Signal Processing Blockset™ 7

Reference

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™ Reference

© COPYRIGHT 1995–2010 The MathWorks, Inc.

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

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

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` 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

April 1995	First printing	New for Version 1.0
May 1997	Second printing	Revised for Version 2.0
January 1998	Third printing	Revised for Version 2.2 (Release 10)
January 1999	Fourth printing	Revised for Version 3.0 (Release 11)
November 2000	Fifth printing	Revised for Version 4.0 (Release 12)
June 2001	Online only	Revised for Version 4.1 (Release 12.1)
July 2002	Sixth printing	Revised for Version 5.0 (Release 13)
April 2003	Seventh printing	Revised for Version 5.1 (Release 13SP1)
June 2004	Online only	Revised for Version 6.0 (Release 14) (Renamed from DSP Blockset User's Guide)
October 2004	Online only	Revised for Version 6.0.1 (Release 14SP1)
March 2005	Online only	Revised for Version 6.1 (Release 14SP2)
September 2005	Online only	Revised for Version 6.2 (Release 14SP3)
March 2006	Online only	Revised for Version 6.3 (Release 2006a)
September 2006	Online only	Revised for Version 6.4 (Release 2006b)
March 2007	Online only	Revised for Version 6.5 (Release 2007a)
September 2007	Online only	Revised for Version 6.6 (Release 2007b)
March 2008	Online only	Revised for Version 6.7 (Release 2008a)
October 2008	Online only	Revised for Version 6.8 (Release 2008b)
March 2009	Online only	Revised for Version 6.9 (Release 2009a)
September 2009	Online only	Revised for Version 6.10 (Release 2009b)
March 2010	Online only	Revised for Version 7.0 (Release 2010a)

Contents

1 Block Reference

2 Blocks — Alphabetical List

3 System Object Reference

Block Reference

Estimation

Linear Prediction

Autocorrelation LPC	Determine coefficients of Nth-order forward linear predictors
Levinson-Durbin	Solve linear system of equations using Levinson-Durbin recursion
LPC to LSF/LSP Conversion	Convert linear prediction coefficients to line spectral pairs or line spectral frequencies
LPC to/from Cepstral Coefficients	Convert linear prediction coefficients to cepstral coefficients or cepstral coefficients to linear prediction coefficients
LPC to/from RC	Convert linear prediction coefficients to reflection coefficients or reflection coefficients to linear prediction coefficients
LPC/RC to Autocorrelation	Convert linear prediction coefficients or reflection coefficients to autocorrelation coefficients
LSF/LSP to LPC Conversion	Convert line spectral frequencies or line spectral pairs to linear prediction coefficients

Parametric Estimation

Burg AR Estimator	Compute estimate of autoregressive (AR) model parameters using Burg method
Covariance AR Estimator	Compute estimate of autoregressive (AR) model parameters using covariance method
Modified Covariance AR Estimator	Compute estimate of autoregressive (AR) model parameters using modified covariance method
Yule-Walker AR Estimator	Compute estimate of autoregressive (AR) model parameters using Yule-Walker method

Power Spectrum Estimation

Burg Method	Power spectral density estimate using Burg method
Covariance Method	Power spectral density estimate using covariance method
Magnitude FFT	Compute nonparametric estimate of spectrum using periodogram method
Modified Covariance Method	Power spectral density estimate using modified covariance method
Periodogram	Power spectral density or mean-square spectrum estimate using periodogram method
Yule-Walker Method	Power spectral density estimate using Yule-Walker method

Filtering

Adaptive Filters

Block LMS Filter	Compute output, error, and weights using LMS adaptive algorithm
Fast Block LMS Filter	Compute output, error, and weights using LMS adaptive algorithm
Kalman Filter	Predict or estimate states of dynamic systems
LMS Filter	Compute output, error, and weights using LMS adaptive algorithm
RLS Filter	Compute filtered output, filter error, and filter weights for given input and desired signal using RLS adaptive filter algorithm

Filter Designs

Arbitrary Response Filter	Design arbitrary response filter
Bandpass Filter	Design bandpass filter
Bandstop Filter	Design bandstop filter
CIC Compensator	Design CIC compensator
CIC Filter	Design Cascaded Integrator-Comb (CIC) Filter

Comb Filter	Design comb Filter
Differentiator Filter	Design differentiator filter
Halfband Filter	Design halfband filter
Highpass Filter	Design highpass filter
Hilbert Filter	Design Hilbert filter
Inverse Sinc Filter	Design inverse sinc filter
Lowpass Filter	Design lowpass Filter
Nyquist Filter	Design Nyquist filter
Octave Filter	Design octave filter
Parametric Equalizer	Design parametric equalizer
Peak-Notch Filter	Design peak or notch filter
Pulse Shaping Filter	Design pulse shaping filter

Filter Implementations

Analog Filter Design	Design and implement analog filters
Biquad Filter	Model biquadratic IIR (SOS) filters
Digital Filter	Filter each channel of input over time using static or time-varying digital filter implementations
Digital Filter Design	Design and implement digital FIR and IIR filters
Filter Realization Wizard	Construct filter realizations using Digital Filter block or Sum, Gain, and Delay blocks
Overlap-Add FFT Filter	Implement overlap-add method of frequency-domain filtering
Overlap-Save FFT Filter	Implement overlap-save method of frequency-domain filtering

Multirate Filters

CIC Decimation	Decimate signal using Cascaded Integrator-Comb filter
CIC Interpolation	Interpolate signal using Cascaded Integrator-Comb filter
Dyadic Analysis Filter Bank	Decompose signals into subbands with smaller bandwidths and slower sample rates or compute discrete wavelet transform (DWT)
Dyadic Synthesis Filter Bank	Reconstruct signals from subbands with smaller bandwidths and slower sample rates or compute inverse discrete wavelet transform (IDWT)
FIR Decimation	Filter and downsample input signals
FIR Interpolation	Upsample and filter input signals
FIR Rate Conversion	Upsample, filter, and downsample input signals
Two-Channel Analysis Subband Filter	Decompose signal into high-frequency subband and low-frequency subband
Two-Channel Synthesis Subband Filter	Reconstruct signal from high-frequency subband and low-frequency subband

Math Functions

Math Operations (p. 1-7)	Use specialized math operations for signal processing applications
Matrices and Linear Algebra (p. 1-7)	Work with matrices
Polynomial Functions (p. 1-10)	Work with polynomials

Math Operations

Complex Exponential	Compute complex exponential function
Cumulative Product	Compute cumulative product of channel, column, or row elements
Cumulative Sum	Compute cumulative sum of channel, column, or row elements
dB Conversion	Convert magnitude data to decibels (dB or dBm)
dB Gain	Apply decibel gain
Difference	Compute element-to-element difference along specified dimension of input
Normalization	Perform vector normalization along rows, columns, or specified dimension

Matrices and Linear Algebra

Linear System Solvers (p. 1-7)	Solve matrix equation AX = B for X
Matrix Factorizations (p. 1-8)	Factor matrices
Matrix Inverses (p. 1-9)	Invert matrices
Matrix Operations (p. 1-9)	Perform basic matrix operations

Linear System Solvers

Backward Substitution	Solve $UX=B$ for X when U is upper triangular matrix
Cholesky Solver	Solve SX=B for X when S is square Hermitian positive definite matrix

Forward Substitution	Solve $LX=B$ for X when L is lower triangular matrix
LDL Solver	Solve $SX=B$ for X when S is square Hermitian positive definite matrix
Levinson-Durbin	Solve linear system of equations using Levinson-Durbin recursion
LU Solver	Solve $AX=B$ for X when A is square matrix
QR Solver	Find minimum-norm-residual solution to AX=B
SVD Solver	Solve $AX=B$ using singular value decomposition

Matrix Factorizations

Cholesky Factorization	Factor square Hermitian positive definite matrix into triangular components
LDL Factorization	Factor square Hermitian positive definite matrices into lower, upper, and diagonal components
LU Factorization	Factor square matrix into lower and upper triangular components
QR Factorization	Factor arbitrary matrix into unitary and upper triangular components
Singular Value Decomposition	Factor matrix using singular value decomposition

Matrix Inverses

Cholesky Inverse	Compute inverse of Hermitian positive definite matrix using Cholesky factorization
LDL Inverse	Compute inverse of Hermitian positive definite matrix using LDL factorization
LU Inverse	Compute inverse of square matrix using LU factorization
Pseudoinverse	Compute Moore-Penrose pseudoinverse of matrix

Matrix Operations

Array-Vector Add	Add vector to array along specified dimension
Array-Vector Divide	Divide array by vector along specified dimension
Array-Vector Multiply	Multiply array by vector along specified dimension
Array-Vector Subtract	Subtract vector from array along specified dimension
Constant Diagonal Matrix	Generate square, diagonal matrix
Create Diagonal Matrix	Create square diagonal matrix from diagonal elements
Extract Diagonal	Extract main diagonal of input matrix
Extract Triangular Matrix	Extract lower or upper triangle from input matrices
Identity Matrix	Generate matrix with ones on main diagonal and zeros elsewhere
Matrix 1-Norm	Compute 1-norm of matrix

Matrix Concatenate	Concatenate input signals of same data type to create contiguous output signal
Matrix Exponential	Compute matrix exponential
Matrix Multiply	Multiply or divide inputs
Matrix Product	Multiply matrix elements along rows, columns, or entire input
Matrix Square	Compute square of input matrix
Matrix Sum	Sum matrix elements along rows, columns, or entire input
Matrix Sum (Obsolete)	Sum matrix elements along rows, columns, or entire input
Overwrite Values	Overwrite submatrix or subdiagonal of input
Permute Matrix	Reorder matrix rows or columns
Reciprocal Condition	Compute reciprocal condition of square matrix in 1-norm
Submatrix	Select subset of elements (submatrix) from matrix input
Toeplitz	Generate matrix with Toeplitz symmetry
Transpose	Compute matrix transpose

Polynomial Functions

Least Squares Polynomial Fit	Compute polynomial coefficients that best fit input data in least-squares sense
Polynomial Evaluation	Evaluate polynomial expression
Polynomial Stability Test	Use Schur-Cohn algorithm to determine whether all roots of input polynomial are inside unit circle

Quantizers

G711 Codec	Quantize narrowband speech input signals
Quantizer	Discretize input at specified interval
Scalar Quantizer Decoder	Convert each index value into quantized output value
Scalar Quantizer Design	Start Scalar Quantizer Design Tool (SQDTool) to design scalar quantizer using Lloyd algorithm
Scalar Quantizer Encoder	Encode each input value by associating it with index value of quantization region
Uniform Decoder	Decode integer input into floating-point output
Uniform Encoder	Quantize and encode floating-point input into integer output
Vector Quantizer Decoder	Find vector quantizer codeword that corresponds to given, zero-based index value
Vector Quantizer Design	Design vector quantizer using Vector Quantizer Design Tool (VQDTool)
Vector Quantizer Encoder	For given input, find index of nearest codeword based on Euclidean or weighted Euclidean distance measure

Signal Management

Buffers (p. 1-12)	Change sample rate or frame rate of signals by buffering or unbuffering
Indexing (p. 1-12)	Manipulate ordering of signals

Buffers

Buffer	Buffer input sequence to smaller or larger frame size
Delay Line	Rebuffer sequence of inputs with one-sample shift
Queue	Store inputs in FIFO register
Stack	Store inputs into LIFO register
Unbuffer	Unbuffer input frame into sequence of scalar outputs

Indexing

Flip	Flip input vertically or horizontally
Multiport Selector	Distribute arbitrary subsets of input rows or columns to multiple output ports
Overwrite Values	Overwrite submatrix or subdiagonal of input
Selector	Select input elements from vector, matrix, or multidimensional signal
Submatrix	Select subset of elements (submatrix) from matrix input
Variable Selector	Select subset of rows or columns from input

Signal Attributes

Check Signal Attributes	Error when input signal does or does not match selected attributes exactly
Convert 1-D to 2-D	Reshape 1-D or 2-D input to 2-D matrix with specified dimensions
Convert 2-D to 1-D	Convert 2-D matrix input to 1-D vector
Data Type Conversion	Convert input signal to specified data type
Frame Conversion	Specify sampling mode of output signal
Inherit Complexity	Change complexity of input to match reference signal

Switches and Counters

Counter	Count up or down through specified range of numbers
Edge Detector	Detect transition from zero to nonzero value
Event-Count Comparator	Detect threshold crossing of accumulated nonzero inputs
Multiphase Clock	Generate multiple binary clock signals
N-Sample Enable	Output ones or zeros for specified number of sample times
N-Sample Switch	Switch between two inputs after specified number of sample periods

Signal Operations

Constant Ramp	Generate ramp signal with length based on input dimensions
Convolution	Compute convolution of two inputs
Delay	Delay discrete-time input by specified number of samples or frames
Downsample	Resample input at lower rate by deleting samples
Interpolation	Interpolate values of real input samples
NCO	Generate real or complex sinusoidal signals
Offset	Truncate vectors by removing or keeping beginning or ending values
Pad	Pad or truncate specified dimension(s)
Peak Finder	Determine whether each value of input signal is local minimum or maximum
Repeat	Resample input at higher rate by repeating values
Sample and Hold	Sample and hold input signal
Triggered Signal From Workspace	Import signal samples from MATLAB® workspace when triggered
Unwrap	Unwrap signal phase
Upsample	Resample input at higher rate by inserting zeros
Variable Fractional Delay	Delay input by time-varying fractional number of sample periods

Variable Integer Delay	Delay input by time-varying integer number of sample periods
Window Function	Compute and/or apply window to input signal
Zero Crossing	Count number of times signal crosses zero in single time step

Signal Processing Sinks

Display	Show value of input
Matrix Viewer	Display matrices as color images
Signal To Workspace	Write simulation data to array in MATLAB workspace
Spectrum Scope	Compute and display periodogram of each input signal
Time Scope	Display time-domain signals
To Audio Device	Write audio data to computer's audio device
To Multimedia File	Write video frames and audio samples to multimedia file
Triggered To Workspace	Write input sample to MATLAB workspace when triggered
UDP Send	Send UDP message
Vector Scope	Display vector or matrix of time-domain, frequency-domain, or user-defined data
Waterfall	View vectors of data over time

Signal Processing Sources

Chirp	Generate swept-frequency cosine (chirp) signal
Constant	Generate constant value
Constant Diagonal Matrix	Generate square, diagonal matrix
Discrete Impulse	Generate discrete impulse
DSP Constant (Obsolete)	Generate discrete- or continuous-time constant signal
From Audio Device	Read audio data from computer's audio device
From Multimedia File	Read multimedia file
Identity Matrix	Generate matrix with ones on main diagonal and zeros elsewhere
Multiphase Clock	Generate multiple binary clock signals
N-Sample Enable	Output ones or zeros for specified number of sample times
Random Source	Generate randomly distributed values
Signal From Workspace	Import signal from MATLAB workspace
Sine Wave	Generate continuous or discrete sine wave
UDP Receive	Receive `uint8` vector as UDP message

Statistics

Autocorrelation	Compute autocorrelation of vector or matrix input
Correlation	Compute cross-correlation of two inputs
Detrend	Remove linear trend from vectors
Histogram	Generate histogram of input or sequence of inputs
Maximum	Find maximum values in input or sequence of inputs
Mean	Find mean value of input or sequence of inputs
Median	Find median value of input
Minimum	Find minimum values in input or sequence of inputs
RMS	Compute root-mean-square value of input or sequence of inputs
Sort	Sort input elements by value
Standard Deviation	Find standard deviation of input or sequence of inputs
Variance	Compute variance of input or sequence of inputs

Transforms

Analytic Signal	Compute analytic signals of discrete-time inputs
Complex Cepstrum	Compute complex cepstrum of input
DCT	Compute discrete cosine transform (DCT) of input

DWT	Discrete wavelet transform (DWT) of input or decompose signals into subbands with smaller bandwidths and slower sample rates
FFT	Compute fast Fourier transform (FFT) of input
IDCT	Compute inverse discrete cosine transform (IDCT) of input
IDWT	Inverse discrete wavelet transform (IDWT) of input or reconstruct signals from subbands with smaller bandwidths and slower sample rates
IFFT	Compute inverse fast Fourier transform (IFFT) of input
Inverse Short-Time FFT	Recover time-domain signals by performing inverse short-time, fast Fourier transform (FFT)
Magnitude FFT	Compute nonparametric estimate of spectrum using periodogram method
Real Cepstrum	Compute real cepstrum of input
Short-Time FFT	Compute nonparametric estimate of spectrum using short-time, fast Fourier transform (FFT) method

2

Blocks — Alphabetical List

Analog Filter Design

Purpose

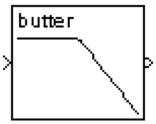
Design and implement analog filters

Library

Filtering / Filter Implementations

`dsparch4`

Description



The Analog Filter Design block designs and implements a Butterworth, Chebyshev type I, Chebyshev type II, or elliptic filter in a highpass, lowpass, bandpass, or bandstop configuration.

The input must be a sample-based, continuous-time, real-valued, scalar signal.

The design and band configuration of the filter are selected from the **Design method** and **Filter type** pop-up menus in the dialog box. For each combination of design method and band configuration, an appropriate set of secondary parameters is displayed.

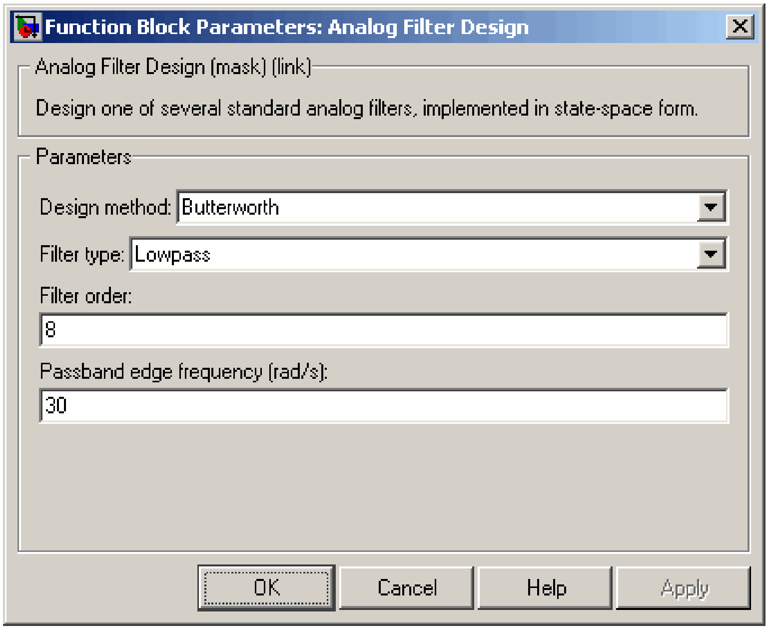
Filter Design	Description
`Butterworth`	The magnitude response of a Butterworth filter is maximally flat in the passband and monotonic overall.
`Chebyshev type I`	The magnitude response of a Chebyshev type I filter is equiripple in the passband and monotonic in the stopband.
`Chebyshev type II`	The magnitude response of a Chebyshev type II filter is monotonic in the passband and equiripple in the stopband.
`Elliptic`	The magnitude response of an elliptic filter is equiripple in both the passband and the stopband.

The following table lists the available parameters for each design/band combination. For lowpass and highpass band configurations, these parameters include the passband edge frequency Ω_p, the stopband edge frequency Ω_s, the passband ripple R_p, and the stopband attenuation R_s. For bandpass and bandstop configurations, the parameters include the lower and upper passband edge frequencies, Ω_{p1} and Ω_{p2}, the lower and upper stopband edge frequencies, Ω_{s1} and Ω_{s2}, the passband ripple R_p, and the stopband attenuation R_s. Frequency values are in rad/s, and ripple and attenuation values are in dB.

	Lowpass	Highpass	Bandpass	Bandstop
Butterworth	Order, Ω_p	Order, Ω_p	Order, Ω_{p1}, Ω_{p2}	Order, Ω_{p1}, Ω_{p2}
Chebyshev Type I	Order, Ω_p, R_p	Order, Ω_p, R_p	Order, Ω_{p1}, Ω_{p2}, R_p	Order, Ω_{p1}, Ω_{p2}, R_p
Chebyshev Type II	Order, Ω_s, R_s	Order, Ω_s, R_s	Order, Ω_{s1}, Ω_{s2}, R_s	Order, Ω_{s1}, Ω_{s2}, R_s
Elliptic	Order, Ω_p, R_p, R_s	Order, Ω_p, R_p, R_s	Order, Ω_{p1}, Ω_{p2}, R_p, R_s	Order, Ω_{p1}, Ω_{p2}, R_p, R_s

The analog filters are designed using the filter design commands in Signal Processing Toolbox™ software's `buttap`, `cheb1ap`, `cheb2ap`, and `ellipap` functions, and are implemented in state-space form. Filters of order 8 or less are implemented in controller canonical form for improved efficiency.

Dialog Box



The parameters displayed in the dialog box vary for different design/band combinations. Only some of the parameters listed below are visible in the dialog box at any one time.

Design method

The filter design method: `Butterworth`, `Chebyshev type I`, `Chebyshev type II`, or `Elliptic`. Tunable.

Filter type

The type of filter to design: `Lowpass`, `Highpass`, `Bandpass`, or `Bandstop`. Tunable.

Filter order

The order of the filter, for lowpass and highpass configurations. For bandpass and bandstop configurations, the order of the final filter is *twice* this value.

Passband edge frequency
The passband edge frequency, in rad/s, for the highpass and lowpass configurations of the Butterworth, Chebyshev type I, and elliptic designs. Tunable.

Lower passband edge frequency
The lower passband frequency, in rad/s, for the bandpass and bandstop configurations of the Butterworth, Chebyshev type I, and elliptic designs. Tunable.

Upper passband edge frequency
The upper passband frequency, in rad/s, for the bandpass and bandstop configurations of the Butterworth, Chebyshev type I, or elliptic designs. Tunable.

Stopband edge frequency
The stopband edge frequency, in rad/s, for the highpass and lowpass band configurations of the Chebyshev type II design. Tunable.

Lower stopband edge frequency
The lower stopband edge frequency, in rad/s, for the bandpass and bandstop configurations of the Chebyshev type II design. Tunable.

Upper stopband edge frequency
The upper stopband edge frequency, in rad/s, for the bandpass and bandstop filter configurations of the Chebyshev type II design. Tunable.

Passband ripple in dB
The passband ripple, in dB, for the Chebyshev Type I and elliptic designs. Tunable.

Stopband attenuation in dB
The stopband attenuation, in dB, for the Chebyshev Type II and elliptic designs. Tunable.

References

Antoniou, A. *Digital Filters: Analysis, Design, and Applications*. 2nd ed. New York, NY: McGraw-Hill, 1993.

Supported Data Types

- Double-precision floating point

See Also

Digital Filter Design	Signal Processing Blockset
`buttap`	Signal Processing Toolbox
`cheb1ap`	Signal Processing Toolbox
`cheb2ap`	Signal Processing Toolbox
`ellipap`	Signal Processing Toolbox

See the following sections for related information:

- "Filters"
- "Analog Filter Design Block"

Purpose

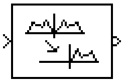
Compute analytic signals of discrete-time inputs

Library

Transforms

`dspxfrm3`

Description



The Analytic Signal block computes the complex analytic signal corresponding to each channel of the real *M*-by-*N* input, *u*

$$y = u + jH\{u\}$$

where $j = \sqrt{-1}$ and $H\{\}$ denotes the Hilbert transform. The real part of the output in each channel is a replica of the real input in that channel; the imaginary part is the Hilbert transform of the input. In the frequency domain, the Fourier transform of the analytic signal doubles the positive frequency content of the original signal while zeroing-out negative frequencies and retaining the DC component.

The block computes the Hilbert transform using an equiripple FIR filter with the order specified by the **Filter order** parameter, *n*. The linear phase filter is designed using the Remez exchange algorithm, and imposes a delay of *n*/2 on the input samples.

The output has the same dimension and frame status as the input.

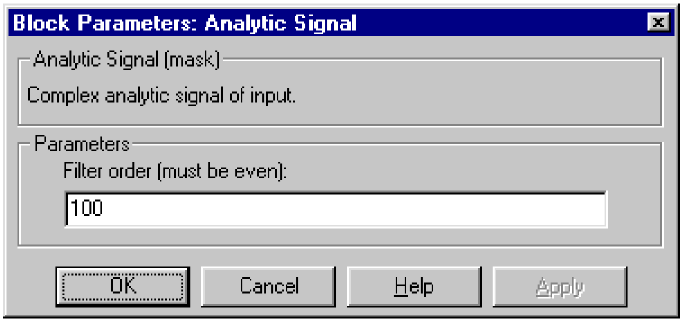
Sample-Based Operation

When the input is sample based, each of the *M*N* matrix elements represents an independent channel. Thus, the block computes the analytic signal for each channel (matrix element) over time.

Frame-Based Operation

When the input is frame based, each of the *N* columns in the matrix contains *M* sequential time samples from an independent channel, and the block computes the analytic signal for each channel over time.

Dialog Box



Filter order

The length of the FIR filter used to compute the Hilbert transform.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

Purpose

Design arbitrary response filter

Library

Filtering / Filter Designs

`dspfdesign`

Description

This block brings the filter design capabilities of the filterbuilder function to the Simulink® environment. Without a Filter Design Toolbox™ license, you can run models that contain this block, and can edit some, but not all, block parameters. To enable the full filter design functionality of this block, you must have a Filter Design Toolbox license.

Dialog Box

See "Arbitrary Response Design Dialog Box — Main Pane" in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset™ Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Purpose

Add vector to array along specified dimension

Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description

The Array-Vector Add block adds the values in the specified dimension of the N-dimensional input array *A* to the values in the input vector *V*.

The length of the input *V* must be the same as the length of the specified dimension of *A*. The Array-Vector Add block adds each element of *V* to the corresponding element along that dimension of *A*.

Consider a 3-dimensional *M*-by-*N*-by-*P* input array *A(i,j,k)* and a *P*-by-1 input vector *V*. When the **Add along dimension** parameter is set to `2`, the output of the block *Y(i,j,k)* is

$$Y(i,j,k) = A(i,j,k) + V(j)$$

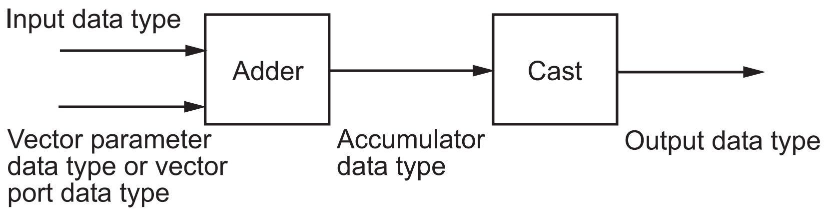
where

$$1 \le i \le M$$
$$1 \le j \le N$$
$$1 \le k \le P$$

The output of the Array-Vector Add block is the same size as the input array, *A*. When both inputs are sample based, the output is sample based; otherwise, the output is frame based. This block accepts real and complex floating-point and fixed-point inputs.

Fixed-Point Data Types

The following diagram shows the data types used within the Array-Vector Add block for fixed-point signals.

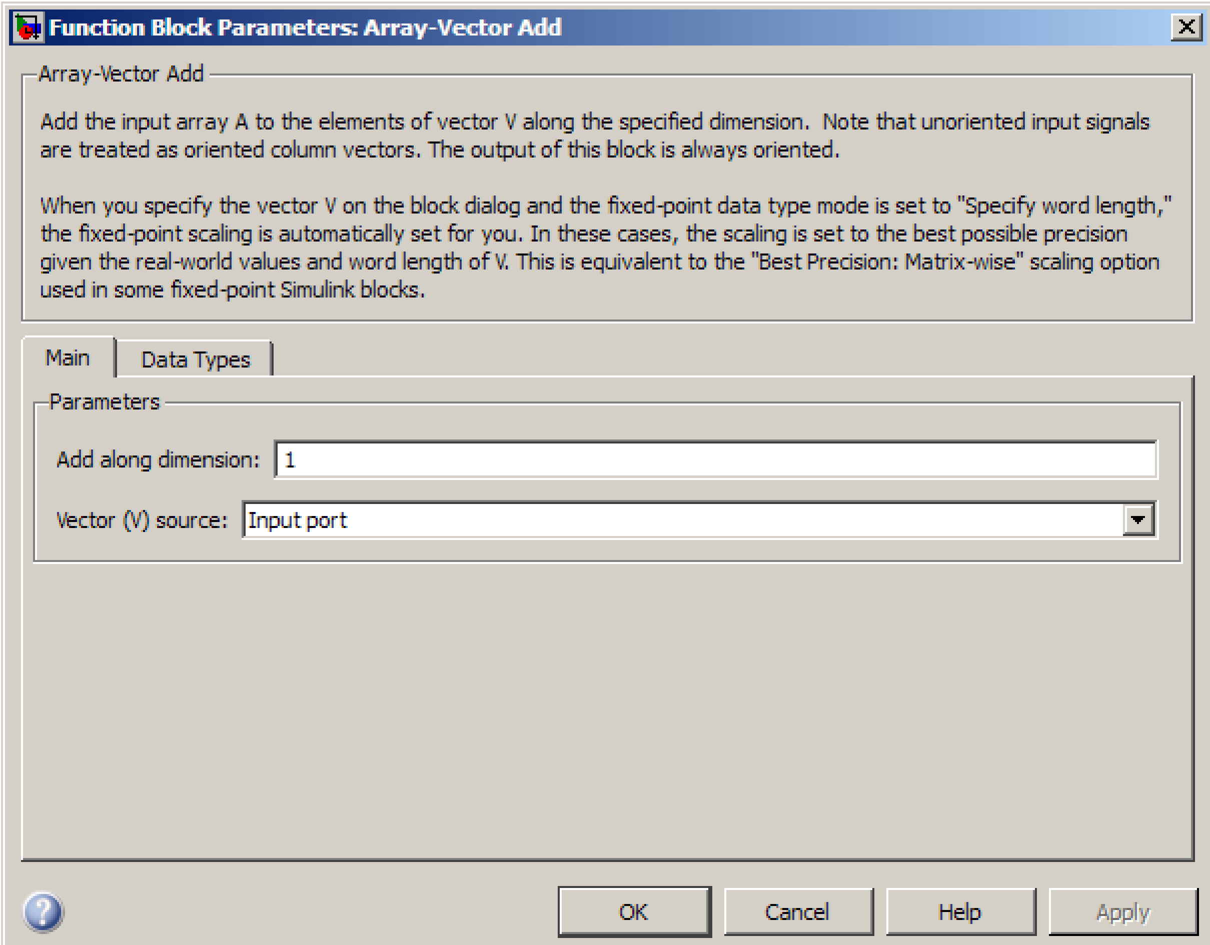


When you specify the vector *V* on the **Main** pane of the block mask, you must specify the data type and scaling properties of its elements in the **Vector (V)** parameter on the **Data Types** tab. When the vector comes in through the block port, its elements inherit their data type and scaling from the driving block.

You can set the vector, accumulator, and output data types in the block dialog as discussed below.

Dialog Box

The **Main** pane of the Array-Vector Add block dialog appears as follows.



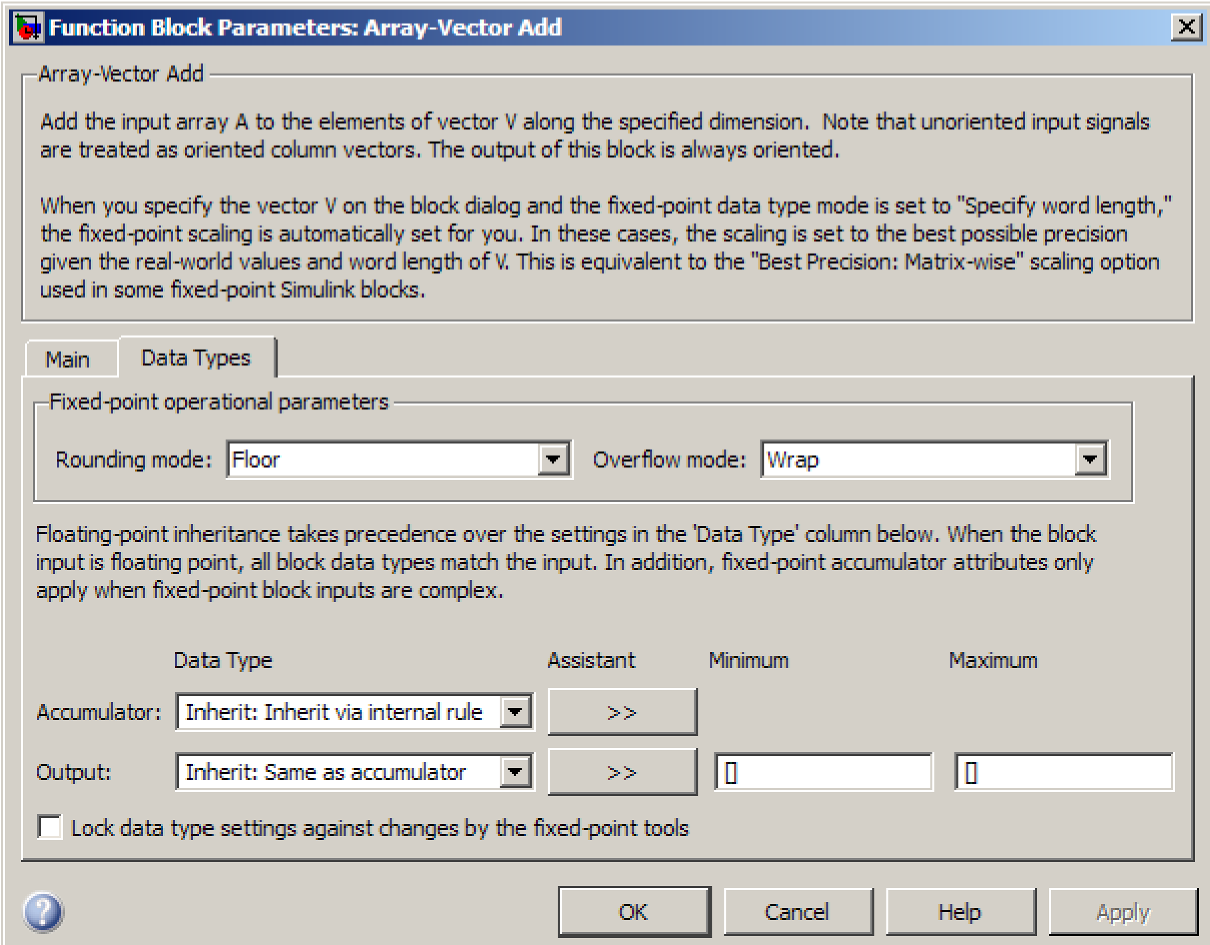
Add along dimension

Specify the dimension along which to add the input array *A* to the elements of vector *V*.

Vector (V) source
Specify the source of the vector, *V*. The vector can come from the `Input port` or from a `Dialog parameter`.

Vector (V)
Specify the vector, *V*. This parameter is visible only when you select `Dialog parameter` for the **Vector (V) source** parameter.

The **Data Types** pane of the Array-Vector Add block dialog appears as follows.



Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Rounding mode
: Select the rounding mode for fixed-point operations.

Overflow mode
: Select the overflow mode for fixed-point operations.

Vector (V)
: Use this parameter to specify the word and fraction lengths for the elements of the vector, *V*. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same word length as input`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Note The **Vector (V)** parameter on the **Data Types** pane is only visible when you select `Dialog parameter` for the **Vector (V) source** parameter on the **Main** pane of the block mask. When the vector comes in through the block's input port, the data type and scaling of its elements are inherited from the driving block.

