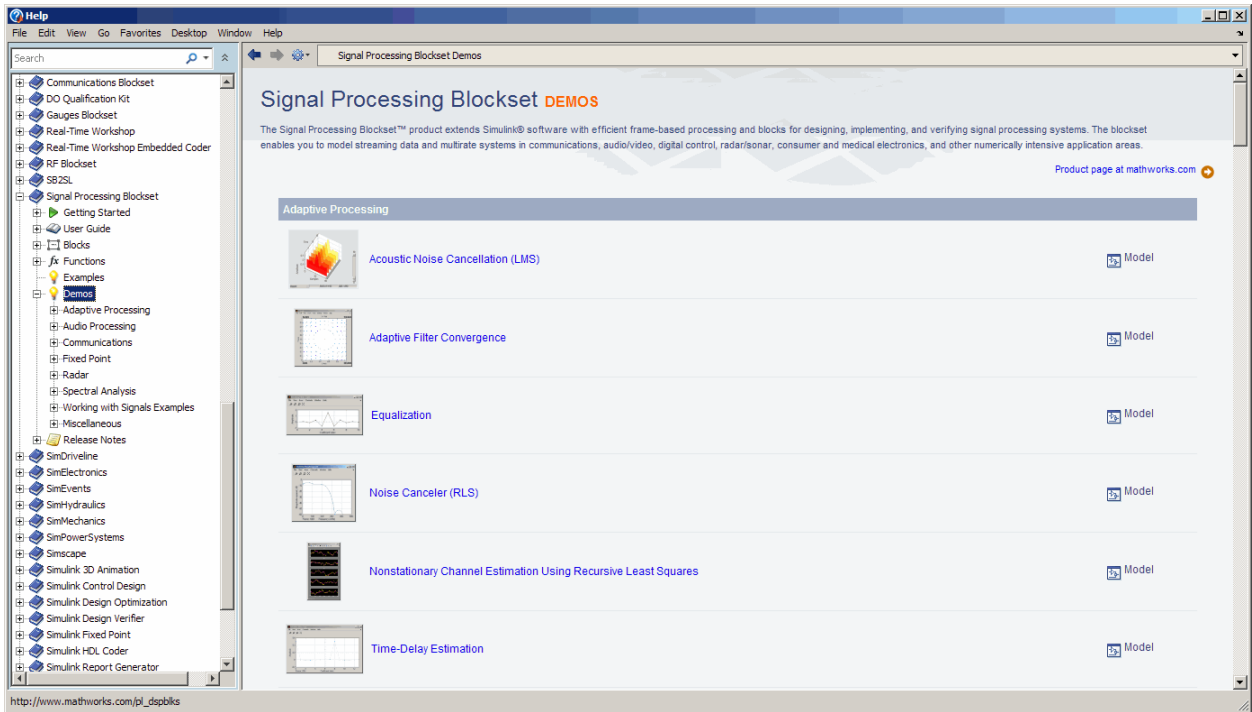
3 To view the description of the Equalization demo, which demonstrates adaptive channel equalization, expand **Adaptive Processing** then click **Equalization**.

4 Click **Open this model** to display the Simulink model for the Equalization demo. The model window opens, and a Results window opens to display the simulation results.

5 Run the model by selecting **Start** from the **Simulation** menu in the model window. The results of the simulation appear in the Results window.
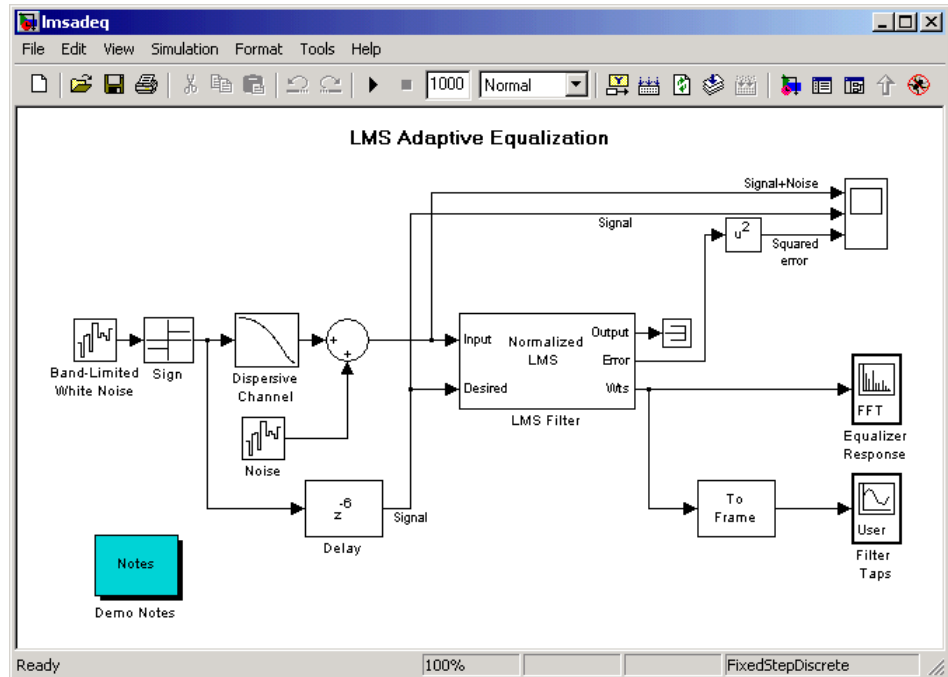
Demos on the Web

The MathWorks Web site contains demos that show you how to use
Signal Processing Blockset software. You can find these demos at
`http://www.mathworks.com/products/sigprocblockset/demos.jsp`.

You can run these demos without having MATLAB or the Signal Processing
Blockset product installed on your system.

Demos on MATLAB Central

MATLAB Central contains files, including demos, contributed by users and
developers of Signal Processing Blockset, MATLAB, Simulink, and other
products. Contributors submit their files to one of a list of categories. You can
browse these categories to find submissions that pertain to Signal Processing
Blockset software or a specific problem that you want to solve. MATLAB
Central is located at `http://www.mathworks.com/matlabcentral/`.

Working with the Documentation

In this section...
"Viewing the Documentation" on page 1-9
"Printing the Documentation" on page 1-10
"Using This Guide" on page 1-10

Viewing the Documentation

You can access the Signal Processing Blockset documentation using files you installed on your system or from the Web using the MathWorks Web site.

Documentation in the Help Browser

This procedure shows you how to use the Help browser to view the blockset documentation installed on your system:

1 In the MATLAB window, from the **Help** menu, click **Product Help**. The Help browser opens.

2 From the list of products in the left pane, click **Signal Processing Blockset**. In the right pane, the Help browser displays the Signal Processing Blockset roadmap page.

3 Under the section titled **Documentation Set**, click **Getting Started**. The Help browser displays the chapters of this manual.

Documentation on the Web

You can also view the documentation from the MathWorks Web site. The documentation available on these Web pages is for the latest release, regardless of whether the release was distributed on a DVD or as a Web download:

1 Navigate to the Product Page at http://www.mathworks.com/products/sigprocblockset/.

2 Click the **Documentation** link on the left side of the page. The blockset documentation is displayed.

Printing the Documentation

The Signal Processing Blockset documentation is also available in printable PDF format. To view the documentation in PDF format:

1 In the MATLAB window, from the **Help** menu, click **Product Help**. The Help browser opens.

2 From the list of products in the left pane, click **Signal Processing Blockset**. In the right pane, the Help browser displays the Signal Processing Blockset roadmap page.

3 Under the **Printable (PDF) Documentation on the Web** section, click the links to view PDF versions of the blockset documentation.

Using This Guide

To help you effectively read and use this guide, here is a brief description of the chapters and a suggested reading path.

Expected Background

This manual assumes that you are already familiar with the following products:

- MATLAB, to write scripts and functions with M-code, and to use functions with the command-line interface

- Simulink, to create simple models as block diagrams and simulate those models

What Chapters Should I Read?

If You Are a New User — In the Getting Started Guide:

- Read Chapter 1, "Introduction" to learn about the installation process, the products required to run Signal Processing Blockset software, and how to view the blockset demos.

- Read Chapter 2, "Concepts, Terminology, and Feature Overview" to learn about blockset functionality, review key concepts and terminology, and find out more about product features.

- Read Chapter 3, "Signal Processing Models" to learn how to build a signal processing model and simulate its behavior.

- Read Chapter 4, "Filters" to create an adaptive noise cancellation system using digital and adaptive filters.

- Read Chapter 5, "Code Generation" to generate ANSI® C code from your signal processing model.

- Read Chapter 6, "Frequency Domain Signals" to learn how to view the spectral content of a speech signal.

If You Are an Experienced User — In the User's Guide:

- Read Chapter 1, "Working with Signals" and Chapter 2, "Advanced Signal Concepts" for details on key operations common to many signal processing tasks.

- Read the following chapters for discussions of how to implement various signal processing operations:

  - Chapter 3, "Filters"
  - Chapter 4, "Transforms"
  - Chapter 5, "Quantizers"
  - Chapter 6, "Statistics, Estimation, and Linear Algebra"
  - Chapter 7, "Working with Fixed-Point Data"

- See the block reference for a description of each block's operation, parameters, and characteristics.

2

Concepts, Terminology, and Feature Overview

Sample Model and Block Libraries

In this section...
"Modeling System Behavior" on page 2-2
"Signal Processing Blockset Blocks" on page 2-5

Modeling System Behavior

Signal Processing Blockset blocks can simulate the behavior of complex signal processing systems. For example, the Acoustic Noise Canceler demo model in this section illustrates some of the capabilities of the blockset. In the model, the signal output at the upper port of the Acoustic Environment subsystem is white noise. The signal output at the lower port contains colored noise and a signal from a .wav file. This demo model uses an adaptive filter to remove the noise from the signal output at the lower port. When you run the model, you hear both noise and a person playing the drums. Over time, the adaptive filter in the model filters out the noise so all you hear is the person playing the drums.

Note Later, this manual shows you how to create a similar model.

---

1 Open the Acoustic Noise Canceler demo model by typing `dspanc` at the MATLAB command prompt. The demo model and the dspanc/Waterfall scope window open. We discuss the scope window later in this procedure.
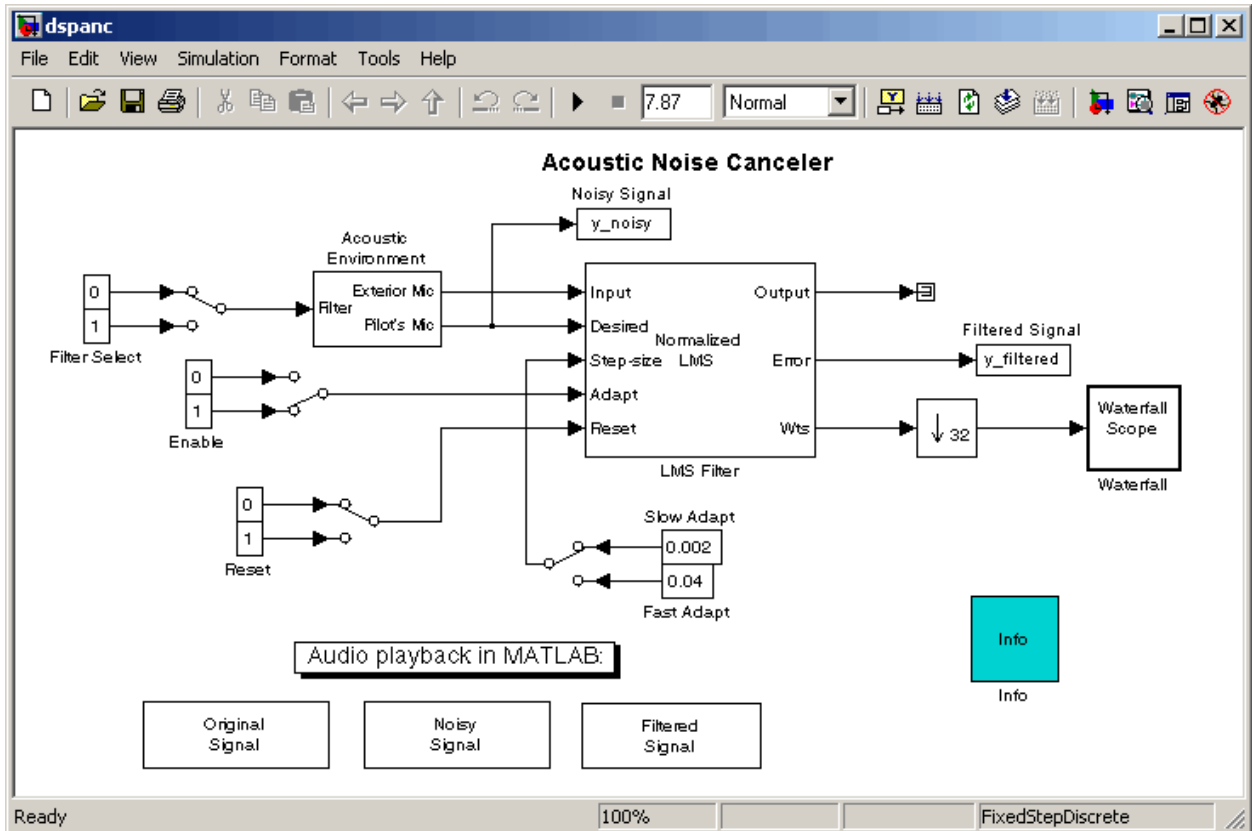
2 Run this demo by selecting **Start** from the **Simulation** menu.

3 After the demo runs, listen to each of the signals by double-clicking the Original Signal, Noisy Signal, and Filtered Signal blocks. Notice that as the filter coefficients change, the noise in the signal decreases and you can hear the drums more clearly.

4 The dspanc/Waterfall scope window displays the behavior of the adaptive filter coefficients. The following figure shows the scope window when the simulation begins. Each plot represents the values of the filter coefficients of a normalized LMS adaptive filter. In the figure, you can see that they are initialized to zero. Also, the color of the plots fades from red to yellow.

The current filter coefficients are plotted in red. The other plots represent the filter coefficients at previous simulation times.
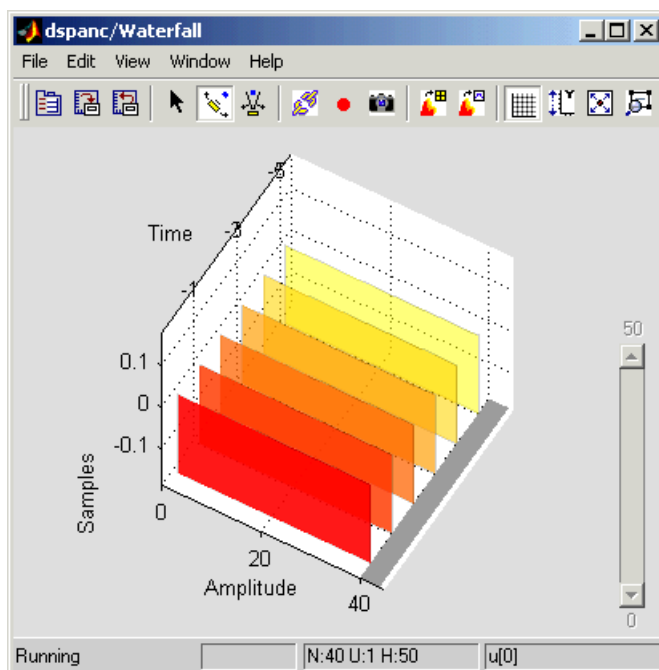
The next figure shows the dspanc/Waterfall scope window when the filter coefficients have reached their steady state.
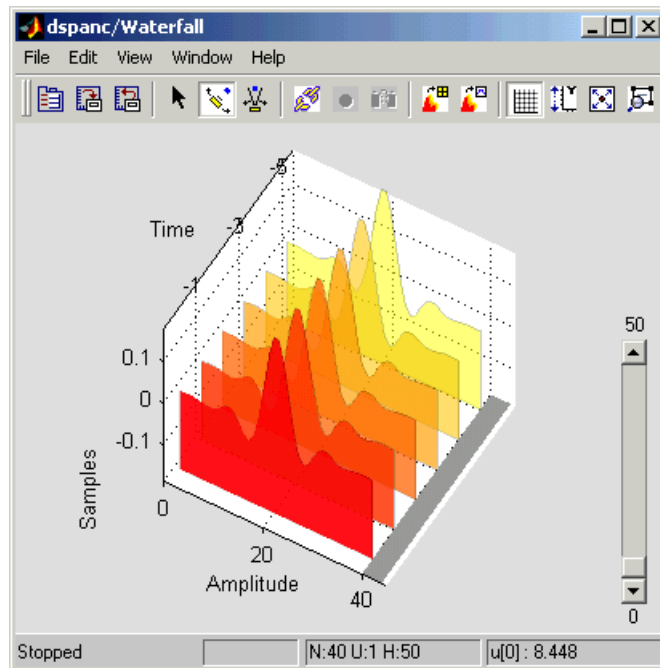
5 To speed up or slow down the rate of filter adaption, double-click the switch attached to the blocks labeled Fast Adapt and Slow Adapt. When you connect the switch to the block labeled Fast Adapt, the filter coefficients reach steady state in a shorter time. To hear the difference in the filtered signals, run the demo using both the fast adapt and the slow adapt rates. Listen to the filtered signal produced by each.

The "Adaptive Filters" section of the *Signal Processing Blockset User's Guide* contains more information on the Acoustic Noise Canceler demo.

Signal Processing Blockset Blocks

The Signal Processing Blockset product organizes a collection of blocks within nested libraries. These libraries are specifically for digital signal processing applications. They include blocks for operations such as multirate and adaptive filtering, matrix manipulation, linear algebra, statistics, and time-frequency transforms. You can locate these blocks using the main Signal Processing Blockset library or the Simulink Library Browser:

Accessing Blocks Directly

You can access the main Signal Processing Blockset library from the MATLAB command line. This procedure shows you how to open this library and locate the source blocks:

1 Open the library by typing `dsplib` at the MATLAB command prompt.
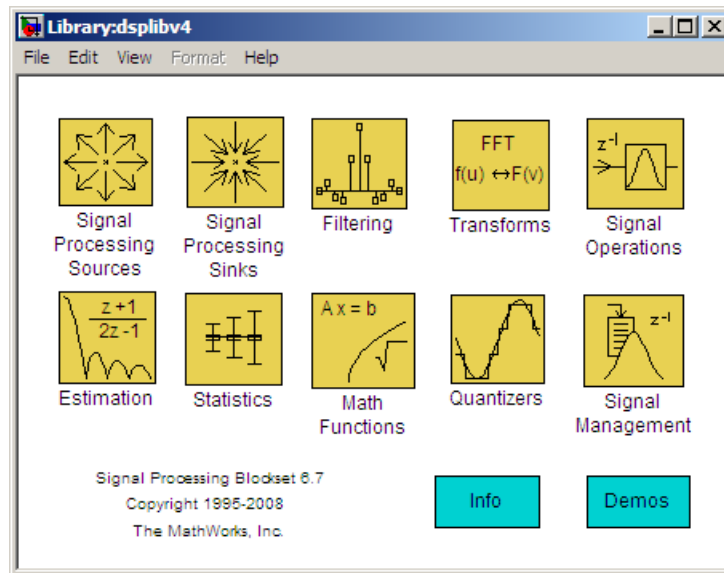
The Signal Processing Blockset libraries are

- Signal Processing Sources — Blocks that create discrete-time or continuous-time signals or import these signals from the MATLAB workspace

- Signal Processing Sinks — Blocks used to display data in a scope or send data to the MATLAB workspace

- Filtering — Blocks used to design digital, analog, adaptive, and multirate filters

- Transforms — Blocks that transform data into different domains

- Signal Operations — Blocks that perform operations such as convolution, downsampling, upsampling, padding, and delaying the input

- Estimation — Blocks for linear prediction, parametric estimation, and power spectrum estimation

- Statistics — Blocks that perform statistical operations such as correlation, maximum, and mean

- Math Functions — Blocks used to perform mathematical operations, matrix operations, and polynomial functions

- Quantizers — Blocks that create scalar and vector quantizers as well as uniform encoders and decoders

- Signal Management — Blocks for buffering, selecting part of a signal, modifying signal attributes, and edge detection

2 Double-click the Signal Processing Sources library. The library displays the blocks it contains. You can use the blocks in this library to create discrete-time or continuous-time signals.
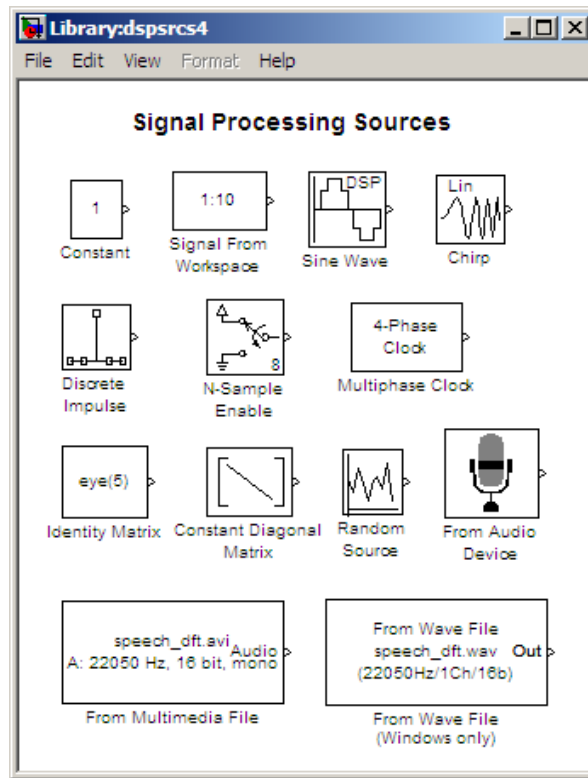
3 Drag any block into a model, double-click the block, and click **Help** to learn more about the block functionality.