Accumulator data type
: Specify the accumulator data type. See "Fixed-Point Data Types" on page 2-11 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Output data type

Specify the output data type. See “Fixed-Point Data Types” on page 2-11 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as accumulator`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User’s Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see “Checking Signal Ranges”)
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
A	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
V	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

Array-Vector Divide	Signal Processing Blockset
Array-Vector Multiply	Signal Processing Blockset
Array-Vector Subtract	Signal Processing Blockset

Array-Vector Divide

Purpose

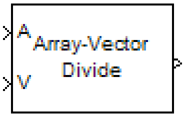
Divide array by vector along specified dimension

Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description



The Array-Vector Divide block divides the values in the specified dimension of the N-dimensional input array *A* by the values in the input vector *V*.

The length of the input *V* must be the same as the length of the specified dimension of *A*. The Array-Vector Divide block divides each element of *V* by the corresponding element along that dimension of *A*.

Consider a 3-dimensional *M*-by-*N*-by-*P* input array *A(i,j,k)* and a *P*-by-1 input vector *V*. When the **Divide along dimension** parameter is set to `2`, the output of the block *Y(i,j,k)* is

$$Y(i,j,k) = \frac{A(i,j,k)}{V(j)}$$

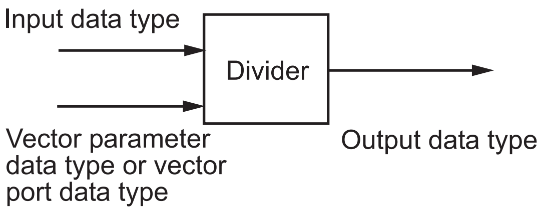
where

$$1 \le i \le M$$
$$1 \le j \le N$$
$$1 \le k \le P$$

The output of the Array-Vector Divide block is the same size as the input array, *A*. When both inputs are sample based, the output is sample based; otherwise, the output is frame based. This block accepts real and complex floating-point and fixed-point input arrays, and real floating-point and fixed-point input vectors.

Fixed-Point Data Types

The following diagram shows the data types used within the Array-Vector Divide block for fixed-point signals.

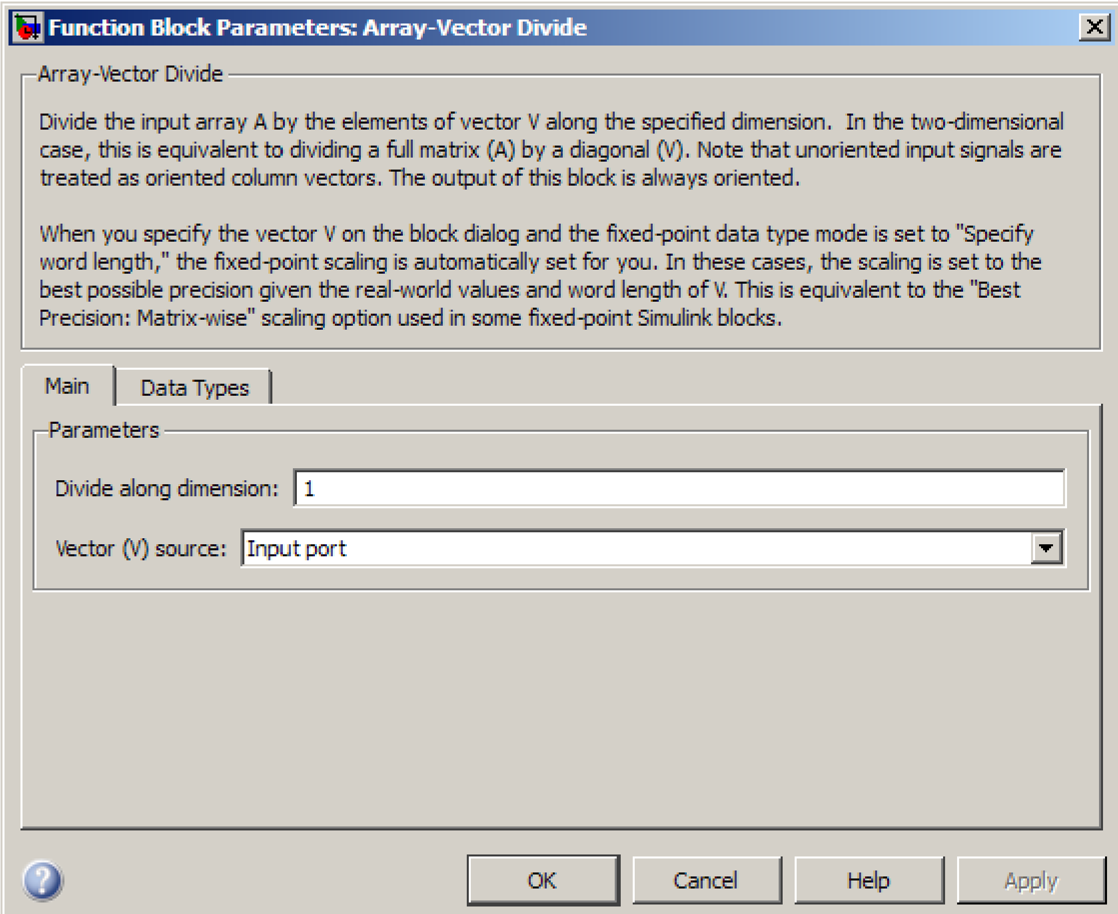


When you specify the vector *V* on the **Main** pane of the block mask, you must specify the data type and scaling properties of its elements in the **Vector (V)** parameter on the **Data Types** tab. When the vector comes in through the block port, its elements inherit their data type and scaling from the driving block.

You can set the vector and output data types in the block dialog as discussed below.

Dialog Box

The **Main** pane of the Array-Vector Divide block dialog appears as follows.



Divide along dimension

Specify the dimension along which to divide the input array *A* by the elements of vector *V*.

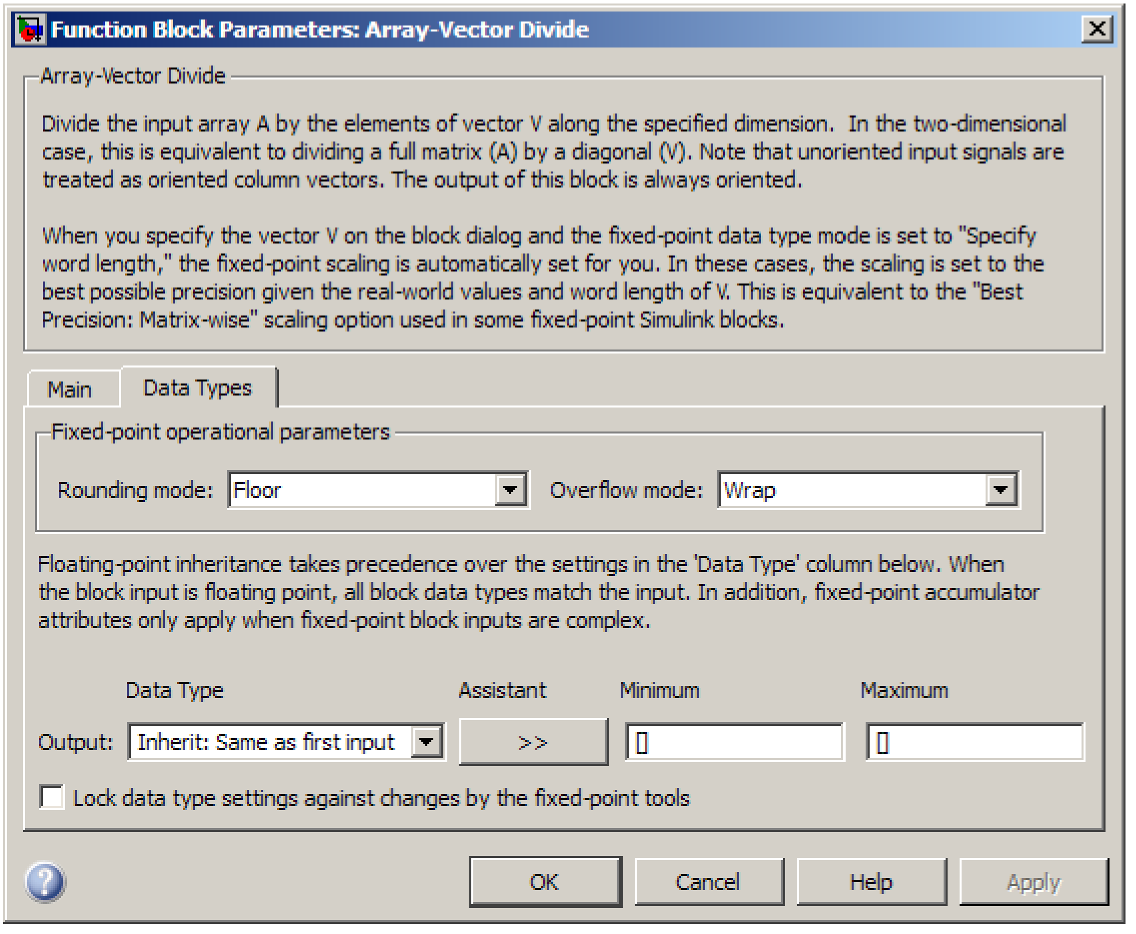
Vector (V) source

Specify the source of the vector, *V*. The vector can come from the `Input port` or from a `Dialog parameter`.

Vector (V)

Specify the vector, *V*. This parameter is visible only when you select `Dialog parameter` for the **Vector (V) source** parameter.

The **Data Types** pane of the Array-Vector Divide block dialog appears as follows.



Array-Vector Divide

Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Rounding mode
: Select the rounding mode for fixed-point operations.

Overflow mode
: Select the overflow mode for fixed-point operations.

Vector (V)
: Use this parameter to specify the word and fraction lengths for the elements of the vector, *V*. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same word length as input`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Note The **Vector (V)** parameter on the **Data Types** pane is only visible when you select `Dialog parameter` for the **Vector (V) source** parameter on the **Main** pane of the block mask. When the vector comes in through the block's input port, the data type and scaling of its elements are inherited from the driving block.

Output data type

Specify the output data type. See “Fixed-Point Data Types” on page 2-20 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as first input`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User’s Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see “Checking Signal Ranges”)
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see “Checking Signal Ranges”)
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
A	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
V	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

Array-Vector Add	Signal Processing Blockset
Array-Vector Multiply	Signal Processing Blockset
Array-Vector Subtract	Signal Processing Blockset

Purpose

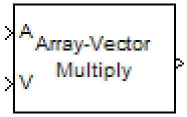
Multiply array by vector along specified dimension

Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description



The Array-Vector Multiply block multiplies the values in the specified dimension of the N-dimensional input array *A* by the values in the input vector *V*.

The length of the input *V* must be the same as the length of the specified dimension of *A*. The Array-Vector Multiply block multiplies each element of *V* by the corresponding element along that dimension of *A*.

Consider a 3-dimensional *M*-by-*N*-by-*P* input array *A(i,j,k)* and a *P*-by-1 input vector *V*. When the **Multiply along dimension** parameter is set to `2`, the output of the block *Y(i,j,k)* is

$$Y(i,j,k) = A(i,j,k) * V(j)$$

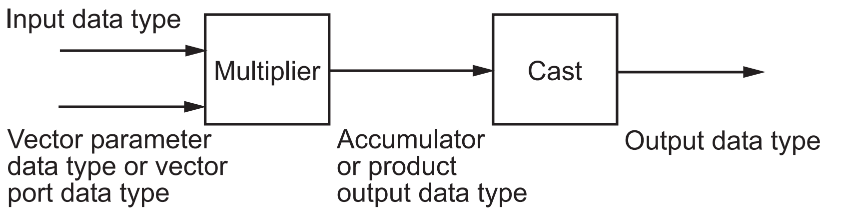
where

$$1 \le i \le M$$
$$1 \le j \le N$$
$$1 \le k \le P$$

The output of the Array-Vector Multiply block is the same size as the input array, *A*. When both inputs are sample based, the output is sample based; otherwise, the output is frame based. This block accepts real and complex floating-point and fixed-point inputs.

Fixed-Point Data Types

The following diagram shows the data types used within the Array-Vector Multiply block for fixed-point signals.



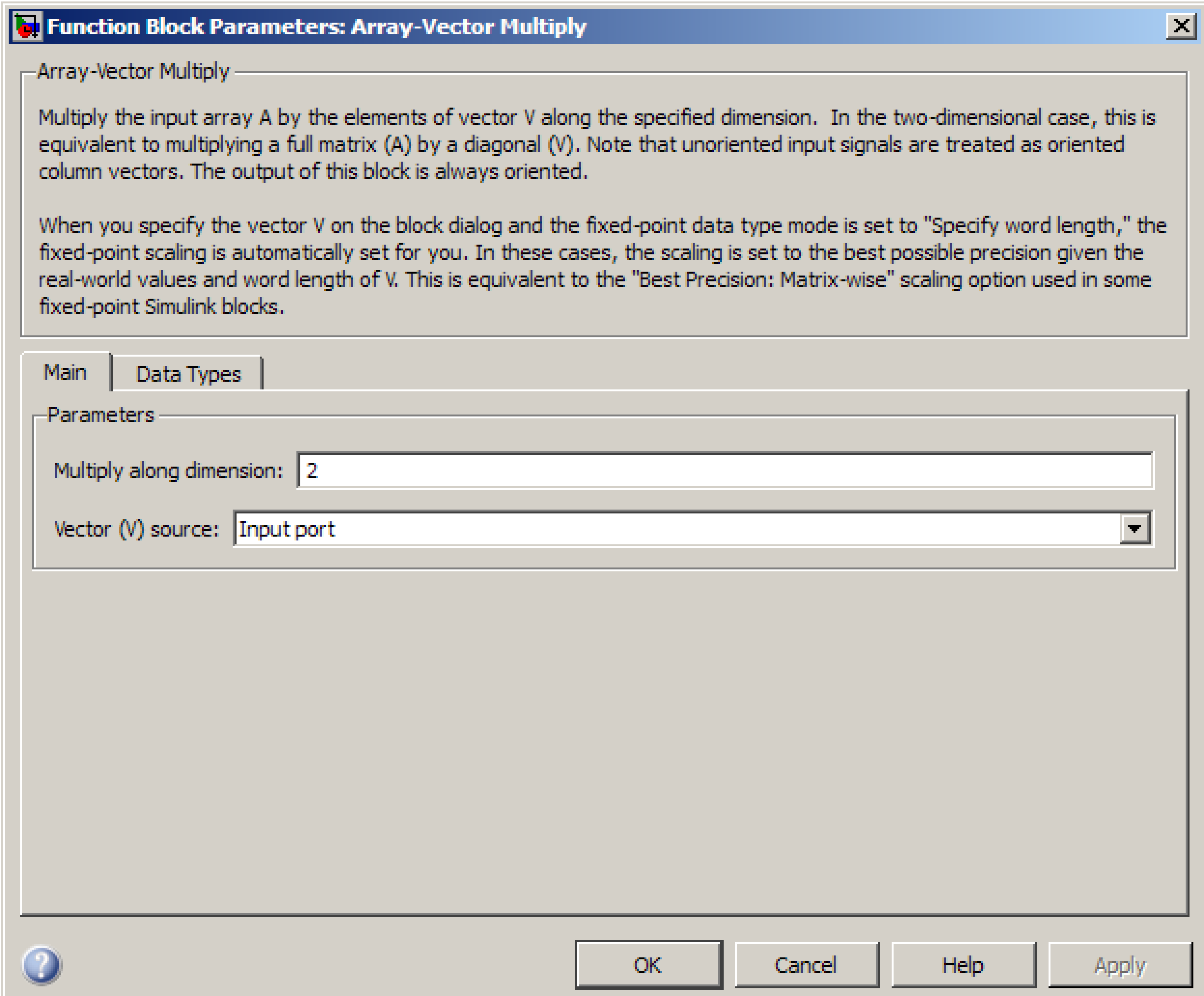
When you specify the vector *V* on the **Main** pane of the block mask, you must specify the data type and scaling properties of its elements in the **Vector (V)** parameter on the **Data Types** tab. When the vector comes in through the block port, its elements inherit their data type and scaling from the driving block.

The output of the multiplier is in the product output data type when at least one of the inputs to the multiplier is real. When both of the inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see "Multiplication Data Types".

You can set the vector, accumulator, product output, and output data types in the block dialog as discussed below.

Dialog Box

The **Main** pane of the Array-Vector Multiply block dialog appears as follows.



Multiply along dimension

Specify the dimension along which to multiply the input array *A* by the elements of vector *V*.

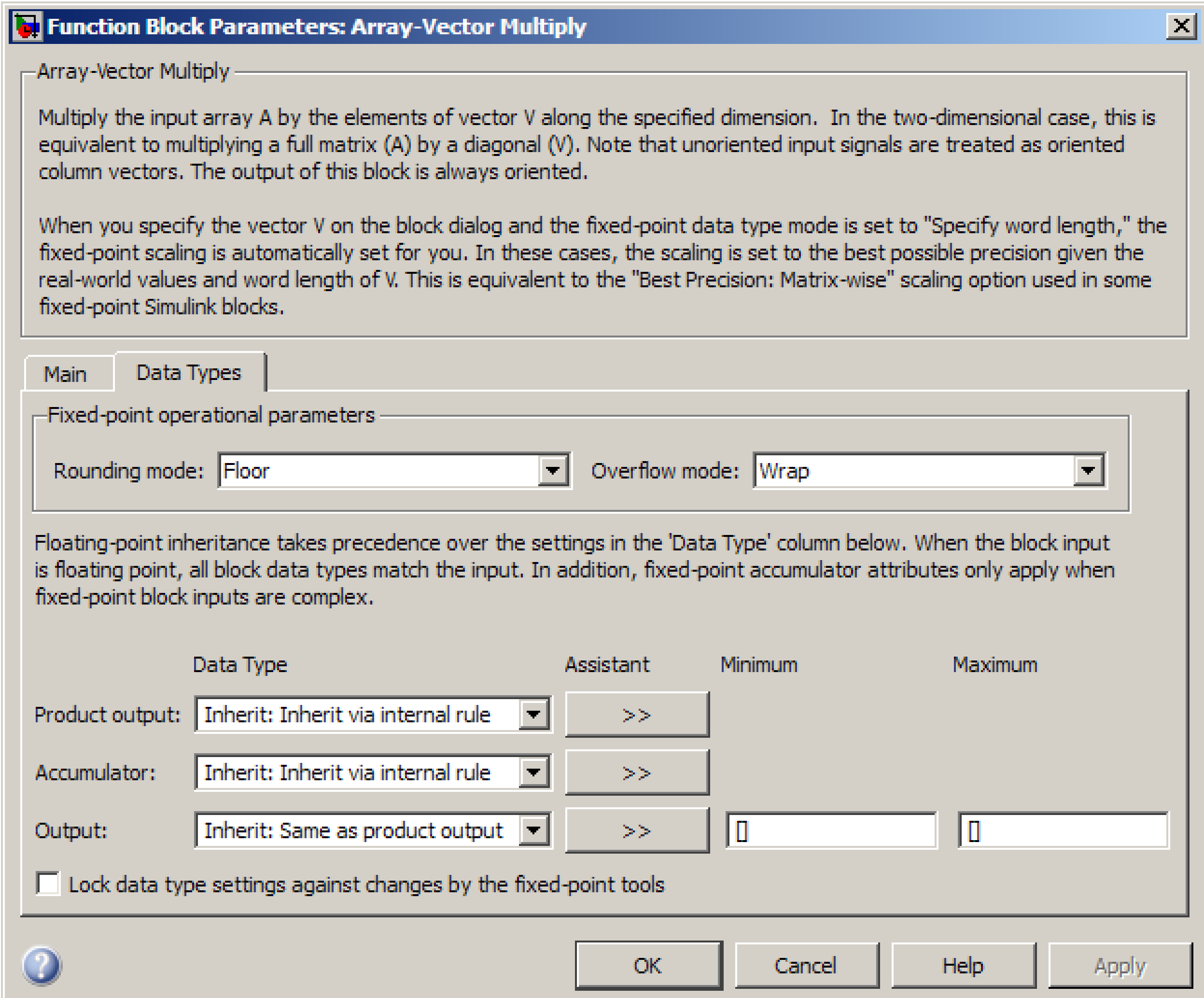
Vector (V) source

Specify the source of the vector, *V*. The vector can come from the `Input port` or from a `Dialog parameter`.

Vector (V)

Specify the vector, *V*. This parameter is visible only when you select `Dialog parameter` for the **Vector (V) source** parameter.

The **Data Types** pane of the Array-Vector Multiply block dialog appears as follows.


Function Block Parameters: Array-Vector Multiply
Array-Vector Multiply
Multiply the input array A by the elements of vector V along the specified dimension. In the two-dimensional case, this is equivalent to multiplying a full matrix (A) by a diagonal (V). Note that unoriented input signals are treated as oriented column vectors. The output of this block is always oriented.
When you specify the vector V on the block dialog and the fixed-point data type mode is set to "Specify word length," the fixed-point scaling is automatically set for you. In these cases, the scaling is set to the best possible precision given the real-world values and word length of V. This is equivalent to the "Best Precision: Matrix-wise" scaling option used in some fixed-point Simulink blocks.
Main
Data Types
Fixed-point operational parameters
Rounding mode:
Floor
Overflow mode:
Wrap
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input. In addition, fixed-point accumulator attributes only apply when fixed-point block inputs are complex.
Data Type
Assistant
Minimum
Maximum
Product output:
Inherit: Inherit via internal rule
Accumulator:
Inherit: Inherit via internal rule
Output:
Inherit: Same as product output
[]
[]
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Rounding mode
: Select the rounding mode for fixed-point operations.

Overflow mode
: Select the overflow mode for fixed-point operations.

Vector (V)
: Use this parameter to specify the word and fraction lengths for the elements of the vector, *V*. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same word length as input`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Note The **Vector (V)** parameter on the **Data Types** pane is only visible when you select `Dialog parameter` for the **Vector (V) source** parameter on the **Main** pane of the block mask. When the vector comes in through the block's input port, the data type and scaling of its elements are inherited from the driving block.

Product output data type

Specify the product output data type. See “Fixed-Point Data Types” on page 2-27 and “Multiplication Data Types” for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Accumulator data type

Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-27 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Output data type

Specify the output data type. See “Fixed-Point Data Types” on page 2-27 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as product output`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User’s Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see “Checking Signal Ranges”)
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see “Checking Signal Ranges”)
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
A	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
V	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

Array-Vector Add	Signal Processing Blockset
Array-Vector Divide	Signal Processing Blockset
Array-Vector Subtract	Signal Processing Blockset

Array-Vector Subtract

Purpose

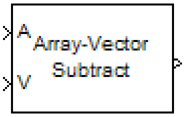
Subtract vector from array along specified dimension

Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description



The Array-Vector Subtract block subtracts the values in the input vector *V* from the values in the specified dimension of the N-dimensional input array *A*.

The length of the input *V* must be the same as the length of the specified dimension of *A*. The Array-Vector Subtract block subtracts each element of *V* from the corresponding element along that dimension of *A*.

Consider a 3-dimensional *M*-by-*N*-by-*P* input array *A(i,j,k)* and a *P*-by-1 input vector *V*. When the **Subtract along dimension** parameter is set to `2`, the output of the block *Y(i,j,k)* is

$$Y(i,j,k) = A(i,j,k) - V(j)$$

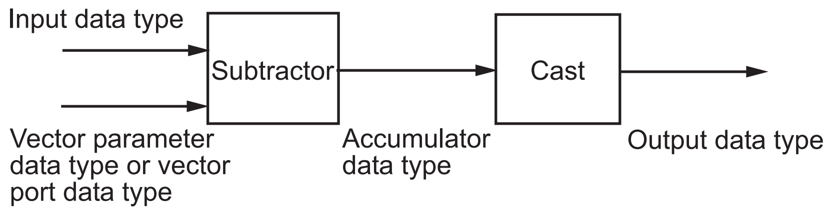
where

$$1 \le i \le M$$
$$1 \le j \le N$$
$$1 \le k \le P$$

The output of the Array-Vector Subtract block is the same size as the input array, *A*. When both inputs are sample based, the output is sample based; otherwise, the output is frame based. This block accepts real and complex floating-point and fixed-point inputs.

Fixed-Point Data Types

The following diagram shows the data types used within the Array-Vector Subtract block for fixed-point signals.



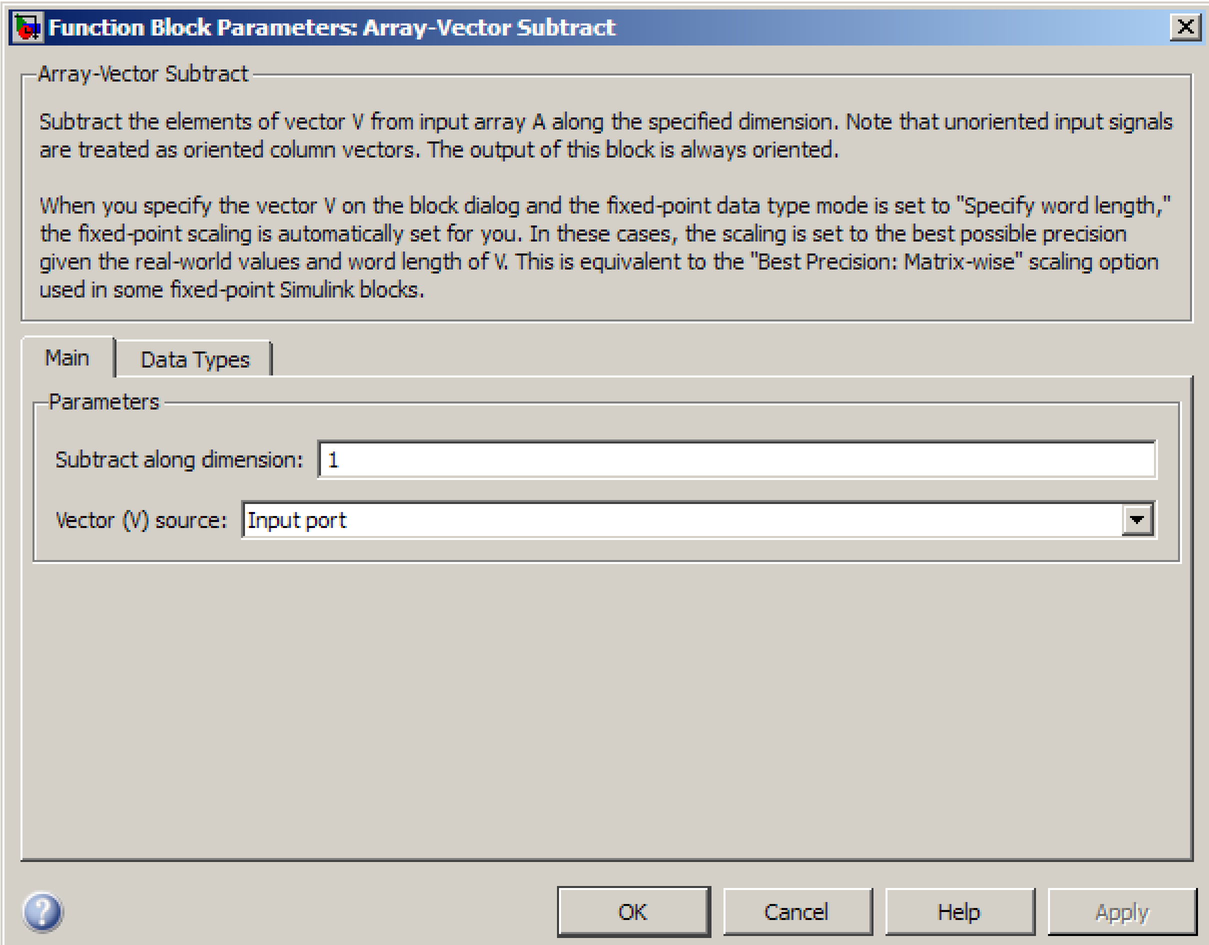
When you specify the vector *V* on the **Main** pane of the block mask, you must specify the data type and scaling properties of its elements in the **Vector (V)** parameter on the **Data Types** tab. When the vector comes in through the block port, its elements inherit their data type and scaling from the driving block.

The output of the subtractor is in the accumulator data type.

You can set the vector, accumulator, and output data types in the block dialog as discussed below.

Dialog Box

The **Main** pane of the Array-Vector Subtract block dialog appears as follows.



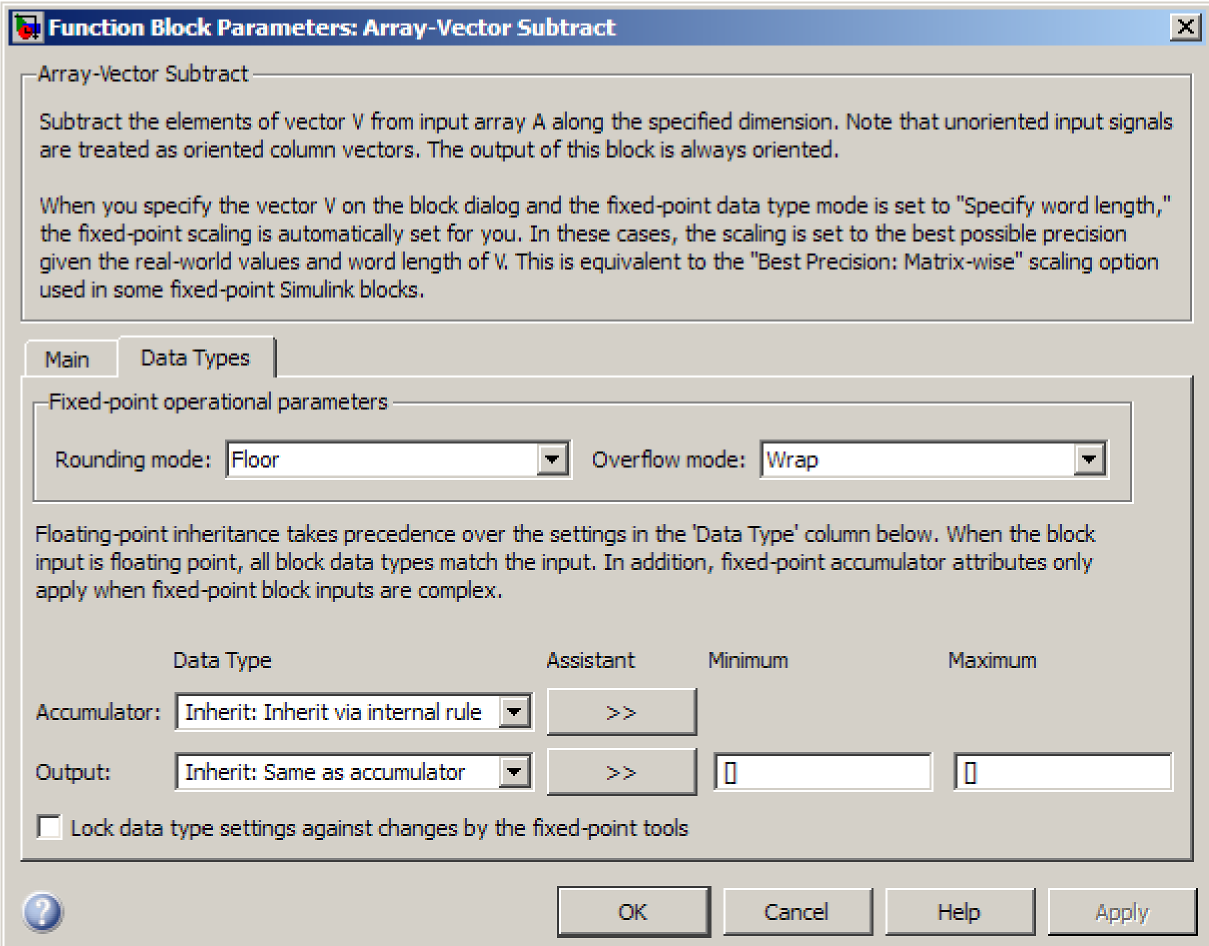
Subtract along dimension

Specify the dimension along which to subtract the elements of vector *V* from the input array *A*.

Vector (V) source
: Specify the source of the vector, *V*. The vector can come from the `Input port` or from a `Dialog parameter`.

Vector (V)
: Specify the vector, *V*. This parameter is visible only when you select `Dialog parameter` for the **Vector (V) source** parameter.

The **Data Types** pane of the Array-Vector Subtract block dialog appears as follows.



Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Rounding mode
Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode for fixed-point operations.

Vector (V)
Use this parameter to specify the word and fraction lengths for the elements of the vector, *V*. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same word length as input`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Note The **Vector (V)** parameter on the **Data Types** pane is only visible when you select `Dialog parameter` for the **Vector (V) source** parameter on the **Main** pane of the block mask. When the vector comes in through the block’s input port, the data type and scaling of its elements are inherited from the driving block.

Accumulator data type
Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-36 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Output data type

Specify the output data type. See "Fixed-Point Data Types" on page 2-36 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as accumulator`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
A	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
V	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

Array-Vector Add	Signal Processing Blockset
Array-Vector Divide	Signal Processing Blockset
Array-Vector Multiply	Signal Processing Blockset

Purpose

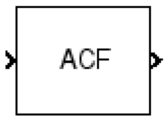
Compute autocorrelation of vector or matrix input

Library

Statistics

`dspstat3`

Description



The Autocorrelation block computes the autocorrelation along each column of a frame-based input, and computes along the first dimension of an N-D sample-based input. The output of the block is always sample-based.

The Autocorrelation block accepts real and complex floating-point and fixed-point inputs except for complex unsigned fixed-point inputs. Fixed-point signals are not supported for the frequency domain.

Autocorrelation of Frame-Based Inputs

When the input to the Autocorrelation block is a frame-based M-by-N matrix u, the output, y, is a sample-based (l+1)-by-N matrix whose jth column has elements

$$y_{i,j} = \sum_{k=0}^{M-l-1} u^*_{k,j} u_{(k+i),j} \qquad 0 \le i \le l$$

where * denotes the complex conjugate, and l represents the maximum lag. $y_{0,j}$ is the zero-lag element in the jth column. When you select **Compute all non-negative lags**, l=M–1. Otherwise, l is specified as a nonnegative integer by the **Maximum non-negative lag (less than input length)** parameter.

Input u is zero when indexed outside of its valid range. When the input is real, the output is real; otherwise, the output is complex.

Autocorrelation of Sample-Based Inputs

When the input is a sample-based N-D array, the block computes the autocorrelation along the first dimension of the input. The output is a sample based N-D array, where the size of the first dimension is l+1, and the sizes of all other dimensions match those of the input array. For example, when the input is an M-by-N-by-P array, the Autocorrelation block outputs an $(l+1)$-by-N-by-P sample-based array.