Accessing Blocks with the Library Browser

Starting the Simulink product displays the Simulink Library Browser. To start Simulink, type `simulink` at the MATLAB command line. One way to explore the Signal Processing Blockset product is to expand the **Signal Processing Blockset** entry in the tree pane of this browser.
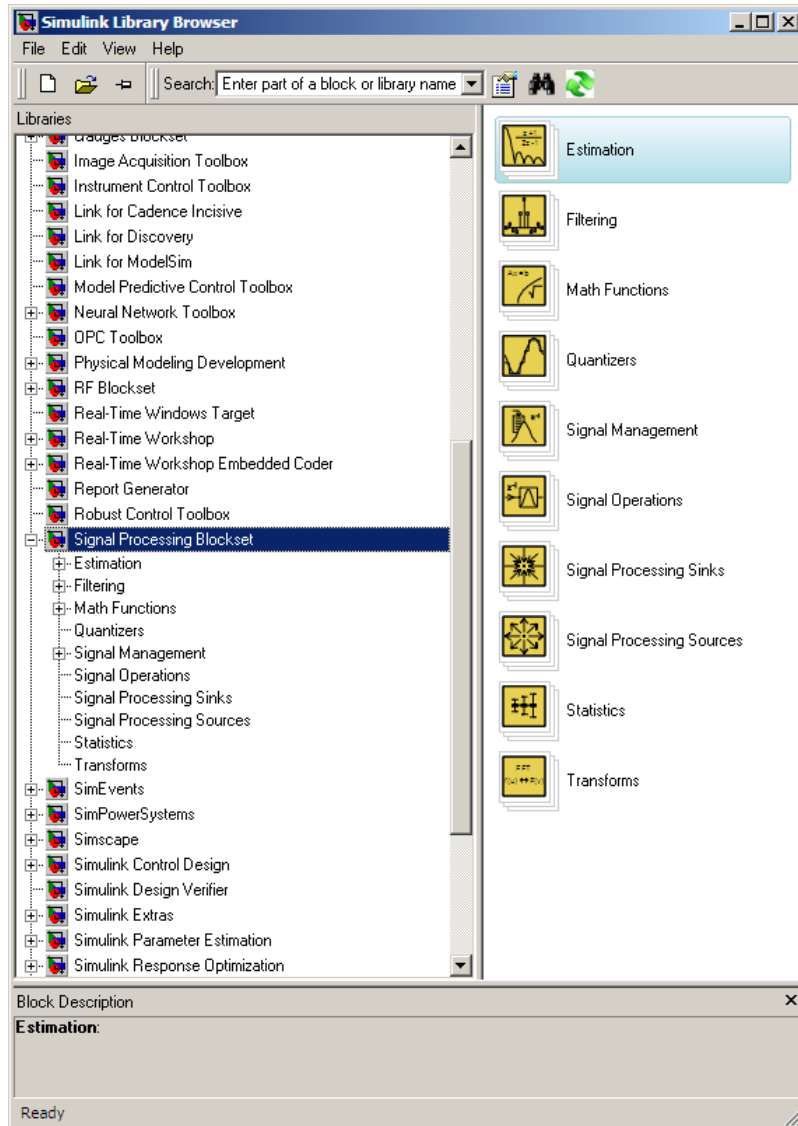
Key Blockset Concepts

Signals

Signals in the Simulink environment can be real or complex valued. You can represent signals with data types such as single-precision floating point, double-precision floating point, or fixed point. Signals can be either sample based or frame based, and single channel or multichannel.

Sample Time

A discrete-time signal is a sequence of values that correspond to particular instants in time. The time instants at which the signal is defined are the signal sample times, and the associated signal values are the signal samples. For a periodically sampled signal, the equal interval between any pair of consecutive sample times is the signal sample period, T_s. The sample rate, F_s, is the reciprocal of the sample period. It represents the number of samples in the signal per second:

$$F_s = \frac{1}{T_s}$$

Note In the block parameter dialog boxes, the term *sample time* refers to the *sample period* of the signal T_s.

State

Some Signal Processing Blockset blocks have state and others do not. If a block does not have state, the block calculates its output using only the current input. If a block has state, the output of the block depends on the current input as well as past inputs and/or outputs.

Sample-Based Signals

A signal is sample based if it propagates through the model one sample at a time. To represent a single-channel sample-based signal, create a 1-by-1-by-T matrix. Each matrix element represents one sample from the channel, and T is the total number of samples in the channel. To represent a multichannel signal with $M*N$ independent channels, create an M-by-N-by-T matrix. Each matrix element represents one sample from a distinct channel, and T is the total number of samples in each channel.

Consider the following model.
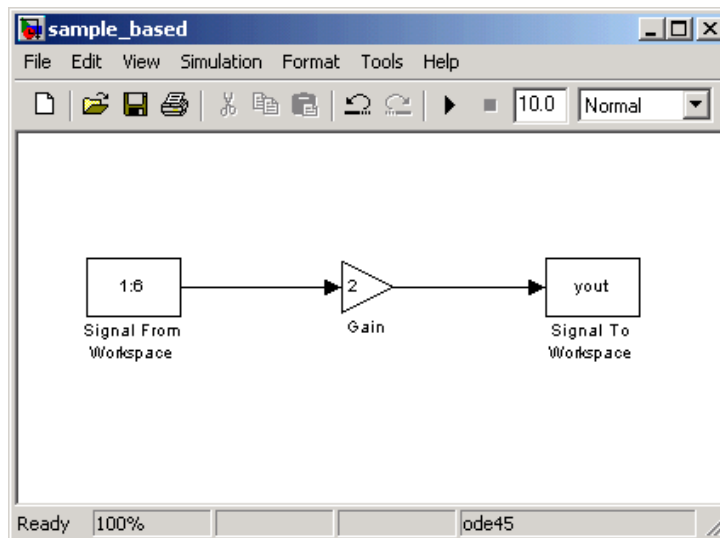
The Signal From Workspace block outputs a sample-based signal. The Gain block multiplies all the samples of the signal by two. Then, the Signal To Workspace block outputs the signal to the MATLAB workspace in a variable

called yout. The following figure is a symbolic representation of how the single-channel, sample-based signal propagates through the model.
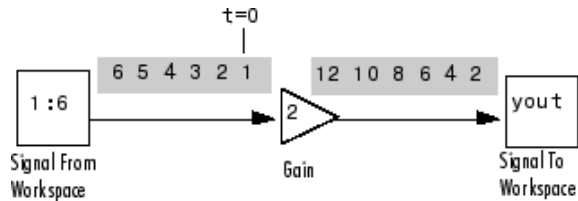
If you type yout at the MATLAB command prompt after you run the model, you see, in part:

```
yout(:,:,1) =

     2

yout(:,:,2) =

     4

yout(:,:,3) =

     6
```

Because yout represents a single-channel, sample-based signal, each sample of the signal is a different page of the output matrix.

Frame-Based Signals

A signal is frame based if it propagatesthrough a model one frame at a time. A frame of data is a collection of sequential samples from a single channel or multiple channels. One frame of a single-channel signal is represented by an *M*-by-1 column vector. One frame of a multichannel signal is represented by an*M*-by-*N* matrix. Each matrix column is a different channel, and the number of rows in the matrix is the number of samples in each frame.

You can typically specify whether a signal is frame based or sample based using a source block from the Signal Processing Sources library. Most other

signal processing blocks preserve the frame status of an input signal, but some do not.

The process of propagating frames of data through a model is *frame-based processing*. Because multiple samples can process at once, the computational time of the model improves. "Working with Signals" in the *Signal Processing Blockset User's Guide* contains more information about frame-based processing.

Consider the following model.
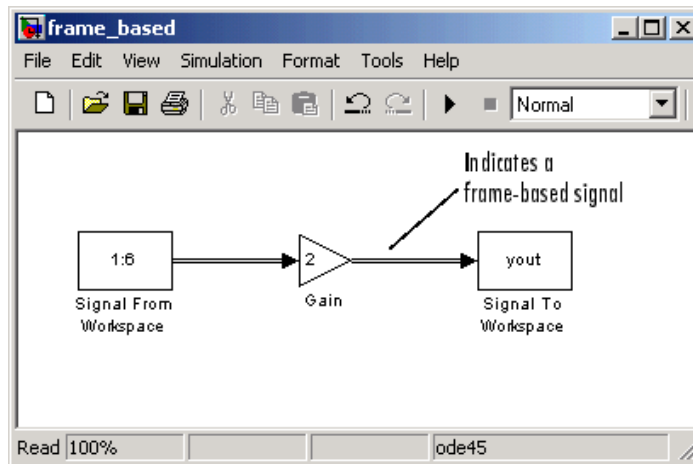
To have the Signal From Workspace block output a frame-based signal, set the **Samples per frame** parameter to 2 and run the model. The lines that connect the blocks become double lines, indicating a frame-based signal; in this example, there are two signals per frame.

The Gain block multiplies all the samples of this signal by two. Then, the Signal To Workspace block outputs the signal to the MATLAB workspace in the form of a variable called yout. The following figure is a symbolic representation of how the frame-based signal propagatesthrough the model.
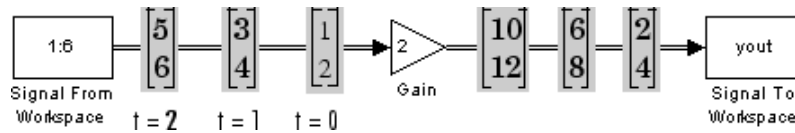
If, after you run the model, you type `yout` at the MATLAB command prompt, the following is a portion of what you would see:

```
yout =

    2
    4
    6
    8
   10
   12
```

Because `yout` represents a single-channel, frame-based signal, the output is a column vector. Once you export your signal values into the MATLAB workspace, they are no longer grouped into frames.

Tunable Parameters

There are some parameters that you can change, or tune, during simulation. To change a tunable parameter during simulation, double-click the block to open its dialog box, change any tunable parameters to the desired settings, and then click **OK**. The simulation now uses the new parameter settings.

Note Opening a dialog box for a source block causes Simulink to pause. While Simulink is paused, you can edit the parameter values. You must close the dialog box to have the changes take effect and allow Simulink to continue.

As a rule, parameters that specify numeric values are tunable. However, some parameters are not tunable, such as parameters that change the operational mode of a block. For example, you cannot directly or indirectly change the following while a simulation is running:

• Number of block ports

- Block sample rate

- Signal data type

For more information on tunable parameters, see the "Tunable Parameters" section of the Simulink documentation.

Signal Processing Blockset Product Features

Frame-Based Operations

Most real-time signal processing systems optimize throughput rates by processing data in "batch" or "frame-based" mode. By propagating multisample frames instead of the individual signal samples, the signal processing system can take advantage of the speed of signal processing algorithm execution, while simultaneously reducing the demands placed on the data acquisition (DAQ) hardware.

For an example of frame-based operations, open the LPC Analysis and Synthesis of Speech demo by typing `dsplpc` at the MATLAB command prompt. To run this demo, from the **Simulation** menu, select **Start**. A frame-based signal is used for computation throughout the model.

For more information about frame-based signals, see "Frame-Based Signals" on page 2-12.

Multirate Processing

Signal Processing Blockset blocks support multirate processing. This means that one port can have a different sample time than another port on the same block. Multirate processing is achieved by port-based sample time support across the blocks. You can find multirate blocks in the Multirate Filters sublibrary, the Signal Operations library, and the Buffers sublibrary.

For more information, see "Inspecting Sample Rates and Frame Rates" in the *Signal Processing Blockset User's Guide* and "Scheduling Considerations" in the Real-Time Workshop documentation.

Fixed-Point Support

Many Signal Processing Blockset blocks have fixed-point support. This allows you to design discrete-time dynamic signal processing systems that use fixed-point arithmetic. Fixed-point support in the blockset includes:

- Signed two's complement fixed-point data types

- Word sizes from 2 to 128 bits in simulation

- Word sizes from 2 to the size of a `long` in the Real-Time Workshop ANSI C code-generation target

- Overflow handling, scaling, and rounding methods

- ANSI C code generation for deployment on a fixed-point embedded processor, with the Real-Time Workshop product. The generated code uses all allowed simulation data types supported by the embedded target, and automatically includes all necessary shift and scaling operations.

Simulating your fixed-point development choices before implementing them in hardware saves time and money. Signal Processing Blockset software provides built-in fixed-point operations that save time in simulation and provide automatically optimized code.

For fixed-point blocks, the Signal Processing Blockset and Real-Time Workshop products produce optimized fixed-point code ready for execution on a fixed-point processor. All the choices you make during simulation with the blockset in terms of scaling, overflow handling, and rounding methods are automatically optimized in the generated code, without the need for time-consuming and costly hand-optimized code.

For more information on fixed-point support in the Signal Processing Blockset product, see "Working with Fixed-Point Data" in the *Signal Processing Blockset User's Guide*.

Real-Time Code Generation

For all Signal Processing Blockset blocks, the Signal Processing Blockset and Real-Time Workshop products produce optimized, compact, ANSI C code.

Chapter 5, "Code Generation" explains this process in more details.

Adaptive and Multirate Filtering

The Adaptive Filters and Multirate Filters sublibraries provide key tools for the construction of advanced signal processing systems. You can use adaptive filter block parameters to tailor signal processing algorithms to application-specific environments.

For an example of adaptive filtering, open the LMS Adaptive Equalization demo by typing `lmsadeq` at the MATLAB command prompt. Equalization is important in the field of communications. It involves estimating and eliminating dispersion present in communication channels. In this demo, the LMS Filter block models the dispersion of the system. The plot of the squared error demonstrates the effectiveness of this adaptive filter.

For more information on adaptive filters, see "Adaptive Filters" on page 4-9. For more information on multirate filters, see "Multirate Filters" in the *Signal Processing Blockset User's Guide*.

Quantization

The process of quantization allows you to represent your input signal with a finite number of values. This process helps you limit the bandwidth of your transmitted signal. The Signal Processing Blockset product has a number of blocks that can help you to design and implement scalar and vector quantizers. In the main Signal Processing Blockset library, open the Quantizers library to view the available blocks. See the block reference pages for any of these blocks to find out more information about their functionality.

For more information about quantization, see "Analysis and Synthesis of Speech" in the *Signal Processing Blockset User's Guide*.

Statistical Operations

Use the blocks in the Statistics library for basic statistical analysis. These blocks calculate measures of central tendency and spread such as mean, standard deviation, and so on. They can also calculate the frequency distribution of input values.

See "Statistics" in the *Signal Processing Blockset User's Guide* for more information.

Linear Algebra

The Matrices and Linear Algebra sublibrary provides Cholesky, LU, LDL, and QR matrix factorization methods and equation solvers based on these methods. It also provides blocks for common matrix operations.

See "Linear Algebra" in the *Signal Processing Blockset User's Guide* for more information.

Parametric Estimation

The Parametric Estimation sublibrary provides four blocks for modeling a signal as the output of an AR system:

- Burg AR Estimator

- Covariance AR Estimator

- Modified Covariance AR Estimator

- Yule-Walker AR Estimator

These blocks allow you to compute the AR system parameters based on forward error minimization, backward error minimization, or both.

In the Comparison of Spectral Analysis Techniques demo, `dspsacomp`, an IIR all-pole filter filters a Gaussian noise sample. Three different blocks, each with its own method, estimate the spectrum of the IIR filter.

Matrix Support

The Signal Processing Blockset product takes full advantage of the matrix format of the Simulink environment. Some typical uses of matrices in signal processing simulations are:

• General two-dimensional array

  A matrix can be used in its traditional mathematical capacity, as a simple structured array of numbers. The Matrices and Linear Algebra sublibrary contains most of the blocks used for general matrix operations.

• Factored submatrices

  A number of the matrix factorization blocks in the Matrix Factorizations sublibrary store the submatrix factors (such as lower and upper submatrices) in a single compound matrix. See the LDL Factorization and LU Factorization blocks for examples.

• Multichannel frame-based signal

  The standard format for multichannel frame-based signals is a matrix, where each column represents a different channel. For example, a matrix with three columns contains three channels of data. The number of rows in the matrix is the number of samples in each frame.

The following sections of the *Signal Processing Blockset User's Guide* provide more information about working with matrices:

• "Creating Sample-Based Signals"

• "Creating Frame-Based Signals"

• "Creating Multichannel Sample-Based Signals"

• "Creating Multichannel Frame-Based Signals"

• "Deconstructing Multichannel Sample-Based Signals"

• "Deconstructing Multichannel Frame-Based Signals"

Data Type Support

All Signal Processing Blockset blocks support single- and double-precision floating-point data types during both simulation and Real-Time Workshop C code generation. Many blocks also support fixed-point and Boolean data types.

To see which data types a particular block supports, refer to the "Supported Data Types" section of the block reference page.

For information about data type support and code generation coverage for all Signal Processing Blockset blocks, use the Signal Processing Blockset Data Type Support Table. To access the table, select **Help > Block Support Table > Signal Processing Blockset** or **Help > Block Support Table > All Tables** from the Simulink model help menu.
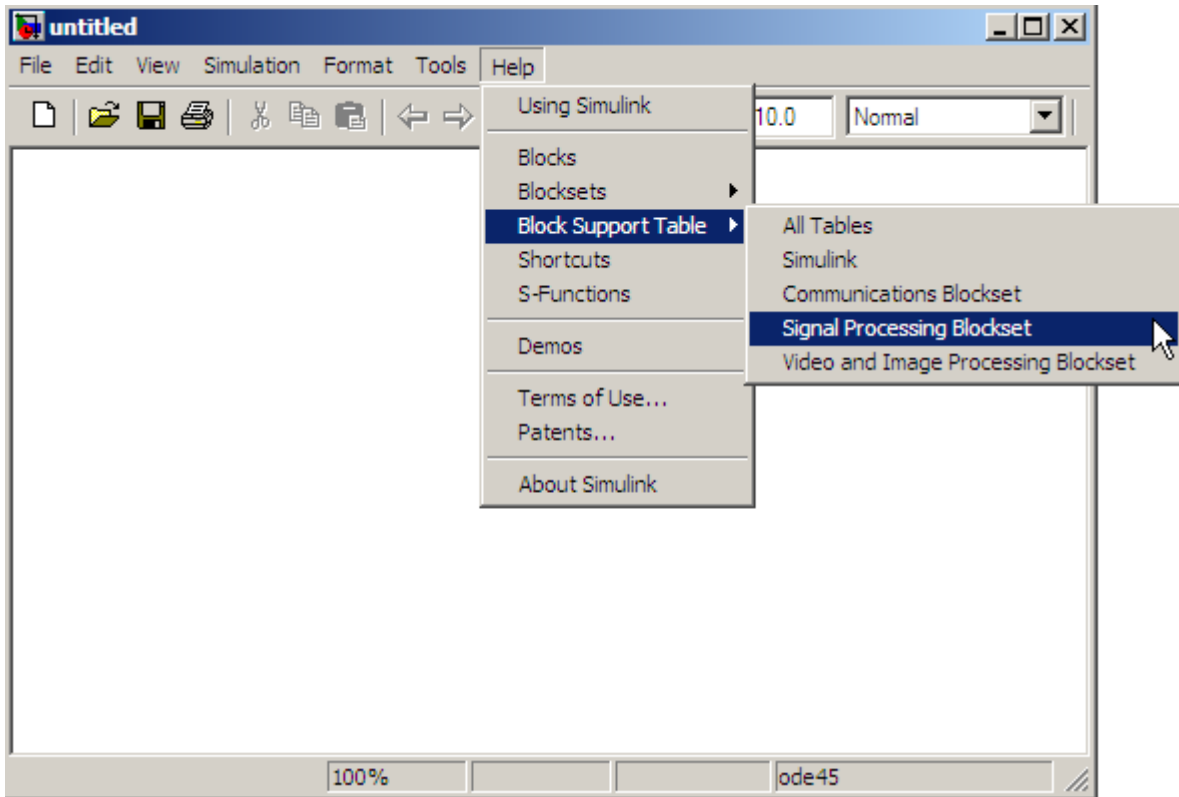
You can also access the Signal Processing Blockset Data Type Support Table by typing showsignalblockdatatypetable at the MATLAB command line.

It is often necessary to convert between different data types when working with Simulink models. The following table lists all data types supported by

Signal Processing Blockset blocks and which function or block to use when converting between data types.

Supported Data Types

Data Types Supported by Signal Processing Blockset Blocks	Functions and Blocks for Converting Data Types	Comments
Double-precision floating point	• `double` • Data Type Conversion block	Simulink built-in data type supported by all Signal Processing Blockset blocks
Single-precision floating point	• `single` • Data Type Conversion block	Simulink built-in data type supported by all Signal Processing Blockset blocks
Boolean	• `boolean` • Data Type Conversion block	Simulink built-in data type.
Integer (8-,16-, or 32-bits)	• `int8, int16, int32` • Data Type Conversion block	Simulink built-in data type.
Unsigned integer (8-,16-, or 32-bits)	• `uint8, uint16, uint32` • Data Type Conversion block	Simulink built-in data type.
Fixed-point data types	• Data Type Conversion block • Simulink Fixed Point product `num2fixpt` function • Functions and GUIs for designing quantized filters with the Filter Design Toolbox™ (compatible with Filter Realization Wizard block)	To learn more about fixed-point data types in the Signal Processing Blockset product, see "Working with Fixed-Point Data" in the *Signal Processing Blockset User's Guide*.

Configuring the Simulink Environment for Signal Processing Models

In this section...
"Using dspstartup.m" on page 2-23
"Settings in dspstartup.m" on page 2-24

Using dspstartup.m

The Signal Processing Blockset product provides an M-file, `dspstartup`, that lets you automatically configure the Simulink environment for signal processing simulation. We recommend these configuration parameters for models that contain Signal Processing Blockset blocks. Because these blocks calculate values directly rather than solving differential equations, you must configure the Simulink solver to behave like a scheduler. The solver, while in scheduler mode, uses a block sample time to determine when the code behind each block executes. For example, if the sample time of a Sine Wave block is `0.05`, the solver executes the code behind this block and every other block with this sample time once every 0.05 seconds.

Note When working with models that contains Signal Processing Blockset blocks, use source blocks that enable you to specify their sample time. When your source block does not have a **Sample time** parameter, you must add a Zero-Order Hold block in your model and use it to specify the sample time. For more information, see "Continuous-Time Source Blocks" in the *Signal Processing Blockset User's Guide.* The exception to this rule is the Constant block, which can have a constant sample time. When it does, Simulink executes this block and records the constant value once, which allows for faster simulations and more compact generated code.

To use the `dspstartup` M-file to configure Simulink for signal processing simulations, you can

• Type `dspstartup` at the MATLAB command line. All new models have settings customized for signal processing applications. Existing models are not affected.

- Place a call to dspstartup within the startup.m file. This is an efficient way to use dspstartup if you want these settings to be in effect every time you start Simulink. For more information about performing automated tasks at startup, see the documentation for the startup command in the MATLAB Function Reference.

The dspstartup M-file executes the following commands:

```
set_param(0, ...
            'SingleTaskRateTransMsg','error', ...
            'multiTaskRateTransMsg', 'error', ...
            'Solver',                'fixedstepdiscrete', ...
            'SolverMode',            'SingleTasking', ...
            'StartTime',             '0.0', ...
            'StopTime',              'inf', ...
            'FixedStep',             'auto', ...
            'SaveTime',              'off', ...
            'SaveOutput',            'off', ...
            'AlgebraicLoopMsg',      'error', ...
            'SignalLogging',         'off');
```

You can edit the dspstartup M-file to change any of these settings or to add your own custom settings. For complete information about these settings, see "Model and Block Parameters" in the Simulink documentation.