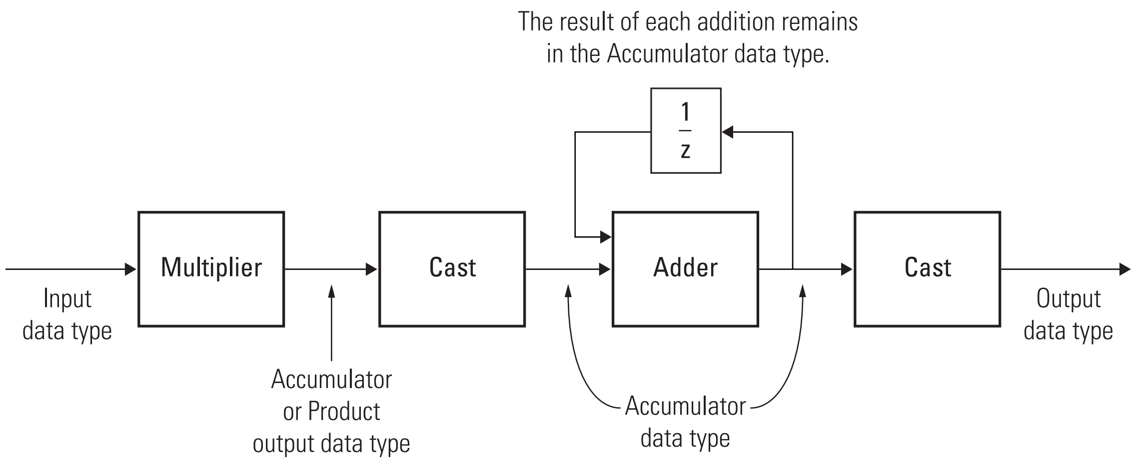
When you select **Compute all non-negative lags**, $l=M-1$. Otherwise, l is specified as a nonnegative integer by the **Maximum non-negative lag (less than input length)** parameter.

Input u is zero when indexed outside of its valid range. When the input is real, the output is real; otherwise, the output is complex.

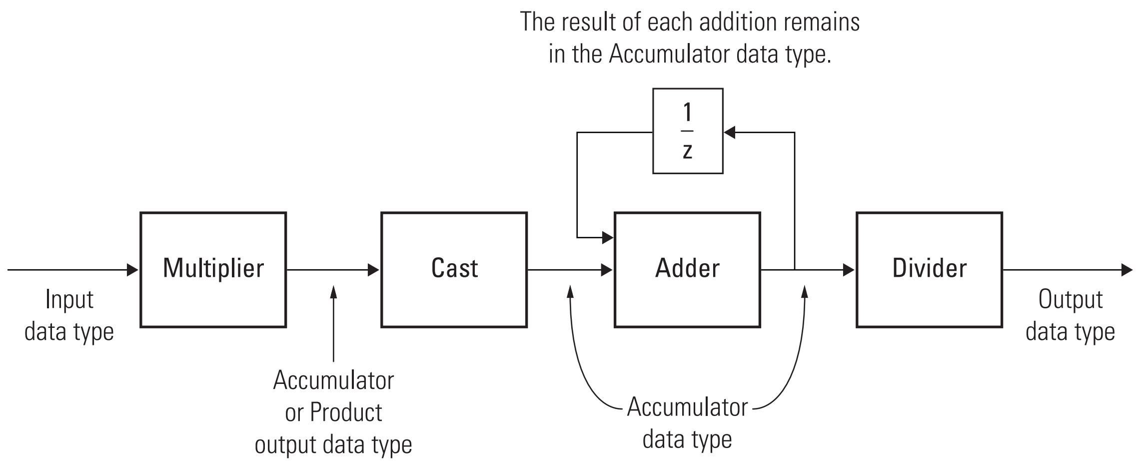
Fixed-Point Data Types

The following diagrams show the data types used within the Autocorrelation block for fixed-point signals (time domain only).

Signal flow when Scaling is "None"



Signal flow when Scaling is other than "None"

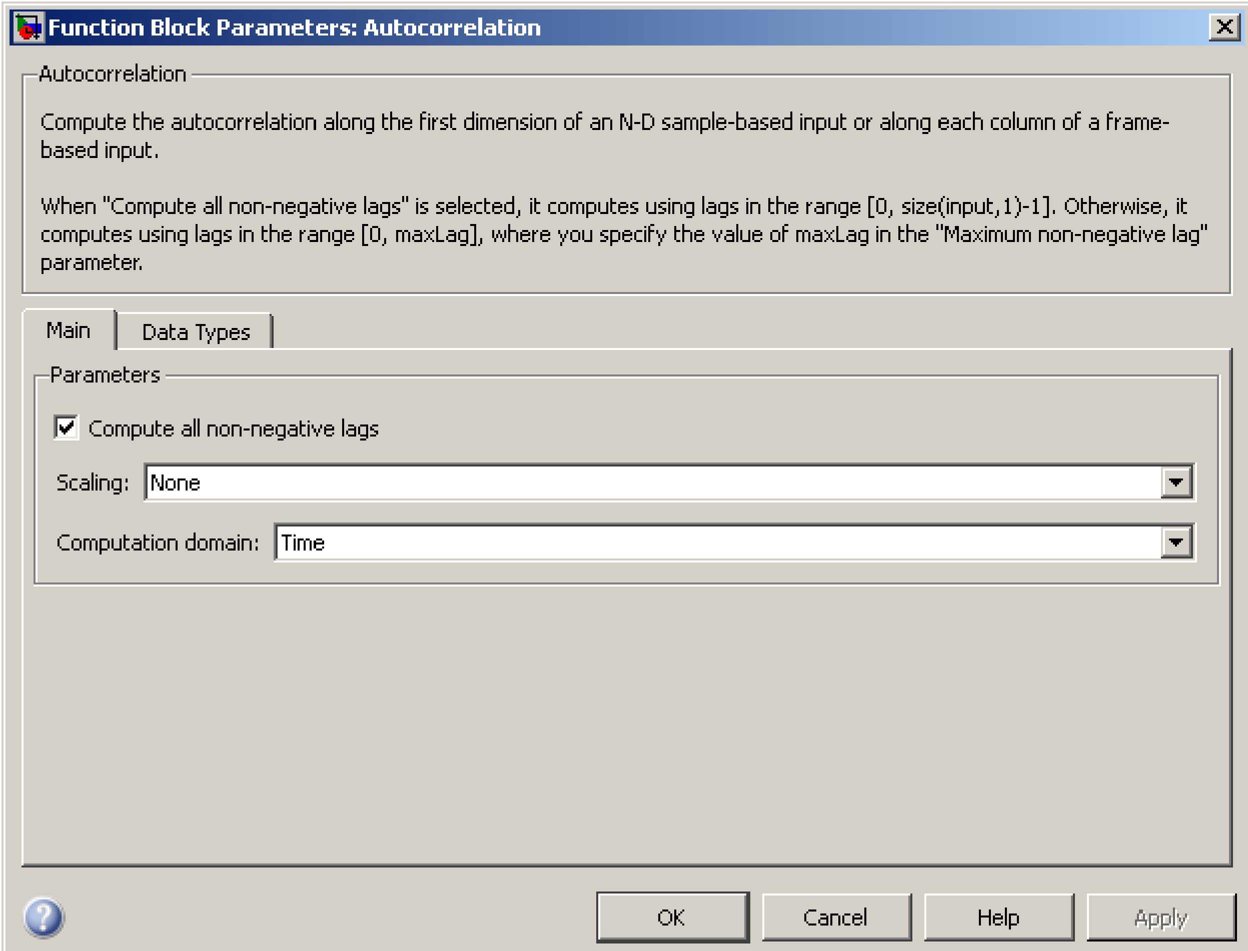


You can set the product output, accumulator, and output data types on the **Data Types** pane of the block dialog as discussed in the next section.

The output of the multiplier is in the product output data type when the input is real. When the input is complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see "Multiplication Data Types".

Dialog Box

The **Main** pane of the Autocorrelation block dialog appears as follows.



Compute all non-negative lags

Select to compute the autocorrelation over all nonnegative lags in the range [0, `length(input)-1`].

Maximum non-negative lag (less than input length)

Specify the maximum positive lag, l, for the autocorrelation. This parameter is enabled when you do not select the **Compute all non-negative lags** check box.

Scaling

This parameter controls the scaling that is applied to the output. The following options are available:

- `None` — Generates the raw autocorrelation $y_{i,j}$ without normalization.
- `Biased` — Generates the biased estimate of the autocorrelation.

$$y_{i,j}^{biased} = \frac{y_{i,j}}{M}$$

- `Unbiased` — Generates the unbiased estimate of the autocorrelation.

$$y_{i,j}^{unbiased} = \frac{y_{i,j}}{M-i}$$

- `Unity at zero-lag` — Normalizes the estimate of the autocorrelation for each channel so that the zero-lag sum is identically 1.

$$y_{0,j} = 1$$

This parameter is tunable, except in the Simulink external mode.

Computation domain

This parameter sets the domain in which the block computes convolutions to one of the following settings:

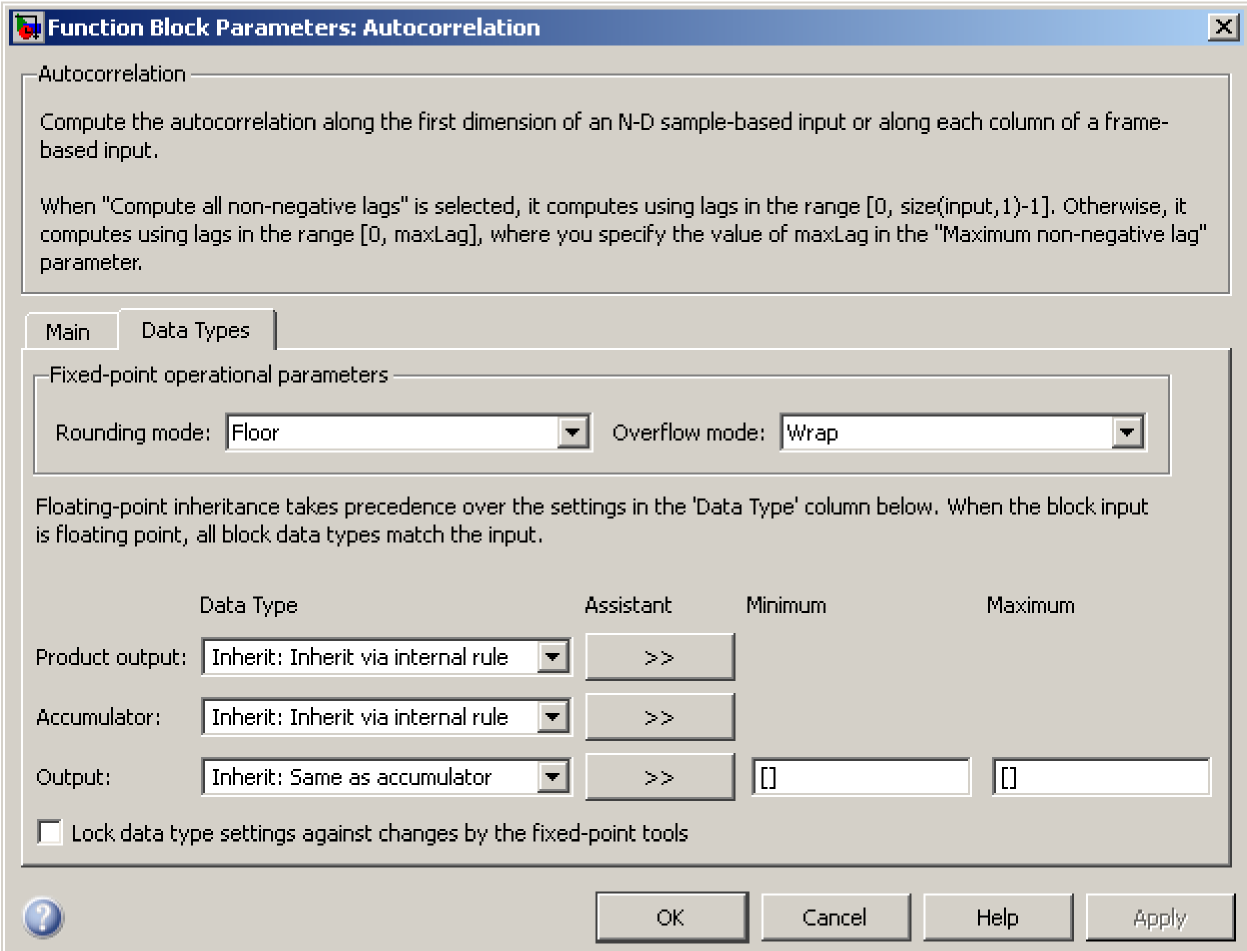
- `Time` — Computes in the time domain, which minimizes memory use

- `Frequency` — Computes in the frequency domain, which might require fewer computations than computing in the time domain, depending on the input length

---

Note This parameter must be set to `Time` for fixed-point signals.

The **Data Types** pane of the Autocorrelation block dialog appears as follows.



Note Fixed-point signals are only supported for the time domain. To use the parameters on this pane, make sure `Time` is selected for the **Computation domain** parameter on the **Main** pane.

Rounding mode

Select the rounding mode for fixed-point operations.

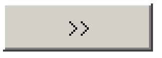
Overflow mode

Select the overflow mode for fixed-point operations.

Product output data type

Specify the product output data type. See "Fixed-Point Data Types" on page 2-46 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Accumulator data type

Specify the accumulator data type. See "Fixed-Point Data Types" on page 2-46 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Output data type

Specify the output data type. See "Fixed-Point Data Types" on page 2-46 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as accumulator`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Correlation	Signal Processing Blockset
`xcorr`	Signal Processing Toolbox

Purpose

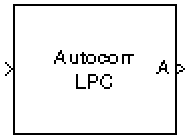
Determine coefficients of Nth-order forward linear predictors

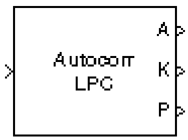
Library

Estimation / Linear Prediction

`dsplp`

Description





The Autocorrelation LPC block determines the coefficients of an *N-step forward linear predictor* for the time-series in each length-M input channel, u, by minimizing the prediction error in the least squares sense. A linear predictor is an FIR filter that predicts the next value in a sequence from the present and past inputs. This technique has applications in filter design, speech coding, spectral analysis, and system identification.

The Autocorrelation LPC block can output the prediction error for each channel as polynomial coefficients, reflection coefficients, or both. It can also output the prediction error power for each channel. The input u can be a scalar, 1-D vector, frame- or sample-based column vector, a sample-based row vector, or a channel-based matrix. Frame-based row vectors are not valid inputs. Frame-based matrices of size M-by-N are treated as N channels of length M.

When you select **Inherit prediction order from input dimensions**, the prediction order, N, is inherited from the input dimensions. Otherwise, you can use the **Prediction order** parameter to specify the value of N. Note that N must be a scalar with a value less than the length of the input channels or the block produces an error.

When **Output(s)** is set to `A`, port A is enabled. For each channel, port A outputs an (N+1)-by-1 column vector, $a = [1\ a_2\ a_3\ \ldots\ a_{N+1}]^T$, containing the coefficients of an Nth-order moving average (MA) linear process that predicts the next value, $\hat{u}_{M+1}$, in the input time-series.

$$\hat{u}_{M+1} = -(a_2 u_M) - (a_3 u_{M-1}) - \ldots - (a_{N+1} u_{M-N+1})$$

When **Output(s)** is set to `K`, port K is enabled. For each channel, port K outputs a length-N column vector whose elements are the prediction error reflection coefficients. When **Output(s)** is set to `A and K`, both port A and K are enabled, and each port outputs its respective set of prediction coefficients for each channel.

When you select **Output prediction error power (P)**, port P is enabled. The prediction error power is output at port P as a vector whose length is the number of input channels.

Algorithm

The Autocorrelation LPC block computes the least squares solution to

$$\min_{i \in \Re^n} \| U\tilde{a} - b \|$$

where $\|\cdot\|$ indicates the 2-norm and

$$U = \begin{bmatrix} u_1 & 0 & \cdots & 0 \\ u_2 & u_1 & \ddots & \vdots \\ \vdots & u_2 & \ddots & 0 \\ \vdots & \vdots & \ddots & u_1 \\ \vdots & \vdots & \vdots & u_2 \\ \vdots & \vdots & \vdots & \vdots \\ u_M & \vdots & \vdots & \vdots \\ 0 & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_M \end{bmatrix}, \tilde{a} = \begin{bmatrix} a_2 \\ \vdots \\ a_n + 1 \end{bmatrix}, b = \begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_M \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Solving the least squares problem via the normal equations

$$U^* U\tilde{a} = U^* b$$

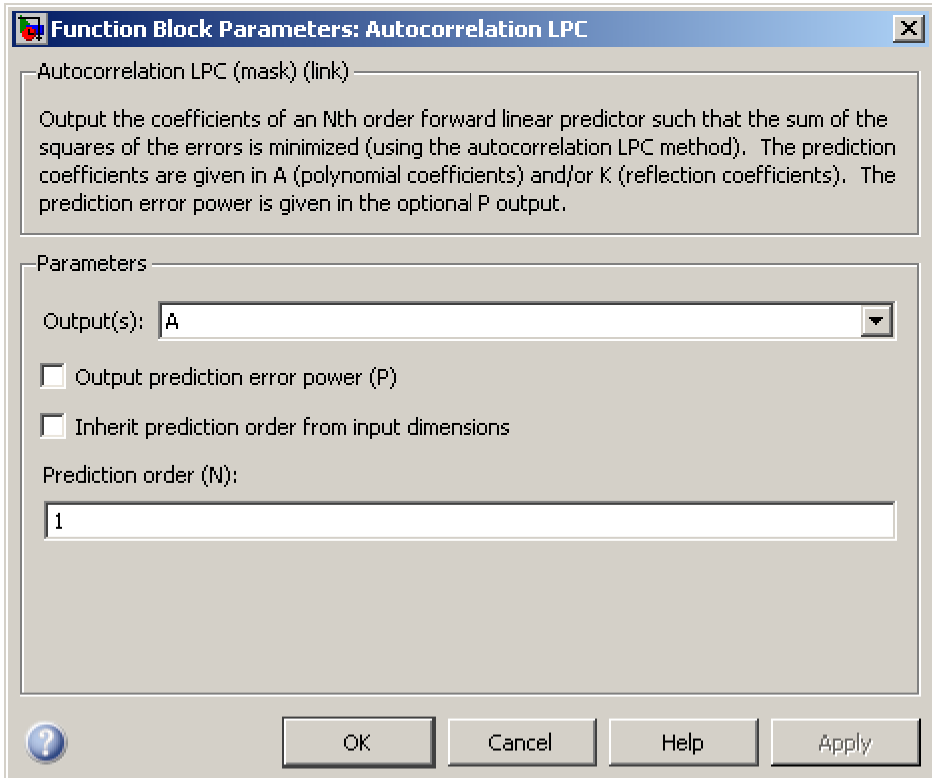
leads to the system of equations

$$\begin{bmatrix} r_1 & r_2^* & \cdots & r_n^* \\ r_2 & r_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & r_2^* \\ r_n & \cdots & r_2 & r_1 \end{bmatrix} \begin{bmatrix} a_2 \\ a_3 \\ \vdots \\ a_{n+1} \end{bmatrix} = \begin{bmatrix} -r_2 \\ -r_3 \\ \vdots \\ -r_{n+1} \end{bmatrix}$$

where $r = [r_1\ r_2\ r_3 \ldots\ r_{n+1}]^T$ is an autocorrelation estimate for u computed using the Autocorrelation block, and * indicates the complex conjugate transpose. The normal equations are solved in $O(n^2)$ operations by the Levinson-Durbin block.

Note that the solution to the LPC problem is very closely related to the Yule-Walker AR method of spectral estimation. In that context, the normal equations above are referred to as the Yule-Walker AR equations.

Dialog Box



Output(s)
: The type of prediction coefficients output by the block. The block can output polynomial coefficients (`A`), reflection coefficients (`K`), or both (`A and K`).

Output prediction error power (P)
: When selected, enables port `P`, which outputs the output prediction error power.

Inherit prediction order from input dimensions
: When selected, the block inherits the prediction order from the input dimensions.

Prediction order (N)

Specify the prediction order, *N*, which must be a scalar. This parameter is disabled when you select the **Inherit prediction order from input dimensions** parameter.

References

Haykin, S. *Adaptive Filter Theory*. 3rd ed. Englewood Cliffs, NJ: Prentice Hall, 1996.

Ljung, L. *System Identification: Theory for the User*. Englewood Cliffs, NJ: Prentice Hall, 1987. Pgs. 278-280.

Proakis, J. and D. Manolakis. *Digital Signal Processing*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Autocorrelation	Signal Processing Blockset
Levinson-Durbin	Signal Processing Blockset
Yule-Walker Method	Signal Processing Blockset
`lpc`	Signal Processing Toolbox

Backward Substitution

Purpose

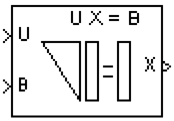
Solve *UX*=*B* for *X* when *U* is upper triangular matrix

Library

Math Functions / Matrices and Linear Algebra / Linear System Solvers

`dspsolvers`

Description



The Backward Substitution block solves the linear system *UX*=*B* by simple backward substitution of variables, where:

- *U* is the upper triangular *M*-by-*M* matrix input to the U port.
- *B* is the *M*-by-*N* matrix input to the B port.

The sample-based output is the *M*-by-*N* matrix *X* that is the solution of the equations. The block does not check the rank of the inputs.

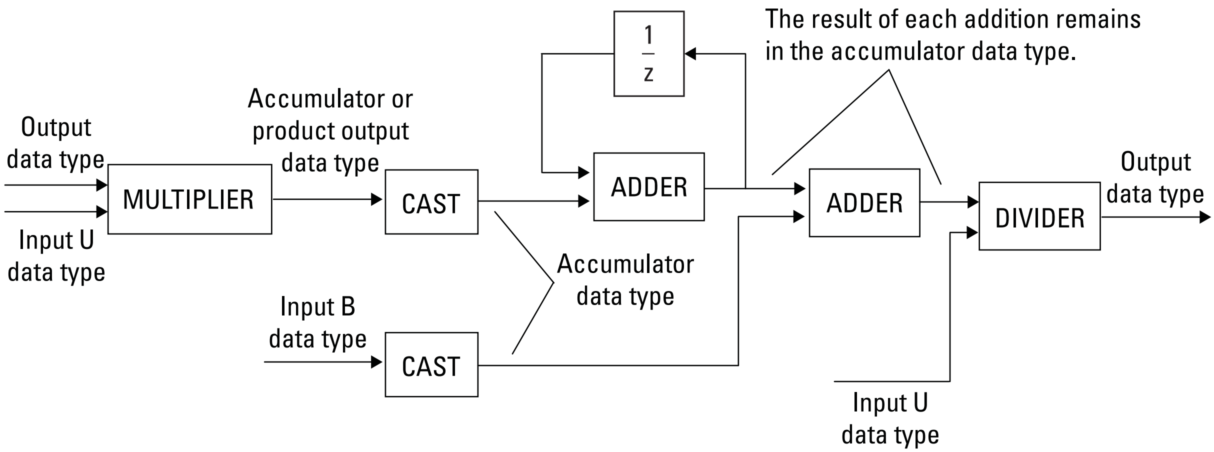
The block uses only the elements in the *upper triangle* of input *U* and ignores the lower elements. When you select the **Input U is unit-upper triangular** check box, the block assumes the elements on the diagonal of *U* are 1s. This is useful when matrix *U* is the result of another operation, such as an LDL decomposition, that uses the diagonal elements to represent the *D* matrix.

The block treats a length-*M* vector input at port B as an *M*-by-1 matrix.

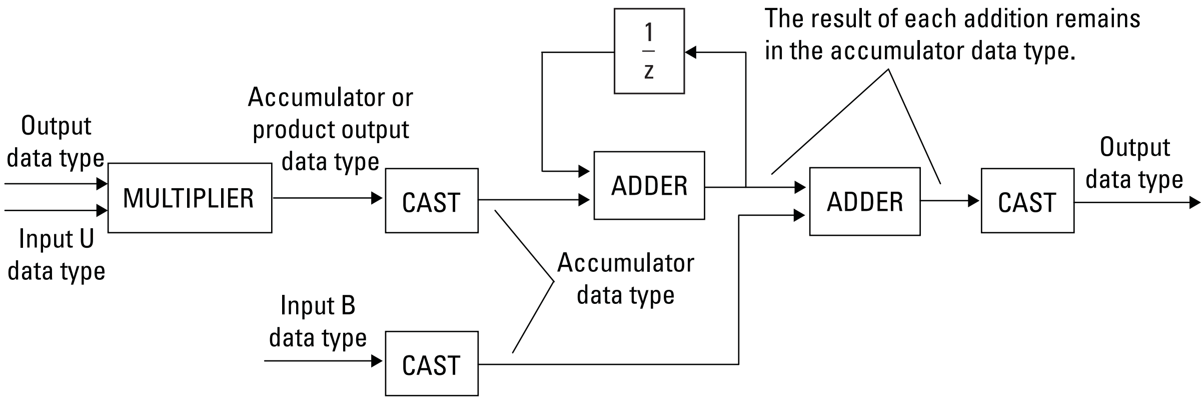
Fixed-Point Data Types

The following diagram shows the data types used within the Backward Substitution block for fixed-point signals.

When input U is not unit-upper triangular:



When input U is unit-upper triangular:

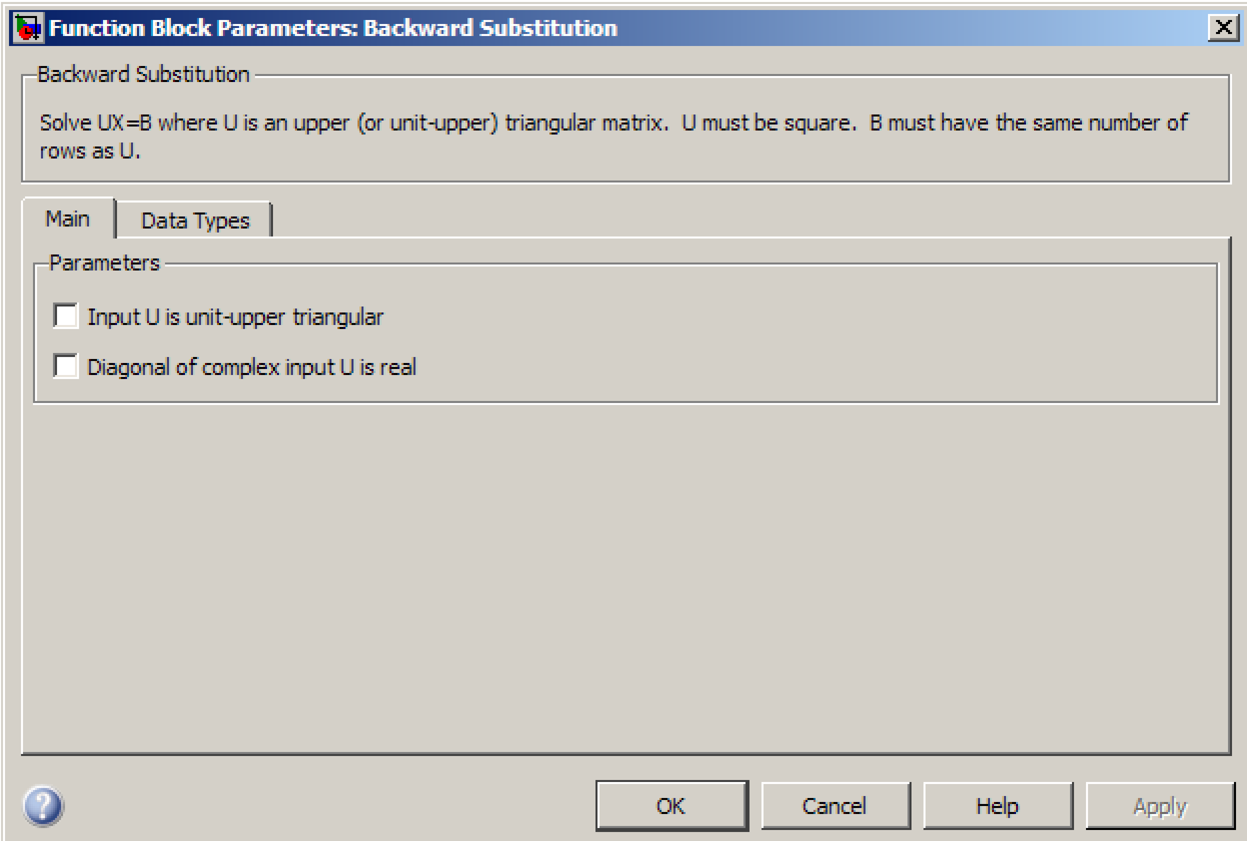


You can set the product output, accumulator, and output data types in the block dialog as discussed in the following section.

The output of the multiplier is in the product output data type when the input is real. When the input is complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see "Multiplication Data Types".

Dialog Box

The **Main** pane of the Backward Substitution block dialog box appears as follows.



Input U is unit-upper triangular

Select this check box only when all elements on the diagonal of *U* have a value of 1. When you do so, the block optimizes its behavior by skipping an unnecessary divide operation.

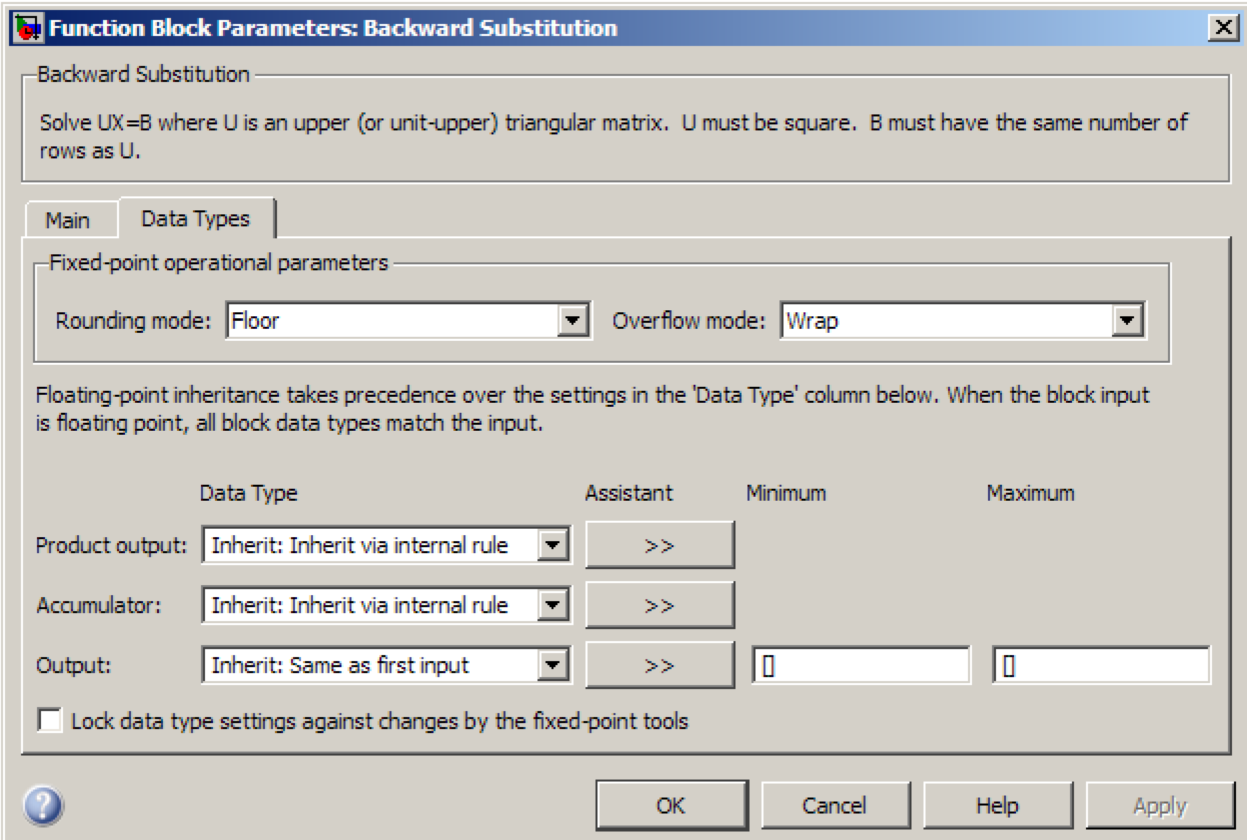
Do not select this check box if there are any elements on the diagonal of *U* that do not have a value of 1. When you clear the **Input U is unit-upper triangular** check box, the block always performs the necessary divide operation.

Diagonal of complex input U is real

Select to optimize simulation speed when the diagonal elements of complex input U are real. This parameter is only visible when **Input U is unit-upper triangular** is not selected.

Note When U is a complex fixed-point signal, you must select either **Input U is unit-upper triangular** or **Diagonal of complex input U is real**. In such a case, any imaginary part of the diagonal of U is ignored.

The **Data Types** pane of the Backward Substitution block dialog appears as follows.



Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode for fixed-point operations.

Product output data type

Specify the product output data type. See “Fixed-Point Data Types” on page 2-60 and “Multiplication Data Types” for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Accumulator data type

Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-60 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Output data type

Specify the output data type. See "Fixed-Point Data Types" on page 2-60 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as first input`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
U	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
B	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
X	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

Cholesky Solver	Signal Processing Blockset
Forward Substitution	Signal Processing Blockset
LDL Solver	Signal Processing Blockset
Levinson-Durbin	Signal Processing Blockset
LU Solver	Signal Processing Blockset
QR Solver	Signal Processing Blockset

See "Linear System Solvers" for related information.

Purpose

Design bandpass filter

Library

Filtering / Filter Designs

`dspfdesign`

Description

This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. Without a Filter Design Toolbox license, you can run models that contain this block, and can edit some, but not all, block parameters. To enable the full filter design functionality of this block, you must have a Filter Design Toolbox license.

Dialog Box

See "Bandpass Filter Design Dialog Box — Main Pane" in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Purpose

Design bandstop filter

Library

Filtering / Filter Designs

`dspfdesign`

Description

This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. Without a Filter Design Toolbox license, you can run models that contain this block, and can edit some, but not all, block parameters. To enable the full filter design functionality of this block, you must have a Filter Design Toolbox license.

Dialog Box

See "Bandstop Filter Design Dialog Box — Main Pane" in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Purpose

Model biquadratic IIR (SOS) filters

Library

Filtering / Filter Implementations

`dsparch4`

Description

The Biquad Filter block independently filters each channel of the input signal with the specified biquadratic IIR filter. The block implements static filters with fixed coefficients that you can tune during simulation.

This block filters each channel of the input signal independently over time, treating each element of the input as an individual channel. The output dimensions always equal those of the input signal. The outputs of this block numerically match the outputs of the Signal Processing Toolbox `dfilt` function.

The Biquad Filter block supports the Simulink state logging feature. See “States” in the *Simulink User’s Guide* for more information. The Digital Filter block also has direct feedthrough. Therefore, if you connect the output of this block back to its input, you get an algebraic loop. For more information on direct feedthrough and algebraic loops, see “Algebraic Loops” in the Simulink documentation.

Coefficient Source and Filter Structures

The Biquad Filter block can operate in three different modes. Select the mode in the **Coefficient source** group box.