Settings in dspstartup.m

A number of the settings in the dspstartup M-file are chosen to improve the performance of the simulation:

- 'Solver' is set to 'fixedstepdiscrete'.

  This selects the fixed-step solver option instead of the Simulink default variable-step solver. This mode enables code generation from the model using the Real-Time Workshop product.

- 'Stop time' is set to 'Inf'.

  The simulation runs until you manually stop it by selecting **Stop** from the **Simulation** menu.

- 'SaveTime' is set to 'off'.

Simulink does not save the `tout` time-step vector to the workspace. The time-step record is not usually needed for analyzing discrete-time simulations, and disabling it saves a considerable amount of memory, especially when the simulation runs for an extended time.

- `'SaveOutput'` is set to `'off'`.

  Simulink Outport blocks in the top level of a model do not generate an output (`yout`) in the workspace.

3

Signal Processing Models

Creating a Block Diagram

You can build signal processing models using blocks from many different Simulink and Signal Processing Blockset libraries. In this section, you move through the tasks needed to create a signal processing model that displays a sine wave over time:

- Opening a new model

- Dragging blocks into the model

- Connecting the blocks

In subsequent procedures, you set the block parameters and run the model. Later in the book, you expand upon this model to create a system capable of adaptive noise cancellation. You also use the Real-Time Workshop product to generate code from this model:

1 Begin building your model. Open the main Signal Processing Blockset library by typing `dsplib` at the MATLAB command prompt.
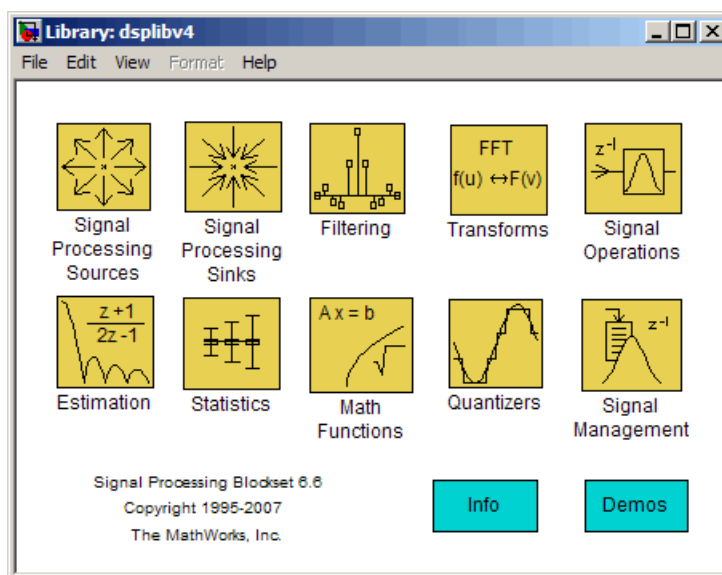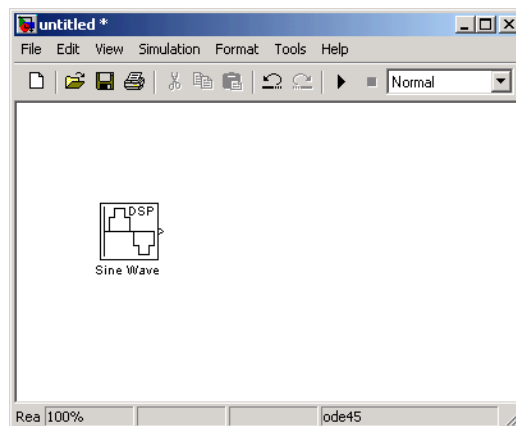
2 Open a new model by selecting **File** > **New** > **Model** in the Signal Processing Blockset library window.
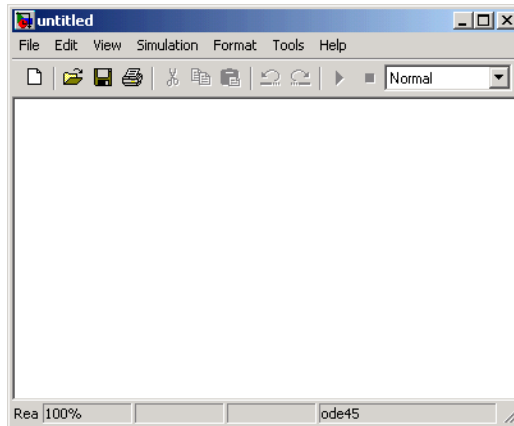

3 Display the Signal Processing Blockset Sources sublibrary by double-clicking the Signal Processing Sources icon in the main library window.
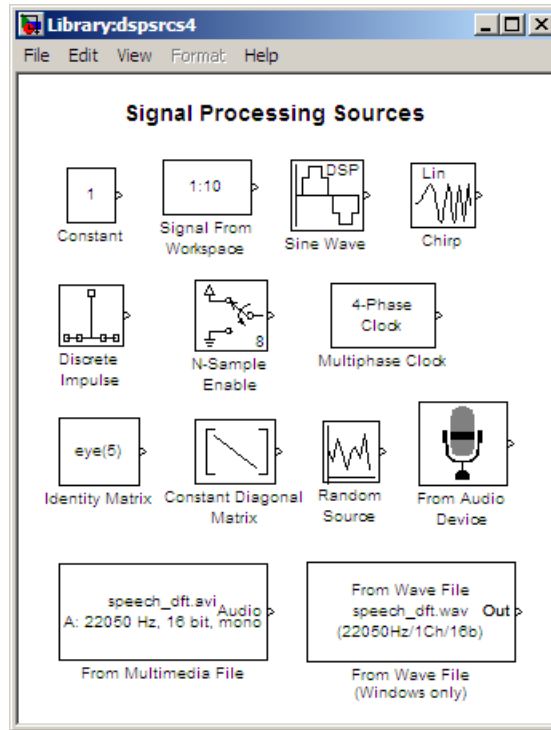
4 Click-and-drag a Sine Wave block into your new model. The Sine Wave block generates a sinusoidal signal.

5 Double-click the Signal Processing Sinks sublibrary, and click-and-drag the
Time Scope block into your model.

6 Connect the two blocks by selecting the Sine Wave block, holding down the
Ctrl key, and then selecting the Time Scope block.
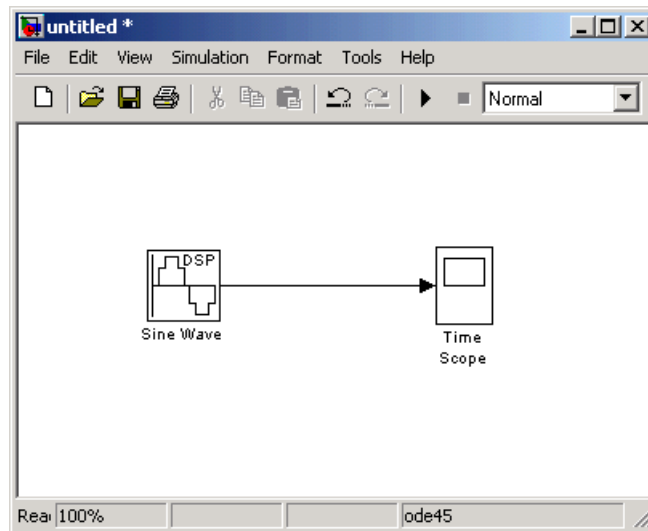
Now that you have created a model, you are ready to set your model
parameters.

Setting the Model Parameters

Once you have built your signal processing model, you can set your model parameters. Nearly all blocks have an associated block parameters dialog box. Double-click the block to display this dialog box. Enter values into this dialog box to ensure that your model accurately simulates the desired behavior of your system.

Note The software provides premade models as starting points to each procedure in this manual. To prevent yourself from overwriting these models, from the **File** menu, select **Save as**. Then, save your modified model in a different folder.

---

1 If the model you created in "Creating a Block Diagram" on page 3-2 is not open on your desktop, you can open an equivalent model by typing

    doc_gstut1

at the MATLAB command prompt.

2 Open the Configuration Parameters dialog box by selecting **Simulation > Configuration Parameters** from the model menu.

3 Set the following **Solver options**:

- **Type**: Fixed-step
- **Solver**: Discrete (no continuous states)

These are the recommended **Solver options** for Signal Processing Blockset models. For more information on recommended settings, see "Configuring the Simulink Environment for Signal Processing Models" on page 2-23.

The Configuration Parameters dialog box should now appear as follows:
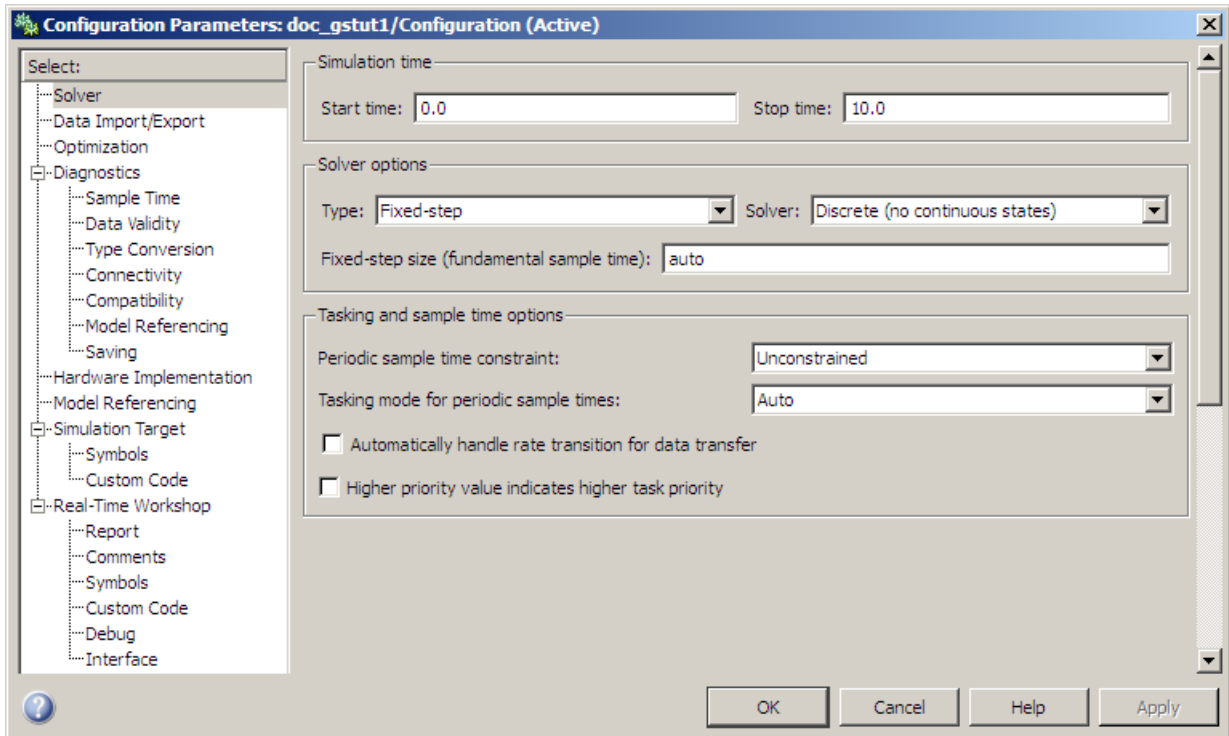
4 Click **OK** to apply the change and close the dialog box.

5 Open the Sine Wave dialog box by double-clicking the Sine Wave block.

6 Set the block parameters as follows:

- **Frequency (Hz)** = 0.5
- **Sample time** = 0.05
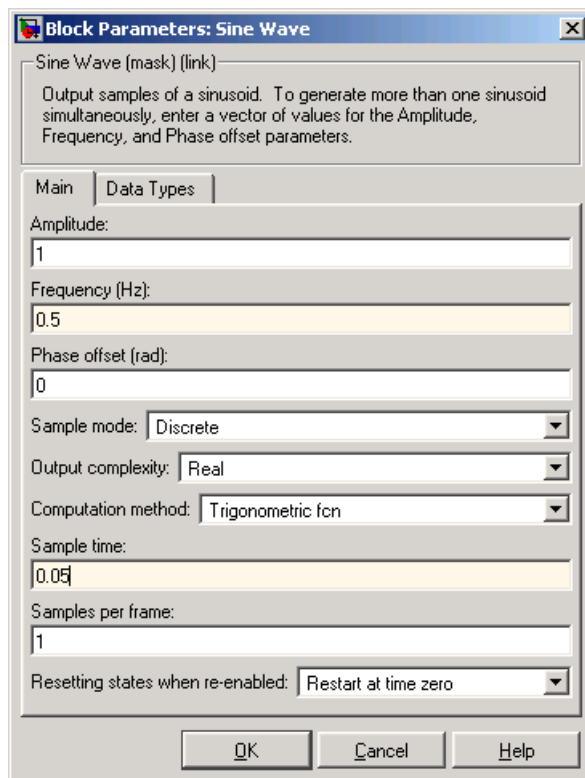
> **Note** On Signal Processing Blockset blocks, the **Sample time** parameter represents the sample period of the signal. The sample period is the amount of time between each sample of the signal.

7 Click **OK** to apply the settings and close the dialog box. Now that you have set your model parameters, you are ready to run your model and view its behavior.
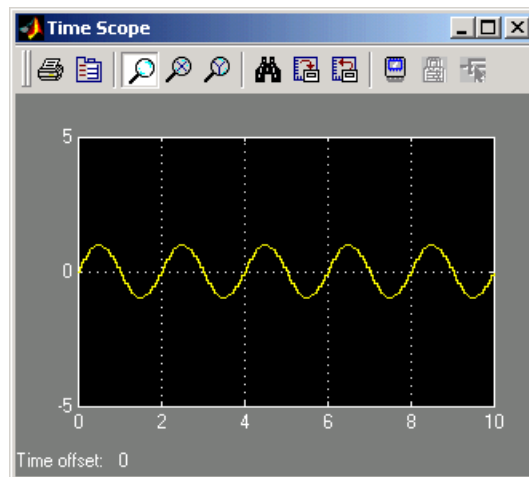
Running the Model

After you set the desired model parameters, you can run your model and view its behavior. The Signal Processing Blockset product has many scope blocks that you can use to display your model output. In this example, you use a Time Scope block to view your sinusoidal signal:

1 If the model you created in "Setting the Model Parameters" on page 3-6 is not open on your desktop, you can open an equivalent model by typing
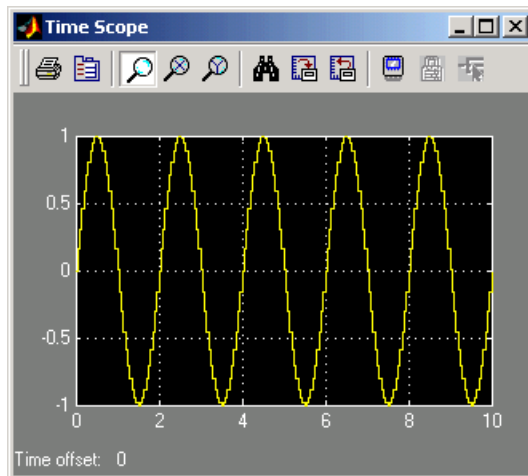
```
doc_gstut2
```

at the MATLAB command prompt.

2 Run the model by selecting **Start** from the **Simulation** menu.

3 Display the sinusoidal signal in the Time Scope window by double-clicking the Time Scope block.

4 Autoscale the output to fit in the scope window by clicking .

You can achieve a more finely sampled output by decreasing the **Sample time** parameter. For example, change the **Sample time** parameter in the Sine Wave block to `0.005`, run the model, and autoscale the output. The Time Scope window should now look similar to the following figure.
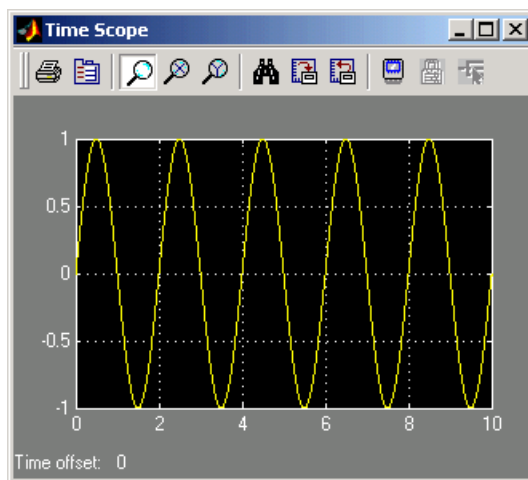
5 Experiment with your model. Change the **Frequency (Hz)** and **Sample time** parameters of the Sine Wave block. Then, run your model to see the effect. Now you are ready to add noise to your sinusoidal signal and view its effect.

Modifying Your Model

A system's input signal can contain noise that was introduced as the signal traveled over a wire or through the air. You can incorporate noise into the model of your system to simulate this real-world noise. Then, you can experiment with ways to eliminate its effect at both low and high frequencies. In this topic, you model a real-world signal by adding noise to your input signal. In the next chapter, you use a filter to convert this noise to low frequency noise and another filter to eliminate this noise from your signal:

1 If the model you worked with in "Running the Model" on page 3-9 is not open on your desktop, you can open an equivalent model by typing

    doc_gstut2

at the MATLAB command prompt.

2 Add a Random Source block to your model from the Signal Processing Sources library to represent the noise in your system. Set the block parameters before you connect the blocks. Double-click the Random Source block and set the block parameters as follows:

- **Source type** = Gaussian
- **Method** = Ziggurat
- **Mean** = 0
- **Variance** = 1
- **Repeatability** = Specify seed
- **Initial seed** = [23341]
- **Sample time** = 0.05

Based on these parameters, the Random Source block produces Gaussian random values using the Ziggurat method. The **Repeatability** and **Initial seed** parameters ensure that the block outputs the same signal each time you run the model. The following figure shows the completed Random Source dialog box.
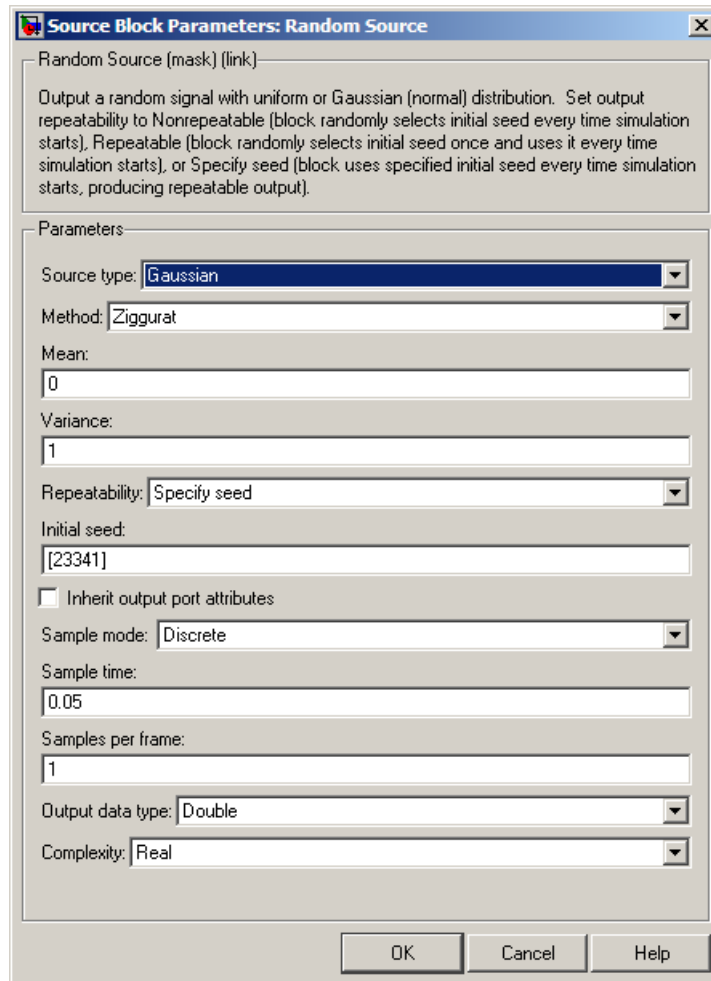
Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters During Simulation" in the online Simulink documentation for details.

3 Add a Sum block to your model from the Simulink Math Operations library to add random noise to your input signal.

4 Set the Sum block parameters. Open the Sum dialog box by double-clicking the Sum block. Change the **List of signs** parameter to ++| and click **OK**.

5 Set the Time Scope parameters. Open the Time Scope window by double-clicking the Time Scope block. Open the Time Scope parameters dialog box by clicking the **Parameters** icon in the Time Scope window.
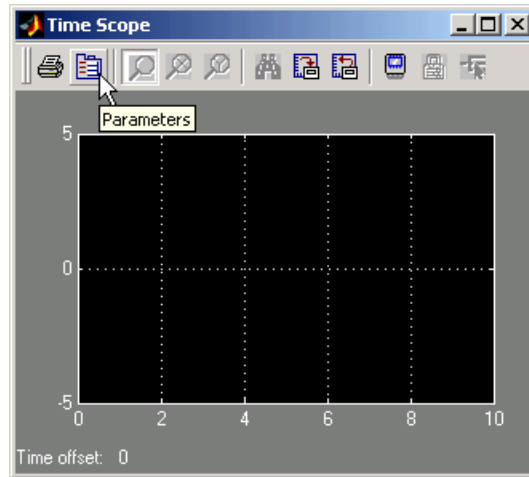
In the Time Scope parameters dialog box, set the **Number of axes** parameter to 2 and click **OK**. Now, the Time Scope window has two plotting windows and the Time Scope block has two input ports.
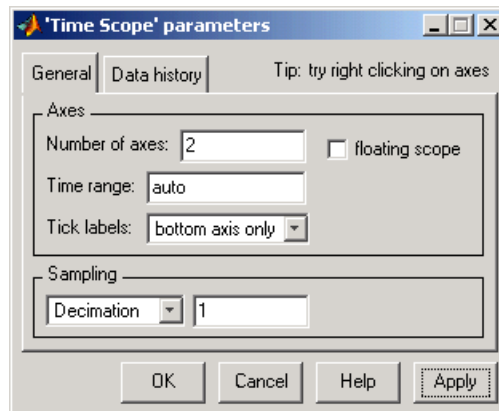
6 Connect the output of the Sine Wave block and the output of the Random Source block to the input of the Sum block. Then, connect the output of the Sum block to the second input of the Time Scope block. When you are finished, your model should look similar to the figure below.
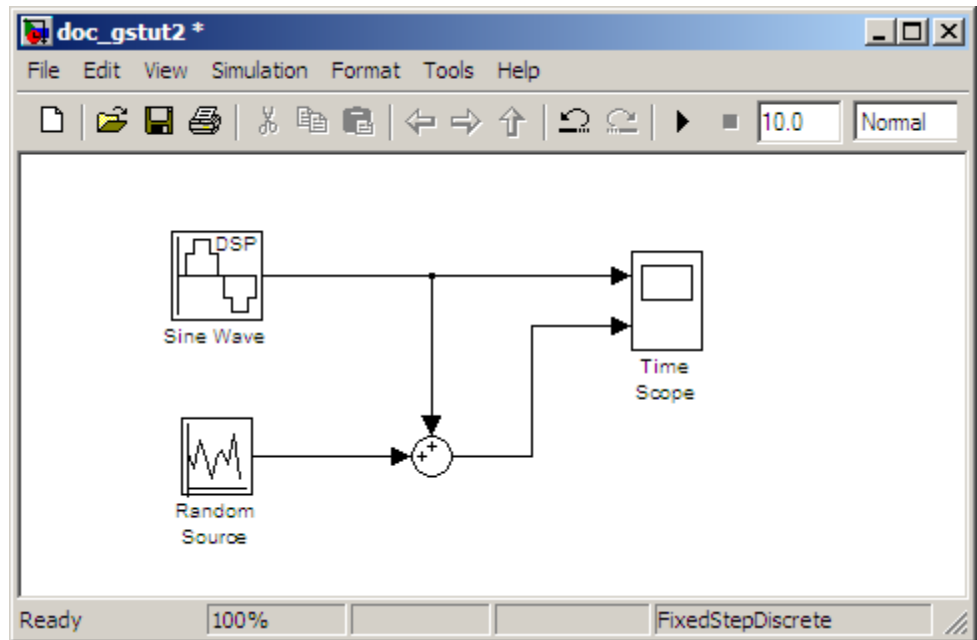
7 Verify the parameters of your Sine Wave block. Open the Sine Wave dialog box by double-clicking the Sine Wave block. Verify that the **Frequency (Hz)** parameter is set to 0.5 and the **Sample time** parameter is set to 0.05. Note that the value of the **Sample time** parameter of the Sine Wave block is the same as the value of the **Sample time** parameter of the Random Source block.