- If you select **Dialog parameters**, you enter information about the filter such as structure and coefficients in the block mask. In this mode, you can choose the following filter structures in the **Filter structure** parameter:
  - `Direct form I`
  - `Direct form I transposed`
  - `Direct form II`
  - `Direct form II transposed`

- If you select **Input port(s)**, you enter information about the filter structure in the block mask using the **Filter structure** parameter, but the filter coefficients come into the block via input ports. The following additional ports appear on the block icon:
  - Num — numerator coefficients
  - Den — denominator coefficients
  - g — scale values
- If you select **Discrete-time filter object (DFILT)**, you specify the filter using a `dfilt` object from the Signal Processing Toolbox product or the Filter Design Toolbox product. This block supports the following `dfilt` structures:
  - `dfilt.df1sos`
  - `dfilt.df1tsos`
  - `dfilt.df2sos`
  - `dfilt.df2tsos`

Specifying the SOS Matrix and Scale Values

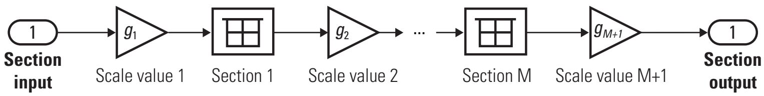
The **SOS matrix (Mx6)** is an M-by-6 matrix, where M is the number of sections in the second-order section filter. Each row of the SOS matrix contains the numerator and denominator coefficients (b_{ik} and a_{ik}) of the corresponding section in the filter.

$$\begin{bmatrix} b_{01} & b_{11} & b_{21} & a_{01} & a_{11} & a_{21} \\ b_{02} & b_{12} & b_{22} & a_{02} & a_{12} & a_{22} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{0M} & b_{1M} & b_{2M} & a_{0M} & a_{1M} & a_{2M} \end{bmatrix}$$

You can use the `ss2sos` and `tf2sos` functions from Signal Processing Toolbox software to convert a state-space or transfer function description of your filter into the second-order section description used by this block.

The **Scale values** parameter specifies the scalar or vector of *M*+1 scale values to be used between SOS sections.

- If you enter a scalar, the value specifies the gain value before the first section of the second-order filter. The rest of the gain values default to 1.
- If you enter a vector of *M*+1 values, each value specifies a separate section of the filter. For example, the first element is the first gain value, the second element is the second gain value, and so on.



Select the **Optimize unity scale values** check box to optimize your simulation when one or more scale values equal 1. Selecting this option removes the unity gains so that the values are treated like Simulink lines or wires. In some fixed-point cases when there are unity scale values, selecting this parameter also omits certain casts. Refer to "Filter Structure Diagrams" on page 2-93 for more information.

Specifying Initial Conditions

The Biquad Filter block initializes the internal filter states to zero by default. You can optionally use the **Initial conditions** or **Initial conditions on zeros side** and **Initial conditions on poles side** parameters to specify nonzero initial states for the filter delays.

To determine the number of initial conditions you must specify and how to specify them, see the following table on valid initial conditions.

Valid Initial Conditions

Initial Condition	Description
Scalar	The block initializes all delay elements in the filter to the scalar value.
Vector or matrix (for applying different delay elements to each channel)	Each vector or matrix element specifies a unique initial condition for a corresponding delay element in a corresponding channel. Where M is the number of sections and N is the number of input channels: • The vector length must equal the number of delay elements in the filter, `max(#_of_zeros, #_of_poles)-1`, or M*2. • The matrix must have the same number of rows as the number of delay elements in the filter, `max(#_of_zeros, #_of_poles)-1`, or (M*2)*N. The matrix must also have one column for each channel of the input signal.

Fixed-Point Data Types

See "Filter Structure Diagrams" on page 2-93. The following constraints apply when you process a fixed-point signal with any of the filter structures supported by this block:

- Inputs and coefficients can be real or complex.
- Numerator and denominator coefficients can be real or complex.
- Specify the coefficients by a M-by-6 matrix in the block mask.
- States are complex when either the inputs or the coefficients are complex.
- Scale values must be real.
- The scale value parameter must be a scalar or a vector of length M+1, where M is the number of sections.
- The **Section input** and **Section output** parameters determine the data type for the section input and output data types.

Examples

Open an example model by typing `doc_biquad_filter_ref` at the MATLAB command line.

Dialog Box

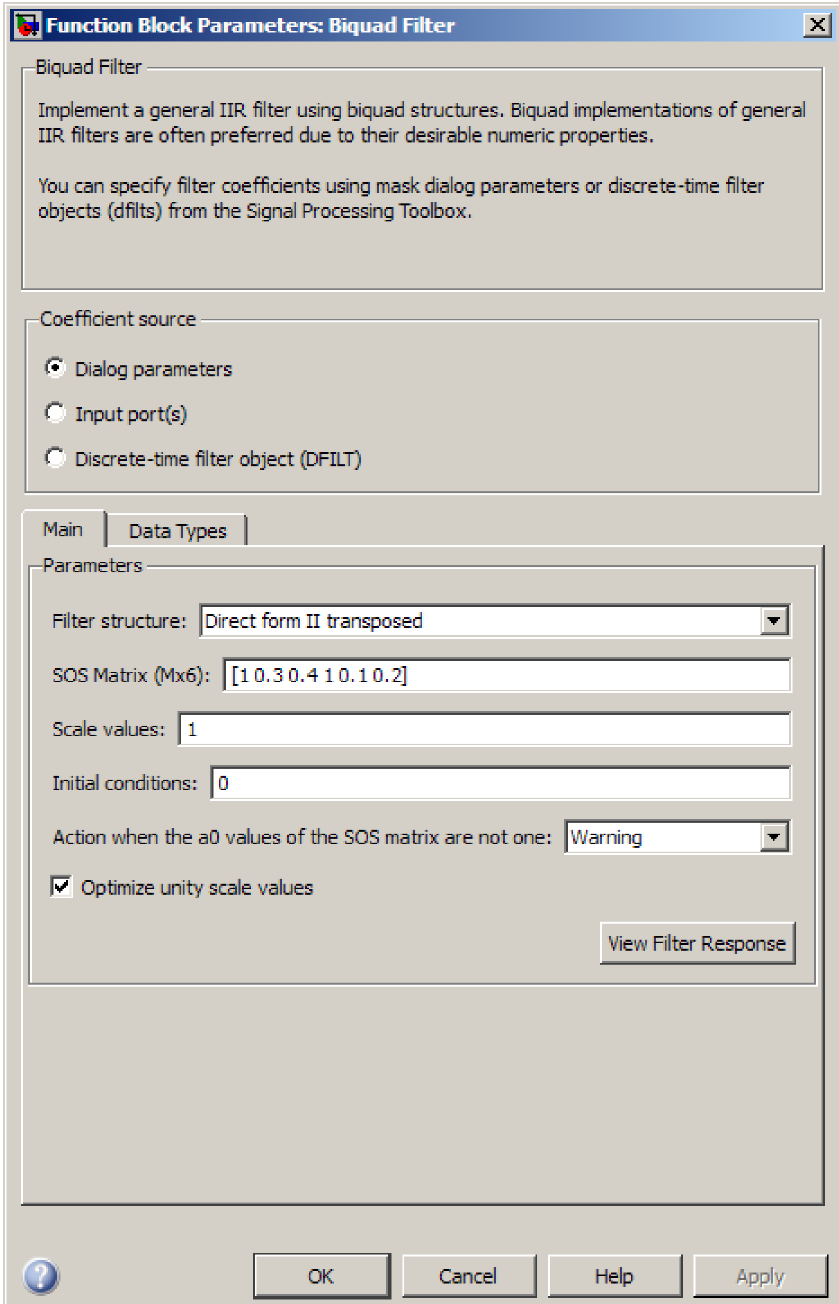
Coefficient Source

The Biquad Filter block can operate in three different modes. Select the mode in the **Coefficient source** group box.

- If you select **Dialog parameters**, you enter information about the filter such as structure and coefficients in the block mask. In this mode, you can choose the following filter structures in the **Filter structure** parameter:
  - `Direct form I`
  - `Direct form I transposed`
  - `Direct form II`
  - `Direct form II transposed`
- If you select **Input port(s)**, you enter information about the filter structure in the block mask using the **Filter structure** parameter, but the filter coefficients come into the block via input ports. The following additional ports appear on the block icon:
  - Num — numerator coefficients
  - Den — denominator coefficients
  - g — scale values
- If you select **Discrete-time filter object (DFILT)**, you specify the filter using a `dfilt` object from the Signal Processing Toolbox product or the Filter Design Toolbox product. This block supports the following `dfilt` structures:
  - `dfilt.df1sos`
  - `dfilt.df1tsos`
  - `dfilt.df2sos`
  - `dfilt.df2tsos`

Specify Filter Characteristics in Dialog

The **Main** pane of the Biquad Filter block dialog appears as follows when **Dialog parameters** is selected in the **Coefficient source** group box.


Function Block Parameters: Biquad Filter
Biquad Filter
Implement a general IIR filter using biquad structures. Biquad implementations of general IIR filters are often preferred due to their desirable numeric properties.
You can specify filter coefficients using mask dialog parameters or discrete-time filter objects (dfilts) from the Signal Processing Toolbox.
Coefficient source
Dialog parameters
Input port(s)
Discrete-time filter object (DFILT)
Main
Data Types
Parameters
Filter structure:
Direct form II transposed
SOS Matrix (Mx6):
[1 0.3 0.4 1 0.1 0.2]
Scale values:
1
Initial conditions:
0
Action when the a0 values of the SOS matrix are not one:
Warning
Optimize unity scale values
View Filter Response
OK
Cancel
Help
Apply

Filter structure

Select the filter structure.

This parameter is only visible when **Dialog parameters** or **Input port(s)** is selected.

SOS Matrix

Specify an M-by-6 matrix, where M is the number of sections in the second-order section filter. Each row of the SOS matrix contains the numerator and denominator coefficients (b_{ik} and a_{ik}) of the corresponding section in the filter.

$$\begin{bmatrix} b_{01} & b_{11} & b_{21} & a_{01} & a_{11} & a_{21} \\ b_{02} & b_{12} & b_{22} & a_{02} & a_{12} & a_{22} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{0M} & b_{1M} & b_{2M} & a_{0M} & a_{1M} & a_{2M} \end{bmatrix}$$

You can use the `ss2sos` and `tf2sos` functions from Signal Processing Toolbox software to check whether your SOS matrix is valid.

This parameter is only visible when **Dialog parameters** is selected.

Scale values

The **Scale values** parameter specifies the scalar or vector of M+1 scale values to be used between SOS sections.

- When you enter a scalar, the value specifies the gain value before the first section of the second-order filter. The rest of the gain values default to 1.
- When you enter a vector of M+1 values, each value specifies a separate section of the filter. For example, the first element is the first gain value, the second element is the second gain value, and so on.

This parameter is only visible when **Dialog parameters** is selected.

Initial conditions

Specify the initial conditions of the filter states. To learn how to specify initial conditions, see "Specifying Initial Conditions" on page 2-75.

This parameter is only visible when **Dialog parameters** or **Input port(s)** is selected and the filter structure is `Direct form II` or `Direct form II transposed`.

Initial conditions on zeros side

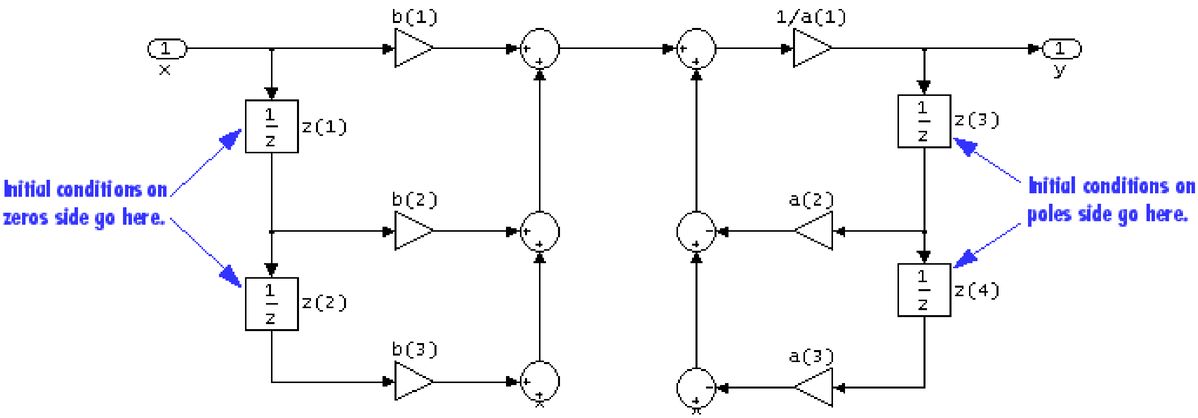
Specify the initial conditions for the filter states on the side of the filter structure with the zeros (b_0, b_1,b_2, ...); see the next diagram.

This parameter is only visible when **Dialog parameters** or **Input port(s)** is selected and the filter structure is `Direct form I` or `Direct form I transposed`. To learn how to specify initial conditions, see"Specifying Initial Conditions" on page 2-75.

Initial conditions on poles side

Specify the initial conditions for the filter states on the side of the filter structure with the poles (a_0, a_1,a_2, ...); see the next diagram.

This parameter is only visible when **Dialog parameters** or **Input port(s)** is selected and the filter structure is `Direct form I` or `Direct form I transposed`. To learn how to specify initial conditions, see "Specifying Initial Conditions" on page 2-75.



Action when the a0 values of the SOS matrix are not one

Specify the action the block should perform when the SOS matrix a_{0j} values do not equal one; `None`, `Error`, or `Warn`.

This parameter is only visible when **Dialog parameters** is selected.

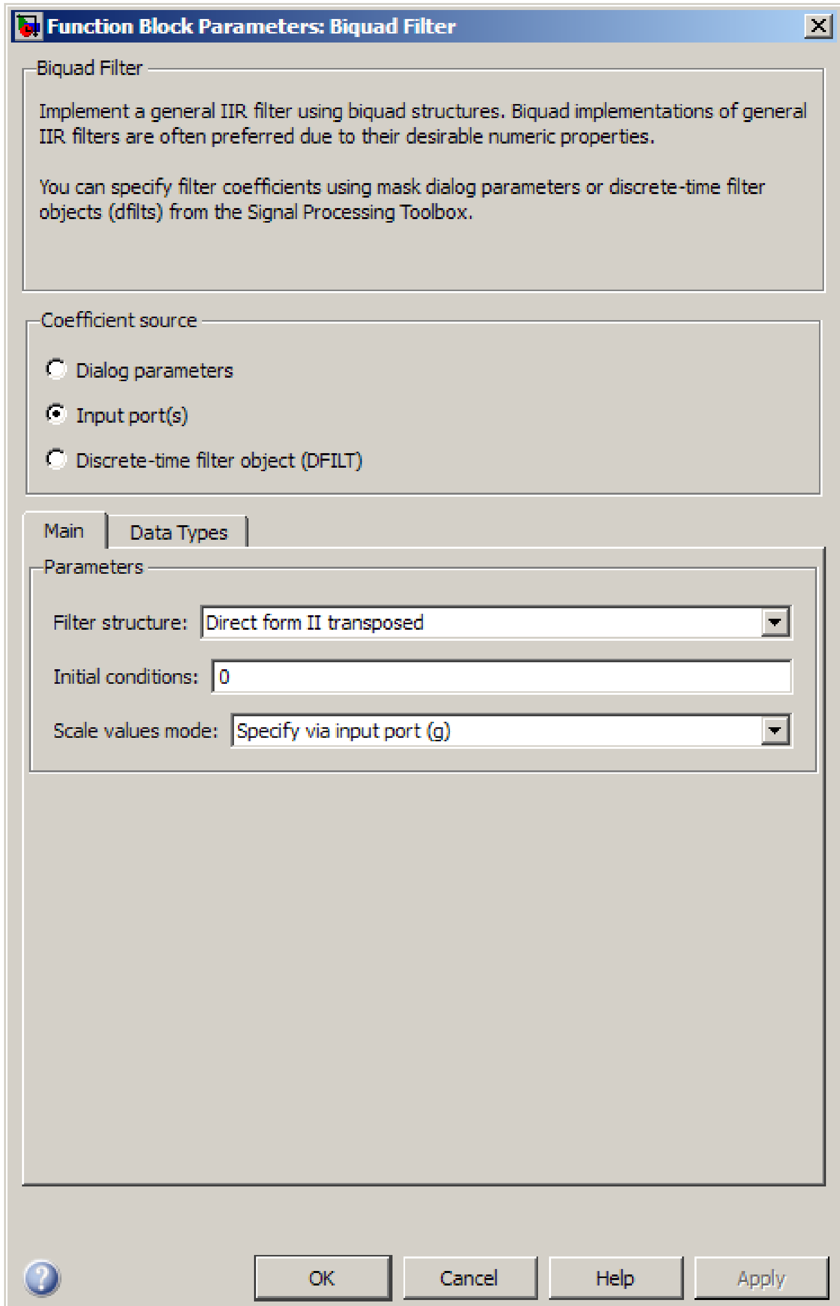
Optimize unity scale values

Select this check box to optimize your simulation when one or more scale values equal 1. Selecting this option removes the unity gains so that the values are treated like Simulink lines or wires. In some fixed-point cases when there are unity scale values, selecting this parameter also omits certain casts. Refer to "Filter Structure Diagrams" on page 2-93 for more information.

This parameter is only visible when **Dialog parameters** is selected.

Specify Filter Characteristics via Input Port

The **Main** pane of the Biquad Filter block dialog appears as follows when **Input port(s)** is selected in the **Coefficient source** group box.


Function Block Parameters: Biquad Filter
Biquad Filter
Implement a general IIR filter using biquad structures. Biquad implementations of general IIR filters are often preferred due to their desirable numeric properties.
You can specify filter coefficients using mask dialog parameters or discrete-time filter objects (dfilts) from the Signal Processing Toolbox.
Coefficient source
Dialog parameters
Input port(s)
Discrete-time filter object (DFILT)
Main
Data Types
Parameters
Filter structure:
Direct form II transposed
Initial conditions:
0
Scale values mode:
Specify via input port (g)
OK
Cancel
Help
Apply

Filter structure

Select the filter structure.

This parameter is only visible when **Dialog parameters** or **Input port(s)** is selected.

Initial conditions

Specify the initial conditions of the filter states. To learn how to specify initial conditions, see "Specifying Initial Conditions" on page 2-75.

This parameter is only visible when **Dialog parameters** or **Input port(s)** is selected and the filter structure is `Direct form II` or `Direct form II transposed`.

Initial conditions on zeros side

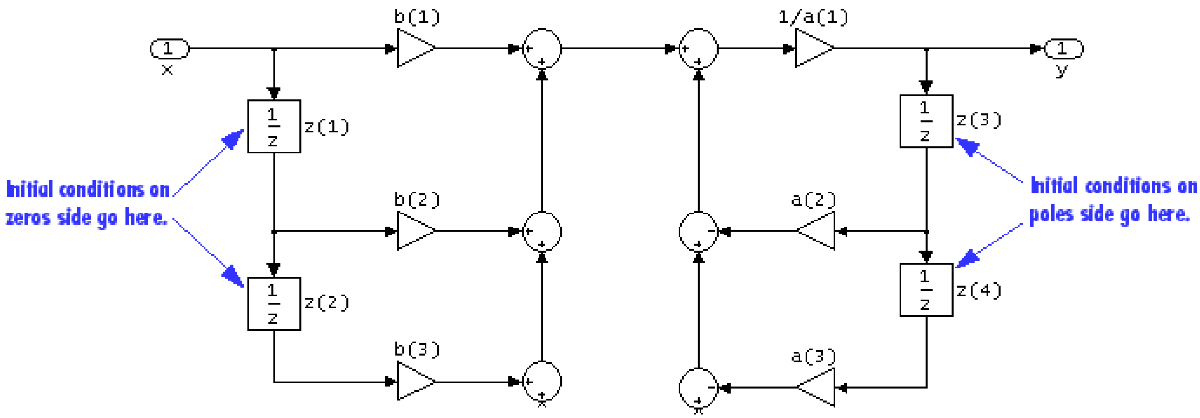
Specify the initial conditions for the filter states on the side of the filter structure with the zeros (b_0, b_1,b_2, ...); see the next diagram.

This parameter is only visible when **Dialog parameters** or **Input port(s)** is selected and the filter structure is `Direct form I` or `Direct form I transposed`. To learn how to specify initial conditions, see "Specifying Initial Conditions" on page 2-75.

Initial conditions on poles side

Specify the initial conditions for the filter states on the side of the filter structure with the poles (a_0, a_1,a_2, ...); see the next diagram.

This parameter is only visible when **Dialog parameters** or **Input port(s)** is selected and the filter structure is `Direct form I` or `Direct form I transposed`. To learn how to specify initial conditions, see "Specifying Initial Conditions" on page 2-75.



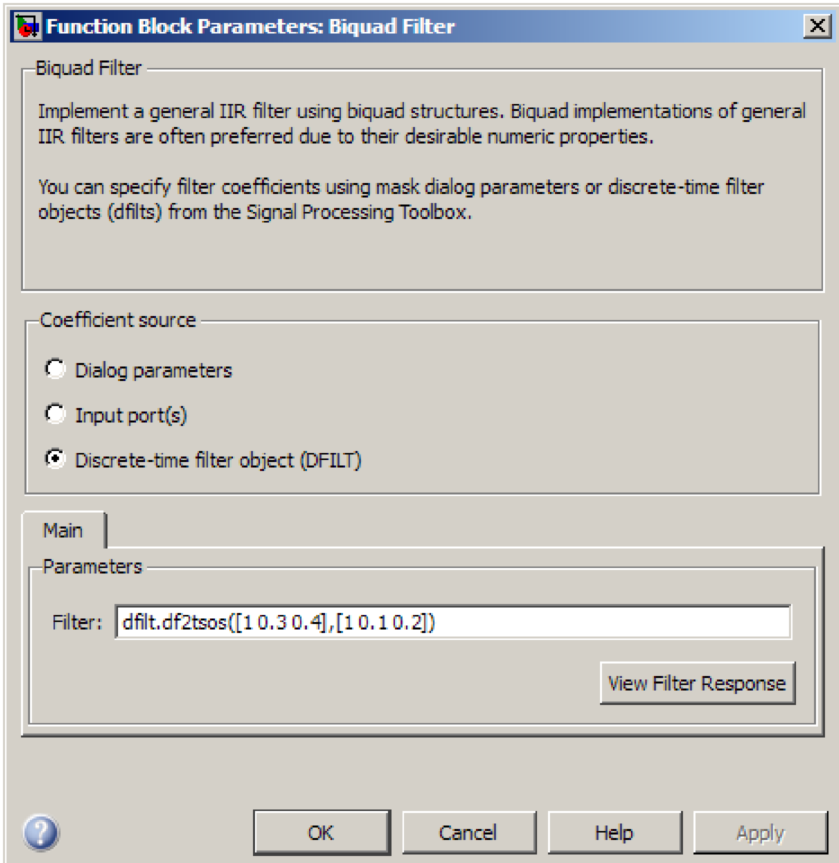
Scale values mode

Choose how to specify the scale values to use between filter sections. When you select `Specify via input port (g)`, you enter the scale values as a 2-D vector at port g. When you select `Assume all are unity and optimize`, all scale values are removed and treated like Simulink lines or wires.

This parameter is only visible when **Input port(s)** is selected.

Specify Discrete-Time Filter Object

The **Main** pane of the Biquad Filter block dialog appears as follows when **Discrete-time filter object (DFILT)** is selected in the **Coefficient source** group box.



Filter

Specify the discrete-time filter object (`dfilt`) that you would like the block to implement. You can do so in one of three ways:

- You can fully specify the `dfilt` object in the block mask.
- You can enter the variable name of a `dfilt` object that is defined in any workspace.

- You can enter a variable name for a `dfilt` object that is not yet defined.

For more information on creating `dfilt` objects, see the `dfilt` function reference page in the Signal Processing Toolbox documentation.

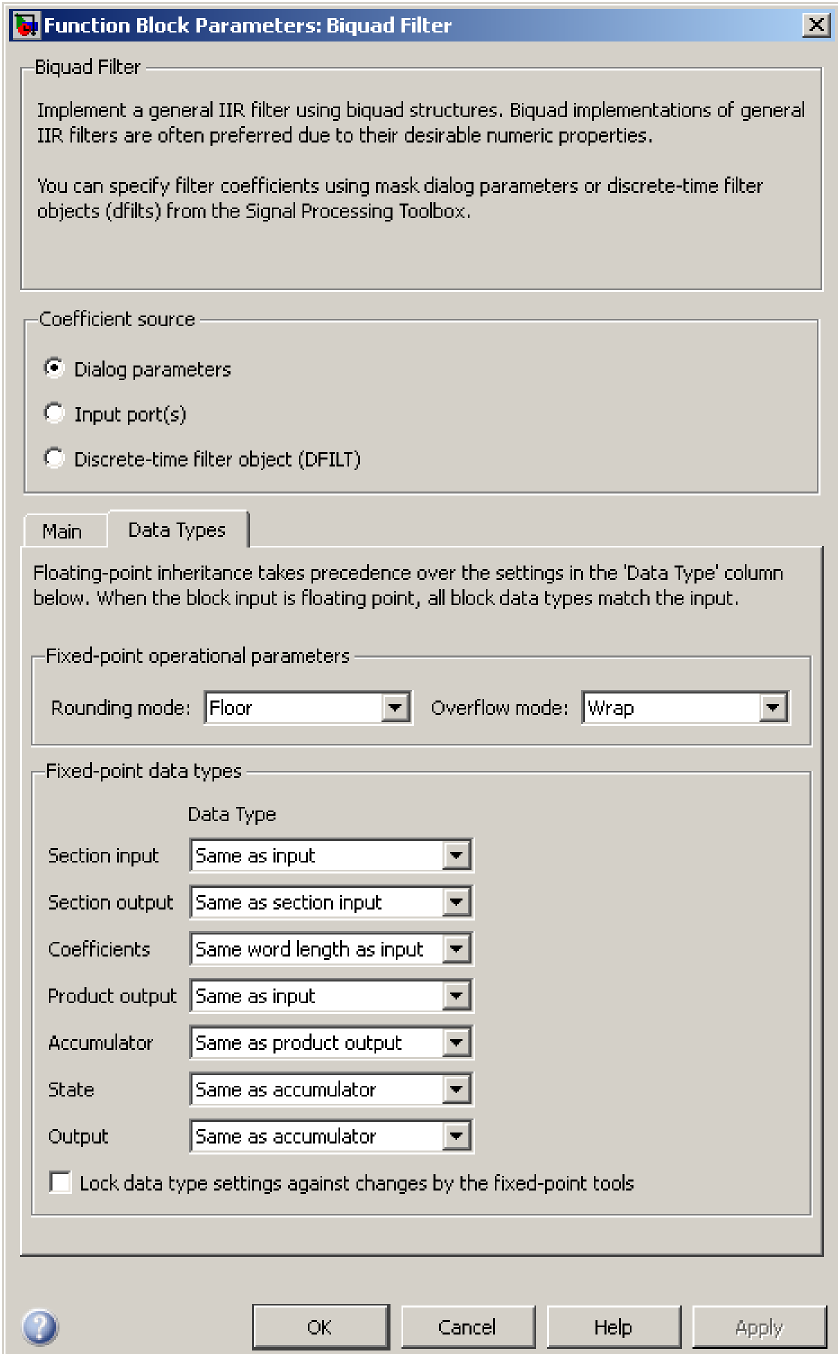
View filter response

This button opens the Filter Visualization Tool (`fvtool`) from the Signal Processing Toolbox product and displays the filter response of the `dfilt` object specified in the **Discrete-time filter object (DFILT)** parameter. For more information on FVTool, see the Signal Processing Toolbox documentation.

Note This button is only available when the Filter Design Toolbox and Fixed-Point Toolbox™ products are installed. If you specify a filter in the **Discrete-time filter object (DFILT)** parameter, you must apply the filter by clicking the **Apply** button before using the **View filter response** button.

Specify Fixed-Point Parameters

The **Data Types** pane of the Biquad Filter block dialog appears as follows. This pane only appears when **Dialog parameters** or **Input port(s)** is selected in the **Coefficient source** group box.


Function Block Parameters: Biquad Filter
Biquad Filter
Implement a general IIR filter using biquad structures. Biquad implementations of general IIR filters are often preferred due to their desirable numeric properties.
You can specify filter coefficients using mask dialog parameters or discrete-time filter objects (dfilts) from the Signal Processing Toolbox.
Coefficient source
Dialog parameters
Input port(s)
Discrete-time filter object (DFILT)
Main
Data Types
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input.
Fixed-point operational parameters
Rounding mode:
Floor
Overflow mode:
Wrap
Fixed-point data types
Data Type
Section input
Same as input
Section output
Same as section input
Coefficients
Same word length as input
Product output
Same as input
Accumulator
Same as product output
State
Same as accumulator
Output
Same as accumulator
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Rounding mode
Select the rounding mode for fixed-point operations. The filter coefficients do not obey this parameter; instead, they always round to `Nearest`.

Overflow mode
Select the overflow mode for fixed-point operations. The filter coefficients do not obey this parameter; instead, they are always saturated.

Multiplicand
Choose how you specify the word length and the fraction length of the multiplicand data type of a `Direct form I transposed` filter structure. See "Fixed-Point Data Types" on page 2-76 and the "Direct Form I Transposed" on page 2-96 filter structure diagram for illustrations depicting the use of the multiplicand data type in this block.

This parameter is only visible when the **Filter structure** parameter is set to `Direct form I transposed`.

- When you select `Same as output`, these characteristics match those of the output of the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Section input
Choose how you specify the word length and the fraction length of the fixed-point data type going into each section of a biquadratic filter. See "Fixed-Point Data Types" on page 2-76 for illustrations depicting the use of the section input data type in this block.

- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word and fraction lengths of the section input and output, in bits.
- When you select `Slope and bias scaling`, you can enter the word lengths, in bits, and the slopes of the section input and output. This block requires power-of-two slope and a bias of zero.

Section output

Choose how you specify the word length and the fraction length of the fixed-point data type coming out of each section of a biquadratic filter. See "Fixed-Point Data Types" on page 2-76 for illustrations depicting the use of the section input data type in this block.

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word and fraction lengths of the section input and output, in bits.
- When you select `Slope and bias scaling`, you can enter the word lengths, in bits, and the slopes of the section input and output. This block requires power-of-two slope and a bias of zero.

Coefficients

Choose how you specify the word length and the fraction length of the filter coefficients (numerator, denominator, and scale value) when **Dialog parameters** is selected in the **Coefficient source** group box. See "Fixed-Point Data Types" on page 2-76 for illustrations depicting the use of the coefficient data types in this block:

- When you select `Same word length as input`, the word length of the filter coefficients match that of the input to the block. In this mode, the block automatically sets the fraction length of the coefficients to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `Specify word length`, you can enter the word length of the coefficients, in bits. In this mode, the block automatically sets the fraction length of the coefficients to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the coefficients, in bits. If applicable, you can enter separate fraction lengths for the numerator and denominator coefficients.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the coefficients. If applicable, you can enter separate slopes for the numerator and denominator coefficients. This block requires power-of-two slope and a bias of zero.
- The filter coefficients do not obey the **Rounding mode** and the **Overflow mode** parameters; instead, they are always saturated and rounded to `Nearest`.

Product output

Use this parameter to specify how you would like to designate the product output word and fraction lengths. See "Fixed-Point Data Types" on page 2-76 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator

Use this parameter to specify how you would like to designate the accumulator word and fraction lengths. See "Fixed-Point Data Types" on page 2-76 and "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Same as product output`, these characteristics match those of the product output.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

State

Use this parameter to specify how you would like to designate the state word and fraction lengths when **Dialog parameters** is selected in the **Coefficient source** group box. See "Fixed-Point Data Types" on page 2-76 for illustrations depicting the use of the state data type in this block.

This parameter is not visible for `Direct form I` and `Direct form I transposed filter` structures.

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Same as accumulator`, these characteristics match those of the accumulator.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Choose how you specify the output word length and fraction length. See “Fixed-Point Data Types” on page 2-76 for illustrations depicting the use of the output data type in this block.

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Same as accumulator`, these characteristics match those of the accumulator.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

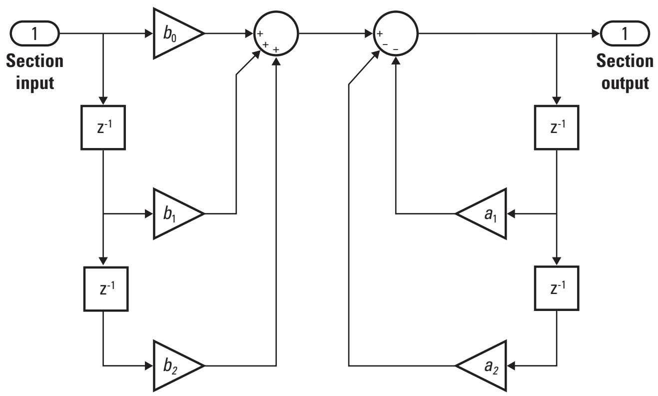
Lock data type settings against changes by the fixed-point tools

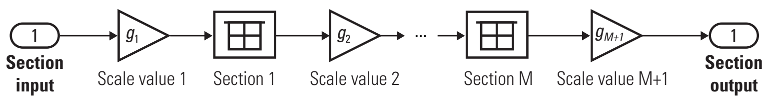
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Filter Structure Diagrams

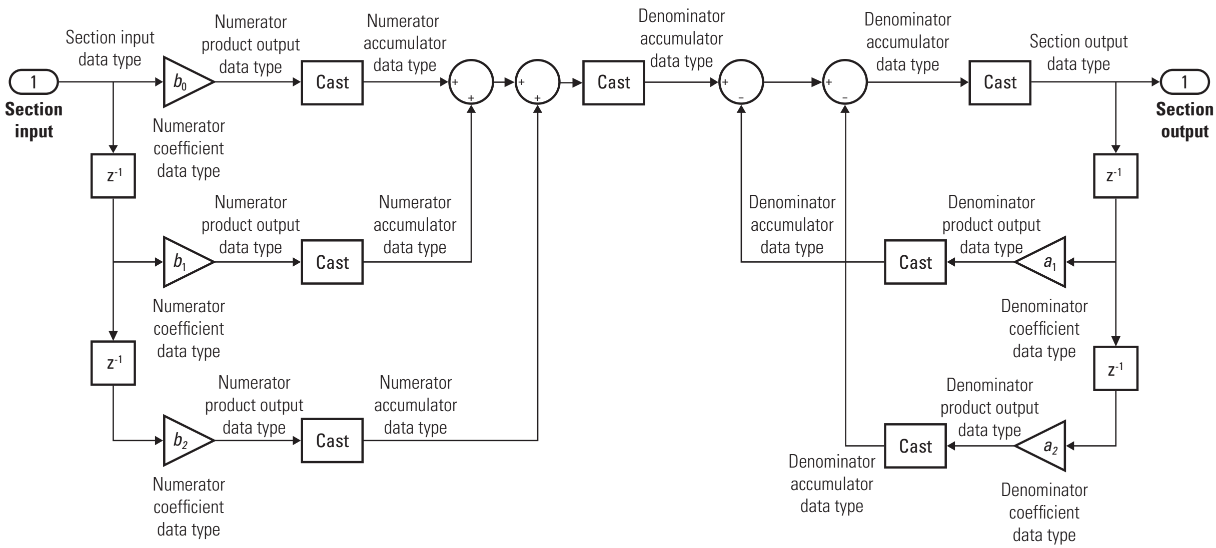
The diagrams in the following sections show the filter structures supported by the Biquad Filter block. They also show the data types used in the filter structures for fixed-point signals. You can set the data types shown in these diagrams in the block dialog box. This is discussed in “Dialog Box” on page 2-77.

Direct Form I



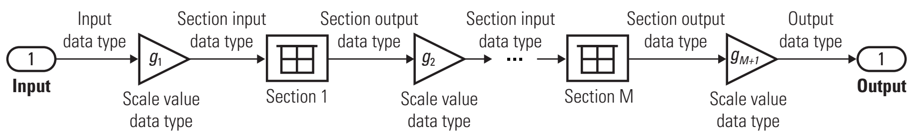


The following diagram shows the data types for one section of the filter for fixed-point signals.