8 Run your model and view the results in the Time Scope window. The block displays the original sinusoidal signal in the top axes and the signal with the noise in the bottom axes.
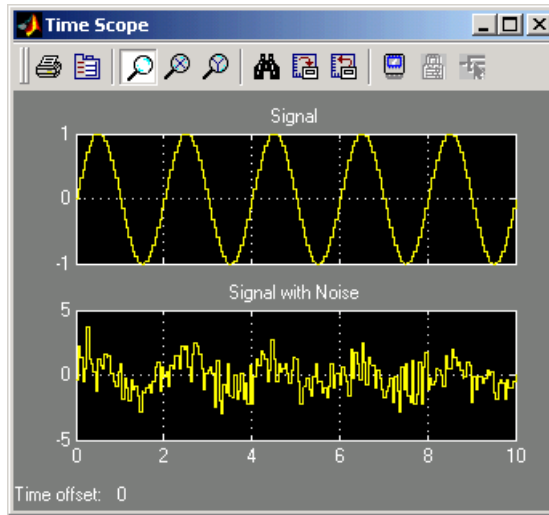
> **Note** You can change the signal labels in the Time Scope window by
> right-clicking the axes and selecting **Axes properties**. In the **Title** text
> box, enter your signal label.

You have now created and run a signal processing model that displays a
sinusoidal signal over time. During this process, you created a digital sine
wave and viewed it in the Time Scope window. You also added noise to your
sinusoidal signal and viewed its effect. In Chapter 4, "Filters", you increase
the complexity of your signal processing model by adding filters to eliminate
the presence of this noise.

4

Filters

- "Digital Filters" on page 4-2
- "Adaptive Filters" on page 4-9

Digital Filters

Designing a Digital Filter

You can design lowpass, highpass, bandpass, and bandstop filters using either the Digital Filter Design block or the Filter Realization Wizard. These blocks are capable of calculating filter coefficients for various filter structures. In Chapter 3, "Signal Processing Models", you added white noise to a sine wave and viewed the resulting signal on a scope. In this section, you use the Digital Filter Design block to convert this white noise to low frequency noise so you can simulate its effect on your system.

As a practical application, suppose a pilot is speaking into a microphone within the cockpit of an airplane. The noise of the wind passing over the fuselage is also reaching the microphone. A sensor is measuring the noise of the wind on the outside of the plane. You want to estimate the wind noise inside the cockpit and subtract it from the input to the microphone so that only the pilot's voice is transmitted. In this chapter, you first learn how to model the low frequency noise that is reaching the microphone. Later, you learn how to remove this noise so that only the pilot's voice is heard.

In this topic, you use a Digital Filter Design block to create low frequency noise, which models the wind noise inside the cockpit:

1 If the model you created in "Modifying Your Model" on page 3-12 is not open on your desktop, you can open an equivalent model by typing

```
doc_gstut3
```

at the MATLAB command prompt. This model contains a Time Scope block that displays the original sine wave and the sine wave with white noise added.
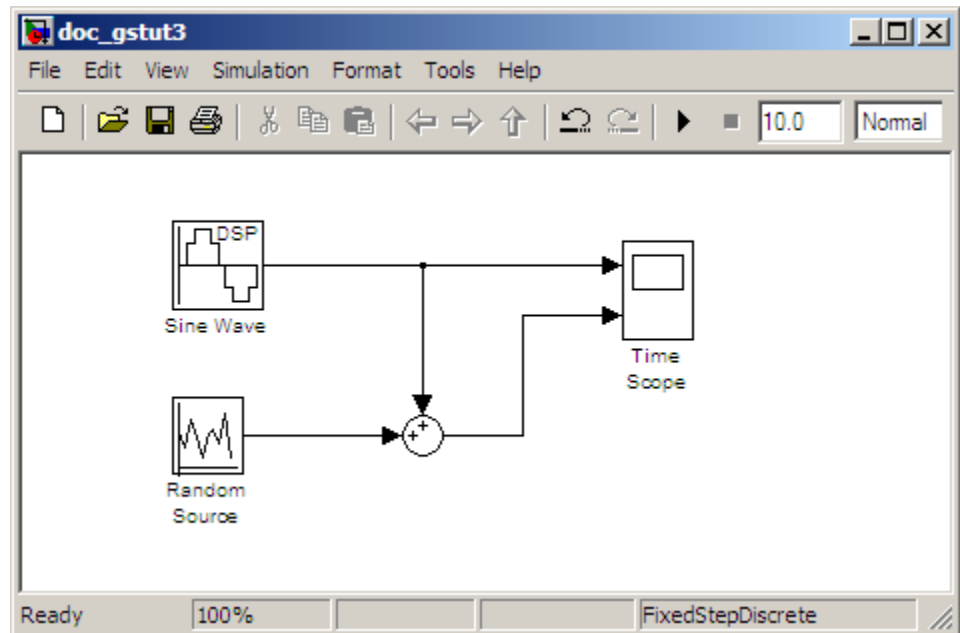
2 Open the Signal Processing Blockset library by typing `dsplib` at the MATLAB command prompt.

3 Convert white noise to low frequency noise by introducing a Digital Filter Design block into your model. In the airplane scenario, the air passing over the fuselage creates white noise that is measured by a sensor. The Random Source block models this noise. The fuselage of the airplane converts this white noise to low frequency noise, a type of colored noise, which is heard inside the cockpit. This noise contains only certain frequencies and is more difficult to eliminate. In this example, you model the low frequency noise using a Digital Filter Design block. This block uses the functionality of the Filter Design and Analysis Tool (FDATool) to design a filter.

Double-click the Filtering library, and then double-click the Filter Designs sublibrary. Click-and-drag the Digital Filter Design block into your model.
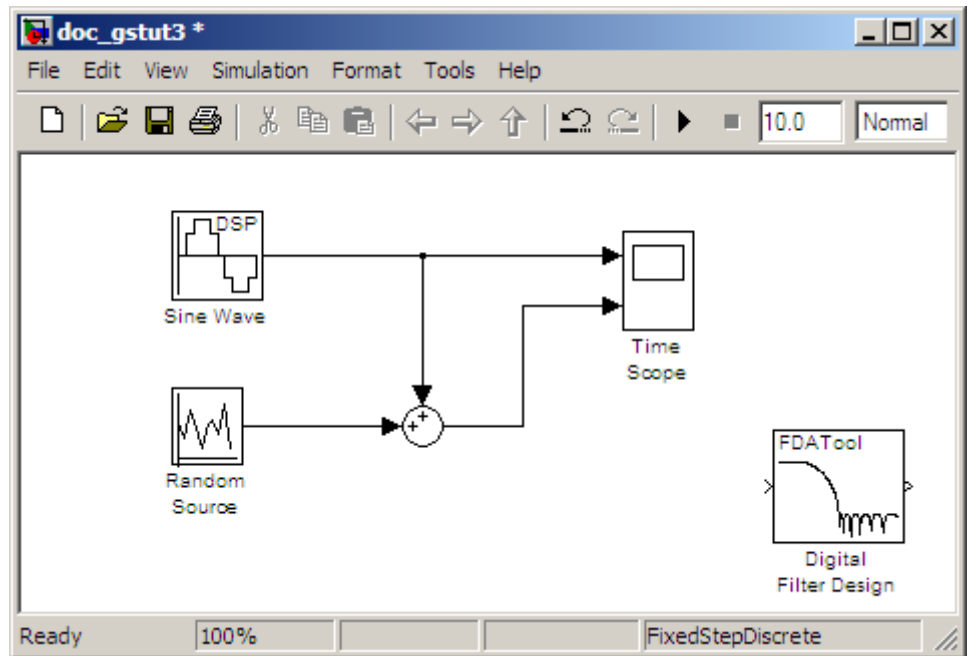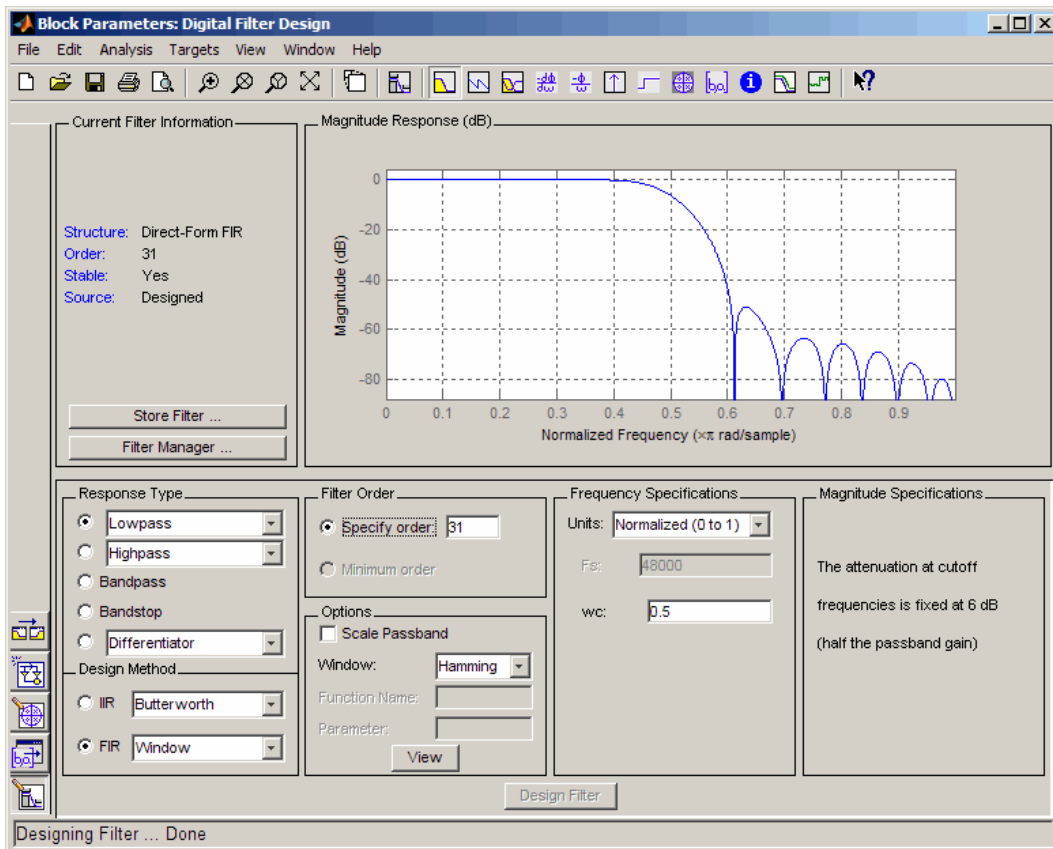
4 Set the Digital Filter Design block parameters to design a lowpass filter
and create low frequency noise. Open the block parameters dialog box by
double-clicking the block. Set the parameters as follows:

- **Response Type** = Lowpass

- **Design Method** = **FIR** and, from the list, choose Window

- **Filter Order** = **Specify order** and enter 31

- **Scale Passband** — Cleared

- **Window** = Hamming

- **Units** = Normalized (0 to 1)

- **wc** = 0.5

Based on these parameters, the Digital Filter Design block designs a
lowpass FIR filter with 32 coefficients and a cutoff frequency of 0.5. The
block multiplies the time-domain response of your filter by a 32 sample
Hamming window.

5 Click **Design Filter** at the bottom center of the dialog box to view the magnitude response of your filter in the **Magnitude Response** pane. The Digital Filter Design dialog box should now look similar to the following figure.

You have now designed a digital lowpass filter using the Digital Filter Design block.

You can experiment with the Digital Filter Design block in order to design a filter of your own. For more information on the block functionality, see the Digital Filter Design block reference page. For more information on the Filter

Design and Analysis Tool, see "FDATool: A Filter Design and Analysis GUI" in the Signal Processing Toolbox documentation.
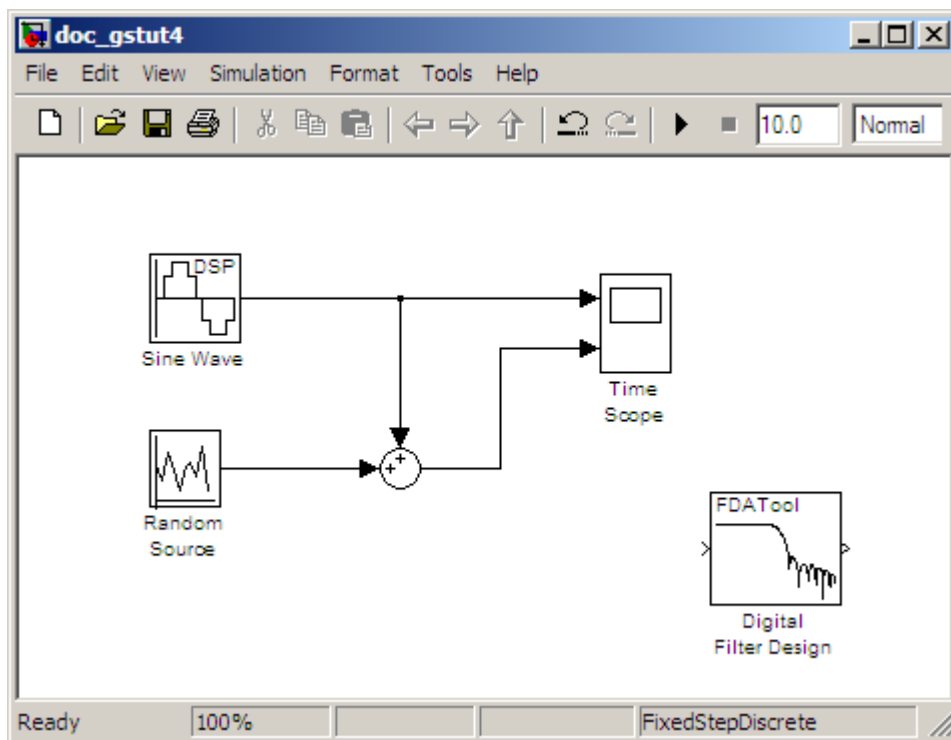
Adding a Digital Filter to Your Model

In this topic, you add the lowpass filter you designed in "Designing a Digital Filter" on page 4-2 to your block diagram. Use this filter, which converts white noise to colored noise, to simulate the low frequency wind noise inside the cockpit:
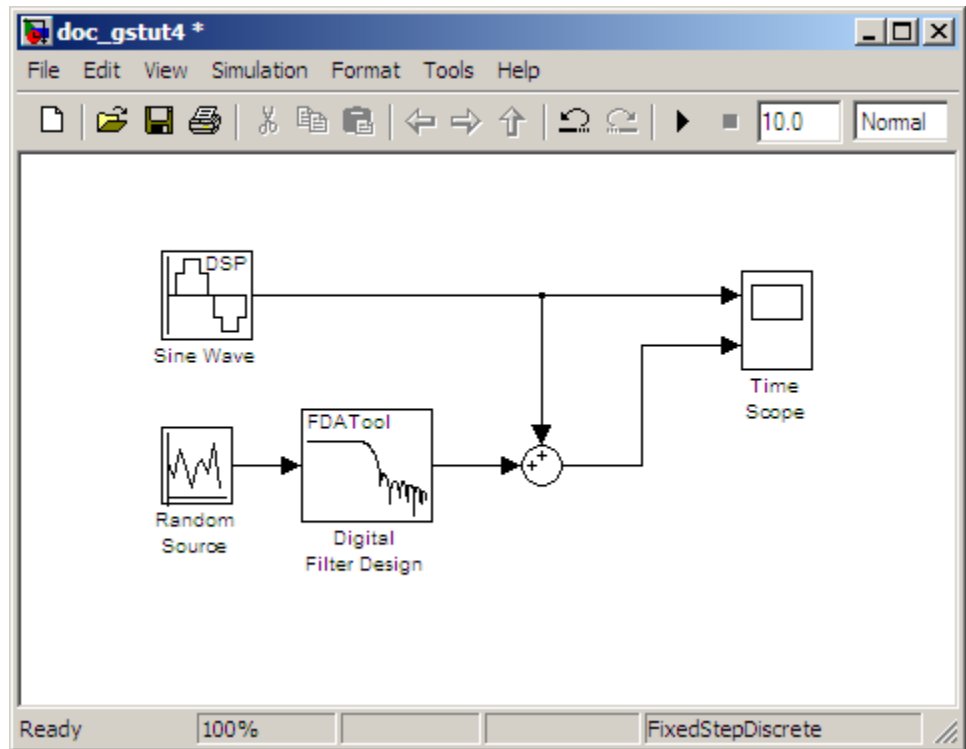
1 If the model you created in "Designing a Digital Filter" on page 4-2 is not open on your desktop, you can open an equivalent model by typing
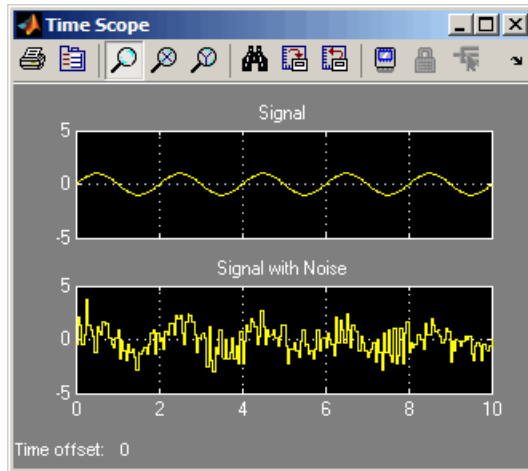
```
doc_gstut4
```

at the MATLAB command prompt.

2 Incorporate the Digital Filter Design block into your block diagram by placing it between the Random Source block and the Sum block.

3 Run your model and view the results in the Time Scope window. This window shows the original input signal and the signal with low frequency noise added to it.

You have now built a digital filter and used it to model the presence of colored noise in your signal. This is analogous to modeling the low frequency noise reaching the microphone in the cockpit of the aircraft. Now that you have added noise to your system, you can experiment with methods to eliminate it.

Adaptive Filters

In this section...
"Designing an Adaptive Filter" on page 4-9
"Adding the Adaptive Filter to Your Model" on page 4-14
"Viewing the Coefficients of Your Adaptive Filter" on page 4-18

## Designing an Adaptive Filter

Adaptive filters track the dynamic nature of a system and allow you to eliminate time-varying signals. The Signal Processing Blockset libraries contain blocks that implement least-mean-square (LMS), Block LMS, Fast Block LMS, and recursive least squares (RLS) adaptive filter algorithms. These filters minimize the difference between the output signal and the desired signal by altering their filter coefficients. Over time, the adaptive filter's output signal more closely approximates the signal you want to reproduce.

In this topic, you design an LMS adaptive filter to remove the low frequency noise in your signal:

1 If the model you created in "Adding a Digital Filter to Your Model" on page 4-6 is not open on your desktop, you can open an equivalent model by typing

  doc_gstut5

at the MATLAB command prompt.
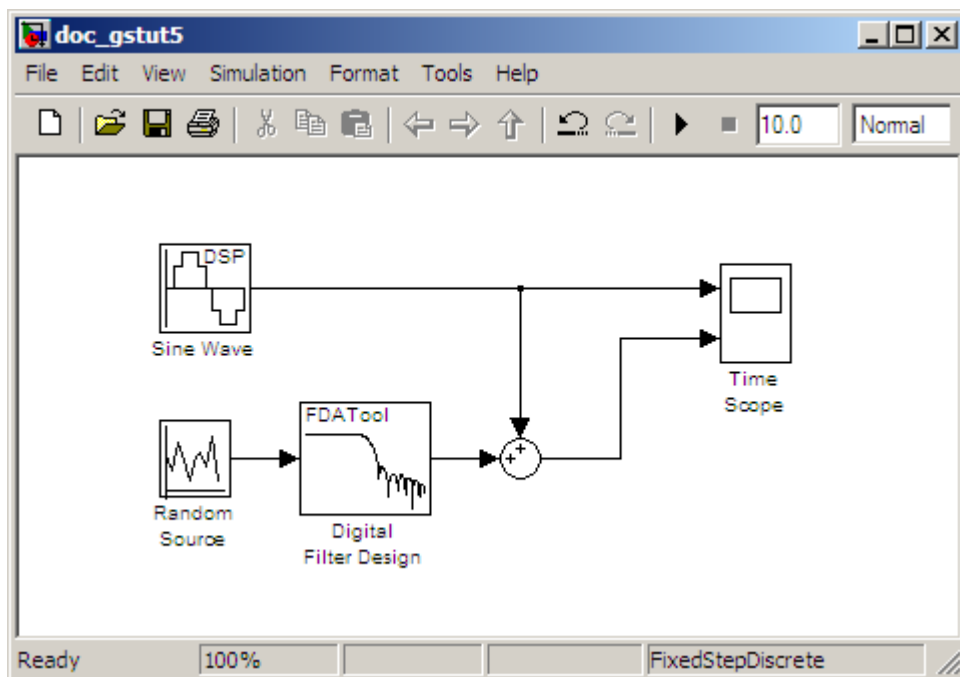
2 Open the Signal Processing Blockset library by typing `dsplib` at the MATLAB command prompt.