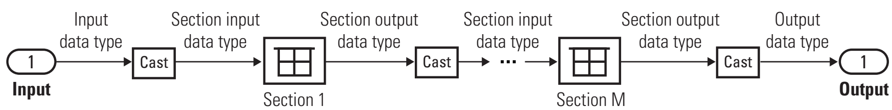


The following diagrams show the fixed-point data types between filter sections.

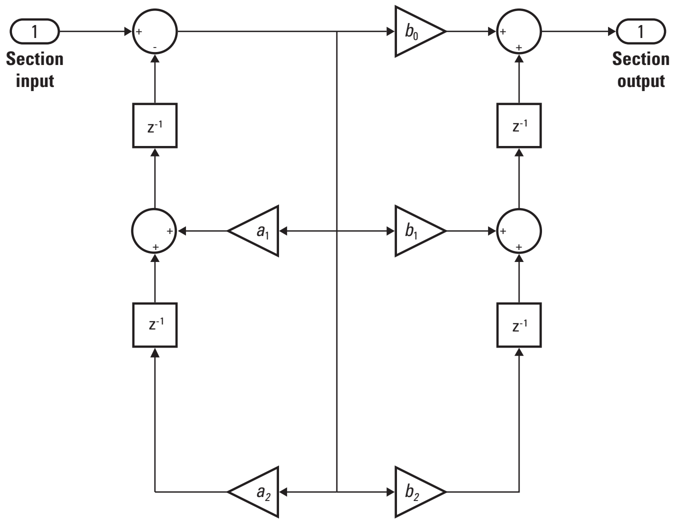
When the data is not optimized:

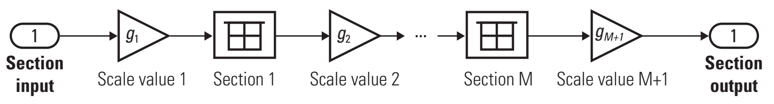


When you select **Optimize unity scale values** and scale values equal 1:

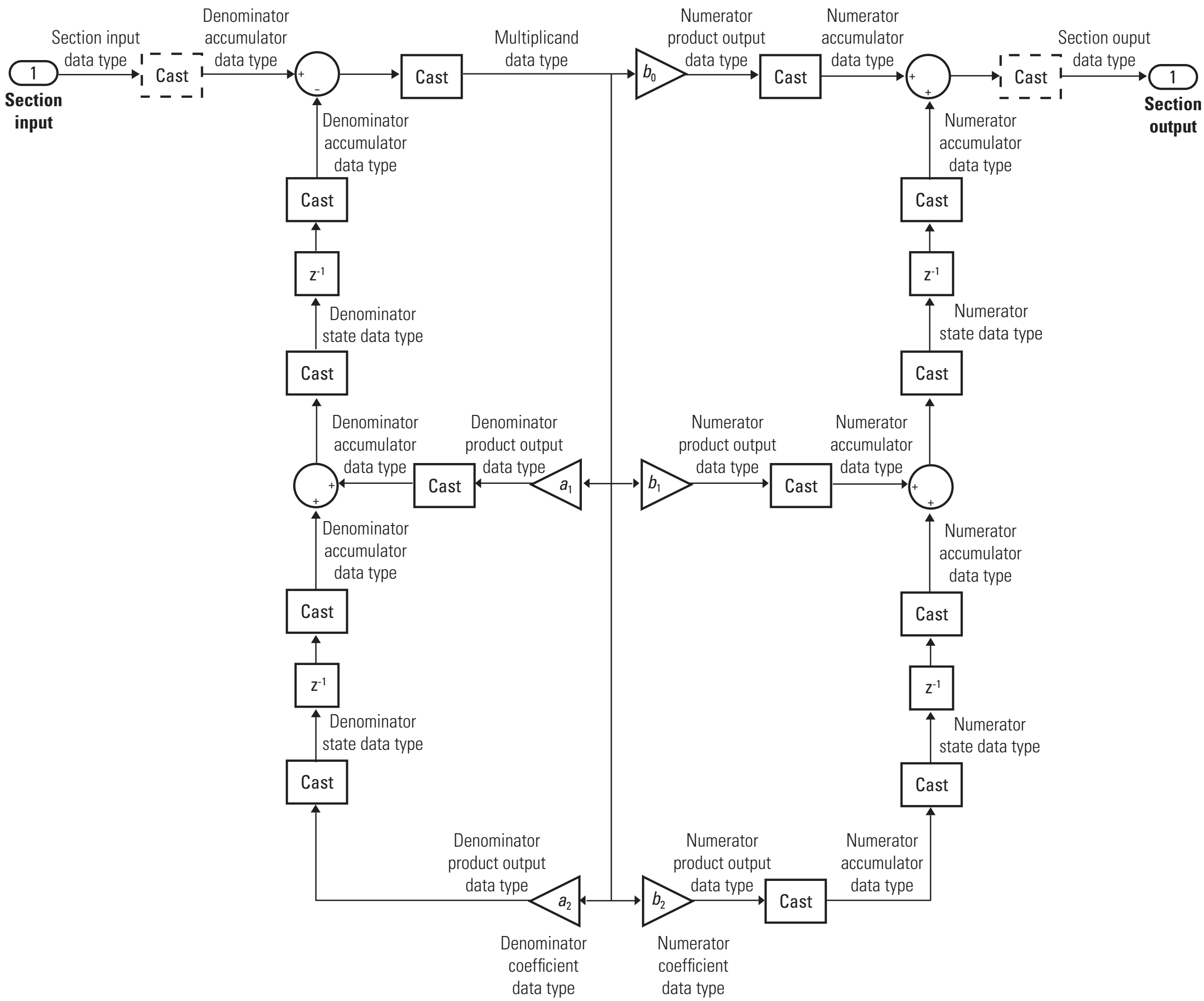


Direct Form I Transposed





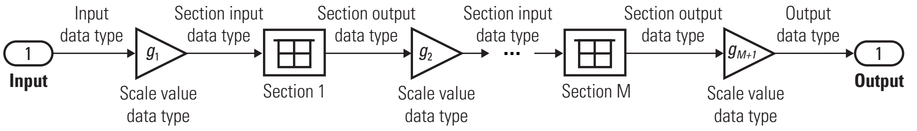
The following diagram shows the data types for one section of the filter for fixed-point signals.



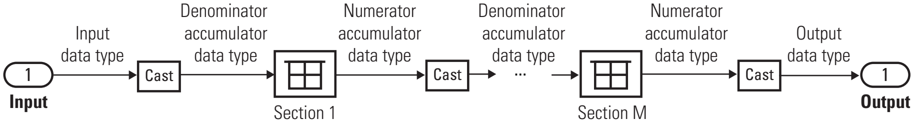
The dashed casts are omitted when **Optimize unity scale values** is selected and scale values equal one.

The following diagrams show the fixed-point data types between filter sections.

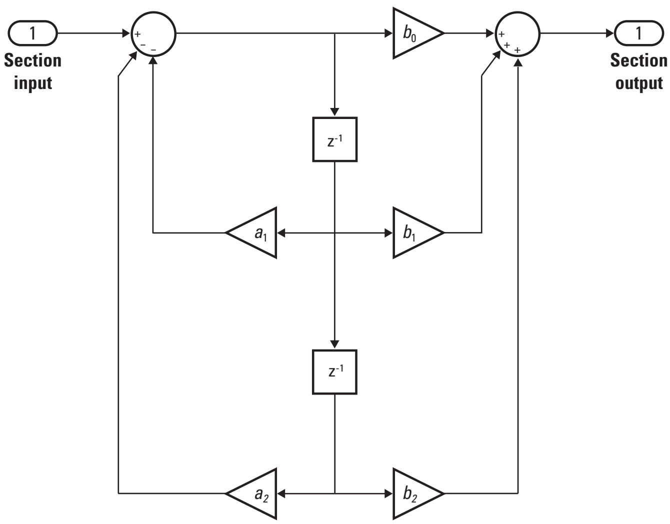
When the data is not optimized:

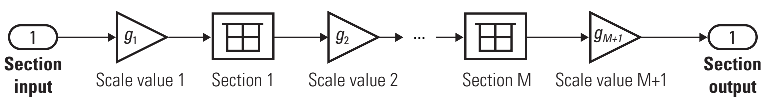


When you select **Optimize unity scale values** and scale values equal 1:

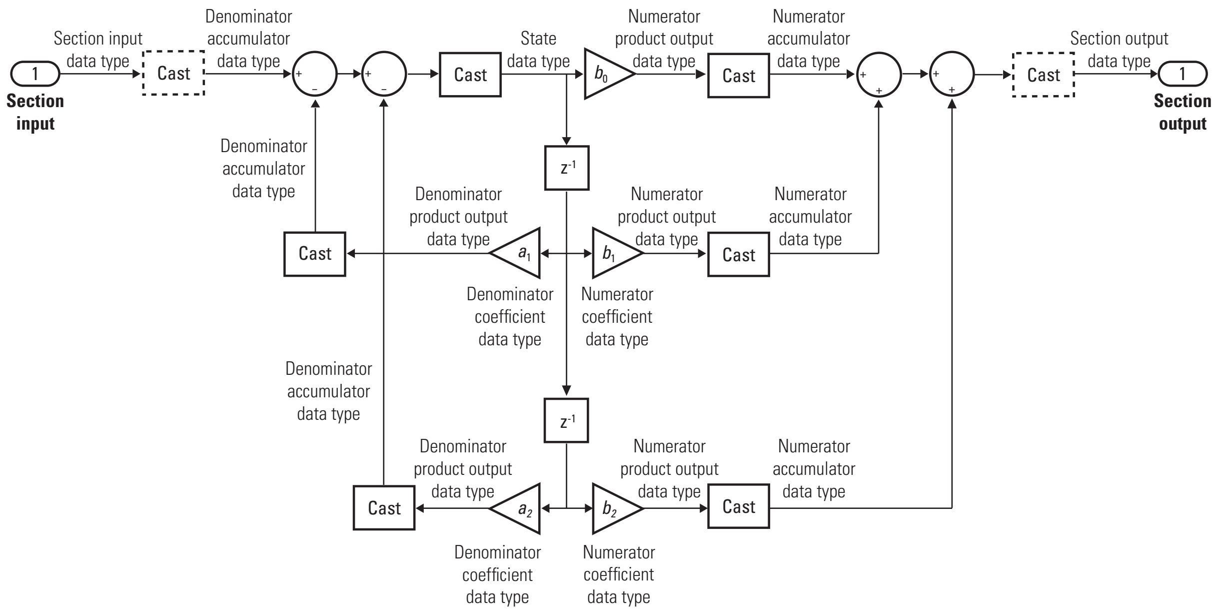


Direct Form II





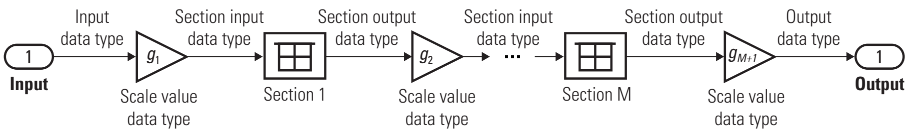
The following diagram shows the data types for one section of the filter for fixed-point signals.



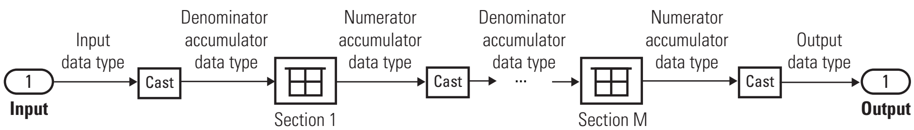
The dashed casts are omitted when **Optimize unity scale values** is selected and scale values equal one.

The following diagrams show the fixed-point data types between filter sections.

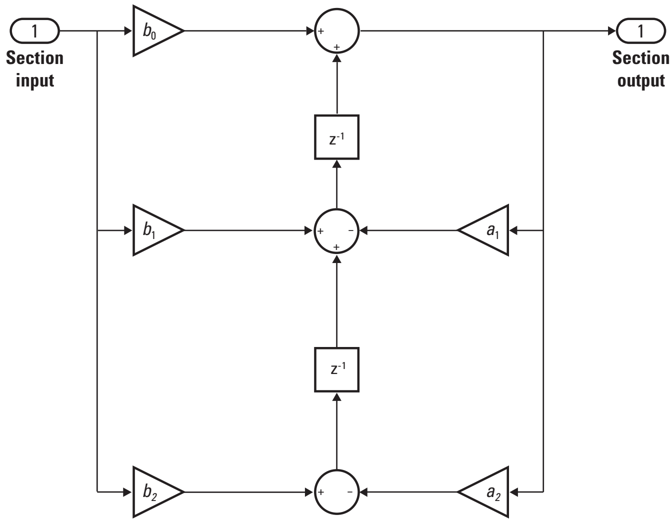
When the data is not optimized:

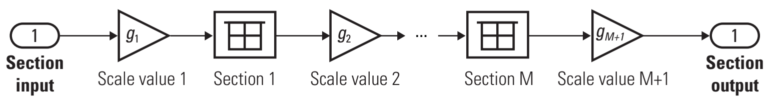


When you select **Optimize unity scale values** and scale values equal 1:

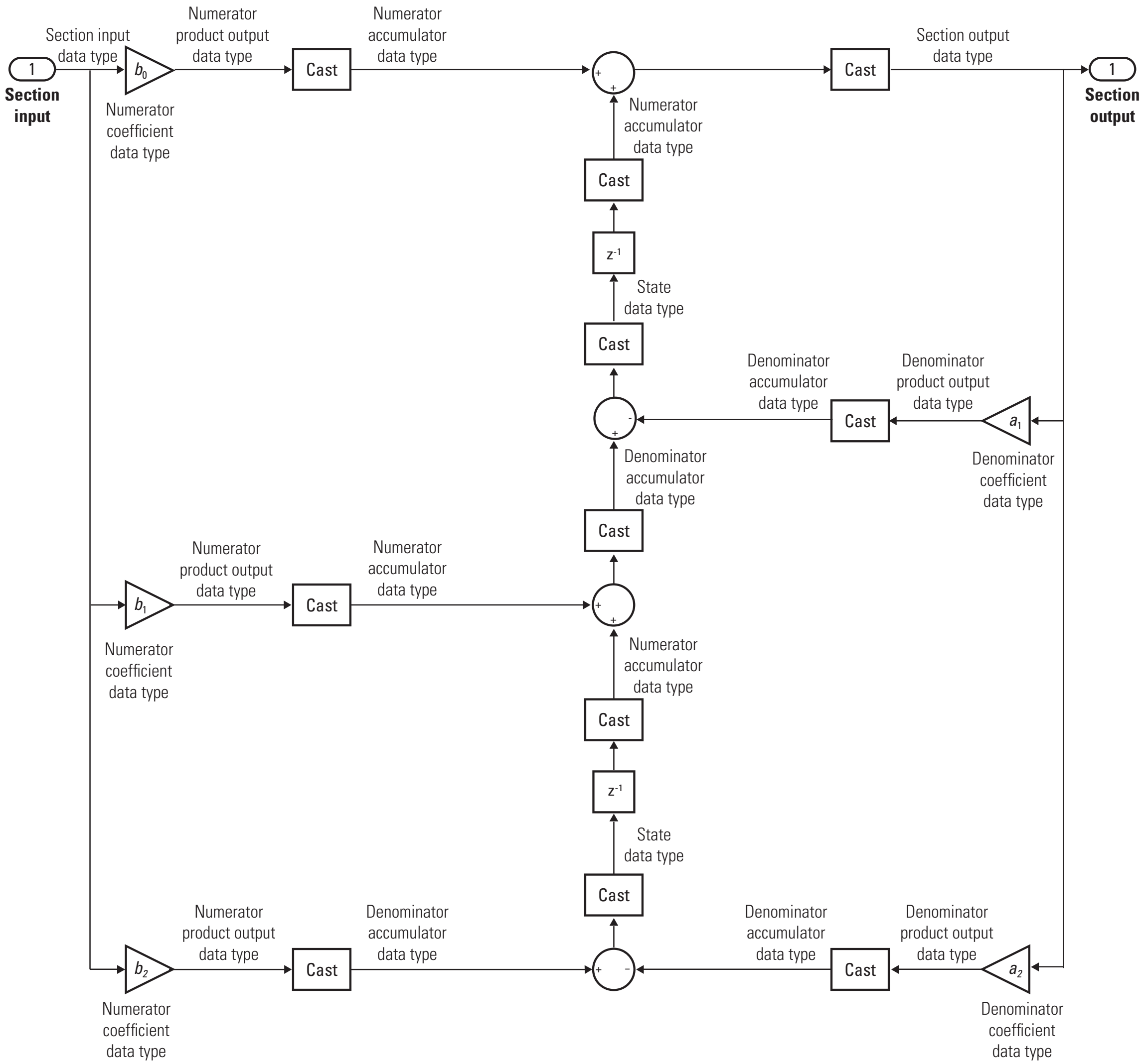


Direct Form II Transposed



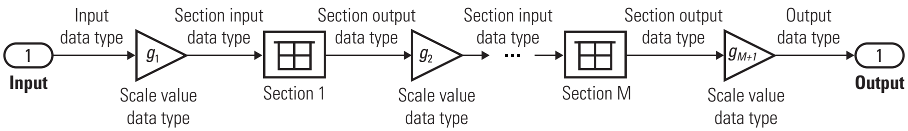


The following diagram shows the data types for one section of the filter for fixed-point signals.

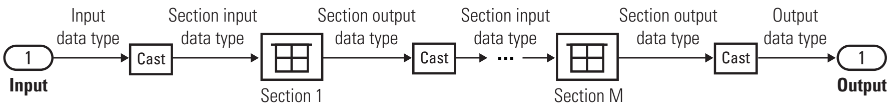


The following diagrams show the fixed-point data types between filter sections.

When the data is not optimized:



When you select **Optimize unity scale values** and scale values equal 1:



Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

`dfilt.df1sos`	Filter Design Toolbox
`dfilt.df1tsos`	Filter Design Toolbox

`dfilt.df2sos`	Filter Design Toolbox
`dfilt.df2tsos`	Filter Design Toolbox

Block LMS Filter

Purpose

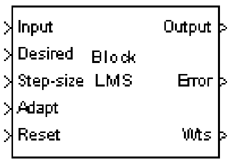
Compute output, error, and weights using LMS adaptive algorithm

Library

Filtering / Adaptive Filters

`dspadpt3`

Description



Block LMS Filter

The Block LMS Filter block implements an adaptive least mean-square (LMS) filter, where the adaptation of filter weights occurs once for every block of samples. The block estimates the filter weights, or coefficients, needed to minimize the error, $e(n)$, between the output signal, $y(n)$, and the desired signal, $d(n)$. Connect the signal you want to filter to the Input port. This input signal can be a sample-based scalar or a single-channel frame-based signal. Connect the signal you want to model to the `Desired` port. The desired signal must have the same data type, frame status, complexity, and dimensions as the input signal. The `Output` port outputs the filtered input signal, which can be sample or frame based. The `Error` port outputs the result of subtracting the output signal from the desired signal.

The block calculates the filter weights using the Block LMS adaptive filter algorithm. This algorithm is defined by the following equations.

$$n = kN + i$$
$$y(n) = \mathbf{w}^T(k-1)\mathbf{u}(n)$$
$$e(n) = d(n) - y(n)$$
$$\mathbf{w}(k) = \mathbf{w}(k-1) + f(\mathbf{u}(n), e(n), \mu)$$

The weight update function for the Block LMS adaptive filter algorithm is defined as

$$f(\mathbf{u}(n), e(n), \mu) = \mu \sum_{i=0}^{N-1} \mathbf{u}^*(kN+i)e(kN+i)$$

The variables are as follows.

Variable	Description
n	The current time index
i	The iteration variable in each block, $0 \le i \le N-1$
k	The block number
N	The block size
$\mathbf{u}(n)$	The vector of buffered input samples at step n
$\mathbf{w}(n)$	The vector of filter-tap estimates at step n
$y(n)$	The filtered output at step n
$e(n)$	The estimation error at time n
$d(n)$	The desired response at time n
μ	The adaptation step size

Use the **Filter length** parameter to specify the length of the filter weights vector.

The **Block size** parameter determines how many samples of the input signal are acquired before the filter weights are updated. The input frame length must be a multiple of the **Block size** parameter.

The adaptation **Step-size (mu)** parameter corresponds to μ in the equations. You can either specify a step-size using the input port, Step-size, or enter a value in the Block Parameters: Block LMS Filter dialog box.

Use the **Leakage factor (0 to 1)** parameter to specify the leakage factor, $0 < 1 - \mu\alpha \le 1$, in the leaky LMS algorithm shown below.

$$\mathbf{w}(k) = (1 - \mu\alpha)\mathbf{w}(k-1) + f(\mathbf{u}(n), e(n), \mu)$$

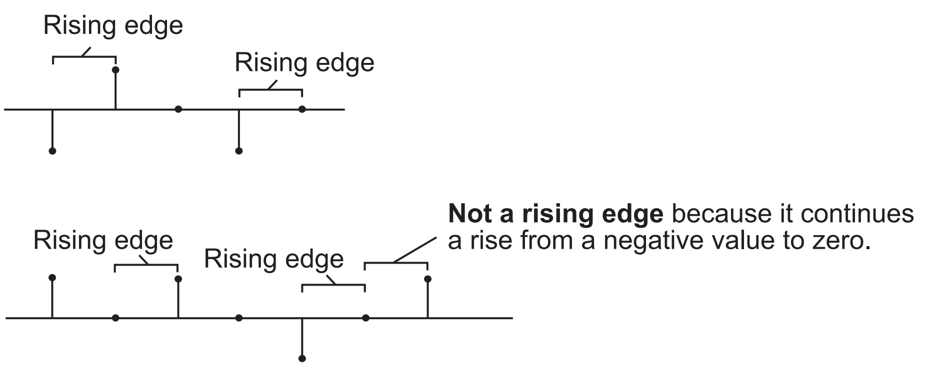
Enter the initial filter weights as a vector or a scalar in the **Initial value of filter weights** text box. When you enter a scalar, the block uses the scalar value to create a vector of filter weights. This vector has length equal to the filter length and all of its values are equal to the scalar value

When you select the **Adapt port** check box, an `Adapt` port appears on the block. When the input to this port is greater than zero, the block continuously updates the filter weights. When the input to this port is zero, the filter weights remain at their current values.

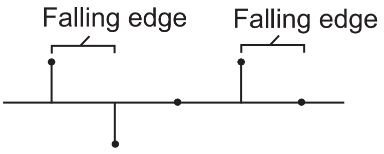
When you want to reset the value of the filter weights to their initial values, use the **Reset input** parameter. The block resets the filter weights whenever a reset event is detected at the Reset port. The reset signal rate must be the same rate as the data signal input.

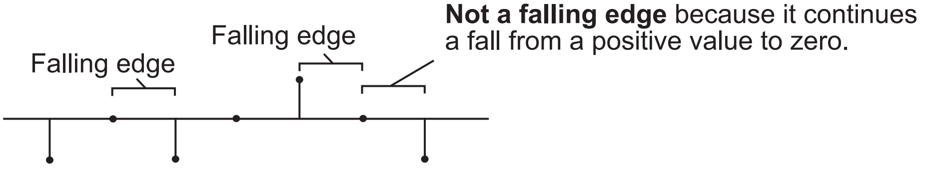
From the **Reset input** list, select `None` to disable the Reset port. To enable the Reset port, select one of the following from the **Reset input** list:

- `Rising edge` — Triggers a reset operation when the Reset input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure).



- `Falling edge` — Triggers a reset operation when the Reset input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)

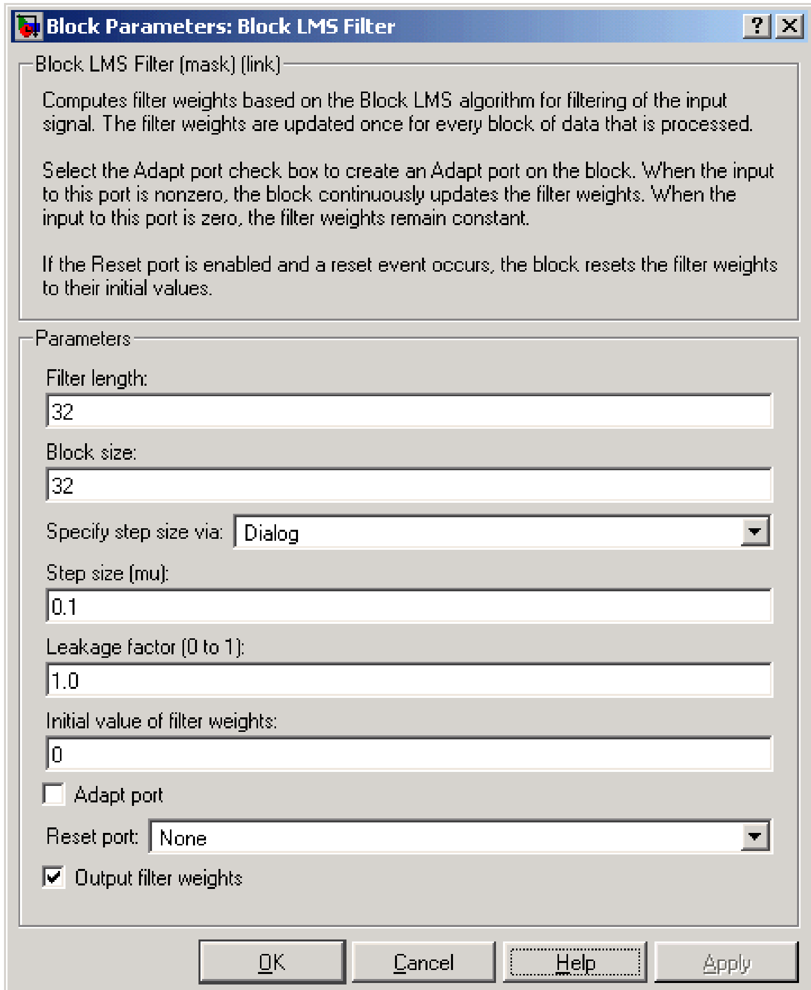




- `Either edge` — Triggers a reset operation when the Reset input is a `Rising edge` or `Falling edge` (as described above)
- `Non-zero sample` — Triggers a reset operation at each sample time that the Reset input is not zero

Select the **Output filter weights** check box to create a `Wts` port on the block. For each iteration, the block outputs the current updated filter weights from this port.

Dialog Box



Filter length

Enter the length of the FIR filter weights vector.

Block size
: Enter the number of samples to acquire before the filter weights are updated. The input frame length must be an integer multiple of the block size.

Specify step-size via
: Select `Dialog` to enter a value for mu in the Block parameters: LMS Filter dialog box. Select `Input port` to specify mu using the Step-size input port.

Step-size (mu)
: Enter the step-size. Tunable.

Leakage factor (0 to 1)
: Enter the leakage factor, $0 < 1 - \mu\alpha \leq 1$. Tunable.

Initial value of filter weights
: Specify the initial values of the FIR filter weights.

Adapt port
: Select this check box to enable the Adapt input port.

Reset port
: Select this check box to enable the Reset input port.

Output filter weights
: Select this check box to export the filter weights from the Wts port.

References

Hayes, M. H. *Statistical Digital Signal Processing and Modeling.* New York: John Wiley & Sons, 1996.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point
Desired	• Must be the same as Input for floating-point signals • Must be any fixed-point data type when Input is fixed point
Step-size	• Must be the same as Input for floating-point signals • Must be any fixed-point data type when Input is fixed point
Adapt	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Reset	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Must be the same as Input for floating-point signals • Must be the same as Desired for fixed-point signals

Port	Supported Data Types
Error	• Must be the same as Input for floating-point signals • Must be the same as Desired for fixed-point signals
Wts	• Must be the same as Input for floating-point signals • Obeys the **Weights** parameter for fixed-point signals

See Also

Fast Block LMS Filter	Signal Processing Blockset
Kalman Adaptive Filter (Obsolete)	Signal Processing Blockset
LMS Filter	Signal Processing Blockset
RLS Filter	Signal Processing Blockset

See "Adaptive Filters" for related information.

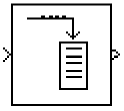
Buffer

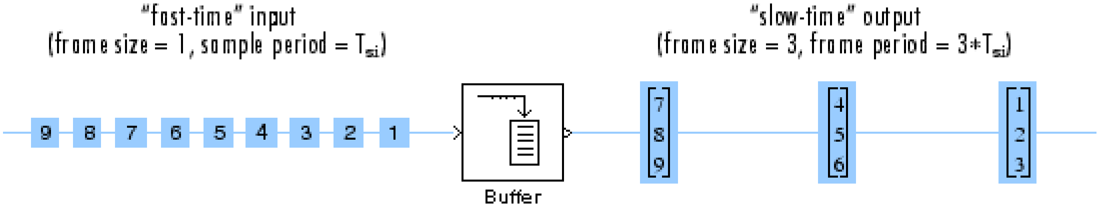
Purpose Buffer input sequence to smaller or larger frame size

Library Signal Management / Buffers

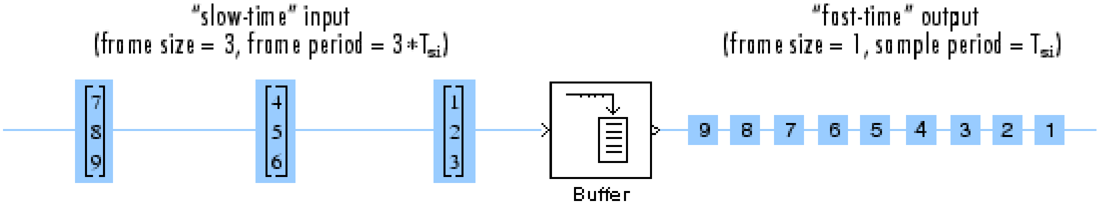
`dspbuff3`

Description The Buffer block redistributes the input samples to a new frame size. Buffering to a larger frame size yields an output with a *slower* frame rate than the input, as illustrated below for scalar input.





Buffering to a smaller frame size yields an output with a *faster* frame rate than the input, as illustrated below for scalar output.



The block coordinates the output *frame size* and *frame rate* of nonoverlapping buffers such that the sample period of the signal is the same at both the input and output: $T_{so} = T_{si}$.

This block supports triggered subsystems when the block's input and output rates are the same.

Sample-based full-dimension matrix inputs are not accepted.

Sample-Based Operation

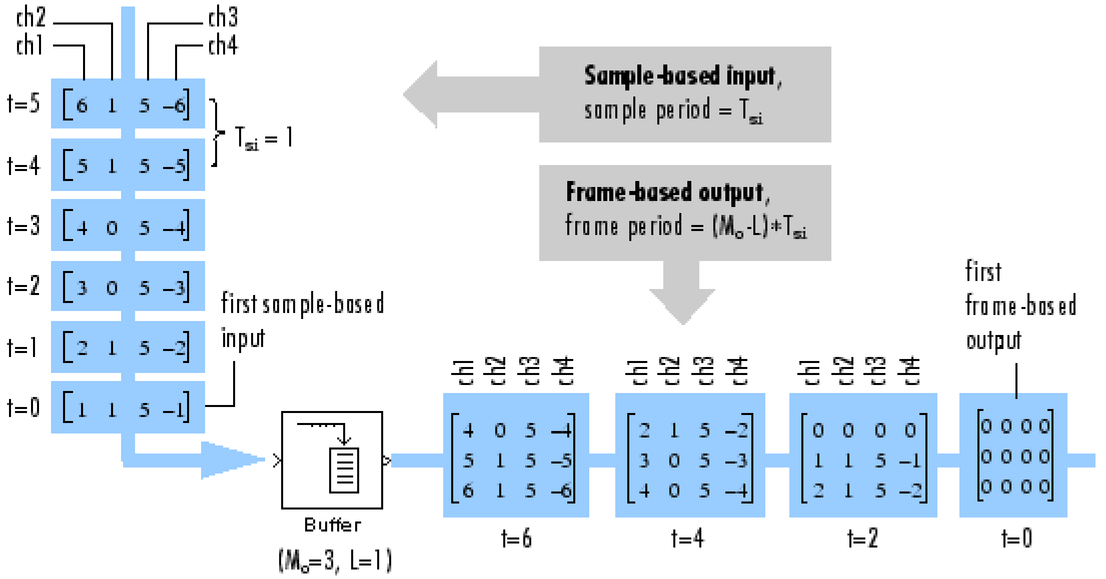
Sample-based inputs are interpreted by the Buffer block as independent channels of data. Therefore, a sample-based length-N vector input is interpreted as N independent samples.

In sample-based operation, the Buffer block creates frame-based outputs from sample-based inputs. A sequence of sample-based length-N vector inputs is buffered into an M_o-by-N matrix, where M_o is specified by the **Output buffer size** parameter (M_o>1). That is, each input vector becomes a *row* in the N-channel frame-based output matrix. When M_o=1, the input is simply passed through to the output, and retains the same dimension.

The **Buffer overlap** parameter, L, specifies the number of samples (rows) from the current output to repeat in the next output, where $L < M_o$. For $0 \leq L < M_o$, the number of *new* input samples that the block acquires before propagating the buffered data to the output is the difference between the **Output buffer size** and **Buffer overlap**, or $M_o - L$.

The output frame period is $(M_o - L)T_{si}$, which is equal to the input sequence sample period, T_{si}, when the **Buffer overlap** is $M_o - 1$. For $L < 0$, the block simply discards L input samples after the buffer fills, and outputs the buffer with period $(M_o - L)T_{si}$, which is longer than the zero-overlap case.

In the model below, the block buffers a four-channel sample-based input using a **Output buffer size** of three and a **Buffer overlap** of one.



Notice that the input vectors do not begin appearing at the output until the second row of the second matrix. This is due to the block's latency. The first output matrix (all zeros in this example) reflects the block's **Initial conditions** setting, while the first row of zeros in the second output is a result of the one-sample overlap between consecutive output frames.

You can use the `rebuffer_delay` function with a frame size of 1 to precisely compute the delay (in samples) for sample-based signals. For the previous example,

```
d = rebuffer_delay(1,3,1)
d =
     4
```

This agrees with the four samples of delay (zeros) per channel shown in the previous figure.

Frame-Based Operation

In frame-based operation, the Buffer block redistributes the samples in the input frame to an output frame with a new size and rate. A sequence of M_i-by-N matrix inputs is buffered into a sequence of M_o-by-N frame-based matrix outputs, where M_o is the output frame size specified by the **Output buffer size** parameter. The output buffer size is the number of consecutive samples from the input frame that are buffered into the output frame. M_o can be greater or less than the input frame size, M_i. Each of the N input channels is buffered independently.