3 Remove the low frequency noise from your signal by adding an LMS Filter block to your system. In the airplane scenario, this is equivalent to subtracting the wind noise inside the cockpit from the input to the microphone. Double-click the Filtering sublibrary, and then double-click the Adaptive Filters library. Add the LMS Filter block into your model.
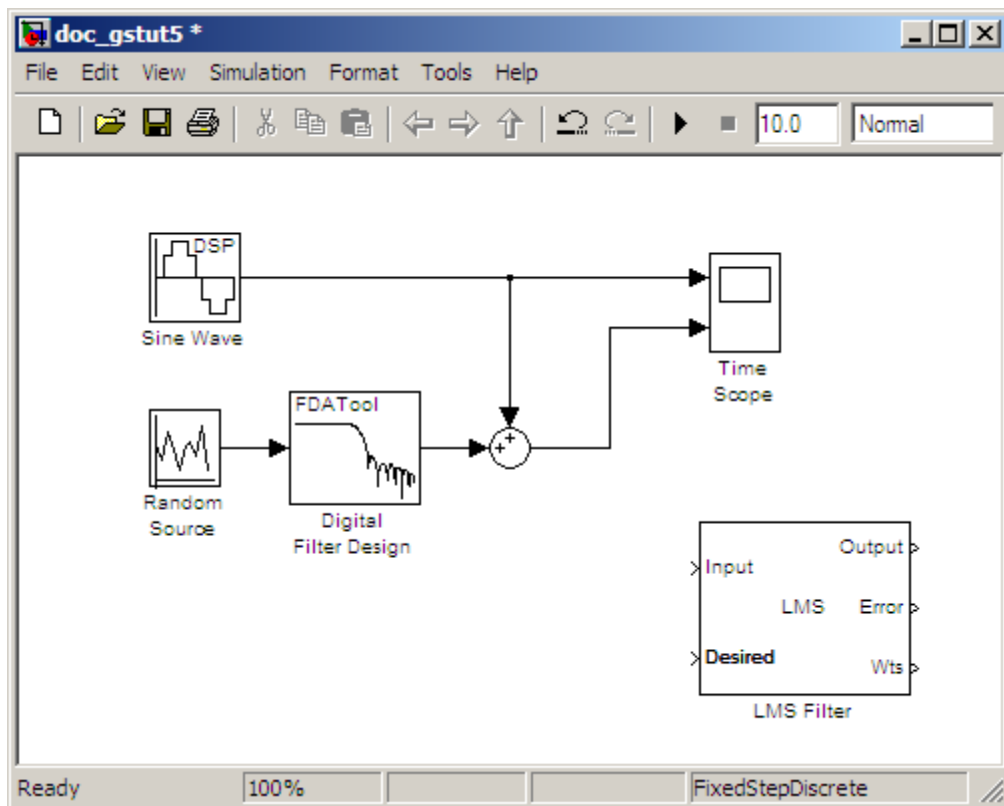
4 Set the LMS Filter block parameters to model the output of the Digital
   Filter Design block. Open its dialog box by double-clicking the block. Set
   the block parameters as follows:

   - **Algorithm** = `Normalized LMS`

   - **Filter length** = `32`

   - **Specify step size via** = `Dialog`

   - **Step size (mu)** = `0.1`

   - **Leakage factor (0 to 1)** = `1.0`

   - **Initial value of filter weights** = `0`

   - Clear the **Adapt port** check box.

- **Reset port** = None
- Select the **Output filter weights** check box.

  The LMS Filter dialog box should now look like the following figure:

Function Block Parameters: LMS Filter

LMS Filter

Adapts the filter weights based on the chosen algorithm for filtering of the input signal.

Select the Adapt port check box to create an Adapt port on the block. When the input to this port is nonzero, the block continuously updates the filter weights. When the input to this port is zero, the filter weights remain constant.

If the Reset port is enabled and a reset event occurs, the block resets the filter weights to their initial values.

Main | Fixed-point

Parameters

Algorithm: Normalized LMS

Filter length: 32

Specify step size via: Dialog

Step size (mu): 0.1

Leakage factor (0 to 1): 1.0

Initial value of filter weights: 0

☐ Adapt port

Reset port: None

☑ Output filter weights

OK | Cancel | Help | Apply

5 Click **Apply**.

Based on these parameters, the LMS Filter block computes the filter weights using the normalized LMS equations. The filter order you specified is the same as the filter order of the Digital Filter Design block. The **Step size (mu)** parameter defines the granularity of the filter update steps. Because you set the **Leakage factor (0 to 1)** parameter to `1.0`, the current filter coefficient values depend on the filter's initial conditions and all of the previous input values. The initial value of the filter weights (coefficients) is zero. Since you selected the **Output filter weights** check box, the Wts port appears on the block. The block outputs the filter weights from this port.

Now that you have set the block parameters of the LMS Filter block, you can incorporate this block into your block diagram.
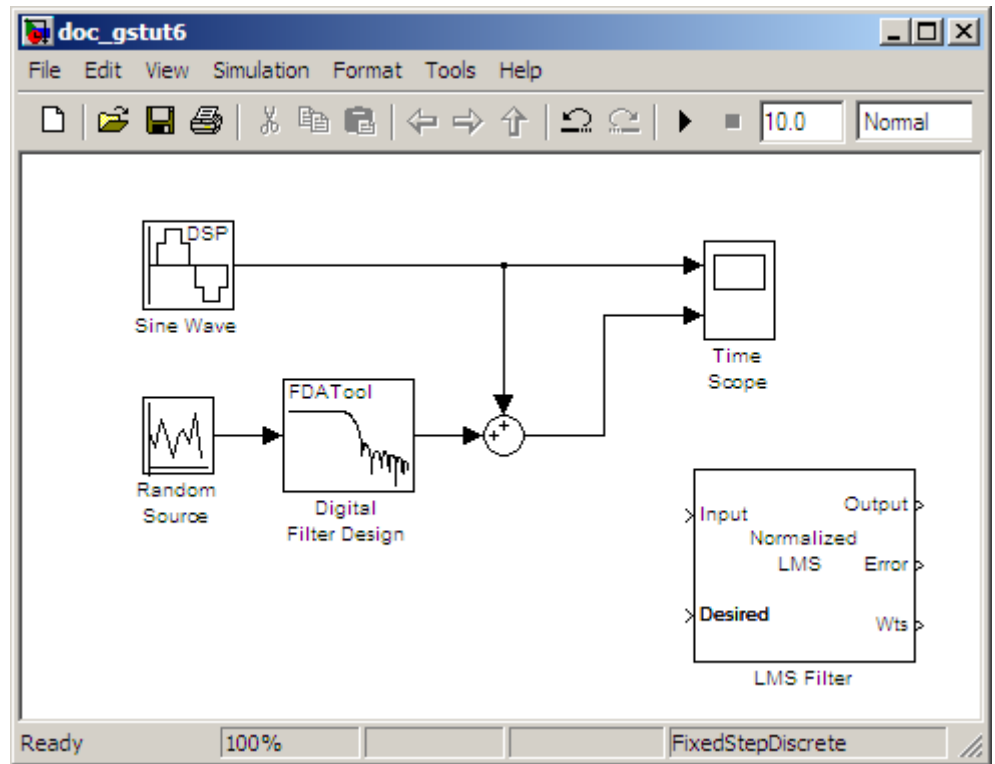
Adding the Adaptive Filter to Your Model

In this topic, you recover your original sinusoidal signal by incorporating the adaptive filter you designed in "Designing an Adaptive Filter" on page 4-9 into your system. In the aircraft scenario, the adaptive filter models the low frequency noise heard inside the cockpit. As a result, you can remove the noise so that the pilot's voice is the only input to the microphone:

1 If the model you created in "Designing an Adaptive Filter" on page 4-9 is not open on your desktop, you can open an equivalent model by typing

    doc_gstut6

at the MATLAB command prompt.

2 Add a Sum block to your model to subtract the output of the adaptive filter from the sinusoidal signal with low frequency noise. From the Simulink Math Operations library, drag a Sum block into your model. Open the Sum dialog box by double-clicking this block. Change the **List of signs** parameter to |+- and then click **OK**.

3 Incorporate the LMS Filter block into your system.

    a Connect the output of the Random Source block to the Input port of the LMS Filter block. In the aircraft scenario, the random noise is the white noise measured by the sensor on the outside of the airplane. The LMS Filter block models the effect of the airplane's fuselage on the noise.

    b Connect the output of the Digital Filter Design block to the Desired port on the LMS Filter block. This is the signal you want the LMS block to reproduce.

c Connect the output of the LMS Filter block to the negative port of the
Sum block you added in step 2.

d Connect the output of the first Sum block to the positive port of the
second Sum block. Your model should now look similar to the following
figure.
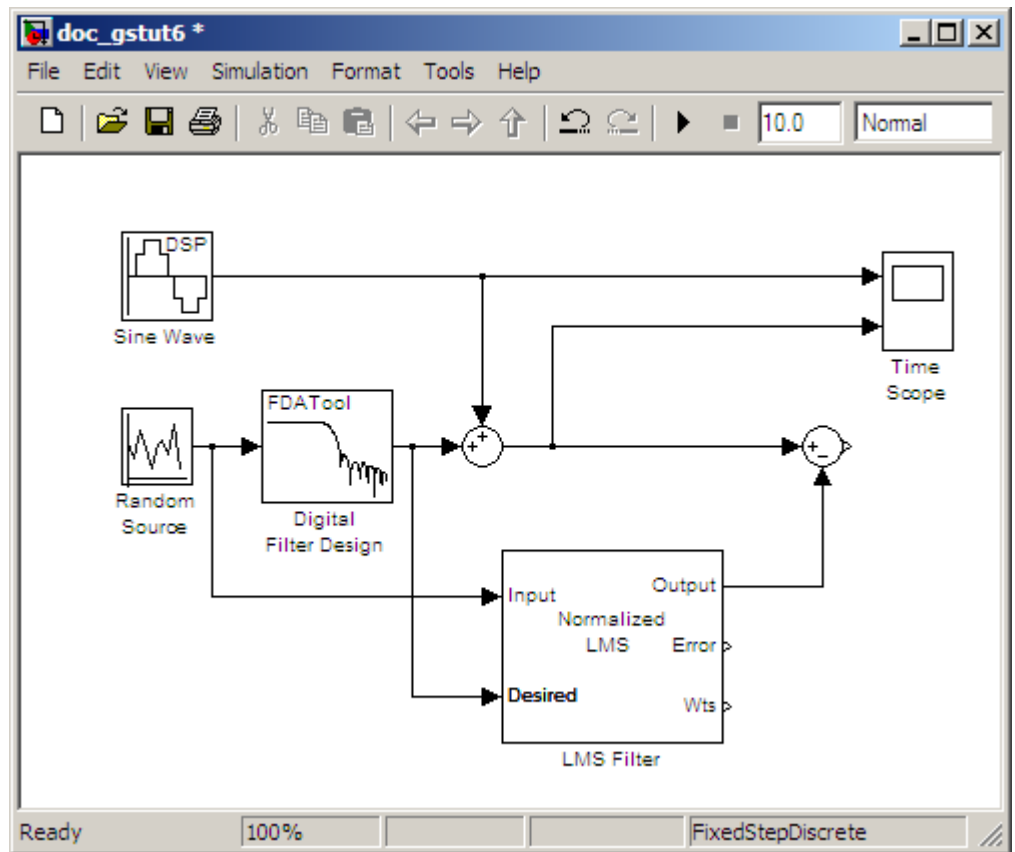
The positive input to the second Sum block is the sum of the input signal
and the low frequency noise, $s(n) + y$. The negative input to the second Sum
block is the LMS Filter block's best estimation of the low frequency noise,
y'. When you subtract the two signals, you are left with an approximation
of the input signal.

$$s(n)_{approx} = s(n) + y - y'$$

In this equation:

- $s(n)$ is the input signal

- $s(n)_{approx}$ is the approximation of the input signal

- y is the noise created by the Random Source block and the Digital Filter Design block

- y' is the LMS Filter block's approximation of the noise

Because the LMS Filter block can only approximate the noise, there is still a difference between the input signal and the approximation of the input signal. In subsequent steps, you set up the Time Scope block so you can compare the original sinusoidal signal with its approximation.

4 Add two additional inputs and axes to the Time Scope block. Open the Time Scope dialog box by double-clicking the Time Scope block. Click the **Parameters** button. For the **Number of axes** parameter, enter 4. Close the dialog box by clicking **OK**.

5 Label the new Time Scope axes. In the Time Scope window, right-click on the third axes and select **Axes properties**. The Time Scope properties: axis 3 dialog box opens. In the **Title** box, enter Approximation of Input Signal. Close the dialog box by clicking **OK**. Repeat this procedure for the fourth axes and label it Error.

6 Connect the output of the second Sum block to the third port of the Time Scope block.

7 Connect the output of the Error port on the LMS Filter block to the fourth port of the Time Scope block. Your model should now look similar to the following figure.
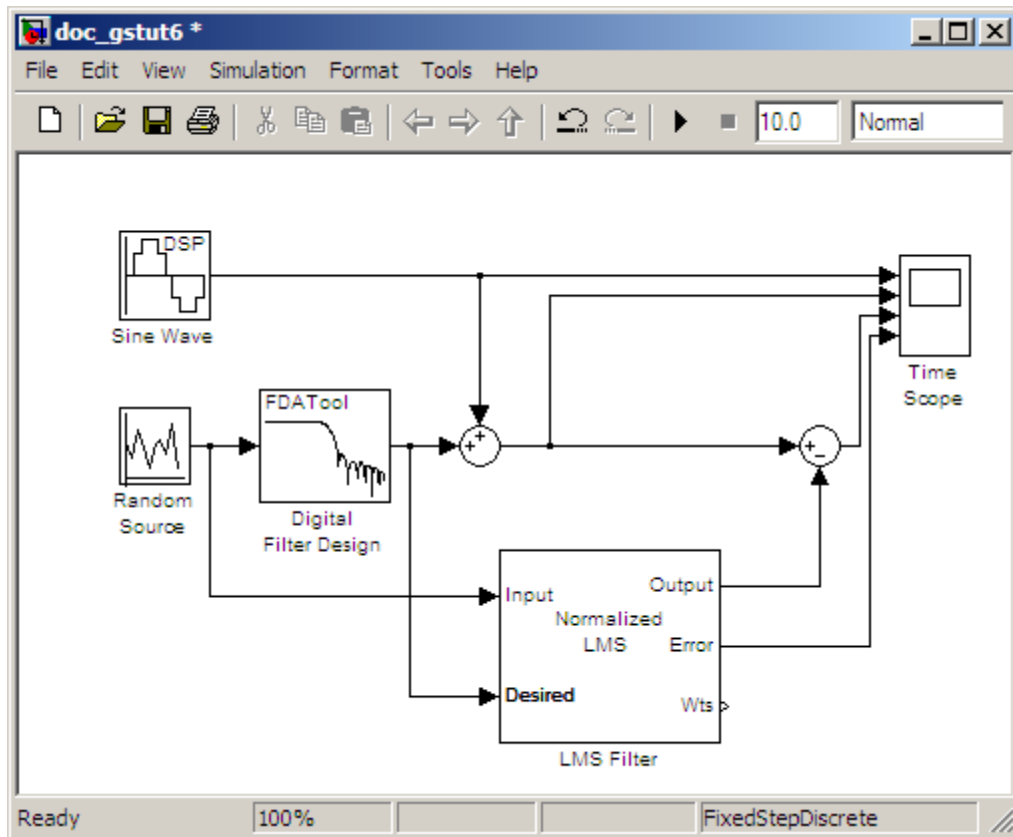
In this example, the output of the Error port is the difference between the LMS filter's desired signal and its output signal. Because the error is never zero, the filter continues to modify the filter coefficients in order to better approximate the low frequency noise. The better the approximation, the more low frequency noise that can be removed from the sinusoidal signal. In the next topic, "Viewing the Coefficients of Your Adaptive Filter" on page 4-18, you learn how to view the coefficients of your adaptive filter as they change with time.

Viewing the Coefficients of Your Adaptive Filter

The coefficients of an adaptive filter change with time in accordance with a chosen algorithm. Once the algorithm optimizes the filter's performance,

these filter coefficients reach their steady-state values. You can view the variation of your coefficients, while the simulation is running, to see them settle to their steady-state values. Then, you can determine whether you can implement these values in your actual system:

1 If the model you created in "Adding the Adaptive Filter to Your Model" on page 4-14 is not open on your desktop, you can open an equivalent model by typing

```
doc_gstut7
```

at the MATLAB command prompt. Note that the Wts port of the adaptive filter, which outputs the filter weights, still needs to be connected.
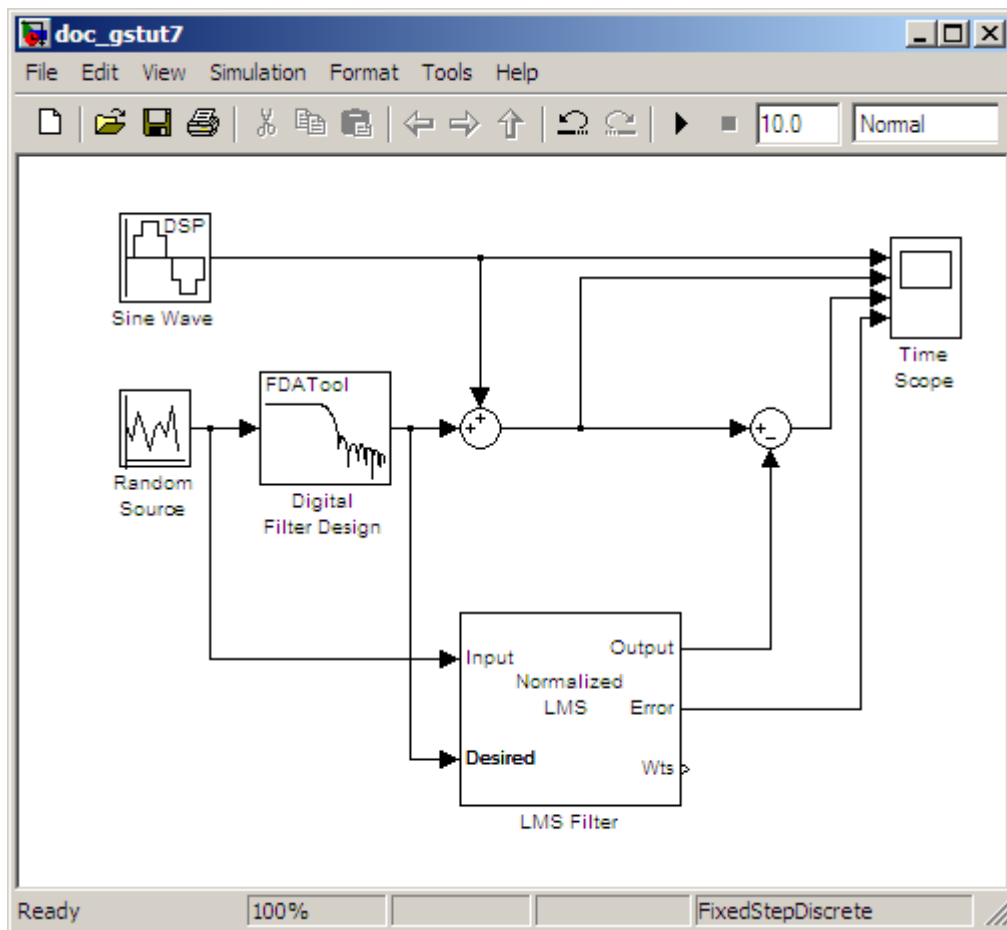
2 Open the Signal Processing Blockset library by typing dsplib at the MATLAB command prompt.

3 View the filter coefficients using a Vector Scope block from the Signal Processing Sinks library.

4 Open the Vector Scope dialog box by double-clicking the block. Set the block parameters as follows:

   a Click the **Scope Properties** tab.

- **Input domain** = `Time`
- **Time display span (number of frames)** = `1`

b Click the **Display Properties** tab.

- Select the following check boxes:

    - **Show grid**
    - **Frame number**
    - **Compact display**
    - **Open scope at start of simulation**

c Click the **Axis Properties** tab.

- **Minimum Y-limit** = `-0.2`
- **Maximum Y-limit** = `0.6`
- **Y-axis label** = `Filter Weights`

d Click the **Line Properties** tab.

- **Line visibilities** = `on`
- **Line style** = `:`
- **Line markers** = `.`
- **Line colors** = `[0 0 1]`

e Click **OK**.