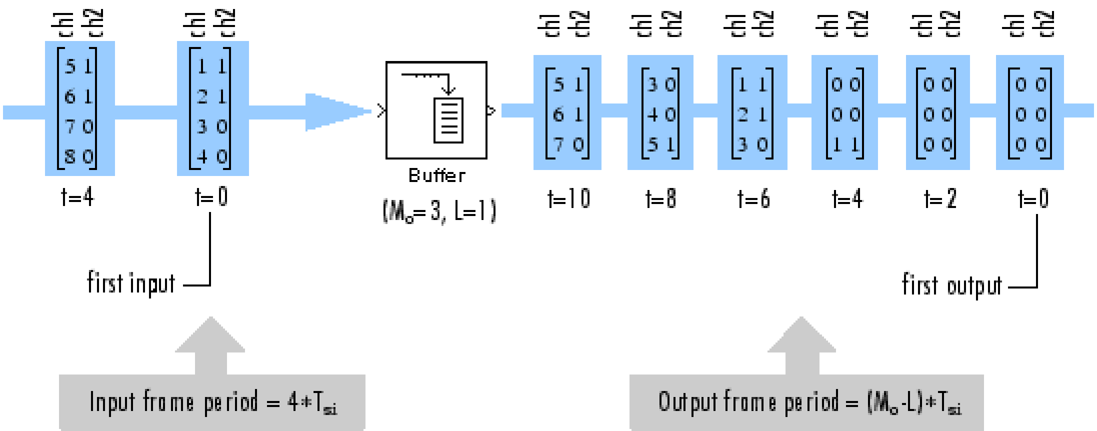
The **Buffer overlap** parameter, L, specifies the number of samples (rows) from the current output to repeat in the next output, where $L < M_o$. For $0 \le L < M_o$, the number of *new* input samples the block acquires before propagating the buffered data to the output is the difference between the **Output buffer size** and **Buffer overlap**, $M_o - L$.

The input frame period is $M_i T_{si}$, where T_{si} is the sample period. The output frame period is $(M_o - L)T_{si}$, which is equal to the sequence sample period when the **Buffer overlap** is $M_o - 1$. The output sample period is therefore related to the input sample period by

$$T_{so} = \frac{(M_o - L)T_{si}}{M_i}$$

Negative **Buffer overlap** values are not permitted.

In the model below, the block buffers a two-channel frame-based input using a **Output buffer size** of three and a **Buffer overlap** of one.



Notice that the sequence is delayed by eight samples, which is the latency of the block in the Simulink multitasking mode for the parameter settings of this example. The first eight output samples therefore adopt the value specified for the **Initial conditions**, which is assumed here to be zero. Use the `rebuffer_delay` function to determine the block's latency for any combination of frame size and overlap.

Latency

Zero-tasking latency means that the first input sample, received at t = 0, appears as the first output sample. In the Simulink single-tasking mode, the Buffer block has zero-tasking latency for the following special cases:

- Scalar input and output ($M_o = M_i = 1$) with zero or negative **Buffer overlap** ($L \le 0$)
- Input frame size is an integer multiple of the output frame size

  $$M_i = kM_o$$

  where k is an integer with zero **Buffer overlap** ($L = 0$); notable cases of this include

- Any input frame size M_i with scalar output ($M_o = 1$) and zero **Buffer overlap** ($L = 0$)
- Equal input and output frame sizes ($M_o = M_i$) with zero **Buffer overlap** ($L = 0$)

For all cases of *sample-based single-tasking* operation other than those listed above, the Buffer block's buffer is initialized to the value(s) specified by the **Initial conditions** parameter, and the block reads from this buffer to generate the first D output samples, where

$$D = \begin{cases} M_o + L & (L \geq 0) \\ M_o & (L < 0) \end{cases}$$

The dimensions of the **Initial conditions** parameter depend on the **Buffer overlap**, L, and whether the input is single-channel or multichannel:

- When $L \neq 0$, the **Initial conditions** parameter must be a scalar.
- When $L = 0$, the **Initial conditions** parameter can be a scalar, or it can be a vector with the following constraints:
  - For single-channel inputs, the **Initial conditions** parameter can be a vector of length M_o if M_i is 1, or a vector of length M_i if M_o is 1.
  - For multichannel inputs, the **Initial conditions** parameter can be a vector of length M_o * N if M_i is 1, or a vector of length M_i * N if M_o is 1.

For *frame-based single-tasking* operation and all *multitasking* operation, use the `rebuffer_delay` function to compute the exact delay, in samples, that the Buffer block introduces for a given combination of buffer size and buffer overlap.

For general buffering between arbitrary frame sizes, the **Initial conditions** parameter must be a scalar value, which is then repeated across all elements of the initial output(s). However, in the special case where the input is 1-by-N, and the block's output is therefore an M_o-by-N matrix, **Initial conditions** can be

- An M_o-by-N matrix
- A length-M_o vector to be repeated across all columns of the initial output(s)
- A scalar to be repeated across all elements of the initial output(s)

In the special case where the output is 1-by-N , which is the result of unbuffering an M_i-by-N frame-based matrix, **Initial conditions** can be

- A vector containing M_i samples to output sequentially for each channel during the first M_i sample times
- A scalar to be repeated across all elements of the initial output(s)

Note For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop® User's Guide*.

Behavior in Enabled Subsystems

The Buffer block cannot be used in an enabled subsystem under the following conditions:

- In a multirate multitasking environment
- When the **Buffer overlap** parameter is set to a negative value

The Buffer block has an internal reservoir that temporarily stores data. When the Buffer block is used in an enabled subsystem, there is the possibility that the reservoir can overrun or underrun. The block implements safeguards against these occurrences.

Overrun occurs when more data enters into the buffer than it can hold. For example, consider buffering a sample-based input to a frame of size three with a buffer that accepts input every second and outputs every three seconds. If you place this buffer inside an enabled subsystem that is disabled every three seconds at t = 3s, t = 6s, and so on, the buffer accumulates data in its internal reservoir without being able to empty it. This condition results in overrun.

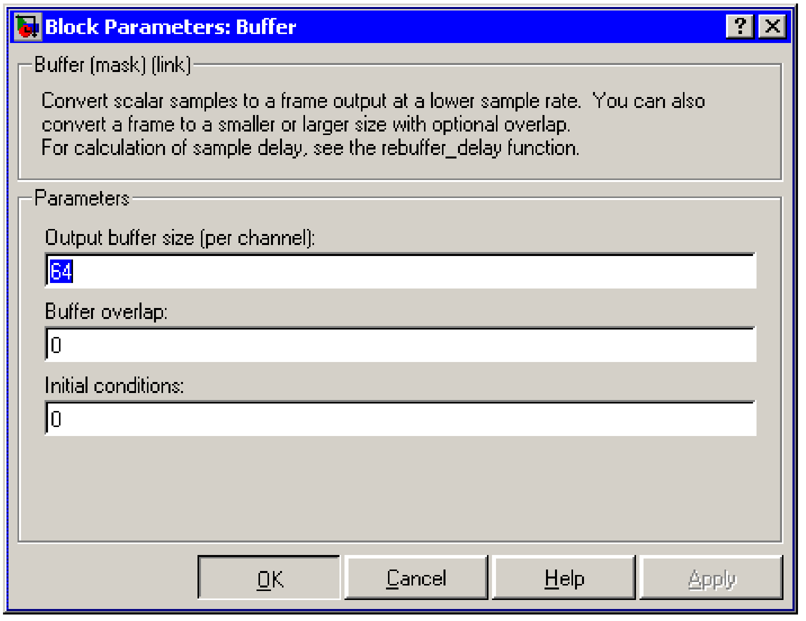
Underrun occurs when the buffer runs out of data to output. For example, again consider buffering a sample-based input to a frame size of three with a buffer that accepts input every second and outputs every three seconds. If you place this buffer inside an enabled subsystem that is disabled at t = 10s, t = 11s, t = 13s, t = 14s, t = 16s, and t = 17s, its internal reservoir becomes drained, and there is no data to output at t = 18s. This condition results in underrun.

To protect from overrun and underrun, the Buffer block keeps a record of the amount of data in its internal reservoir. When the Buffer block reads data, the amount of data in its reservoir goes up. When data is output from the Buffer block, the amount of data in its reservoir goes down. To protect from overrun, the oldest samples in the reservoir are discarded whenever amount of data in the reservoir is larger than the actual buffer size. To protect from underrun, the most recent samples are repeated whenever an output is due and there is no data in the reservoir.

Examples

See “Frame Rebuffering Blocks” and “Converting Frame Status” in the *Signal Processing Blockset User’s Guide*.

Dialog Box



Output buffer size

Specify the number of consecutive samples, M_o, from each channel to buffer into the output frame.

Buffer overlap

Specify the number of samples, L, by which consecutive output frames overlap.

Initial conditions

Specify the value of the block's initial output for cases of nonzero latency; a scalar, vector, or matrix.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Delay Line	Signal Processing Blockset
Unbuffer	Signal Processing Blockset
`rebuffer_delay`	Signal Processing Blockset

See "Converting Sample and Frame Rates" and "Converting Frame Status" for more information.

Burg AR Estimator

Purpose

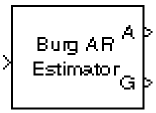
Compute estimate of autoregressive (AR) model parameters using Burg method

Library

Estimation / Parametric Estimation

`dspparest3`

Description



The Burg AR Estimator block uses the Burg method to fit an autoregressive (AR) model to the input data by minimizing (least squares) the forward and backward prediction errors while constraining the AR parameters to satisfy the Levinson-Durbin recursion.

The input is a sample-based vector (row, column, or 1-D) or frame-based vector (column only) representing a frame of consecutive time samples from a single-channel signal, which is assumed to be the output of an AR system driven by white noise. The block computes the normalized estimate of the AR system parameters, $A(z)$, independently for each successive input frame.

$$H(z) = \frac{G}{A(z)} = \frac{G}{1 + a(2)z^{-1} + \ldots + a(p+1)z^{-p}}$$

When you select the **Inherit estimation order from input dimensions** parameter, the order, p, of the all-pole model is one less that the length of the input vector. Otherwise, the order is the value specified by the **Estimation order** parameter.

The **Output(s)** parameter allows you to select between two realizations of the AR process:

- `A` — The top output, A, is a column vector of length p+1 with the same frame status as the input, and contains the normalized estimate of the AR model polynomial coefficients in descending powers of z.

  ```
  [1 a(2) ... a(p+1)]
  ```

- `K` — The top output, K, is a column vector of length p with the same frame status as the input, and contains the reflection coefficients (which are a secondary result of the Levinson recursion).
- `A and K` — The block outputs both realizations.

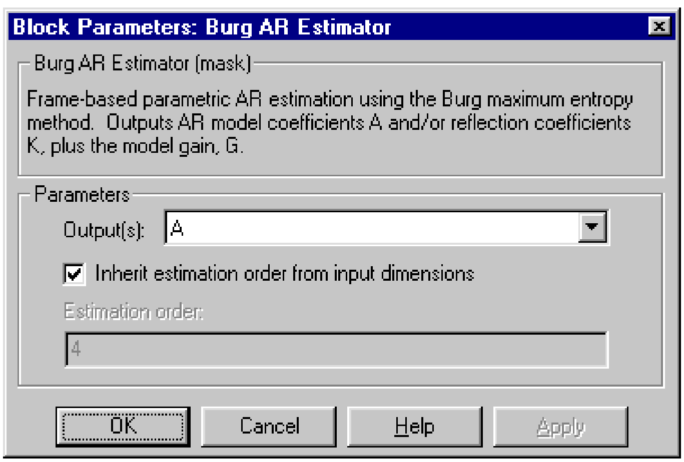
The scalar gain, G, is provided at the bottom output (`G`).

The following table compares the features of the Burg AR Estimator block to the Covariance AR Estimator, Modified Covariance AR Estimator, and Yule-Walker AR Estimator blocks.

	Burg AR Estimator	**Covariance AR Estimator**	**Modified Covariance AR Estimator**	**Yule-Walker AR Estimator**
`Characteristics`	Does not apply window to data	Does not apply window to data	Does not apply window to data	Applies window to data
	Minimizes the forward and backward prediction errors in the least squares sense, with the AR coefficients constrained to satisfy the L-D recursion	Minimizes the forward prediction error in the least squares sense	Minimizes the forward and backward prediction errors in the least squares sense	Minimizes the forward prediction error in the least squares sense (also called “autocorrelation method”)
`Advantages`	Always produces a stable model			Always produces a stable model

	Burg AR Estimator	Covariance AR Estimator	Modified Covariance AR Estimator	Yule-Walker AR Estimator
`Disadvantages`		May produce unstable models	May produce unstable models	Performs relatively poorly for short data records
`Conditions for Nonsingularity`		Order must be less than or equal to half the input frame size	Order must be less than or equal to 2/3 the input frame size	Because of the biased estimate, the autocorrelation matrix is guaranteed to positive-definite, hence nonsingular

Dialog Box



Output(s)
: The realization to output, model coefficients, reflection coefficients, or both.

Inherit estimation order from input dimensions
: When selected, sets the estimation order p to one less than the length of the input vector.

Estimation order
: The order of the AR model, p. This parameter is enabled when you do not select **Inherit estimation order from input dimensions**.

References

Kay, S. M. *Modern Spectral Estimation: Theory and Application.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

Marple, S. L., Jr., *Digital Spectral Analysis with Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1987.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point
A	• Double-precision floating point • Single-precision floating point
G	• Double-precision floating point • Single-precision floating point

See Also

Burg Method	Signal Processing Blockset
Covariance AR Estimator	Signal Processing Blockset
Modified Covariance AR Estimator	Signal Processing Blockset

Yule-Walker AR Estimator	Signal Processing Blockset
`arburg`	Signal Processing Toolbox

Purpose

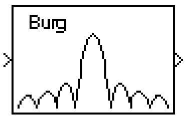
Power spectral density estimate using Burg method

Library

Estimation / Power Spectrum Estimation

`dspspect3`

Description



The Burg Method block estimates the power spectral density (PSD) of the input frame using the Burg method. This method fits an autoregressive (AR) model to the signal by minimizing (least squares) the forward and backward prediction errors. Such minimization occurs with the AR parameters constrained to satisfy the Levinson-Durbin recursion.

The input is a sample-based vector (row, column, or 1-D) or frame-based vector (column only). This input represents a frame of consecutive time samples from a single-channel signal. The block outputs a column vector containing the estimate of the power spectral density of the signal at N_{fft} equally spaced frequency points. The frequency points are in the range $[0,F_s)$, where F_s is the sampling frequency of the signal.

When you select the **Inherit estimation order from input dimensions** parameter, the order of the all-pole model is one less than the input frame size. Otherwise, the **Estimation order** parameter specifies the order. The block computes the spectrum from the FFT of the estimated AR model parameters.

Selecting the **Inherit FFT length from estimation order** parameter specifies that N_{fft} is one greater than the estimation order. Clearing the **Inherit FFT length from estimation order** check box, allows you to use the **FFT length** parameter to specify N_{fft} as a power of 2. The block zero-pads or wraps the input to N_{fft} before computing the FFT. The output is always sample based.

When you select the **Inherit sample time from input** check box, the block computes the frequency data from the sample period of the input signal. For the block to produce valid output, the following conditions must hold:

- The input to the block is the original signal, with no samples added or deleted (by insertion of zeros, for example).
- The sample period of the time-domain signal in the simulation equals the sample period of the original time series.

If these conditions do not hold, clear the **Inherit sample time from input** check box. You can then specify a sample time using the **Sample time of original time series** parameter.

The Burg Method and Yule-Walker Method blocks return similar results for large frame sizes. The following table compares the features of the Burg Method block to the Covariance Method, Modified Covariance Method, and Yule-Walker Method blocks.

	Burg	Covariance	Modified Covariance	Yule-Walker
Characteristics	Does not apply window to data	Does not apply window to data	Does not apply window to data	Applies window to data
	Minimizes the forward and backward prediction errors in the least squares sense, with the AR coefficients constrained to satisfy the L-D recursion	Minimizes the forward prediction error in the least squares sense	Minimizes the forward and backward prediction errors in the least squares sense	Minimizes the forward prediction error in the least squares sense (also called *autocorrelation method*)

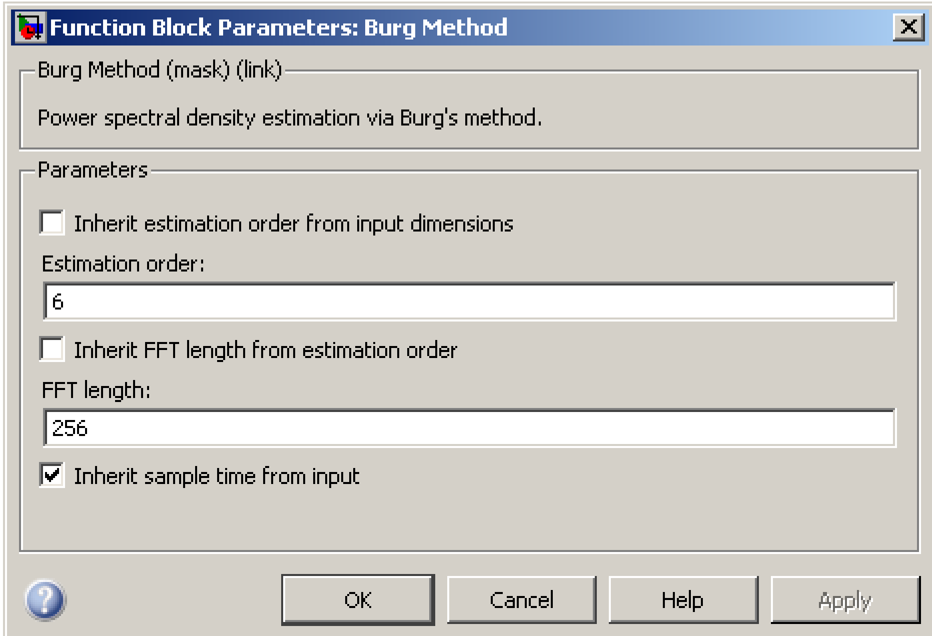
	Burg	Covariance	Modified Covariance	Yule-Walker
Advantages	High resolution for short data records	Better resolution than Y-W for short data records (more accurate estimates)	High resolution for short data records	Performs as well as other methods for large data records
	Always produces a stable model	Able to extract frequencies from data consisting of p or more pure sinusoids	Able to extract frequencies from data consisting of p or more pure sinusoids	Always produces a stable model
			Does not suffer spectral line-splitting	
Disadvantages	Peak locations highly dependent on initial phase	May produce unstable models	May produce unstable models	Performs relatively poorly for short data records
	May suffer spectral line-splitting for sinusoids in noise, or when order is very large	Frequency bias for estimates of sinusoids in noise	Peak locations slightly dependent on initial phase	Frequency bias for estimates of sinusoids in noise
	Frequency bias for estimates of sinusoids in noise		Minor frequency bias for estimates of sinusoids in noise	

	Burg	Covariance	Modified Covariance	Yule-Walker
Conditions for Nonsingularity		Order must be less than or equal to half the input frame size	Order must be less than or equal to 2/3 the input frame size	Because of the biased estimate, the autocorrelation matrix is guaranteed to be positive-definite, hence nonsingular

Examples

The `dspsacomp` demo compares the Burg method with several other spectral estimation methods.

Dialog Box



Inherit estimation order from input dimensions
Selecting this check box sets the estimation order to one less than the length of the input vector.

Estimation order
The order of the AR model. This parameter becomes visible only when you clear the **Inherit estimation order from input dimensions** check box.

Inherit FFT length from estimation order
When selected, the FFT length is one greater than the estimation order. To specify the number of points on which to perform the FFT, clear the **Inherit FFT length from estimation order** check box. You can then specify a power-of-two FFT length using the **FFT length** parameter.

FFT length

Enter the number of data points on which to perform the FFT, N_{fft}. When N_{fft} is larger than the input frame size, the block zero-pads each frame as needed. When N_{fft} is smaller than the input frame size, the block wraps each frame as needed. This parameter becomes visible only when you clear the **Inherit FFT length from input dimensions** check box.

Inherit sample time from input

If you select the **Inherit sample time from input** check box, the block computes the frequency data from the sample period of the input signal. For the block to produce valid output, the following conditions must hold:

- The input to the block is the original signal, with no samples added or deleted (by insertion of zeros, for example).
- The sample period of the time-domain signal in the simulation equals the sample period of the original time series.

If these conditions do not hold, clear the **Inherit sample time from input** check box. You can then specify a sample time using the **Sample time of original time series** parameter.

Sample time of original time series

Specify the sample time of the original time-domain signal. This parameter becomes visible only when you clear the **Inherit sample time from input** check box.

References

Kay, S. M. *Modern Spectral Estimation: Theory and Application.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

Orfanidis, S. J. *Introduction to Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1995.

Orfanidis, S. J. *Optimum Signal Processing: An Introduction.* 2nd ed. New York, NY: Macmillan, 1985.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point
Output	• Double-precision floating point • Single-precision floating point

See Also

Burg AR Estimator	Signal Processing Blockset
Covariance Method	Signal Processing Blockset
Modified Covariance Method	Signal Processing Blockset
Short-Time FFT	Signal Processing Blockset
Yule-Walker Method	Signal Processing Blockset
`spectrum.burg`	Signal Processing Toolbox

See “Power Spectrum Estimation” for related information.

Check Signal Attributes

Purpose

Error when input signal does or does not match selected attributes exactly

Library

Signal Management / Signal Attributes

`dspsigattribs`

Description

The Check Signal Attributes block terminates the simulation with an error when the input characteristics differ from the characteristics you specify in the block parameters.

When you set **Error when input** to `Does not match attributes exactly`, the block generates an error when the input fails to match *any* of the specified attributes. Only signals that possess *all* of the specified attributes propagate to the output unaltered and do not cause the block to generate an error.

When you set **Error when input** to `Matches attributes exactly`, the block generates an error only when the input possesses *all* specified attributes. Signals that do not possess *all* of the specified attributes propagate to the output unaltered, and do not cause the block to generate an error.

Signal Attributes

The Check Signal Attributes block can test for up to five different signal attributes, as specified by the following parameters. When you select `Ignore` for any parameter, the block does not check the signal for the corresponding attribute. For example, when you set **Complexity** to `Ignore`, neither real nor complex inputs cause the block to generate an error. The attributes are:

- **Complexity**

  Check whether the input is real or complex. You can display this information in a model by attaching a Probe block with **Probe complex signal** selected. Alternatively, you can select **Port Data Types** from the **Format** > **Port/Signal Displays** menu.

- **Frame status**

Check whether the signal is frame based or sample based. The Simulink environment displays sample-based signals using a single line and frame-based signals using a double line.

- **Dimensionality**

  Check the dimensionality of the input for compliance or noncompliance with the attributes in the subordinate **Dimension** menu. See the following table. *M* and *N* are positive integers unless otherwise indicated.

Dimensions	Is...	Is not...
1-D	1-D vector, 1-D scalar	*M*-by-*N* matrix, 1-by-*N* matrix (row vector), *M*-by-1 matrix (column vector), 1-by-1 matrix (2-D scalar)
2-D	*M*-by-*N* matrix, 1-by-*N* matrix (row vector), *M*-by-1 matrix (column vector), 1-by-1 matrix (2-D scalar)	1-D vector, 1-D scalar
Scalar (1-D or 2-D)	1-D scalar, 1-by-1 matrix (2-D scalar)	1-D vector with length>1, *M*-by-*N* matrix with *M*>1 and/or *N*>1
Vector (1-D or 2-D)	1-D vector, 1-D scalar, 1-by-*N* matrix (row vector), *M*-by-1 matrix (column vector), 1-by-1 matrix (2-D scalar) Vector (1-D or 2-D) or scalar	*M*-by-*N* matrix with *M*>1 and *N*>1
Row Vector (2-D)	1-by-*N* matrix (row vector), 1-by-1 matrix (2-D scalar) Row vector (2-D) or scalar	1-D vector, 1-D scalar, *M*-by-*N* matrix with *M*>1
Column Vector (2-D)	*M*-by-1 matrix (column vector), 1-by-1 matrix (2-D scalar) Column vector (2-D) or scalar	1-D vector, 1-D scalar, *M*-by-*N* matrix with *N*>1

Dimensions	Is...	Is not...
Full matrix	M-by-N matrix with M>1 and N>1	1-D vector, 1-D scalar, 1-by-N matrix (row vector), M-by-1 matrix (column vector), 1-by-1 matrix (2-D scalar)
Square matrix	M-by-N matrix with M=N, 1-D scalar, 1-by-1 matrix (2-D scalar	M-by-N matrix with M≠N, 1-D vector, 1-by-N matrix (row vector), M-by-1 matrix (column vector)

When you select **Signal Dimensions** from the **Format** > **Port/Signal Displays** menu, Simulink displays the size of a 1-D vector signal as an unbracketed integer, and displays the dimension of a 2-D signal as a pair of bracketed integers, `[MxN]`. Simulink *does not display* any size information for a 1-D or 2-D scalar signal. You can also display dimension information for a signal in a model by attaching a Probe block with **Probe signal dimensions** selected.

- **Data type**

  Check the signal data type for compliance (`Is...`) or noncompliance (`Is not...`) with the attributes in the subordinate **General data type** menu. See the following table. You can individually select any of the specific data types listed in the `Is...` column from the subordinate **Specific data type** menu.

General Data Type	Is...	Is not...
Boolean	`boolean`	`single`, `double`, `uint8`, `int8`, `uint16`, `int16`, `uint32`, `int32`, fixed point, enumerated
Enumerated	A user-defined enumerated data type. See "Using Enumerated Data" in the Simulink documentation.	`boolean`, `single`, `double`, `uint8`, `int8`, `uint16`, `int16`, `uint32`, `int32`, fixed point
Floating point	`single`, `double`	`boolean`, `uint8`, `int8`, `uint16`, `int16`, `uint32`, `int32`, fixed point, enumerated
Floating point or Boolean	`single`, `double`, `boolean`	`uint8`, `int8`, `uint16`, `int16`, `uint32`, `int32`, fixed point, enumerated
Fixed point	fixed point, `uint8`, `int8`, `uint16`, `int16`, `uint32`, `int32`	`boolean`, `single`, `double`, enumerated
Integer	Signed integer `int8`, `int16`, `int32` Unsigned integer `uint8`, `uint16`, `uint32`	`boolean`, `single`, `double`, fixed point, enumerated

To display data type information, in your model window, from the **Format** menu, point to **Port/Signal Displays** and select **Port Data Types**.

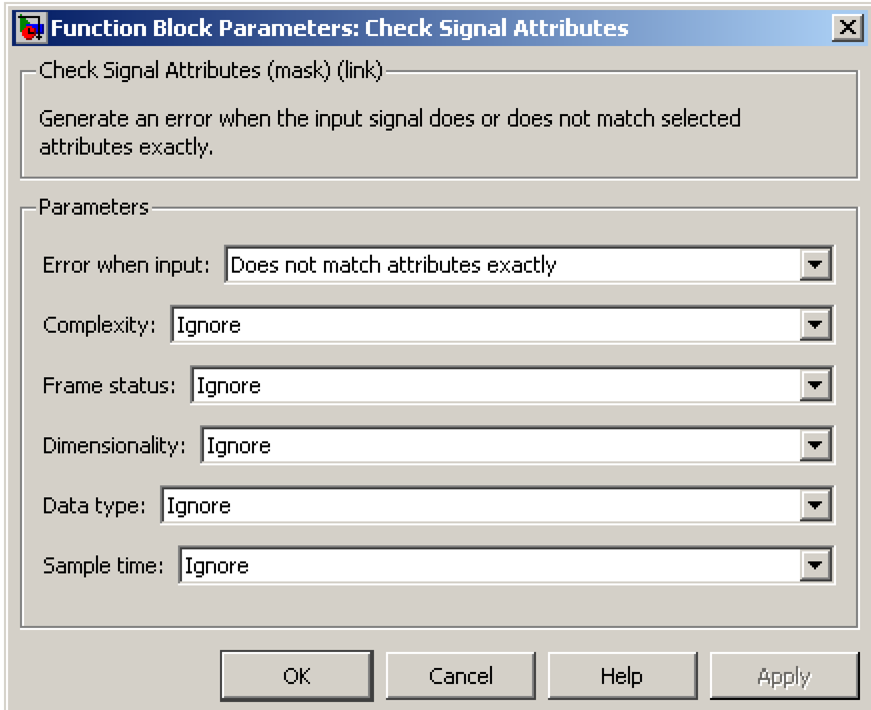
- **Sample time**

  Check whether the signal is discrete time or continuous time. When you select **Colors** from the **Format** > **Sample Time Display** menu, Simulink displays continuous-time signal lines in black or grey and discrete-time signal lines in colors corresponding to the relative rate.

  When you attach a Probe block with **Probe sample time** enabled to a continuous-time signal, the block icon displays the string `Ts:[0 To]`, where `To` is the sample time offset. Valid values of `To` for continuous-time signals are `0` and `1`. When `To` is `0`, updates occur at every major and minor time step. When `To` is `1`, updates occur only at major time steps and the sample time is *fixed in minor time step*.

  When you attach a Probe block with **Probe sample time** enabled to a discrete-time signal, the block icon displays the string `Ts:[Ts To]` for sample-based signals, and `Tf:[Tf To]` for frame-based signals. `Ts` and `Tf` are the positive sample period and frame period, respectively. `To` is the offset, such that 0 ≤ *`offset`* < *`period`*. Frame-based signals are almost always discrete time.

Dialog Box



Error when input

Specify whether the block generates an error when the input *does* or *does not* possess *all* of the required attributes.

Complexity

Specify the complexity for which you want to check the input, `Real` or `Complex`. When you select `Ignore` from the list, the block does not check the complexity of the input.

Frame status

Specify the frame status for which you want to check the input, `Sample-based` or `Frame-based`. When you select `Ignore` from the list, the block does not check the frame status of the input.

Dimensionality
Specify whether you want to check the input for compliance or noncompliance with the attributes in the subordinate **Dimensions** menu. When you select `Ignore` from the list, the block does not check the dimensionality of the input.

Dimensions
Specify the dimensions for which you want to check the input. This parameter is only visible when you set the **Dimensionality** parameter to `Is...` or `Is not...`.

Data type
Specifies whether you want to check the input for compliance or noncompliance with the attributes in the subordinate **General data type** menu. When you select `Ignore` from the list, the block does not check the input data type.

General data type
Specify the general data type for which you want to check the input. This parameter is only visible when you set the **Data type** to `Is...` or `Is not...`.

Specific floating-point
Specify the floating-point data type for which you want to check the input. This parameter is only visible when you set the **General data type** to `Floating-point` or `Floating-point or boolean`.

Specific fixed-point
Specify the fixed-point data type for which you want to check the input. This parameter is only visible when you set the **General data type** to `Fixed-point`.

Specific integer
Specify the integer data type for which you want to check the input. This parameter is only visible when you set the **General data type** to `Integer`.

Sample time

Specify the sample time for which you want to check the input, `Discrete` or `Continuous`. When you select `Ignore` from the list, the block does not check the sample time of the input.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8, 16, and 32-bit signed integers • 8, 16, and 32-bit unsigned integers • Enumerated
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8, 16, and 32-bit signed integers • 8, 16, and 32-bit unsigned integers • Enumerated

See Also

Buffer	Signal Processing Blockset
Convert 1-D to 2-D	Signal Processing Blockset
Convert 2-D to 1-D	Signal Processing Blockset
Data Type Conversion	Simulink

Frame Status Conversion (Obsolete)	Signal Processing Blockset
Inherit Complexity	Signal Processing Blockset
Probe	Simulink
Reshape	Simulink
Submatrix	Signal Processing Blockset

Purpose

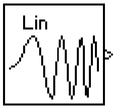
Generate swept-frequency cosine (chirp) signal

Library

Signal Processing Sources

`dspsrcs4`

Description



The Chirp block outputs a swept-frequency cosine (chirp) signal with unity amplitude and continuous phase. To specify the desired output chirp signal, you must define its instantaneous frequency function, also known as the output frequency sweep. The frequency sweep can be linear, quadratic, or logarithmic, and repeats once every **Sweep time** by default. See other sections of this reference page for more details about the block.

Sections of This Reference Page

- Variables Used in This Reference Page on page 2-146
- "Setting the Output Frame Status" on page 2-146
- "Shaping the Frequency Sweep by Setting Frequency Sweep and Sweep Mode" on page 2-147
- "Unidirectional and Bidirectional Sweep Modes" on page 2-148
- "Setting Instantaneous Frequency Sweep Values" on page 2-149
- "Block Computation Methods" on page 2-150
- "Cautions Regarding the Swept Cosine Sweep" on page 2-153
- "Dialog Box" on page 2-155
- "Examples" on page 2-157
- "Supported Data Types" on page 2-168
- "See Also" on page 2-168

Variables Used in This Reference Page

f_0	**Initial frequency** parameter (Hz)
$f_i(t_g)$	**Target frequency** parameter (Hz)
t_g	**Target time** parameter (seconds)
T_{sw}	**Sweep time** parameter (seconds)
ϕ_0	**Initial phase** parameter (radians)
$\psi(t)$	Phase of the chirp signal (radians)
$f_i(t)$	User-specified output instantaneous frequency function (Hz); user-specified sweep
$f_{i(actual)}(t)$	Actual output instantaneous frequency function (Hz); actual output sweep
$y_{chirp}(t)$	Output chirp function

Setting the Output Frame Status

Use **Samples per frame** parameter to set the block's output frame status, as summarized in the following table. The **Sample time** parameter sets the sample time of both sample- and frame-based outputs.

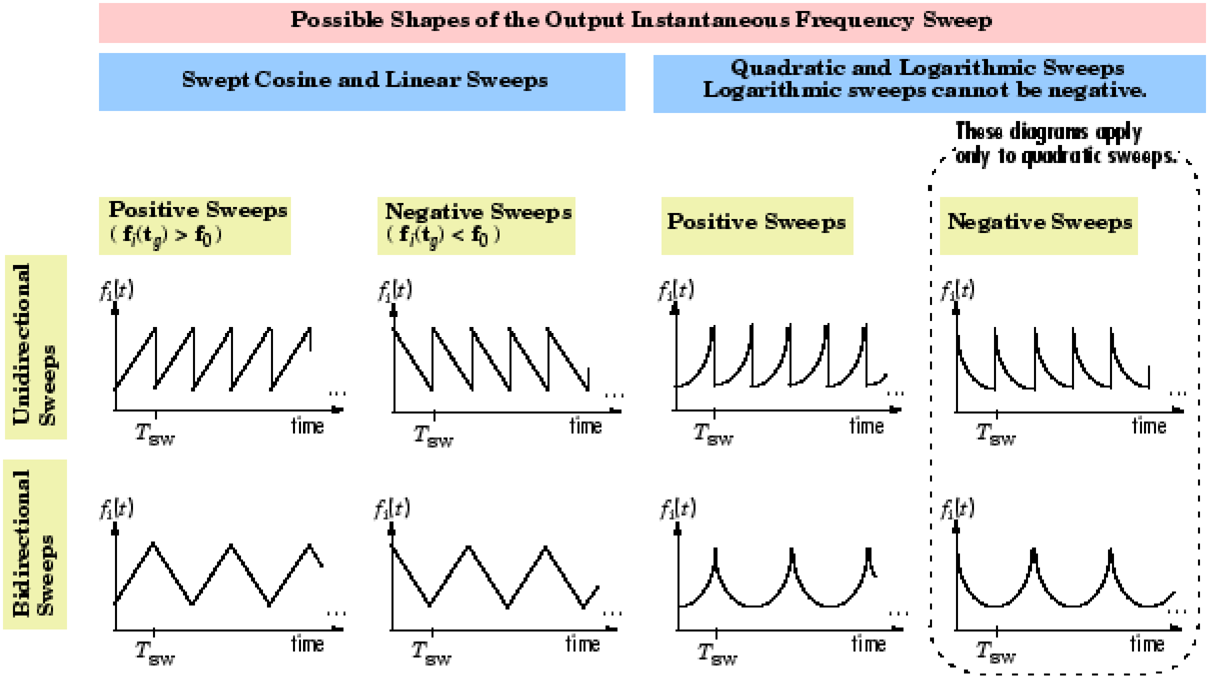
Setting of Samples Per Frame Parameter	Output Frame Status
1	Sample based
n (any integer greater than 1)	Frame based, frame size n

Shaping the Frequency Sweep by Setting Frequency Sweep and Sweep Mode

The basic shape of the output instantaneous frequency sweep, $f_i(t)$, is set by the **Frequency sweep** and **Sweep mode** parameters, described in the following table.

Parameters for Setting Sweep Shape	Possible Setting	Parameter Description
Frequency sweep	Linear Quadratic Logarithmic Swept cosine	Determines whether the sweep frequencies vary linearly, quadratically, or logarithmically. Linear and swept cosine sweeps both vary linearly.
Sweep mode	Unidirectional Bidirectional	Determines whether the sweep is unidirectional or bidirectional. For details, see "Unidirectional and Bidirectional Sweep Modes" on page 2-148

The following diagram illustrates the possible shapes of the frequency sweep that you can obtain by setting the **Frequency sweep** and **Sweep mode** parameters.



For information on how to set the frequency values in your sweep, see "Setting Instantaneous Frequency Sweep Values" on page 2-149.

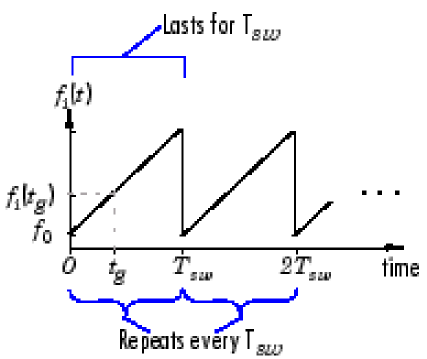
Unidirectional and Bidirectional Sweep Modes

The **Sweep mode** parameter determines whether your sweep is unidirectional or bidirectional, which affects the shape of your output frequency sweep (see "Shaping the Frequency Sweep by Setting Frequency Sweep and Sweep Mode" on page 2-147). The following table describes the characteristics of unidirectional and bidirectional sweeps.

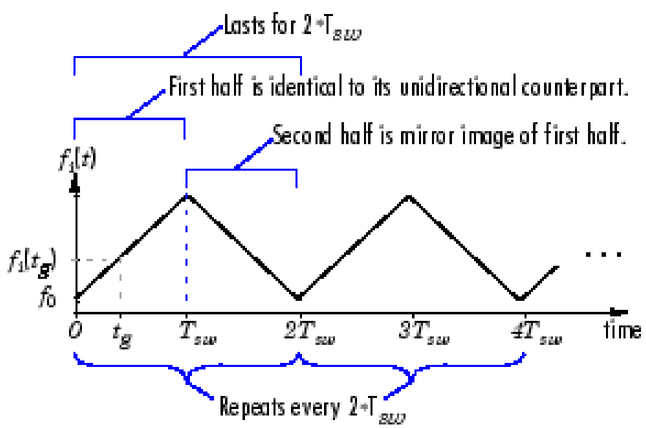
Sweep Mode Parameter Settings	Sweep Characteristics
`Unidirectional`	• Lasts for one **Sweep time**, T_{sw} • Repeats once every T_{sw}
`Bidirectional`	• Lasts for twice the **Sweep time**, $2*T_{sw}$ • Repeats once every $2*T_{sw}$ • First half is identical to its unidirectional counterpart. • Second half is a mirror image of the first half.

The following diagram illustrates a linear sweep in both sweep modes. For information on setting the frequency values in your sweep, see "Setting Instantaneous Frequency Sweep Values" on page 2-149.

Unidirectional Linear Sweep



Bidirectional Linear Sweep



Setting Instantaneous Frequency Sweep Values

Set the following parameters to tune the frequency values of your output frequency sweep. Note that because this is a source block, the simulation pauses while the block dialog box is open. You must close the dialog box by clicking **OK** to resume the simulation.

- **Initial frequency** (Hz), f_0
- **Target frequency** (Hz), $f_i(t_g)$
- **Target time** (seconds), t_g

The following table summarizes the sweep values at specific times for all **Frequency sweep** settings. For information on the formulas used to compute sweep values at other times, see "Block Computation Methods" on page 2-150.

Instantaneous Frequency Sweep Values

Frequency Sweep	Sweep Value at $t = 0$	Sweep Value at $t = t_g$	Time when Sweep Value Is Target Frequency, $f_i(t_g)$
Linear	f_0	$f_i(t_g)$	t_g
Quadratic	f_0	$f_i(t_g)$	t_g
Logarithmic	f_0	$f_i(t_g)$	t_g
Swept cosine	f_0	$2f_i(t_g) - f_0$	$t_g/2$

Block Computation Methods

The Chirp block uses one of two formulas to compute the block output, depending on the **Frequency Sweep** parameter setting. For details, see the following sections:

- "Equations for Output Computation" on page 2-151
- "Output Computation Method for Linear, Quadratic, and Logarithmic Frequency Sweeps" on page 2-152

- "Output Computation Method for Swept Cosine Frequency Sweep" on page 2-153

Equations for Output Computation

The following table shows the equations used by the block to compute the user-specified output frequency sweep, $f_i(t)$, the block output, $y_{chirp}(t)$, and the actual output frequency sweep, $f_{i(actual)}(t)$. The only time the user-specified sweep is not the actual output sweep is when the **Frequency sweep** parameter is set to `Swept cosine`.

Note The following equations apply only to unidirectional sweeps in which $f_i(0) < f_i(t_g)$. To derive equations for other cases, you might find it helpful to examine the following table and the diagram in "Shaping the Frequency Sweep by Setting Frequency Sweep and Sweep Mode" on page 2-147.

The table below contains the following variables:

- $f_i(t)$ — the user-specified frequency sweep
- $f_{i(actual)}(t)$ — the actual output frequency sweep, usually equal to $f_i(t)$
- $y(t)$ — the Chirp block output
- $\psi(t)$ — the phase of the chirp signal, where $\psi(0) = 0$, and $2\pi f_i(t)$ is the derivative of the phase

$$f_i(t) = \frac{1}{2\pi} \cdot \frac{d\psi(t)}{dt}$$

- ϕ_0 — the **Initial phase** parameter value, where $y_{chirp}(0) = \cos(\phi_0)$

Equations Used by the Chirp Block for Unidirectional Positive Sweeps

Frequency Sweep	Block Output Chirp Signal	User-Specified Frequency Sweep, $f_i(t)$	β	Actual Frequency Sweep, $f_{i(actual)}(t)$
`Linear`	$y(t) = \cos(\psi(t) + \phi_0)$	$f_i(t) = f_0 + \beta t$	$\beta = \dfrac{f_i(t_g) - f_0}{t_g}$	$f_{i(actual)}{}^{(t)} = f_i{}^{(t)}$
`Quadratic`	Same as `Linear`	$f_i(t) = f_0 + \beta t^2$	$\beta = \dfrac{f_i(t_g) - f_0}{2}$	$f_{i(actual)}{}^{(t)} = f_i{}^{(t)}$
`Logarithmic`	Same as `Linear`	$F_i(t) = f_0 \left(\dfrac{f_i(t_g)}{f_0} \right)^{\frac{t}{t_g}}$ Where $f_i(t_g) > f_0 > 0$	N/A	$f_{i(actual)}(t) = f_i(t)$
`Swept cosine`	$y(t) = \cos(2\pi f_i(t) t + \phi_0)$	Same as `Linear`	Same as `Linear`	$f_{i(actual)}(t) = f_i(t) + \beta t$

Output Computation Method for Linear, Quadratic, and Logarithmic Frequency Sweeps

The derivative of the phase of a chirp function gives the instantaneous frequency of the chirp function. The Chirp block uses this principle to calculate the chirp output when the **Frequency Sweep** parameter is set to `Linear`, `Quadratic`, or `Logarithmic`.

$$y_{chirp}(t) = \cos(\psi(t) + \phi_0)$$

Linear, quadratic, or logarithmic chirp signal with phase $\psi(t)$

$$f_i(t) = \frac{1}{2\pi} \cdot \frac{d\psi(t)}{dt}$$

Phase derivative is instantaneous frequency

For instance, if you want a chirp signal with a linear instantaneous frequency sweep, you should set the **Frequency Sweep** parameter to

Linear, and tune the linear sweep values by setting other parameters appropriately. Note that because this is a source block, the simulation pauses while the block dialog box is open. You must close the dialog box by clicking **OK** to resume the simulation. The block outputs a chirp signal, the phase derivative of which is the specified linear sweep. This ensures that the instantaneous frequency of the output is the linear sweep you desired. For equations describing the linear, quadratic, and logarithmic sweeps, see "Equations for Output Computation" on page 2-151.

Output Computation Method for Swept Cosine Frequency Sweep

To generate the swept cosine chirp signal, the block sets the swept cosine chirp output as follows.

$$y_{chirp}(t) = \cos(\psi(t) + \phi_0) = \cos(2\pi f_i(t)t + \phi_0)$$

Swept cosine chirp output (Instantaneous frequency equation, shown above, does not hold.)

Note that the instantaneous frequency equation, shown above, does not hold for the swept cosine chirp, so the user-defined frequency sweep, $f_i(t)$, is not the actual output frequency sweep, $f_{i(actual)}(t)$, of the swept cosine chirp. Thus, the swept cosine output might not behave as you expect. To learn more about swept cosine chirp behavior, see "Cautions Regarding the Swept Cosine Sweep" on page 2-153 and "Equations for Output Computation" on page 2-151.

Cautions Regarding the Swept Cosine Sweep

When you want a linearly swept chirp signal, we recommend you use a linear frequency sweep. Though a swept cosine frequency sweep also yields a linearly swept chirp signal, the output might have unexpected frequency content. For details, see the following two sections.

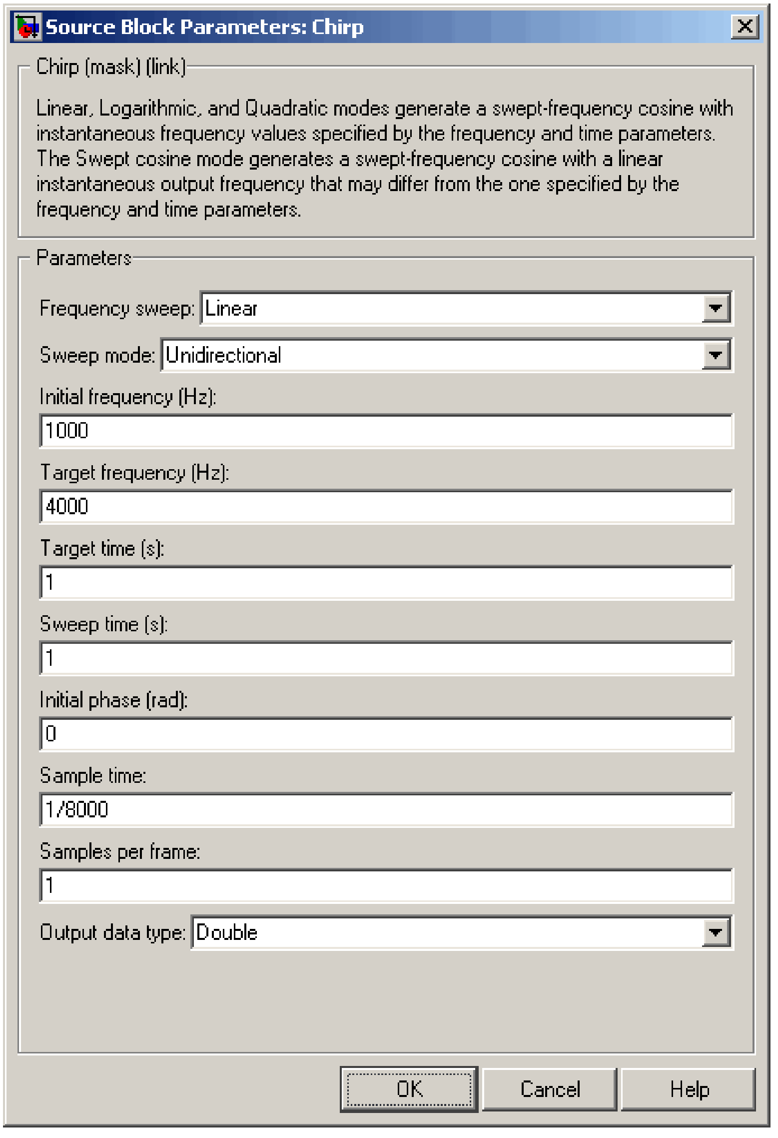
Swept Cosine Instantaneous Output Frequency at the Target Time is not the Target Frequency

The swept cosine sweep value at the **Target time** is not necessarily the **Target frequency**. This is because the user-specified sweep is not the actual frequency sweep of the swept cosine output, as noted in "Output Computation Method for Swept Cosine Frequency Sweep" on page 2-153. See the table Instantaneous Frequency Sweep Values on page 2-150 for the actual value of the swept cosine sweep at the **Target time**.

Swept Cosine Output Frequency Content May Greatly Exceed Frequencies in the Sweep

In **Swept cosine** mode, you should not set the parameters so that $1/T$sw is very large compared to the values of the **Initial frequency** and **Target frequency** parameters. In such cases, the actual frequency content of the swept cosine sweep might be closer to $1/T$sw, far exceeding the **Initial frequency** and **Target frequency** parameter values.

Dialog Box


Source Block Parameters: Chirp
Chirp (mask) (link)
Linear, Logarithmic, and Quadratic modes generate a swept-frequency cosine with instantaneous frequency values specified by the frequency and time parameters. The Swept cosine mode generates a swept-frequency cosine with a linear instantaneous output frequency that may differ from the one specified by the frequency and time parameters.
Parameters
Frequency sweep: Linear
Sweep mode: Unidirectional
Initial frequency (Hz):
1000
Target frequency (Hz):
4000
Target time (s):
1
Sweep time (s):
1
Initial phase (rad):
0
Sample time:
1/8000
Samples per frame:
1
Output data type: Double
OK
Cancel
Help

Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters During Simulation" in the online Simulink documentation for details.

Frequency sweep
: The type of output instantaneous frequency sweep, $f_i(t)$: `Linear`, `Logarithmic`, `Quadratic`, or `Swept cosine`.

Sweep mode
: The directionality of the chirp signal: `Unidirectional` or `Bidirectional`.

Initial frequency (Hz)
: For `Linear`, `Quadratic`, and `Swept cosine` sweeps, the initial frequency, f_0, of the output chirp signal. For `Logarithmic` sweeps, **Initial frequency** is one less than the actual initial frequency of the sweep. Also, when the sweep is `Logarithmic`, you must set the **Initial frequency** to be less than the **Target frequency**. Tunable.

Target frequency (Hz)
: For `Linear`, `Quadratic`, and `Logarithmic` sweeps, the instantaneous frequency, $f_i(t_g)$, of the output at the **Target time**, t_g. For a `Swept cosine` sweep, **Target frequency** is the instantaneous frequency of the output at half the **Target time**, $t_g/2$. When **Frequency sweep** is `Logarithmic`, you must set the **Target frequency** to be greater than the **Initial frequency**. Tunable.

Target time (s)
: For `Linear`, `Quadratic`, and `Logarithmic` sweeps, the time, t_g, at which the **Target frequency**, $f_i(t_g)$, is reached by the sweep. For a `Swept cosine` sweep, **Target time** is the time at which the sweep reaches $2f_i(t_g)$ - f_0. You must set **Target time** to be *no greater than* **Sweep time** , $T_{sw} \geq t_g$. Tunable.

Sweep time (s)
: In `Unidirectional` **Sweep mode**, the **Sweep time**, T_{sw}, is the period of the output frequency sweep. In `Bidirectional` **Sweep**

mode, the **Sweep time** is half the period of the output frequency sweep. You must set **Sweep time** to be no less than **Target time**, $T_{sw} \geq t_g$. Tunable.

Initial phase (rad)

The phase, ϕ_0 , of the cosine output at t=0; $y_{chirp}(t) = \cos(\phi_0)$. Tunable.

Sample time

The sample period, T_s, of the output. The output frame period is M_o*T_s.

Samples per frame

The number of samples, M_o, to buffer into each output frame. When the value of this parameter is 1, the block outputs a sample-based signal.

Output data type

The data type of the output, single-precision or double-precision.

Examples

The first few examples demonstrate how to use the Chirp block's main parameters, how to view the output in the time domain, and how to view the output spectrogram:

- "Example 1: Setting a Final Frequency Value for Unidirectional Sweeps" on page 2-158
- "Example 2: Bidirectional Sweeps" on page 2-161
- "Example 3: When Sweep Time is Greater Than Target Time" on page 2-163

Examples 4 and 5 illustrate Chirp block settings that might produce unexpected outputs:

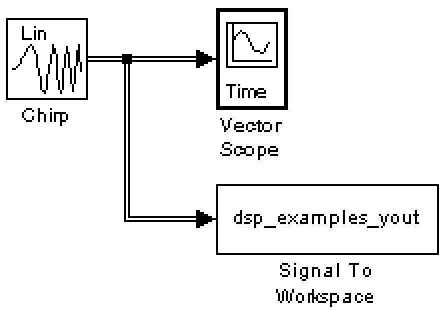
- "Example 4: Output Sweep with Negative Frequencies" on page 2-165
- "Example 5: Output Sweep with Frequencies Greater Than Half the Sampling Frequency" on page 2-166

Example 1: Setting a Final Frequency Value for Unidirectional Sweeps

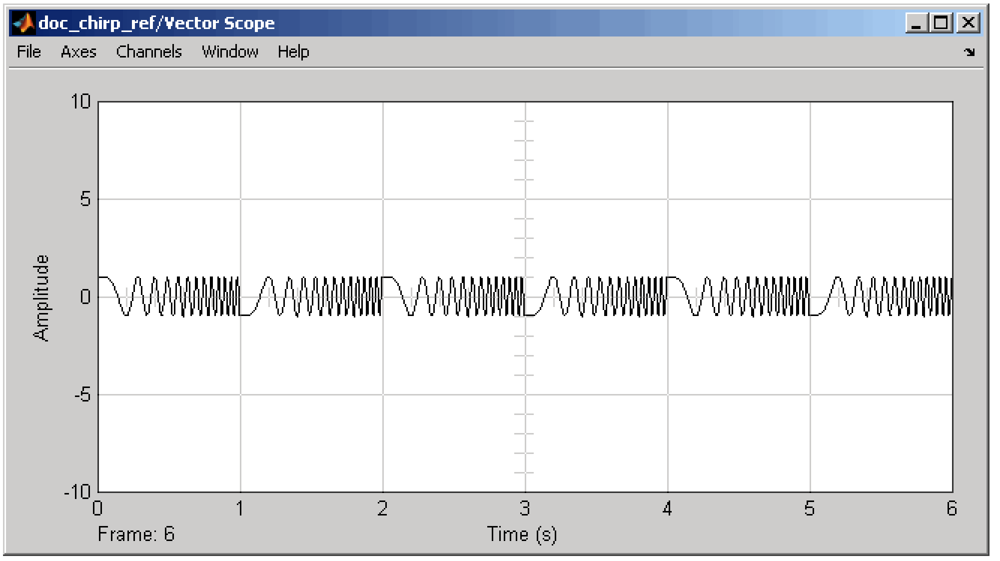
Often times, you might want a unidirectional sweep for which you know the initial and final frequency values. You can specify the final frequency of a unidirectional sweep by setting **Target time** equal to **Sweep time**, in which case the **Target frequency** becomes the final frequency in the sweep. The following model demonstrates this method.

This technique might not work for swept cosine sweeps. For details, see "Cautions Regarding the Swept Cosine Sweep" on page 2-153.

Open the Example 1 model by typing doc_chirp_ref at the MATLAB command line. You can also rebuild the model yourself; see the following list for model parameter settings (leave unlisted parameters in their default states).

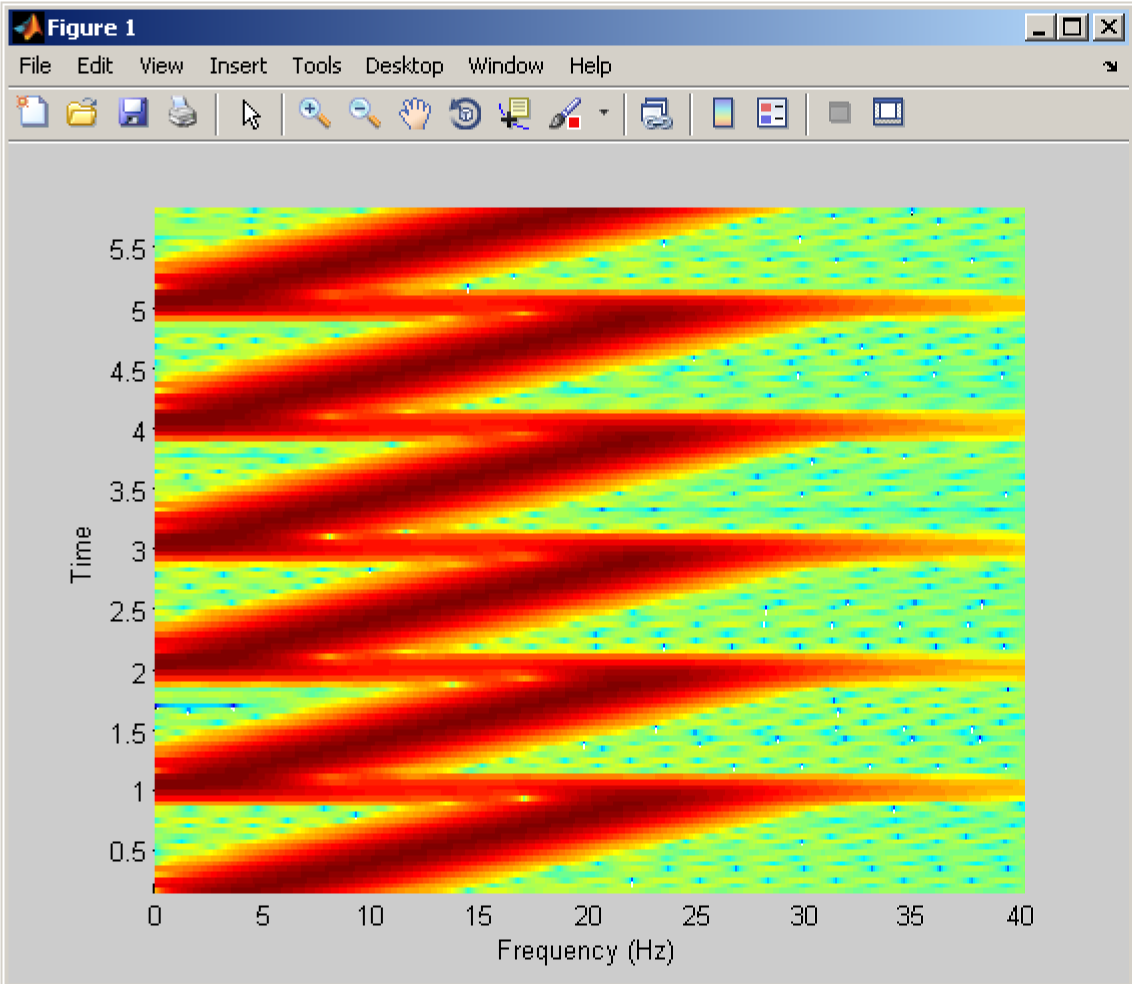


Since **Target time** is set to equal **Sweep time** (1 second), the **Target frequency** (25 Hz) is the final frequency of the unidirectional sweep. Run your model to see the time domain output:



Type the following command to view the chirp output spectrogram:

```
spectrogram(dsp_examples_yout,hamming(128),...
     110,[0:.01:40],400)
```



Chirp Block Parameters for Example 1	
Frequency sweep	`Linear`
Sweep mode	`Unidirectional`

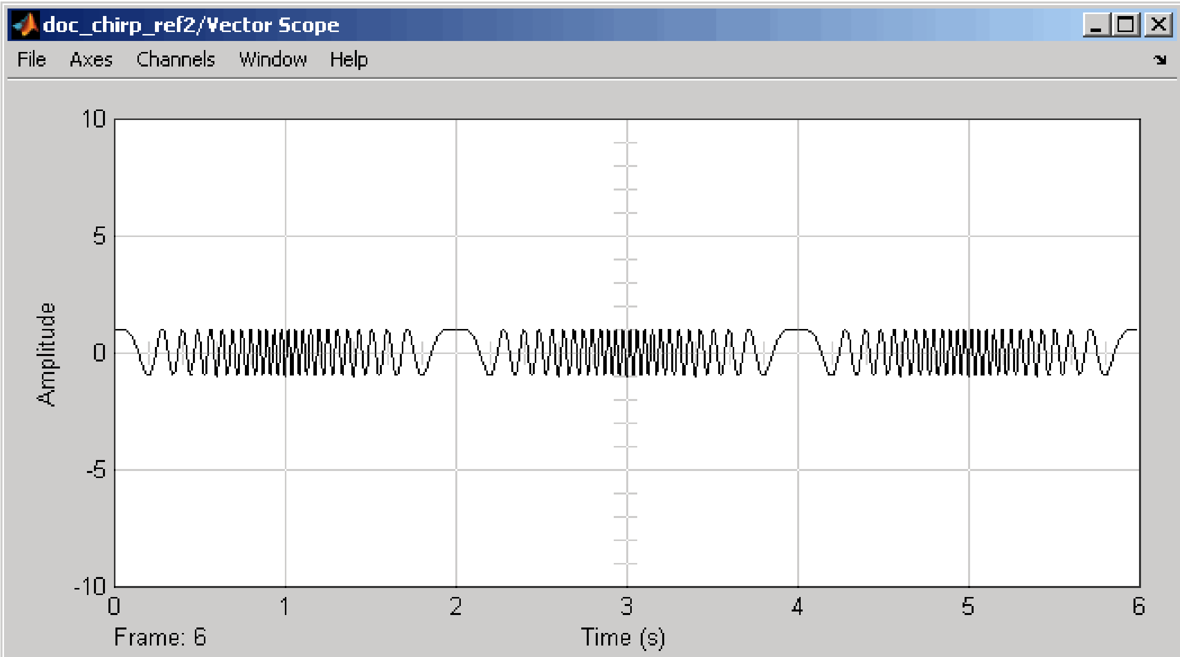
Initial frequency	0
Target frequency	25
Target time	1
Sweep time	1
Initial phase	0
Sample time	1/400
Samples per frame	400
Vector Scope Block Parameters for Example 1	
Input domain	`Time`
Time display span	6
Signal To Workspace Block Parameters for Example 1	
Variable name	`dsp_examples_yout`
Configuration Dialog Parameters for Example 1	
Stop time	5

Example 2: Bidirectional Sweeps

Change the **Sweep mode** parameter in the Example 1 model to `Bidirectional`, and leave all other parameters the same to view the following bidirectional chirp. Note that in the bidirectional sweep, the period of the sweep is twice the **Sweep time** (2 seconds), whereas it was one **Sweep time** (1 second) for the unidirectional sweep in Example 1.

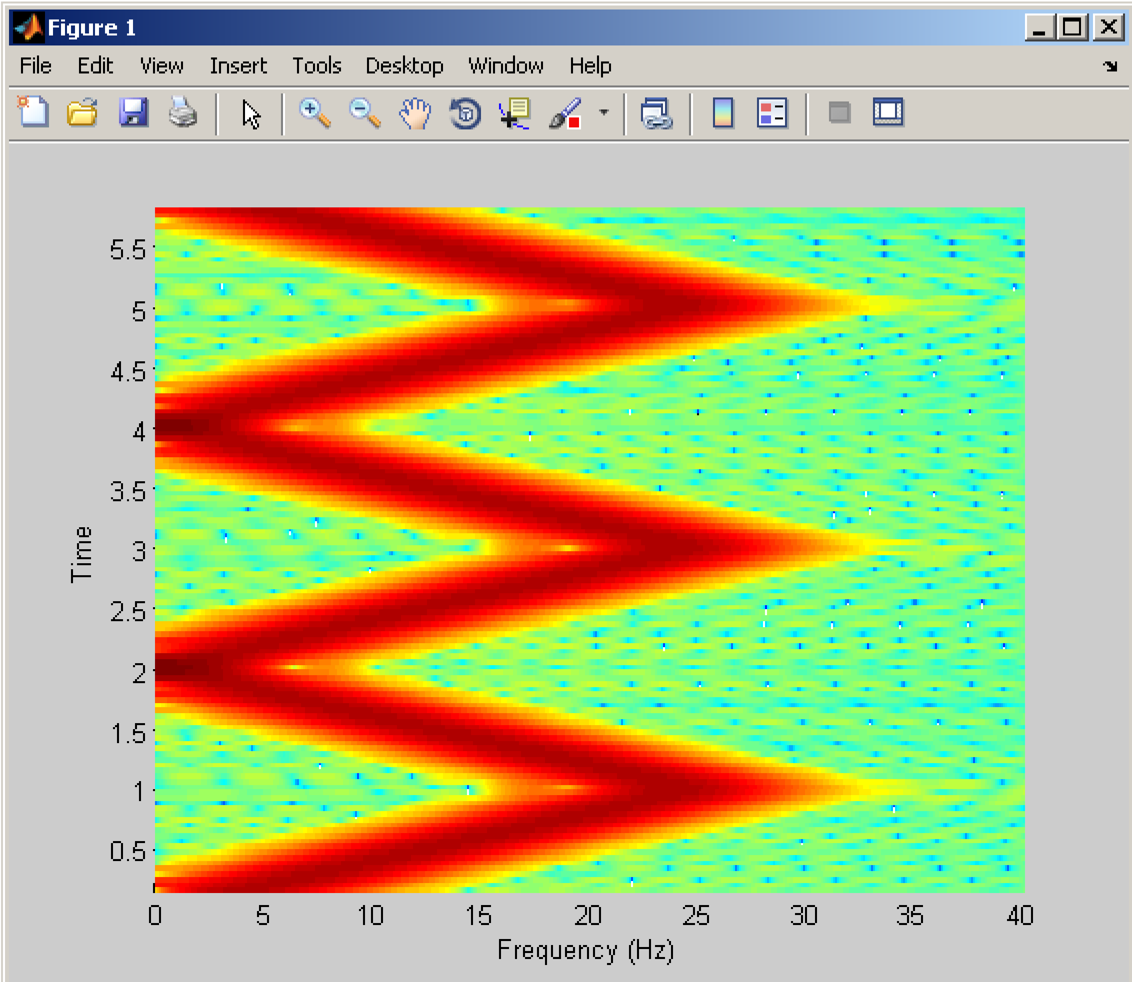
Open the Example 2 model by typing `doc_chirp_ref2` at the MATLAB command line.

Run your model to see the time domain output:



Type the following command to view the chirp output spectrogram:

```
spectrogram(dsp_examples_yout,hamming(128),...
        110,[0:.01:40],400)
```



Example 3: When Sweep Time is Greater Than Target Time

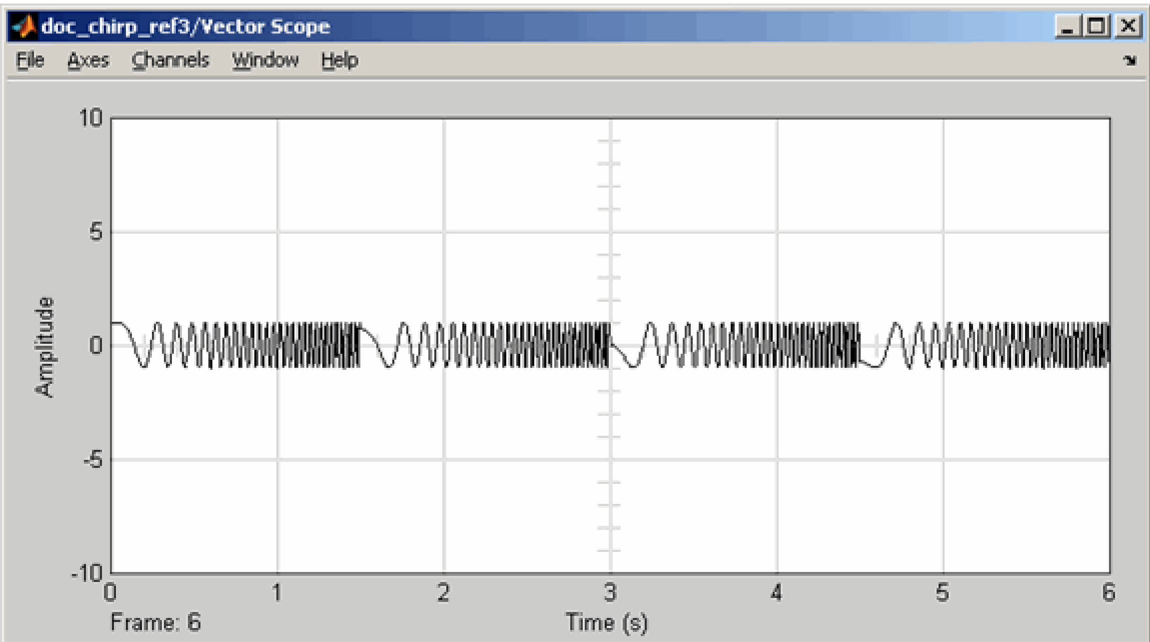
Setting **Sweep time** to `1.5` and leaving the rest of the parameters as in the Example 1 model gives the following output. The sweep still reaches the **Target frequency** (25 Hz) at the **Target time** (1 second), but since

Sweep time is greater than **Target time**, the sweep continues on its linear path until one **Sweep time** (1.5 seconds) is traversed.

Unexpected behavior might arise when you set **Sweep time** greater than **Target time**; see "Example 4: Output Sweep with Negative Frequencies" on page 2-165 for details.

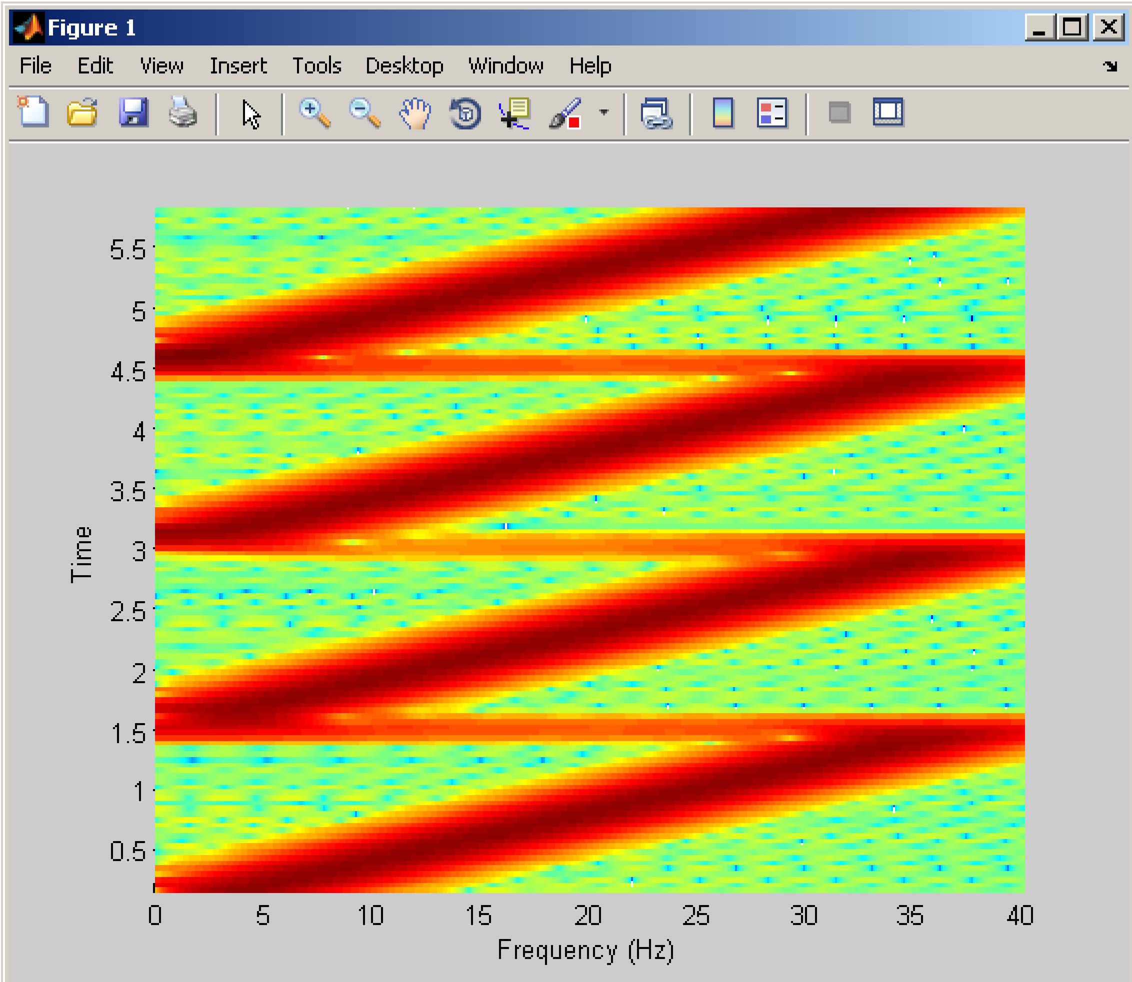
Open the Example 3 model by typing `doc_chirp_ref3` at the MATLAB command line.