5 Connect the Wts port of the LMS Filter block to the Vector Scope block.
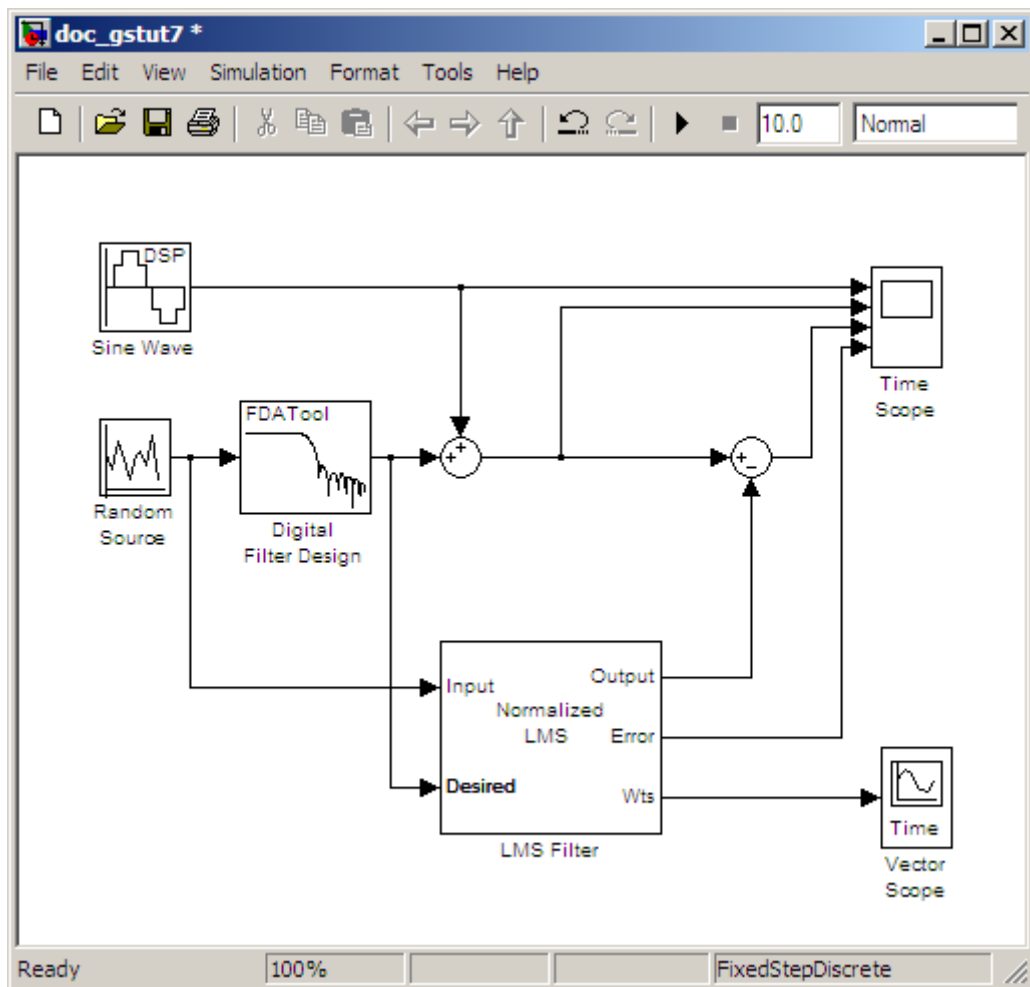
6 Set the configuration parameters:

   a Open the Configuration Parameters dialog box by selecting
      Configuration Parameters from the **Simulation** menu, and navigate
      to the **Solver** pane.

   b Enter inf for the **Stop time** parameter.

   c Choose Fixed-step from the **Type** list.

**d** Choose `Discrete (no continuous states)` from the **Solver** list.

We recommend these configuration parameters for models that contain Signal Processing Blockset blocks. Because these blocks calculate values directly rather than solving differential equations, you must configure the Simulink Solver to behave like a scheduler. The Solver, while in scheduler mode, uses a block's sample time to determine when the code behind each block is executed. For example, the sample time of the Sine Wave and Random Source blocks in this model is `0.05`. The Solver executes the code behind these blocks, and every other block with this sample time, once every 0.05 second.

Note When working with models that contain Signal Processing Blockset blocks, use source blocks that enable you to specify their sample time. If your source block does not have a **Sample time** parameter, you must add a Zero-Order Hold block in your model and use it to specify the sample time. For more information, see "Continuous-Time Source Blocks" in the *Signal Processing Blockset User's Guide*. The exception to this rule is the Constant block, which can have a constant sample time. When it does, Simulink executes this block and records the constant value once, which allows for faster simulations and more compact generated code.

---

7 Close the dialog box by clicking **OK**.

8 Open the Time Scope window by double-clicking the Time Scope block.

9 Run your model and view the behavior of your filter coefficients in the Vector Scope window, which opens automatically when your simulation starts. Over time, you see the filter coefficients change and approach their steady-state values, shown below.
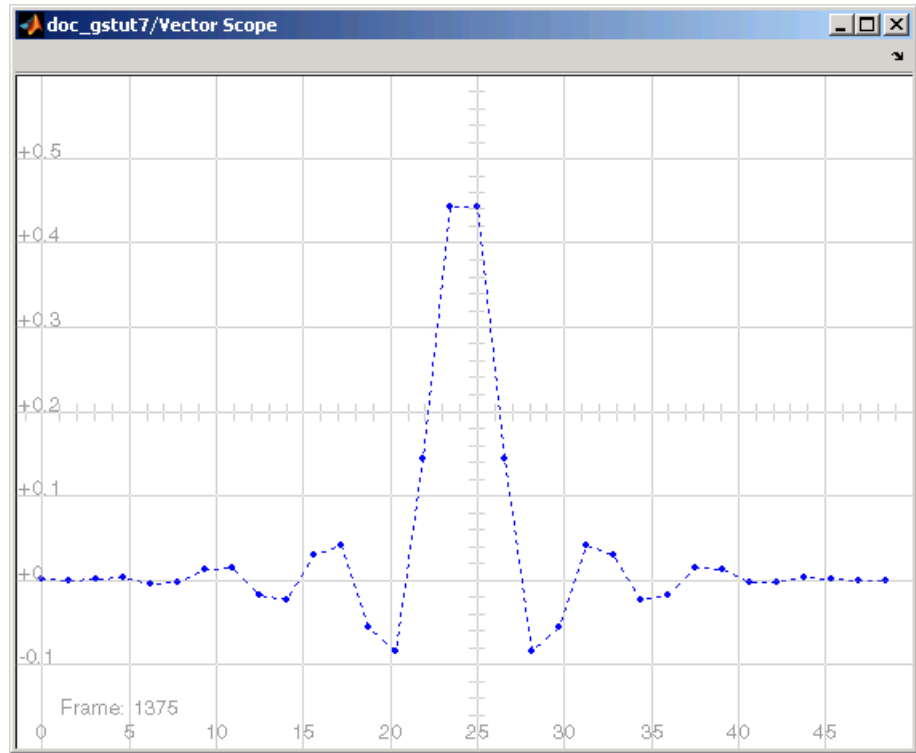
You can simultaneously view the behavior of the system in the Time Scope window. Over time, you see the error decrease and the approximation of the input signal more closely match the original sinusoidal input signal.
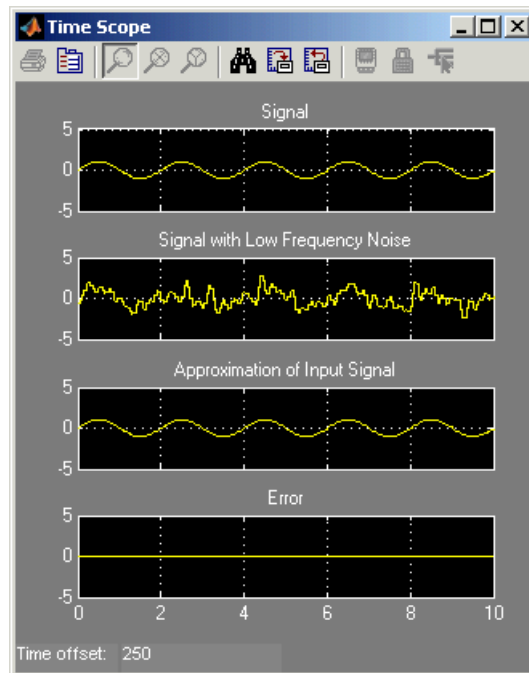
You have now created a model capable of adaptive noise cancellation. So far, you have learned how to design a lowpass filter using the Digital Filter Design block. You also learned how to create an adaptive filter using the LMS Filter block. The Signal Processing Blockset product has other blocks capable of designing and implementing digital and adaptive filters. For more information on the filtering capabilities of this product, see "Filters" in the *Signal Processing Blockset User's Guide*.

Because all blocks in this model have the same sample time, this model is single rate and Simulink ran it in `SingleTasking` solver mode. If the blocks in your model have different sample times, your model is multirate and Simulink might run it in `MultiTasking` solver mode. For more information on solver modes, see "Recommended Settings for Discrete-Time Simulations" in the *Signal Processing Blockset User's Guide*.

In Chapter 5, "Code Generation", you use the Real-Time Workshop product to generate code from your model.

Code Generation

- "Understanding Code Generation" on page 5-2
- "Generating Code" on page 5-4

Understanding Code Generation

In this section...
"Code Generation with the Real-Time Workshop Product" on page 5-2
"Highly Optimized Generated ANSI C Code" on page 5-3

Code Generation with the Real-Time Workshop Product

You can use the Signal Processing Blockset, Real-Time Workshop, and Real-Time Workshop® Embedded Coder™ products together to generate code that you can use to implement your model for a practical application. For instance, you can create an executable from your Simulink model to run on a target chip.

This chapter introduces you to the basic concepts of code generation using these tools. For more information on code generation, see "Building Executables" in the Real-Time Workshop documentation.

Shared Library Dependencies

For the blocks listed in the table below, copy the shared library files from the machine where the blockset software is installed to a folder on the system path of the destination machine.

Block	**Dependent library file**
To Wave File	towavefile.dll
From Wave File	fromwavefile.dll
To Multimedia File	tommfile.dll
	SldirectShow.dll
From Multimedia File	frommmfile.dll
	SldirectShow.dll

Block	Dependent library file
To Audio Device	libaudiodevice.dll
	libportaudio.dll
From Audio Device	libaudiodevice.dll
	libportaudio.dll

Highly Optimized Generated ANSI C Code

All Signal Processing Blockset blocks generate highly optimized ANSI C code. This C code is often suitable for embedded applications, and includes the following optimizations:

- **Function reuse (run-time libraries)** — The generated code reuses common algorithmic functions via calls to run-time functions. Run-time functions are highly optimized ANSI/ISO C functions that implement core algorithms such as FFT and convolution. Run-time functions are precompiled into ANSI/ISO C run-time libraries, and enable the blocks to generate smaller, faster code that requires less memory.

- **Parameter reuse (Real-Time Workshop run-time parameters)** — In many cases, if there are multiple instances of a block that all have the same value for a specific parameter, each block instance points to the same variable in the generated code. This process reduces memory requirements.

- **Blocks have parameters that affect code optimization** — Various blocks, such as the FFT and Sine Wave blocks, have parameters that enable you to optimize the simulation for memory or for speed. These optimizations also apply to code generation.

- **Other optimizations** — Use of contiguous input and output arrays, reusable inputs, overwritable arrays, and inlined algorithms provide smaller generated C code that is more efficient at run time.

Generating Code

In this section...
"Setting Up the Build Folder" on page 5-4
"Setting Configuration Parameters" on page 5-5
"Generating Code" on page 5-10
"Viewing the Generated Code" on page 5-11

Note You must have both the Signal Processing Blockset and Real-Time Workshop products installed on your computer to complete this section's procedures.

Setting Up the Build Folder

First, you need to create a filter folder and put a local copy of your model in it. The Real-Time Workshop software creates a build folder within this filter folder during code generation. The build folder name is *model_target_*rtw, derived from the name of the source model and the selected target. The build folder contains generated source code and other files created during the build process. This procedure assumes that your filter folder resides on drive D: (PC) or your home folder (UNIX):

1 Set up your filter folder by typing

```
!mkdir d:\filter_example
```

on a PC, or

```
!mkdir ~/filter_example
```

on UNIX.

The "!" character passes the command that follows it to the operating system, which creates the folder.

2 Make this your working folder by typing cd d:\filter_example.

3 If the model you created in "Viewing the Coefficients of Your Adaptive Filter" on page 4-18 is not open on your desktop, you can open an equivalent model by typing

    doc_gstut8

at the MATLAB command prompt.

4 Save this model as gstut8.mdl in your new working folder.

Setting Configuration Parameters

Before you can generate code, you must set several model parameters using the Configuration Parameters dialog box. To learn how to configure your model and Real-Time Workshop product settings so that your generated code accurately reflects your system, see the following topics:

- "Selecting a Solver Algorithm" on page 5-5
- "Selecting a Target Configuration" on page 5-6
- "Controlling Other Code Generation Options" on page 5-9

In these procedures, you continue to work with gstut8.mdl, the model you saved in your working folder in "Setting Up the Build Folder" on page 5-4.

Selecting a Solver Algorithm

Specify parameters that enable Simulink software to solve your model:

1 Open the Configuration Parameters dialog box for this model by selecting **Configuration Parameters** from the **Simulation** menu.

2 In the **Select** pane, click **Solver**. Set the parameters as follows:

- **Start Time** = 0.0
- **Stop Time** = 60.0
- **Type** = Fixed-step
- **Solver** = Discrete (no continuous states)
- **Fixed step size (fundamental sample time)** = 0.01

> • **Tasking mode for periodic sample times** = SingleTasking

3 Click **Apply**.

When you are finished setting the parameters, the **Solver** pane should look
similar to the following figure.
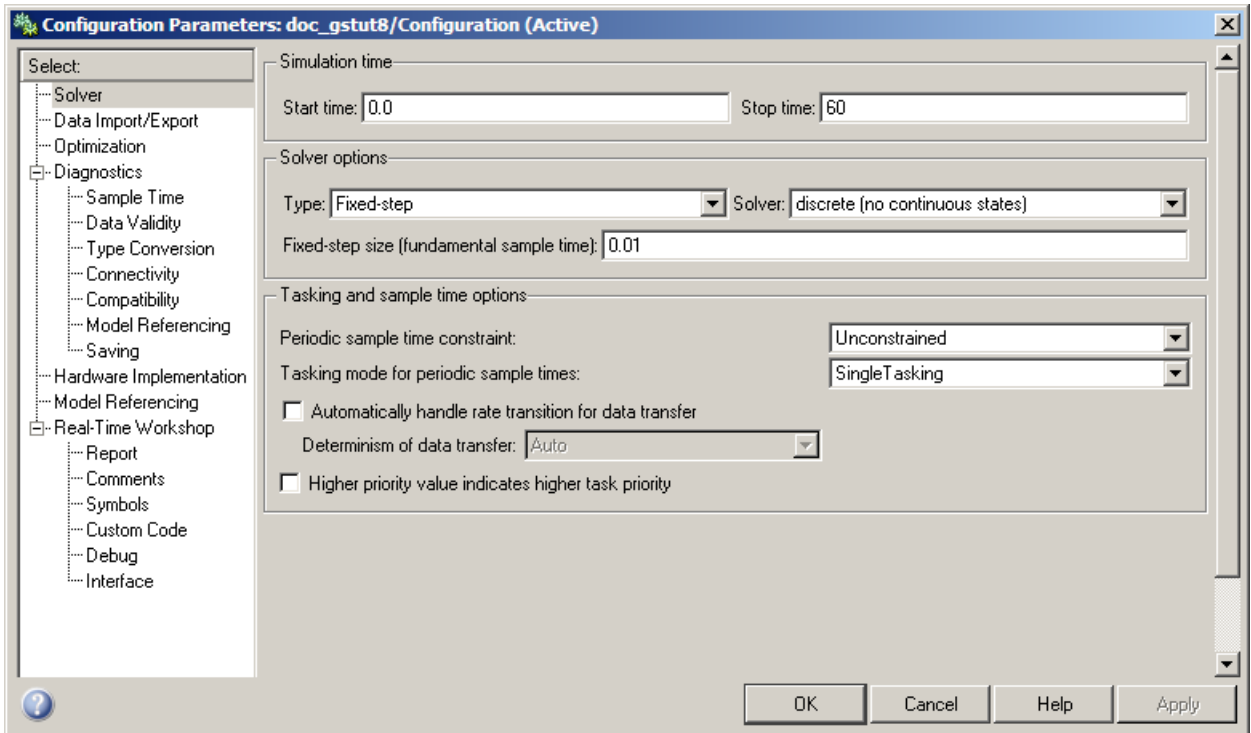
Selecting a Target Configuration
All Signal Processing Blockset blocks support the following code generation
targets:

- Generic Real-Time (GRT) target

- Embedded Real-Time (ERT) target

The MathWorks supplies the Generic Real-Time (GRT) target with the Real-Time Workshop product. This target uses the real-time code format and supports external mode communication. You can use this target as a starting point when creating a custom rapid prototyping target, or for validating the generated code on your workstation.

The MathWorks supplies the Embedded Real-Time (ERT) target with the Real-Time Workshop Embedded Coder product. This target configuration generates model code for execution on an independent embedded real-time system. For more information, see `http://www.mathworks.com/products/rtwembedded/`.

A target configuration consists of system target file, a template makefile, and a `make` command. In most situations, rather than specifying these parameters individually, you use the ready-to-run generic real-time target configuration. This GRT target is designed to build a stand-alone executable program that runs on your workstation:

1 If you have not already done so, open the Configuration Parameters dialog box. In the **Select** pane, click **Real-Time Workshop**.
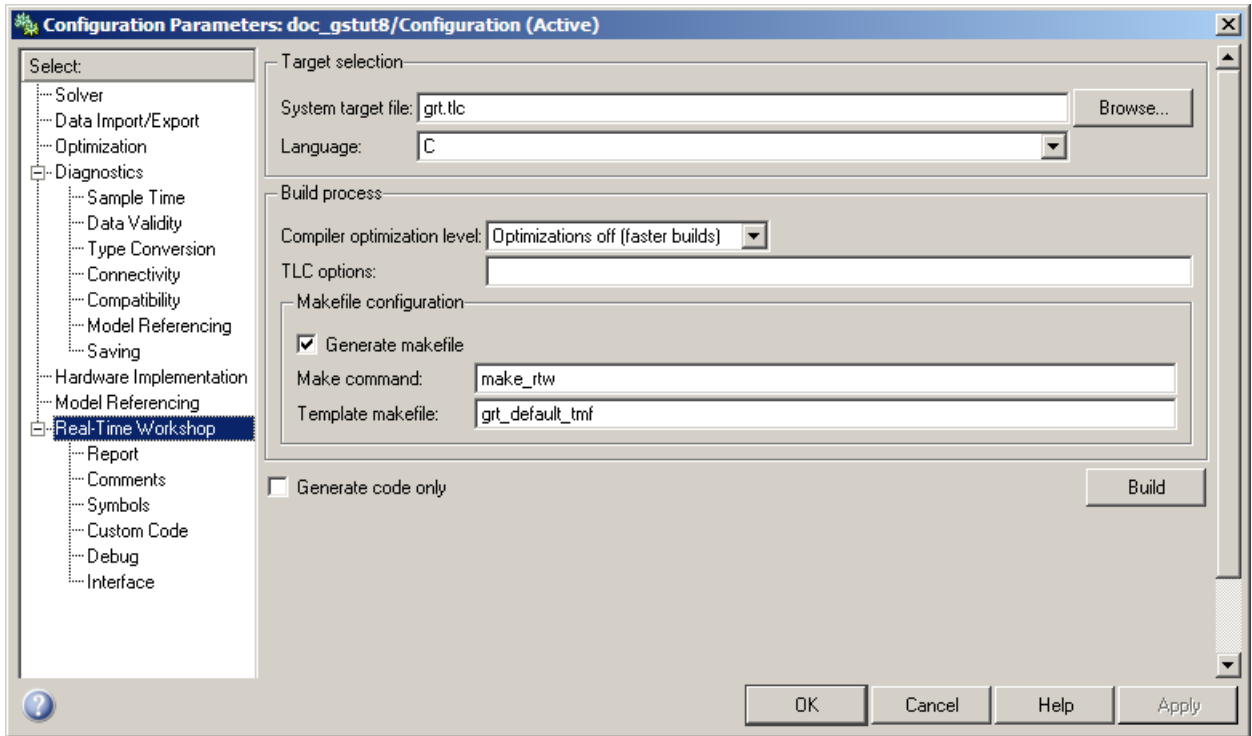
2 Make sure that the **Generate code only** check box is not selected. If you select this check box, the Real-Time Workshop product does not generate an executable after it has created source code.

3 Select a target configuration. If the **System target file** is not already specified as `grt.tlc`, open the **System target file browser** dialog box by clicking **Browse** next to the **System target file** box. The **System target file browser** displays a list of all currently available target configurations. When you select a target configuration, the appropriate system target file, template makefile, and `make` command are automatically chosen.

From the list of available configurations, select `Generic Real-Time Target` and then click **OK**.
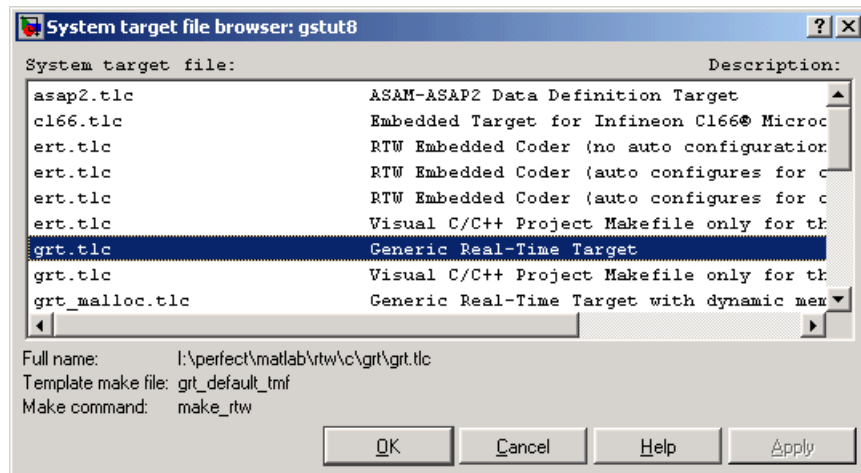
The **Real-Time Workshop** pane now displays the correct system target file (grt.tlc), make command (make_rtw). and template makefile (grt_default_tmf).

Controlling Other Code Generation Options

There are a number of additional options that you can set using the Configuration Parameters dialog box:

1 If you have not already done so, open the Configuration Parameters dialog box.

2 Prevent data from being logged to the MATLAB workspace. In the **Select** pane, click **Data Import/Export**, and then clear the **Time** and **Output** check boxes. These check boxes control whether or not the respective variables are sent to the workspace.

3 Create a navigable summary of source files when the model is built. In the **Select** pane, expand **Real-Time Workshop**. Click **Report**, and select the **Create Code Generation report** and **Launch report automatically** check boxes.

4 Make blocks that were eliminated as a result of optimizations appear as comments in the generated code. In the **Select** pane, expand **Real-Time**

Workshop. Click **Comments**, and then select the **Show eliminated blocks** check box.

5 View progress information during code generation. In the **Select** pane, click **Debug**, and then select the **Verbose build** check box. When this option is selected, the MATLAB Command Window displays progress information during code generation. The compiler also reports its progress there.

6 Apply these settings and close this dialog box by clicking **OK**.

7 Save the model in your working folder. Your configuration parameters are saved within the model file.

Once you have configured your model, you are ready to generate code.

Generating Code

After you set the configuration parameters, you can generate code from your model. In this procedure, you continue to work with `gstut8.mdl`, the model you saved in your working folder in "Setting Configuration Parameters" on page 5-5:

1 Open the Configuration Parameters dialog box. In the **Select** pane, click **Real-Time Workshop**.

2 To start the build process, click **Build** on the **Real-Time Workshop** pane. A number of messages concerning code generation and compilation appear in the MATLAB Command Window. The initial messages are

```
### Starting Real-Time Workshop build procedure for model: gstut8
### Generating code into build folder:
d:\filter_example\gstut8_grt_rtw
```

The content of the succeeding messages depends on your compiler and operating system. The final message is

```
### Successful completion of Real-Time Workshop build procedure for model: gstut8
```

3 Type `dir` at the MATLAB command prompt. The working folder now contains a build folder, `gstut8_grt_rtw`, which the Real-Time Workshop product created. The generated source files are in this folder.

You have now generated code from your signal processing model file.

Viewing the Generated Code

Once you have generated code from your Simulink model, you can view this code in an HTML report. You asked for this report by selecting the **Create code generation report** check box. The HTML report is a navigable summary of source files that you can view in the Report window:

1 View the code automatically generated by the Real-Time Workshop product. Select the Real-Time Workshop Report window. Use the links on the left side of the report to view the different source and header files that were generated.

2 To display one of the generated files, click the `gstut8.c` link in the Generated Source Files list. The generated code appears as an HTML file in the right side of the Real-Time Workshop Report window. The Real-Time Workshop product produces an HTML file for each source file in a folder named `\html` within the build folder.
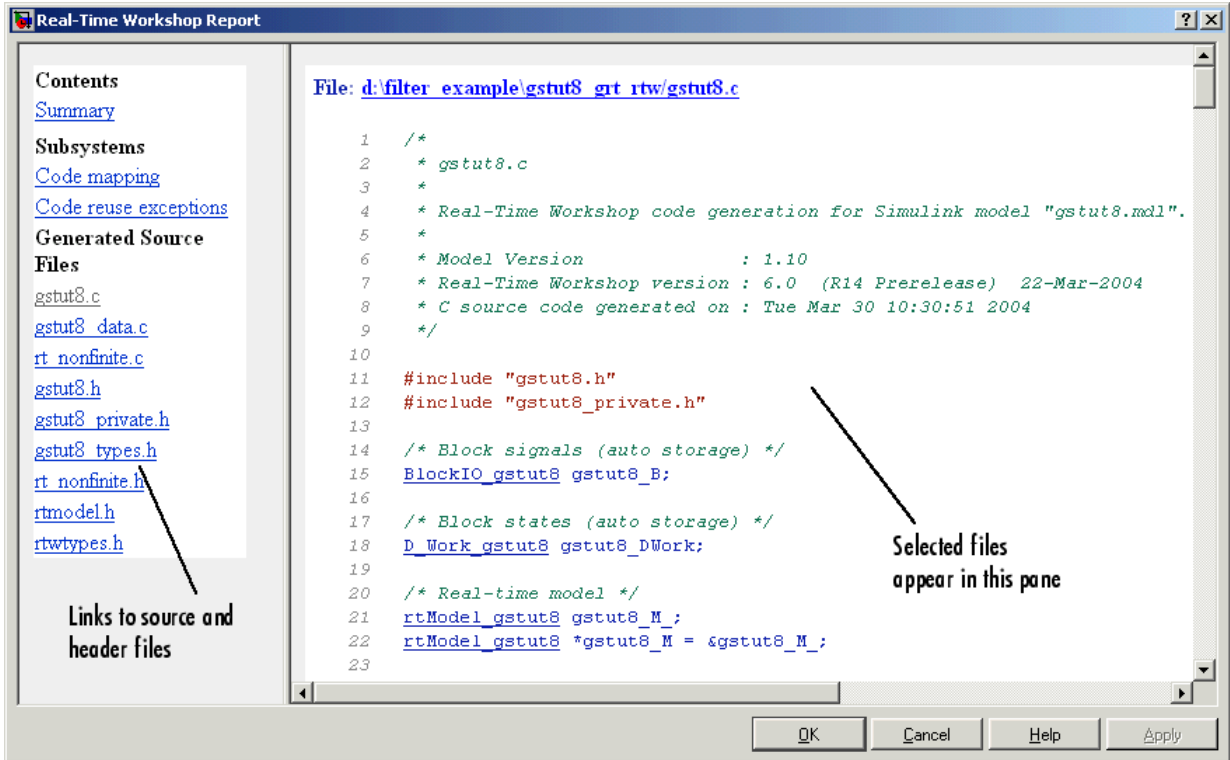
You have now generated ANSI/ISO C code from your signal processing model and viewed this code in an HTML report. For more information on code generation, see the Real-Time Workshop documentation. For information on generating fixed-point code, refer to "Code Generation" in the Simulink Fixed Point documentation.

6

Frequency Domain Signals

Power Spectrum Estimates

In this section...
"Creating the Block Diagram" on page 6-2
"Setting the Model Parameters" on page 6-3
"Viewing the Power Spectrum Estimates" on page 6-9

Creating the Block Diagram

Up until now, you have been dealing with signals in the time domain. The Signal Processing Blockset product is also capable of working with signals in the frequency domain. You can use the software to perform fast Fourier transforms (FFTs), power spectrum analysis, short-time FFTs, and many other frequency-domain applications.

The power spectrum of a signal represents the contribution of every frequency of the spectrum to the power of the overall signal. It is useful because many signal processing applications, such as noise cancellation and system identification, are based on frequency-specific modifications of signals.

First, assemble and connect the blocks needed to calculate the power spectrum of your speech signal:

1 Open a new Simulink model.

2 Add the following blocks to your model. Subsequent topics describe how to use these blocks.

Block	Library
Signal From Workspace	Signal Processing Sources
Buffer	Signal Management / Buffers
Periodogram	Estimation / Power Spectrum Estimation
Vector Scope	Signal Processing Sinks

3 Connect the blocks as shown in the next figure.
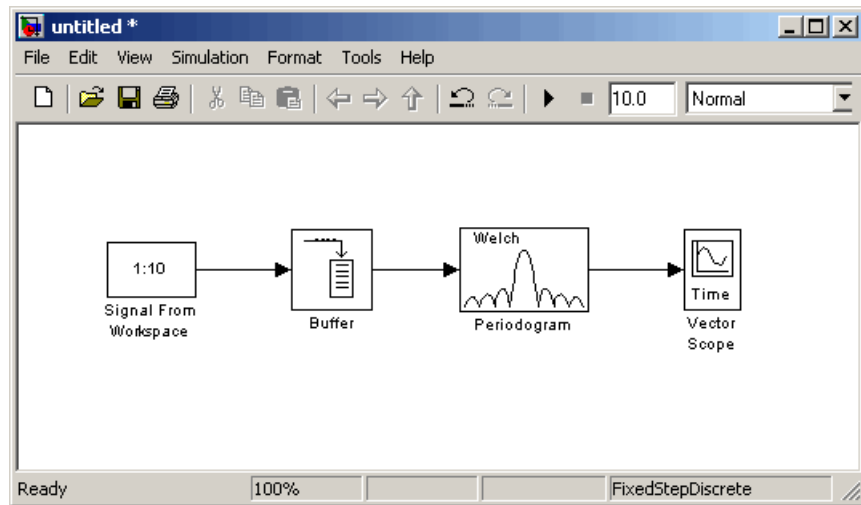
Once you have assembled the blocks needed to calculate the power spectrum of your speech signal, you can set the block parameters.

Setting the Model Parameters

Now that you have assembled the blocks needed to calculate the power spectrum of your speech signal, you need to set the block parameters. These parameter values ensure that the model calculates the power spectrum of your signal accurately:

1 If the model you created in "Creating the Block Diagram" on page 6-2 is not open on your desktop, you can open an equivalent model by typing

   doc_gstut9

at the MATLAB command prompt.

2 Load the speech signal into the MATLAB workspace by typing load mtlb at the MATLAB command prompt. This speech signal is a woman's voice saying "MATLAB."

3 Use the Signal From Workspace block to import the speech signal from the MATLAB workspace into your Simulink model. Open the Signal

From Workspace dialog box by double-clicking the block. Set the block parameters as follows:

- **Signal** = mtlb
- **Sample time** = 1/8000
- **Samples per frame** = 80
- **Form output after final data value by** = Setting to zero

Once you are done setting these parameters, the Signal From Workspace dialog box should look similar to the figure below. Click **OK** to apply your changes.
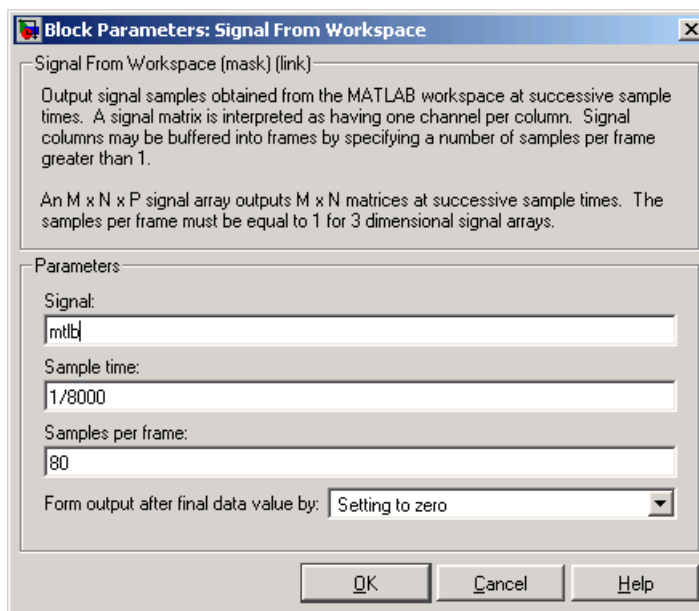
The Signal Processing Blockset product is capable of frame-based processing. In other words, Signal Processing Blockset blocks can process multiple samples of data at one time. This improves the computational speed of your model. In this case, by setting the **Samples per frame** parameter to 80, you are telling the Signal From Workspace block to output a frame that contains 80 signal samples at each simulation time step. Note

that the sample period of the input signal is 1/8000 seconds. Also, after the block outputs the final signal value, all other outputs are zero.

4 Use the Buffer block to buffer the input signal into frames that contain 128 samples. Open the Buffer dialog box by double-clicking the block. Set the block parameters as follows:

- **Output buffer size (per channel)** = 128

- **Buffer overlap** = 48

- **Initial conditions** = 0

Once you are done setting these parameters, the Buffer dialog box should look similar to the figure below. Click **OK** to apply your changes.
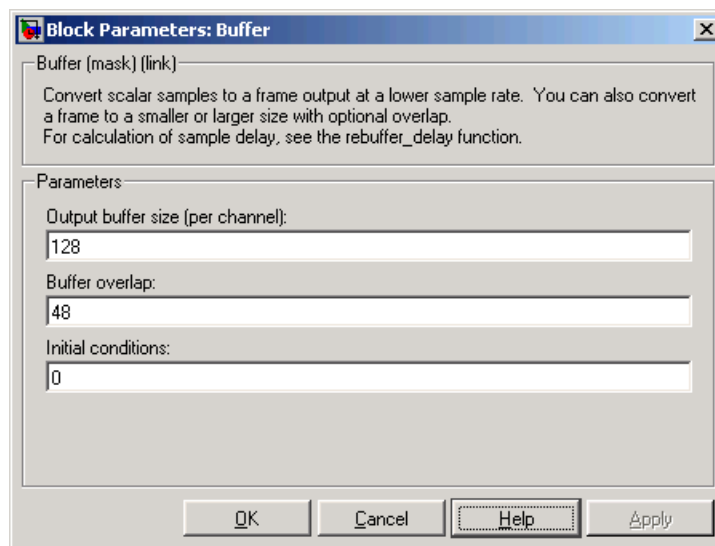
Based on these parameters, the first output frame contains 48 initial condition values followed by the first 80 samples from the first input frame. The second output frame contains the last 48 values from the previous frame followed by the second 80 samples from the second input frame, and so on. You are buffering your input signal into an output signal with 128 samples per frame to minimize the estimation noise added to your signal. Also, because 128 is a power of 2, this operation optimizes the FFT performed by the Periodogram block.

5 Use the Periodogram block to compute a nonparametric estimate of the power spectrum of the speech signal. Open the Periodogram dialog box by double-clicking the block and set the block parameters as follows:

- **Measurement** = Power spectral density

- **Window** = Hamming

- **Window sampling** = Periodic

- Select the **Inherit FFT length from input dimensions** check box.

- **Number of spectral averages** = 2

Once you are done setting these parameters, the Periodogram dialog box should look similar to the figure below. Click **OK** to apply your changes.
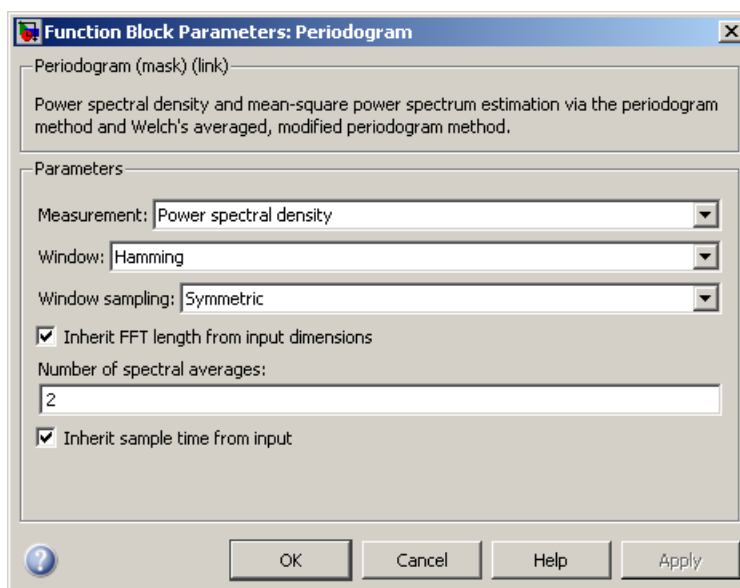
Based on these parameters, the block applies a Hamming window periodically to the input speech signal and averages two spectra at one time. The length of the FFT is assumed to be 128, which is the number of samples per frame being output from the Buffer block.

6 Use the Vector Scope block to view the power spectrum of the speech signal. Open the Vector Scope dialog box by double-clicking the block. Set the block parameters as follows:

- **Input domain** = Frequency

- Click the **Axis Properties** tab.

- Clear the **Inherit sample time from input** check box.

- **Sample time of original time series** = 1/8000

- **Y-axis label** = Magnitude-squared, dB

Once you are done setting these parameters, the **Axis Properties** pane of the Vector Scope dialog box should look similar to the figure below. As you can see by the **Y-axis scaling** parameter, the decibel amplitude is plotted in a vector scope window.
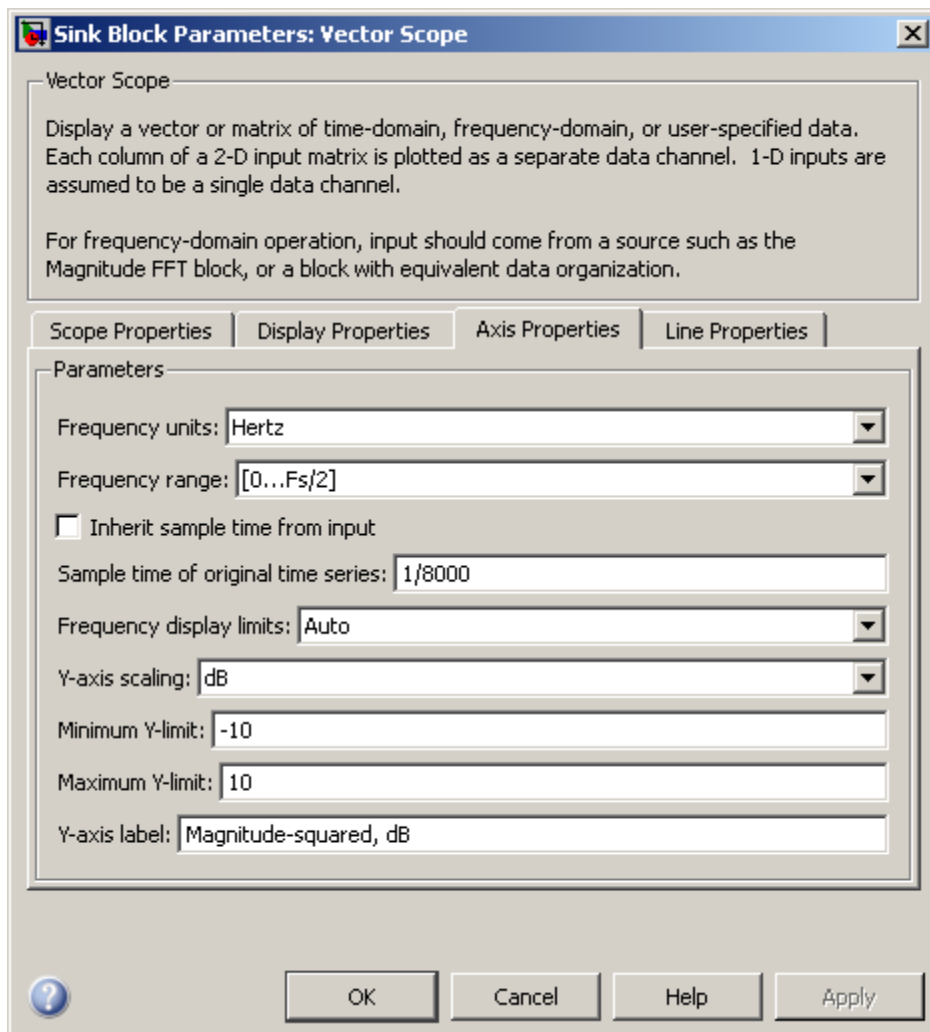
Because you are buffering the input with a nonzero overlap, you have altered the sample time of the signal. As a result, you need to specify the sample time of the original time series. Otherwise, the overlapping buffer samples lead the block to believe that the sample time is shorter than it actually is.

After you have set the block parameter values, you can calculate and view the power spectrum of the speech signal.

Viewing the Power Spectrum Estimates

In the previous topics, you created a power spectrum model and set its parameters. In this topic, you simulate the model and view the power spectrum of your speech signal:

1 If the model you created in "Setting the Model Parameters" on page 6-3 is not open on your desktop, you can open an equivalent model by typing

    doc_gstut10

at the MATLAB command prompt.

2 Set the configuration parameters. Open the Configuration Parameters dialog box by selecting **Configuration Parameters** from the **Simulation** menu. Select **Solver** from the menu on the left side of the dialog box, and set the parameters as follows:

- **Stop time** = 0.5
- **Type** = Fixed-step
- **Solver** = Discrete (no continuous states)
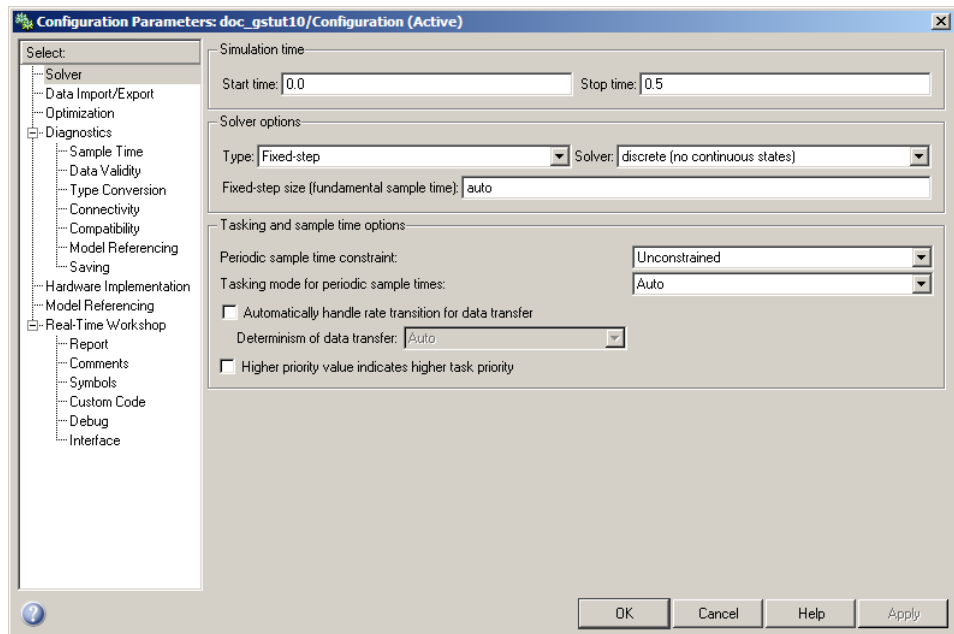
3 Apply these parameters and close the Configuration Parameters dialog box by clicking **OK**. These parameters are saved only when you save your model.

4 If you have not already done so, load the speech signal into the MATLAB workspace by typing `load mtlb`.

5 Run the model to open the Vector Scope window. The data is not immediately visible at the end of the simulation. To autoscale the *y*-axis to fit the data, in the Vector Scope window, right-click and choose **Autoscale**. The following figure shows the data displayed in the Vector Scope window.
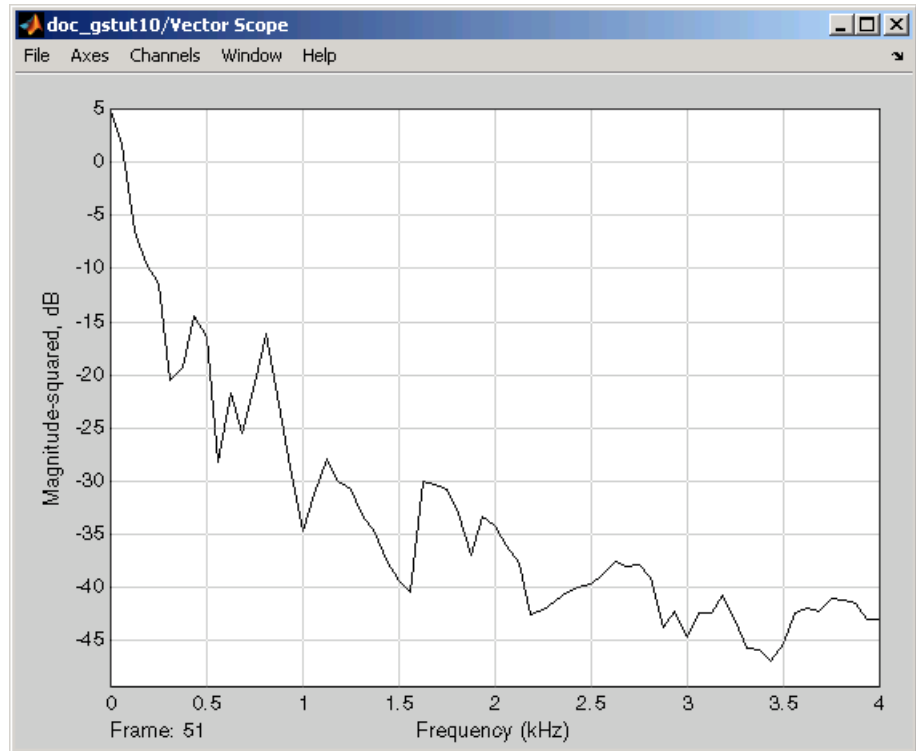
During the simulation, the Vector Scope window displays a series of frames output from the Periodogram block. Each of these frames corresponds to a window of the original speech signal. The data in each frame represents the power spectrum, or contribution of every frequency to the power of the original speech signal, for a given window.

In the next section, "Spectrograms" on page 6-12, you use these power spectrums to create a spectrogram of the speech signal.

Spectrograms

In this section...
"Modifying the Block Diagram" on page 6-12
"Setting the Model Parameters" on page 6-14
"Viewing the Spectrogram of the Speech Signal" on page 6-18

Modifying the Block Diagram

Spectrograms are color-based visualizations of the evolution of the power spectrum of a speech signal as this signal is swept through time. Spectrograms use the periodogram power spectrum estimation method and are widely used by speech and audio engineers. You can use them to develop a visual understanding of the frequency content of your speech signal while a particular sound is being vocalized.

In the previous section, you built a model capable of calculating the power spectrum of a speech signal that represents a woman saying "MATLAB." In this topic, you modify this model to view the spectrogram of your signal:

1 If the model you created in "Viewing the Power Spectrum Estimates" on page 6-9 is not open on your desktop, you can open an equivalent model by typing

    doc_gstut11

at the MATLAB command prompt.
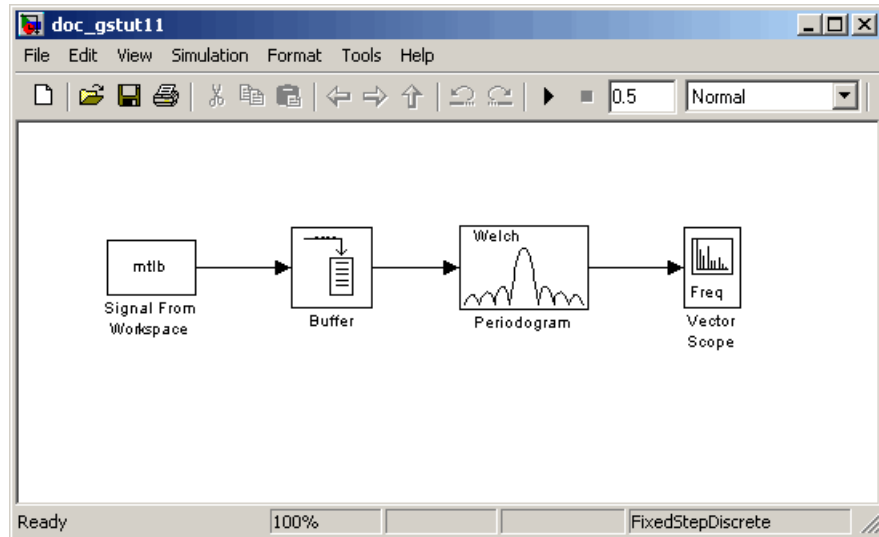
2 Add the following blocks to your model. Subsequent topics describe how to use these blocks.

Block	Library
Selector	Simulink / Signal Routing
dB Conversion	Math Functions / Math Operations
Buffer	Signal Management / Buffers
Transpose	Math Functions / Matrices and Linear Algebra / Matrix Operations
Matrix Viewer	Signal Processing Sinks

3 Connect the blocks as shown in the figure below. These blocks extract the positive frequencies of each power spectrum and concatenate them into a matrix that represents the spectrogram of the speech signal.
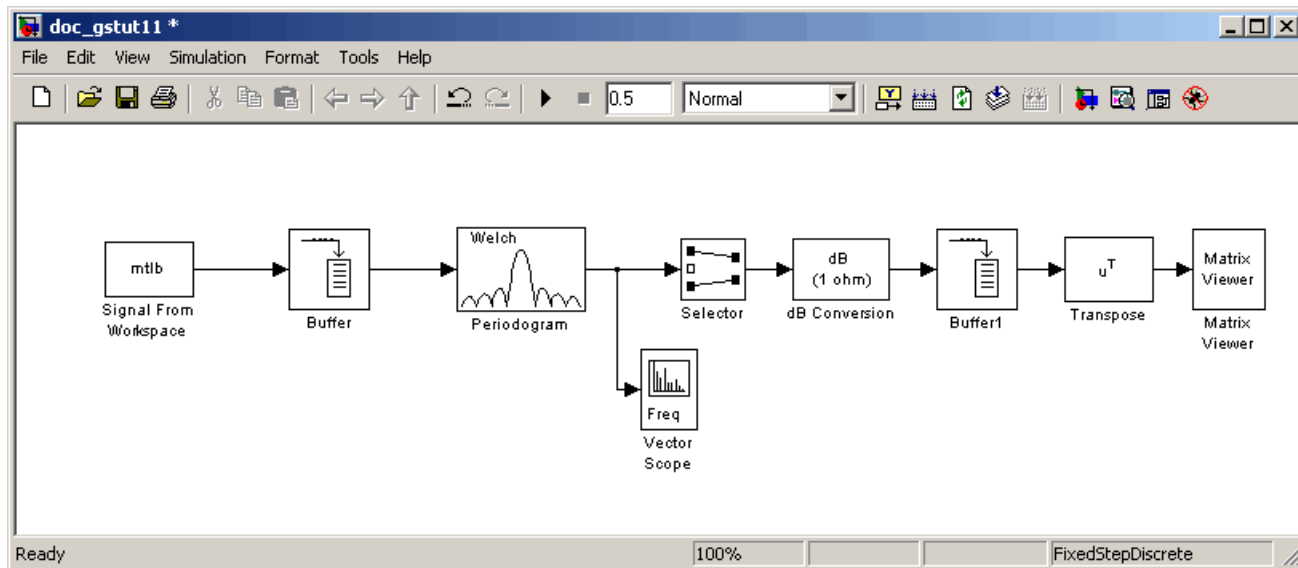
Once you have assembled the blocks needed to view the spectrogram of your speech signal, you can set the block parameters.

Setting the Model Parameters

In the previous topic, you assembled the blocks you need to view the spectrogram of your speech signal. Now you must set the block parameters:

1 If the model you created in "Modifying the Block Diagram" on page 6-12 is not open on your desktop, you can open an equivalent model by typing

    doc_gstut12

at the MATLAB command prompt.

2 Use the Selector block to extract the first 64 elements, or the positive frequencies, of each power spectrum. Open the Selector dialog box by double-clicking the block. Set the block parameters as follows:

- **Number of input dimensions** = 1

- **Index mode** = One-based

- **Index option** = Index vector (dialog)
- **Index** = 1:64
- **Input port size** = 128

At each time instance, the input to the Selector block is a vector of 128 elements. The block assigns one-based indices to these elements and extracts the first 64. Once you are done setting these parameters, the Selector dialog box should look similar to the figure below. To apply your changes, click **OK**.
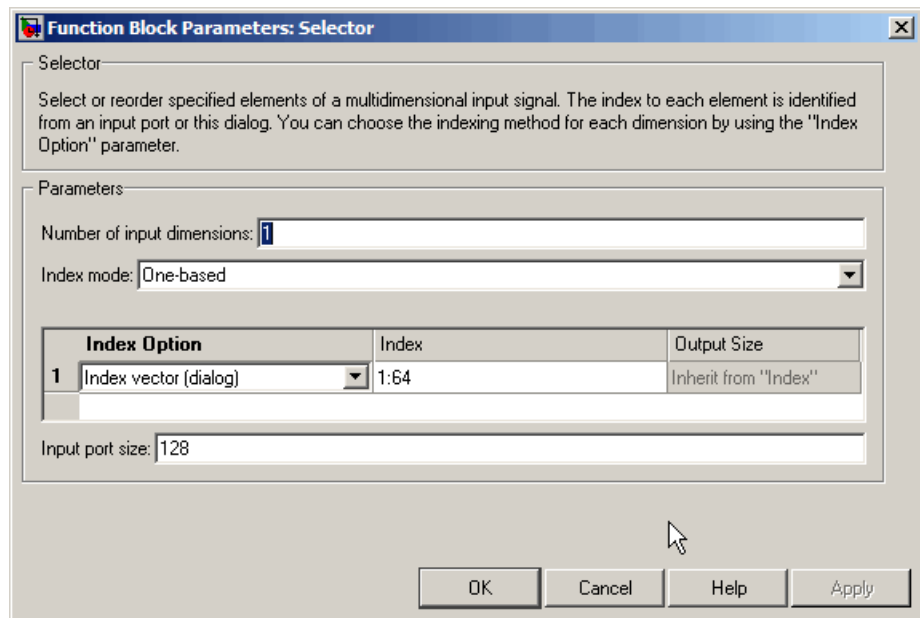
3 The dB Conversion block converts the magnitude of the input FFT signal to decibels. Leave this block at its default parameters.

4 Use the Buffer1 block to concatenate the individual power spectrums into a matrix. Open the Buffer1 dialog box by double-clicking the block. Set the block parameters as follows:

- **Output buffer size (per channel)** = 48
- **Buffer overlap** = 46

- **Initial conditions** = -70

Once you are done setting these parameters, the Buffer1 dialog box should look similar to the figure below. To apply your changes, click **OK**.
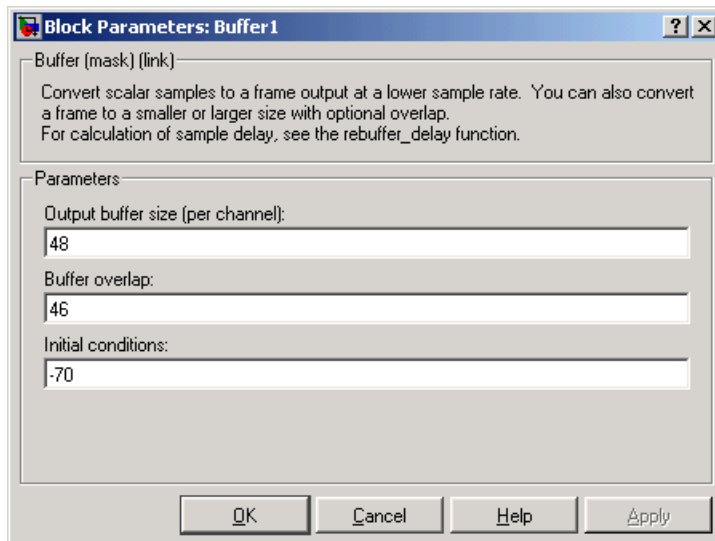
Based on these parameters, the Buffer1 block buffers the 64-by-1 frame-based input signal 48 times in order to create a 64-by-48 frame-based signal. In other words, it collects the power spectrums calculated at each time and concatenates them into a matrix. The block then outputs the transpose of this matrix. To ensure that your spectrogram represents smooth movement through time, set the value of the **Buffer overlap** parameter slightly less than the value of the **Output buffer size (per channel)** parameter. The **Initial conditions** parameter represents the initial values in the buffer; -70 represents silence.

5 The Transpose block transposes the input signal back to its original orientation. Leave this block at its default parameters.

6 The Matrix Viewer enables you to view the spectrogram of the speech signal. Open the Matrix Viewer dialog box by double-clicking the block. Set the block parameters as follows:

- Click the **Image Properties** tab.

- **Colormap matrix** = jet(256)
- **Minimum input value** = -70
- **Maximum input value** = 15
- Select the **Display colorbar** check box.

Once you are done setting these parameters, the **Image Properties** pane should look similar to the figure below.
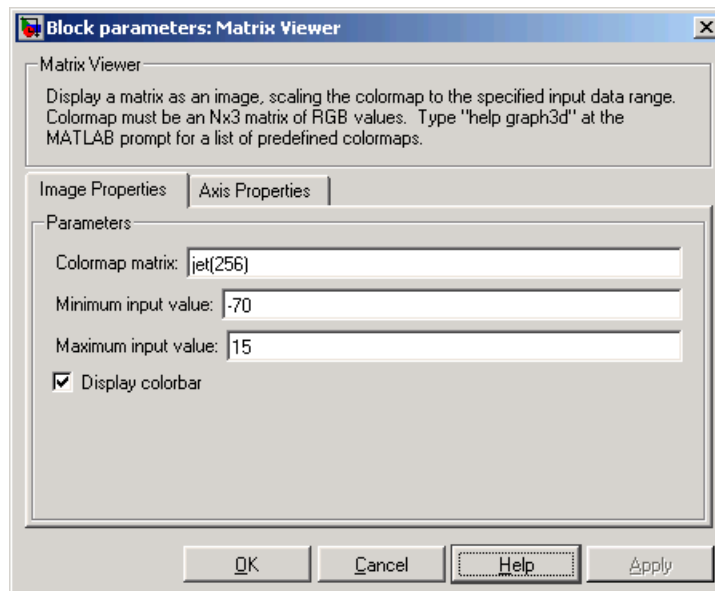
- Click the **Axis Properties** tab.
- **Axis origin** = Lower left corner
- **X-axis title** = Time Index
- **Y-axis title** = Frequency Index
- **Colorbar title** = dB Magnitude

In this case, you are assuming that the power spectrum values do not exceed 15 dB. Once you are done setting these parameters, the **Axis Properties** pane should look similar to the figure below. To apply your changes, click **OK**.
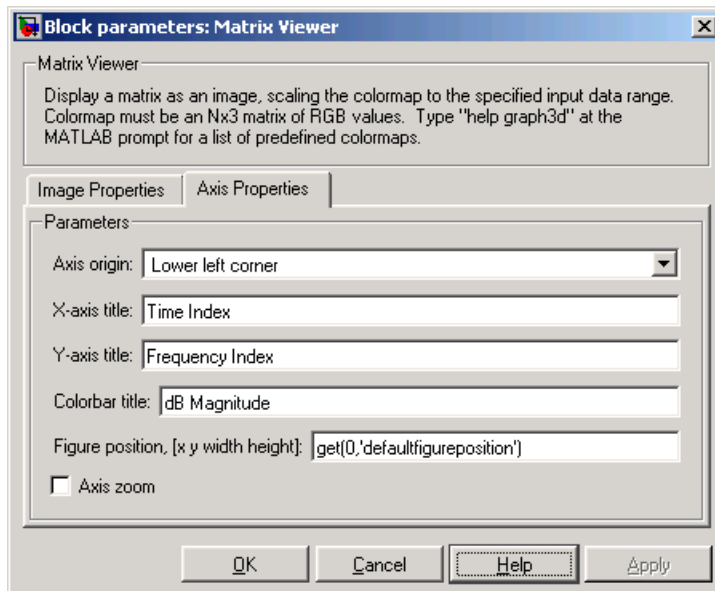
After you have set the parameter values, you can calculate and view the spectrogram of the speech signal.

Viewing the Spectrogram of the Speech Signal

In the topic "Viewing the Power Spectrum Estimates" on page 6-9, you used a Vector Scope block to display the power spectrum of your speech signal. In this topic, you view the spectrogram of your speech signal using a Matrix Viewer block. The speech signal represents a woman's voice saying "MATLAB":

1 If the model you created in "Setting the Model Parameters" on page 6-14 is not open on your desktop, you can open an equivalent model by typing

    doc_gstut13

at the MATLAB command prompt.
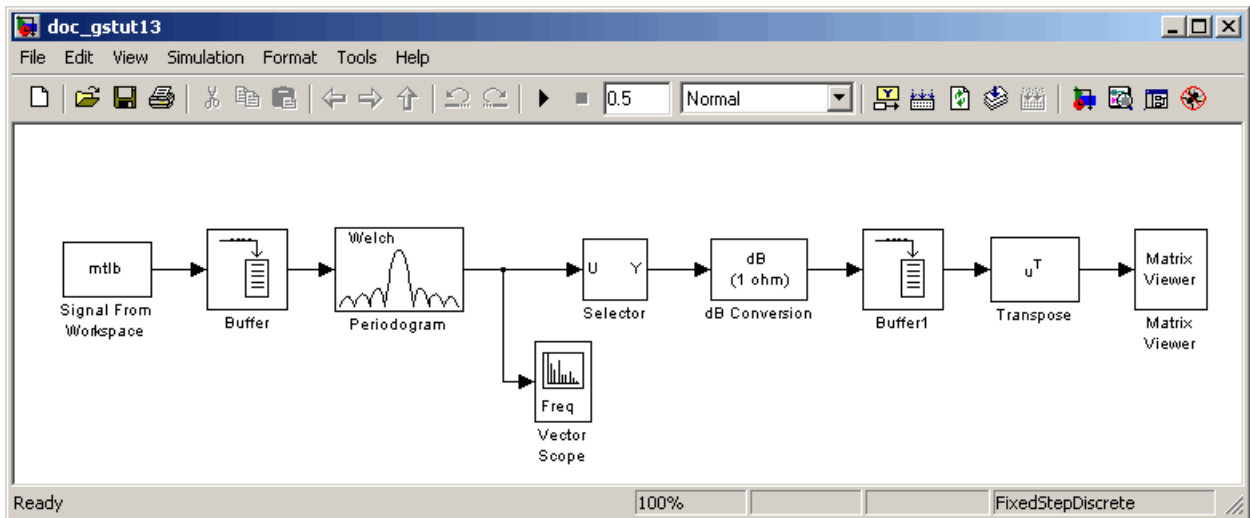
2 Run the model. During the simulation, the Vector Scope window displays a sequence of power spectrums, one for each window of the original speech signal. The power spectrum is the contribution of every frequency to the power of the speech signal.
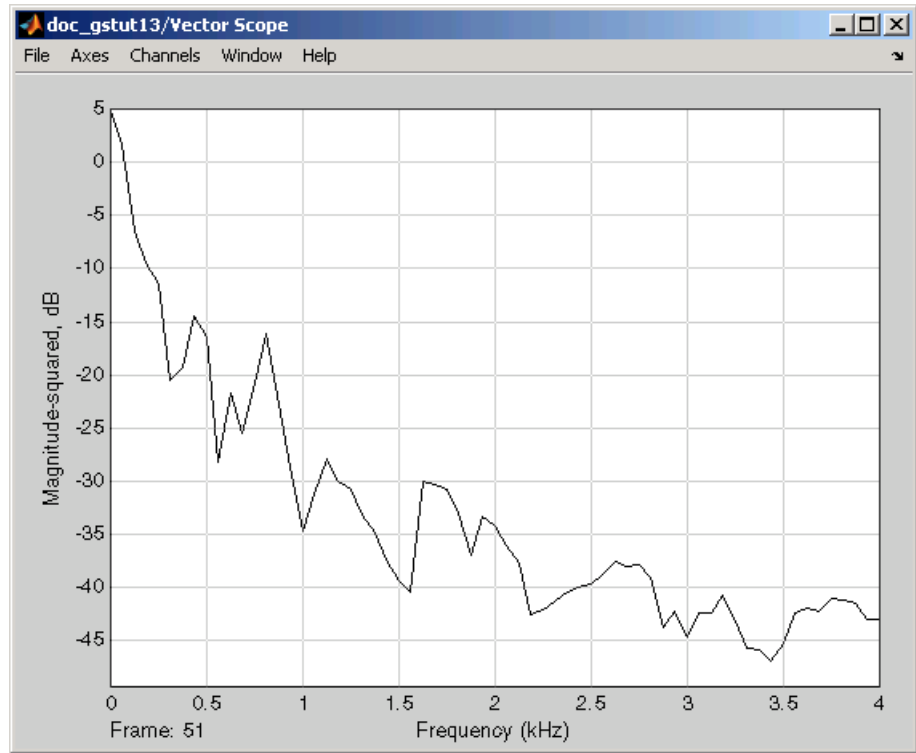
The Matrix Viewer window, shown below, displays the spectrogram of the speech signal. This spectrogram is calculated using the Periodogram power spectrum estimation method. Note the harmonics that are visible in the signal when the vowels are spoken. Most of the signal's energy is concentrated in these harmonics; therefore, two distinct peaks are visible in the spectrogram.
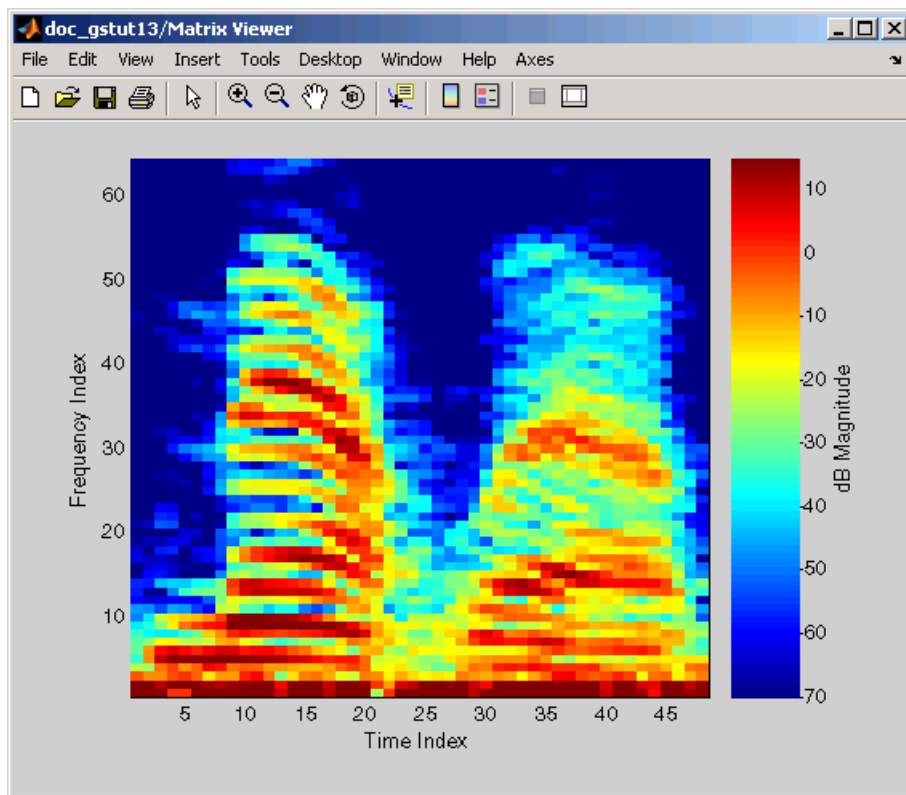
In this example, you viewed the spectrogram of your speech signal using a Matrix Viewer block. You can find additional Signal Processing Blockset product examples in the Help browser. To access these examples, click the **Contents** tab, double-click **Signal Processing Blockset**, and then click **Examples**. A list of the examples in the Signal Processing Blockset documentation appears in the right pane of the Help browser.

For information about Signal Processing Blockset demos, see "Product Demos" on page 1-5. For additional information about Signal Processing Blockset product functionality, see the Signal Processing Blockset User's Guide.

Index