Run your model to see the time domain output:



Type the following command to view the chirp output spectrogram:

```
spectrogram(dsp_examples_yout,hamming(128),...
      110,[0:.01:40],400)
```

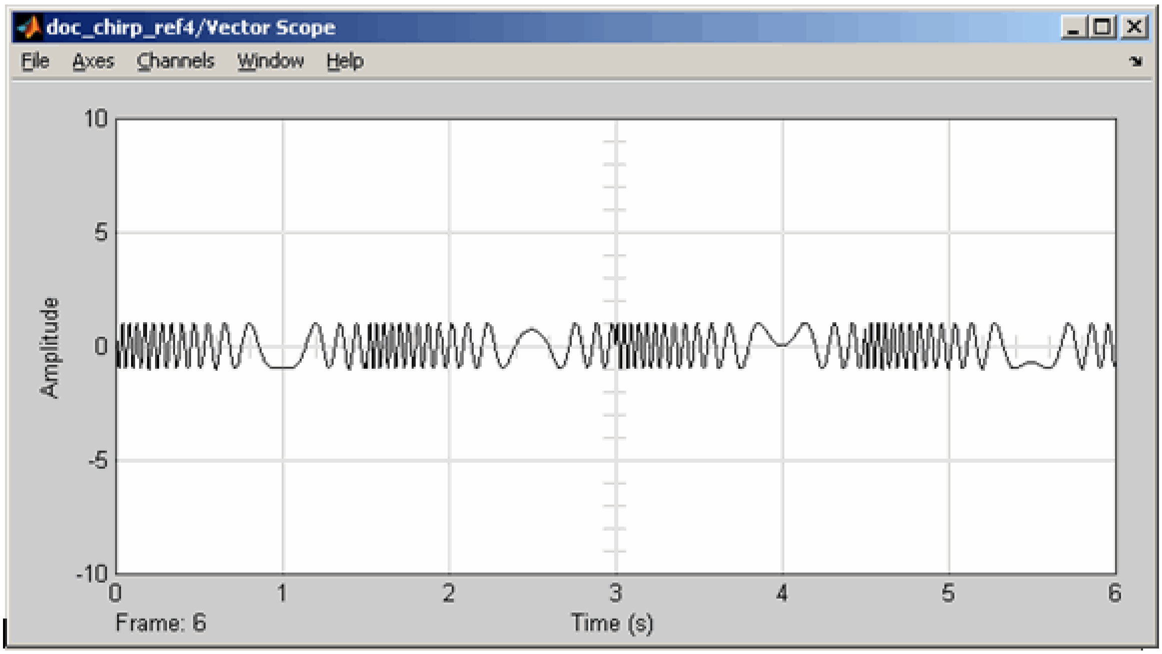


Example 4: Output Sweep with Negative Frequencies

Modify the Example 1 model by changing **Sweep time** to `1.5`, **Initial frequency** to `25`, and **Target frequency** to `0`. *The output chirp of this example might not behave as you expect* because the sweep contains negative frequencies between 1 and 1.5 seconds. The sweep reaches the **Target frequency** of 0 Hz at one second, then continues on its negative slope, taking on negative frequency values until it traverses one **Sweep time** (1.5 seconds).

Open the Example 4 model by typing `doc_chirp_ref4` at the MATLAB command line.

Run your model to see the time domain output:

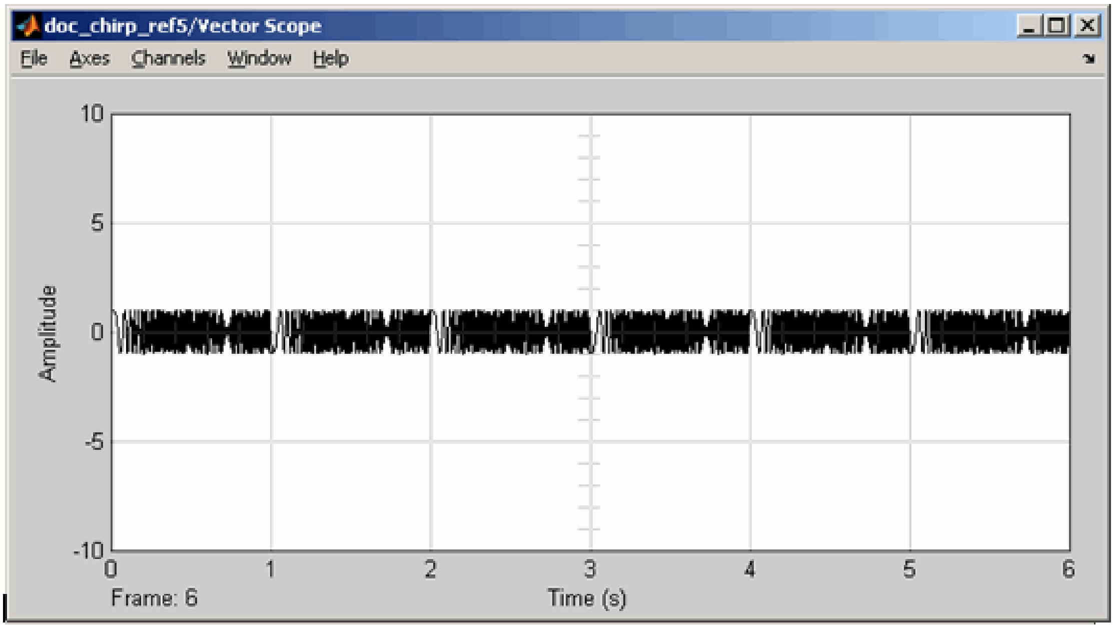


Example 5: Output Sweep with Frequencies Greater Than Half the Sampling Frequency

Modify the Example 1 model by changing the **Target frequency** parameter to `275`. *The output chirp of this model might not behave as you expect* because the sweep contains frequencies greater than half the sampling frequency (`200` Hz), which causes aliasing. If you unexpectedly get a chirp output with a spectrogram resembling the one following, your chirp's sweep might contain frequencies greater than half the sampling frequency.

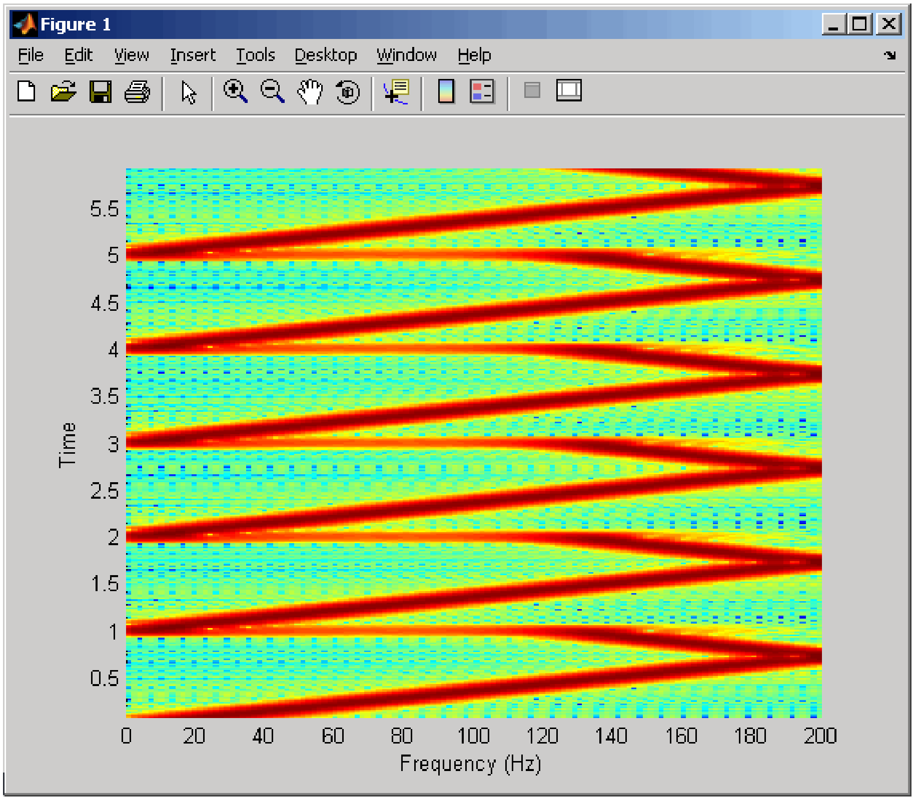
Open the Example 5 model by typing `doc_chirp_ref5` at the MATLAB command line.

Run your model to see the time domain output:



Type the following command to view the chirp output spectrogram:

```
spectrogram(dsp_examples_yout,hamming(64),...
        60,256,400)
```



Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Signal From Workspace	Signal Processing Blockset
Signal Generator	Simulink
Sine Wave	Signal Processing Blockset

`chirp`	Signal Processing Toolbox
`spectrogram`	Signal Processing Toolbox

Cholesky Factorization

Purpose

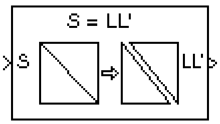
Factor square Hermitian positive definite matrix into triangular components

Library

Math Functions / Matrices and Linear Algebra / Matrix Factorizations

`dspfactors`

Description



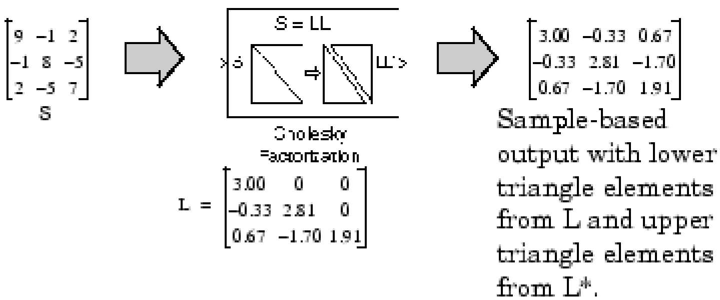
The Cholesky Factorization block uniquely factors the square Hermitian positive definite input matrix S as

$$S = LL^*$$

where L is a lower triangular square matrix with positive diagonal elements and L^* is the Hermitian (complex conjugate) transpose of L. The block outputs a matrix with lower triangle elements from L and upper triangle elements from L^*. The output is always sample based. The output is not in the same form as the output of the MATLAB `chol` function. In order to convert the output of the Cholesky Factorization block to the MATLAB form, use the following equation:

```
R = triu(LL');
```

Here, `LL'` is the output of the Cholesky Factorization block. Due to roundoff error, these equations do not produce a result that is exactly the same as the MATLAB result.



Block Output Composed of L and L^*

Input Requirements for Valid Output

The block output is valid only when its input has the following characteristics:

- Hermitian — The block does *not* check whether the input is Hermitian; it uses only the diagonal and upper triangle of the input to compute the output.
- Real-valued diagonal entries — The block disregards any imaginary component of the input's diagonal entries.
- Positive definite — Set the block to notify you when the input is not positive definite as described in "Response to Nonpositive Definite Input" on page 2-171.

Response to Nonpositive Definite Input

To generate a valid output, the block algorithm requires a positive definite input (see "Input Requirements for Valid Output" on page 2-171). Set the **Non-positive definite input** parameter to determine how the block responds to a nonpositive definite input:

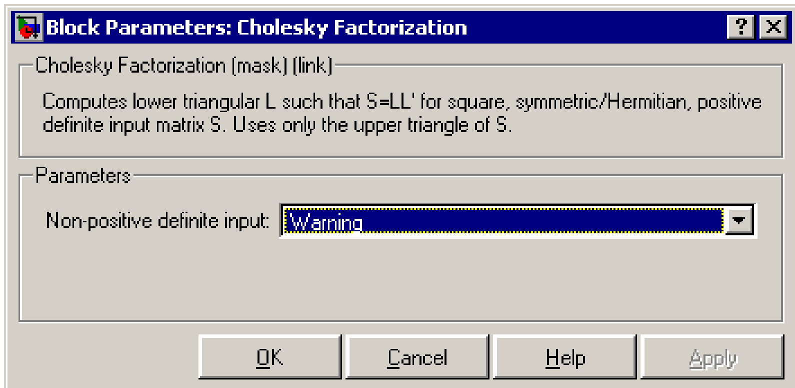
- `Ignore` — Proceed with the computation and *do not* issue an alert. The output is *not* a valid factorization. A partial factorization will be present in the upper left corner of the output.
- `Warning` — Display a warning message in the MATLAB Command Window, and continue the simulation. The output is *not* a valid factorization. A partial factorization will be present in the upper left corner of the output.
- `Error` — Display an error dialog and terminate the simulation.

Note The **Non-positive definite input** parameter is a diagnostic parameter. Like all diagnostic parameters on the Configuration Parameters dialog box, it is set to `Ignore` in the code generated for this block by Real-Time Workshop code generation software.

Performance Comparisons with Other Blocks

Note that L and L^* share the same diagonal in the output matrix. Cholesky factorization requires half the computation of Gaussian elimination (LU decomposition), and is always stable.

Dialog Box



Non-positive definite input

Response to nonpositive definite matrix inputs: `Ignore`, `Warning`, or `Error`. See "Response to Nonpositive Definite Input" on page 2-171.

References

Golub, G. H., and C. F. Van Loan. *Matrix Computations*. 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

Supported Data Types

Port	Supported Data Types
S	• Double-precision floating point • Single-precision floating point
LL'	• Double-precision floating point • Single-precision floating point

See Also

Autocorrelation LPC	Signal Processing Blockset
Cholesky Inverse	Signal Processing Blockset
Cholesky Solver	Signal Processing Blockset
LDL Factorization	Signal Processing Blockset
LU Factorization	Signal Processing Blockset
QR Factorization	Signal Processing Blockset
`chol`	MATLAB

See “Matrix Factorizations” for related information.

Cholesky Inverse

Purpose

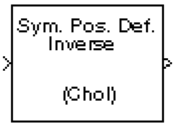
Compute inverse of Hermitian positive definite matrix using Cholesky factorization

Library

Math Functions / Matrices and Linear Algebra / Matrix Inverses

`dspinverses`

Description



The Cholesky Inverse block computes the inverse of the Hermitian positive definite input matrix S by performing Cholesky factorization.

$$S^{-1} = (LL^*)^{-1}$$

L is a lower triangular square matrix with positive diagonal elements and L^* is the Hermitian (complex conjugate) transpose of L. Only the diagonal and upper triangle of the input matrix are used, and any imaginary component of the diagonal entries is disregarded. Cholesky factorization requires half the computation of Gaussian elimination (LU decomposition), and is always stable. The output is always sample based.

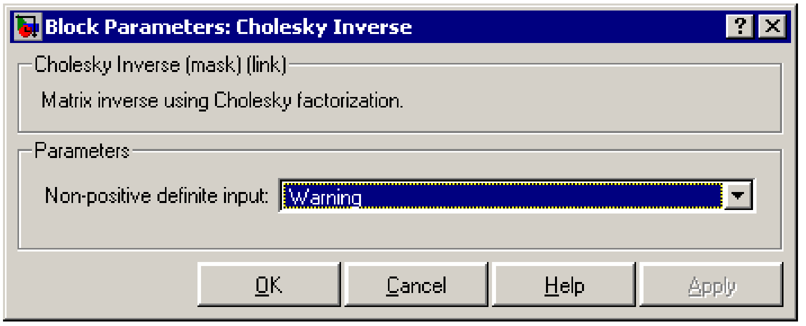
Response to Nonpositive Definite Input

The algorithm requires that the input be Hermitian positive definite. When the input is not positive definite, the block reacts with the behavior specified by the **Non-positive definite input** parameter. The following options are available:

- `Ignore` — Proceed with the computation and *do not* issue an alert. The output is *not* a valid inverse.
- `Warning` — Display a warning message in the MATLAB Command Window, and continue the simulation. The output is *not* a valid inverse.
- `Error` — Display an error dialog box and terminate the simulation.

Note The **Non-positive definite input** parameter is a diagnostic parameter. Like all diagnostic parameters on the Configuration Parameters dialog box, it is set to `Ignore` in the code generated for this block by Real-Time Workshop code generation software.

Dialog Box



Non-positive definite input
Response to nonpositive definite matrix inputs: `Ignore`, `Warning`, or `Error`. See "Response to Nonpositive Definite Input" on page 2-174.

References

Golub, G. H., and C. F. Van Loan. *Matrix Computations*. 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Cholesky Factorization	Signal Processing Blockset
Cholesky Solver	Signal Processing Blockset

LDL Inverse	Signal Processing Blockset
LU Inverse	Signal Processing Blockset
Pseudoinverse	Signal Processing Blockset
`inv`	MATLAB

See “Matrix Inverses” for related information.

Purpose

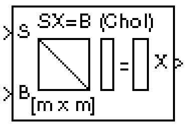
Solve $SX=B$ for X when S is square Hermitian positive definite matrix

Library

Math Functions / Matrices and Linear Algebra / Linear System Solvers

`dspsolvers`

Description



The Cholesky Solver block solves the linear system $SX=B$ by applying Cholesky factorization to input matrix at the `S` port, which must be square (M-by-M) and Hermitian positive definite. Only the diagonal and upper triangle of the matrix are used, and any imaginary component of the diagonal entries is disregarded. The input to the `B` port is the right side M-by-N matrix, B. The output is the unique solution of the equations, M-by-N matrix X, and is always sample based.

A length-M vector input for right side B is treated as an M-by-1 matrix.

Response to Nonpositive Definite Input

When the input is not positive definite, the block reacts with the behavior specified by the **Non-positive definite input** parameter. The following options are available:

- `Ignore` — Proceed with the computation and *do not* issue an alert. The output is *not* a valid solution.
- `Warning` — Proceed with the computation and display a warning message in the MATLAB Command Window. The output is *not* a valid solution.
- `Error` — Display an error dialog box and terminate the simulation.

Note The **Non-positive definite input** parameter is a diagnostic parameter. Like all diagnostic parameters on the Configuration Parameters dialog box, it is set to `Ignore` in the code generated for this block by Real-Time Workshop code generation software.

Algorithm

Cholesky factorization uniquely factors the Hermitian positive definite input matrix S as

$$S = LL^*$$

where L is a lower triangular square matrix with positive diagonal elements.

The equation $SX=B$ then becomes

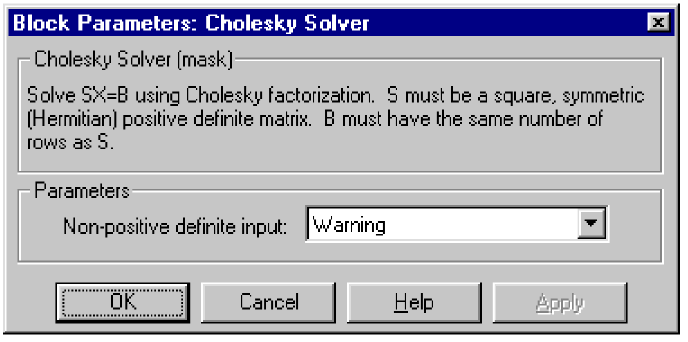
$$LL^*X = B$$

which is solved for X by making the substitution $Y = L^*X$, and solving the following two triangular systems by forward and backward substitution, respectively.

$$LY = B$$

$$L^*X = Y$$

Dialog Box



Non-positive definite input

Response to nonpositive definite matrix inputs: `Ignore`, `Warning`, or `Error`. See "Response to Nonpositive Definite Input" on page 2-177.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Autocorrelation LPC	Signal Processing Blockset
Cholesky Factorization	Signal Processing Blockset
Cholesky Inverse	Signal Processing Blockset
LDL Solver	Signal Processing Blockset
LU Solver	Signal Processing Blockset
QR Solver	Signal Processing Blockset
`chol`	MATLAB

See "Linear System Solvers" for related information.

CIC Compensator

Purpose

Design CIC compensator

Library

Filtering / Filter Designs

`dspfdesign`

Description

This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. You must have a Filter Design Toolbox license to design filters with this block. However, you can run models containing this block without a license. This allows you to run a model sent to you by a colleague who has designed a filter using this block, even if you do not have the Filter Design Toolbox product.

Dialog Box

See "CIC Compensator Filter Design Dialog Box — Main Pane" in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Purpose

Decimate signal using Cascaded Integrator-Comb filter

Library

Filtering / Multirate Filters

`dspmlti4`

Description

The CIC Decimation block performs a sample rate decrease (decimation) on an input signal by an integer factor. Cascaded Integrator-Comb (CIC) filters are a class of linear phase FIR filters comprised of a comb part and an integrator part.

The transfer function of a CIC decimator filter is

$$H(z) = H_I^{\,N}(z)H_c^{\,N}(z) = \frac{\left(1 - z^{-RM}\right)^N}{\left(1 - z^{-1}\right)^N} = \left[\sum_{k=0}^{RM-1} z^{-k}\right]^N$$

where

- H_I is the transfer function of the integrator part of the filter.
- H_C is the transfer function of the comb part of the filter.
- N is the number of sections. The number of sections in a CIC filter is defined as the number of sections in either the comb part *or* the integrator part of the filter, not as the total number of sections throughout the entire filter.
- R is the decimation factor.
- M is the differential delay.

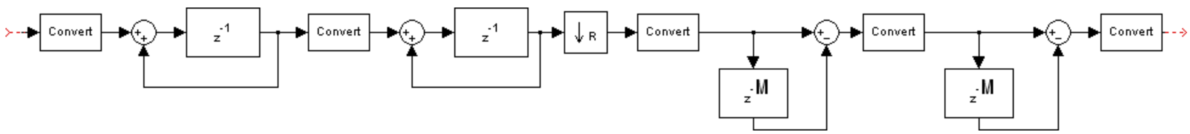
The CIC Decimation block supports real and complex fixed-point inputs. Each channel of a complex input is treated as two real input channels.

CIC Filter Structure

The filter structures supported by the CIC Decimation and CIC Interpolation blocks exactly match those created by Filter Design Toolbox `mfilt` CIC objects. If you have the Filter Design Toolbox and

Fixed-Point Toolbox products installed, you can create an `mfilt` object in any workspace to specify in the **Multirate filter variable** parameter of this block. Otherwise, you can specify the CIC filter completely using only block dialog parameters.

This block can be used to create the following CIC filter structure. This decimator has a latency of *N*, where *N* is the number of sections in either the comb or the integrator part of the filter.



Examples

The GSM Digital Down Converter demo provides an example of using the CIC Decimation block.

Dialog Box

Coefficient Source

The CIC Decimation block can operate in two different modes. Select the mode in the **Coefficient source** group box. If you select

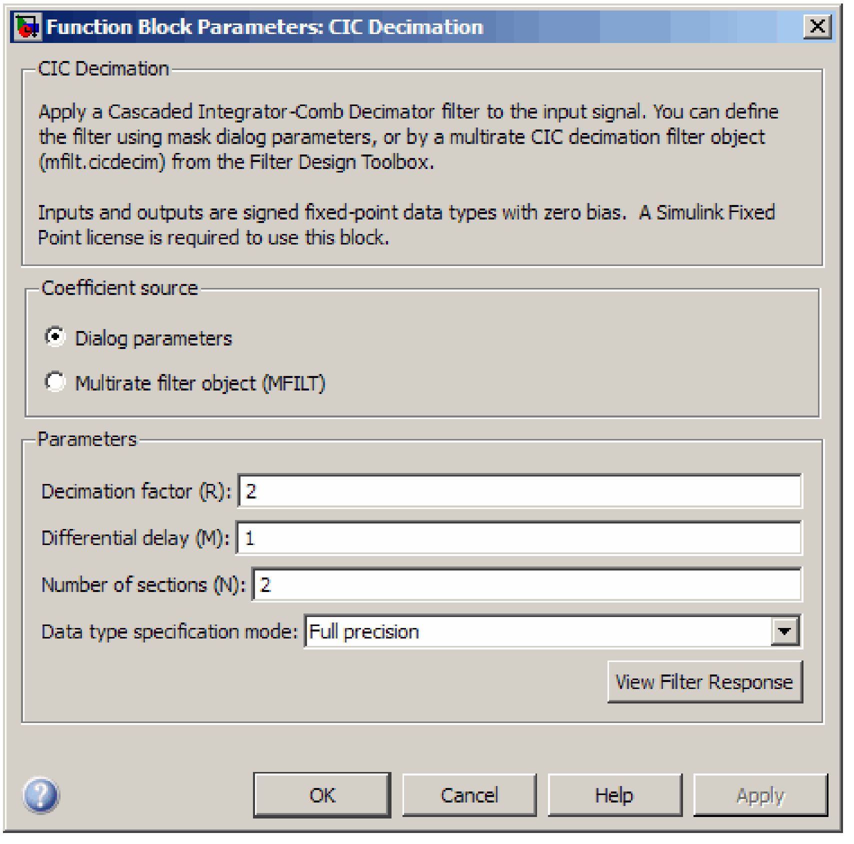
- **Dialog parameters**, you enter information about the filter such as structure and coefficients in the block mask.
- **Multirate filter object (MFILT)**, you specify the filter using a Filter Design Toolbox `mfilt` object, if you have the Filter Design Toolbox and Fixed-Point Toolbox products installed.

Different items appear on the CIC Decimation block dialog depending on whether you select **Dialog parameters** or **Multirate filter object (MFILT)** in the **Coefficient source** group box. See the following sections for details:

- "Specify Filter Characteristics in Dialog" on page 2-184
- "Specify Multirate Filter Object" on page 2-188

Specify Filter Characteristics in Dialog

The **Main** pane of the CIC Decimation block dialog appears as follows when **Dialog parameters** is selected in the **Coefficient source** group box.



Decimation factor (R)
Specify the decimation factor of the filter.

Differential delay (M)

Specify the differential delay of the comb part of the filter, `M`, as shown in the diagram in "CIC Filter Structure" on page 2-182.

Number of sections (N)

Specify the number of filter sections. This number is equal to the number of sections in either the comb part of the filter or in the integrator part of the filter. This value is not equal to the total number of sections in the comb and integrator parts combined.

Data type specification mode

Choose how you specify the fixed-point word length and fraction length of the filter sections and/or output.

- `Full precision` — In this mode, the word and fraction lengths of the filter sections and outputs are automatically selected for you. All word lengths are set to

  $$\text{word length} = \text{ceil}(N * \log_2(M * R)) + I$$

  where

  - I = input word length
  - M = differential delay
  - N = number of sections
  - R = decimation factor

  All fraction lengths are set to the input fraction length.

- `Minimum section word lengths` — In this mode, you specify the word length of the filter output in the **Output word length** parameter. The word lengths of the filter sections and all fraction lengths are automatically selected for you such that each of the section word lengths is as small as possible. The precision of each filter section is less than in `Full precision` mode, but the range of each section is preserved.

- `Specify word lengths` — In this mode you specify the word lengths of the filter sections and output in the **Section word**

lengths and **Output word length** parameters. The fraction lengths of the filter sections and output are automatically selected for you such that when least significant bits are discarded at each section, the range of that section is preserved.

- `Binary point scaling` — In this mode you fully specify the word and fraction lengths of the filter sections and output in the **Section word lengths**, **Section fraction lengths**, **Output word length**, and **Output fraction length** parameters.

Section word lengths
Specify the word length, in bits, of the filter sections.

This parameter is only visible if `Specify word lengths` or `Binary point scaling` is selected for the **Data type specification mode** parameter.

Section fraction lengths
Specify the fraction length of the filter sections.

This parameter is only visible if `Binary point scaling` is selected for the **Data type specification mode** parameter.

Output word length
Specify the word length, in bits, of the filter output.

This parameter is only visible if `Minimum section word lengths`, `Specify word lengths`, or `Binary point scaling` is selected for the **Data type specification mode** parameter.

Output fraction length
Specify the fraction length of the filter output.

This parameter is only visible if `Binary point scaling` is selected for the **Data type specification mode** parameter.

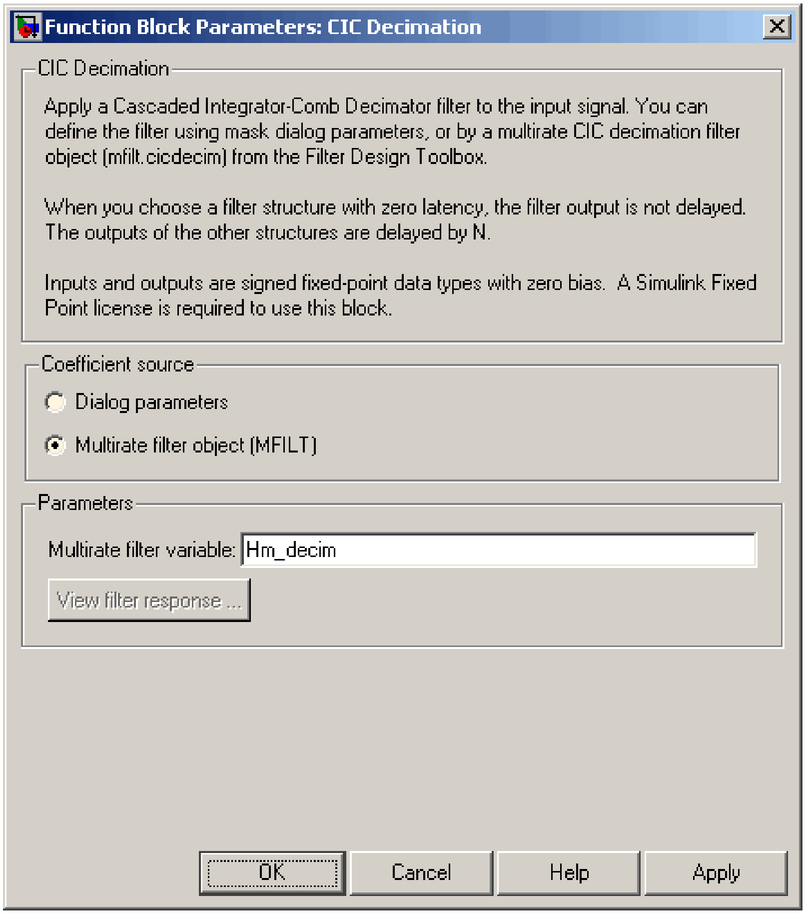
View filter response
This button opens the Filter Visualization Tool (`fvtool`) from the Signal Processing Toolbox product and displays the filter response

of the filter defined in the block. For more information on FVTool, see the Signal Processing Toolbox documentation.

Note This button is only available when the Filter Design Toolbox and Fixed-Point Toolbox products are installed. If you specify a filter in the **Multirate filter variable** parameter, you must apply the filter by clicking the **Apply** button before using the **View filter response** button.

Specify Multirate Filter Object

The **Main** pane of the CIC Decimation block dialog appears as follows when **Multirate filter object (MFILT)** is specified in the **Coefficient source** group box.



Multirate filter variable

Specify the multirate filter object (`mfilt`) that you would like the block to implement. You can do this in one of three ways:

- You can fully specify the `mfilt` object in the block mask.
- You can enter the variable name of a `mfilt` object that is defined in any workspace.
- You can enter a variable name for a `mfilt` object that is not yet defined, as shown in the default value.

For more information on creating `mfilt` objects, see the `mfilt` function reference page in the Filter Design Toolbox documentation.

View filter response

This button opens the Filter Visualization Tool (`fvtool`) from the Signal Processing Toolbox product and displays the filter response of the `mfilt` object specified in the **Multirate filter variable** parameter. For more information on FVTool, see the Signal Processing Toolbox documentation.

Note This button is only available when the Filter Design Toolbox and Fixed-Point Toolbox products are installed. If you specify a filter in the **Multirate filter variable** parameter, you must apply the filter by clicking the **Apply** button before using the **View filter response** button.

References

[1] Hogenauer, E.B., "An Economical Class of Digital Filters for Decimation and Interpolation," *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-29(2): pp. 155-162, 1981.

[2] Meyer-Baese, U., *Digital Signal Processing with Field Programmable Gate Arrays*, Springer Verlag, 2001.

[3] Harris, Fredric J., *Multirate Signal Processing for Communication Systems*, Prentice Hall PTR, 2004.

Supported Data Types

- Fixed point (signed only)
- 8-, 16-, and 32-bit signed integers

See Also

CIC Interpolation	Signal Processing Blockset
FIR Decimation	Signal Processing Blockset
FIR Interpolation	Signal Processing Blockset
`filter`	Filter Design Toolbox
`mfilt.cicdecim`	Filter Design Toolbox
`mfilt.cicinterp`	Filter Design Toolbox

Purpose

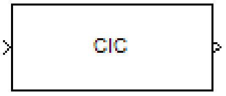
Design Cascaded Integrator-Comb (CIC) Filter

Library

Filtering / Filter Designs

`dspfdesign`

Description



This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. You must have a Filter Design Toolbox license to design filters with this block. However, you can run models containing this block without a license. This allows you to run a model sent to you by a colleague who has designed a filter using this block, even if you do not have the Filter Design Toolbox product.

Dialog Box

See “CIC Filter Design Dialog Box — Main Pane” in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Fixed point

CIC Interpolation

Purpose

Interpolate signal using Cascaded Integrator-Comb filter

Library

Filtering / Multirate Filters

`dspmlti4`

Description

The CIC Interpolation block performs a sample rate increase (interpolation) on an input signal by an integer factor. Cascaded Integrator-Comb (CIC) filters are a class of linear phase FIR filters comprised of a comb part and an integrator part.

The transfer function of a CIC interpolator filter is

$$H(z) = H_I{}^N(z)H_C{}^N(z) = \frac{(1 - z^{-RM})^N}{(1 - z^{-1})^N} = \left[\sum_{k=0}^{RM-1} z^{-k}\right]^N$$

where

- H_I is the transfer function of the integrator part of the filter.
- H_C is the transfer function of the comb part of the filter.
- N is the number of sections. The number of sections in a CIC filter is defined as the number of sections in either the comb part *or* the integrator part of the filter, not as the total number of sections throughout the entire filter.
- R is the interpolation factor.
- M is the differential delay.

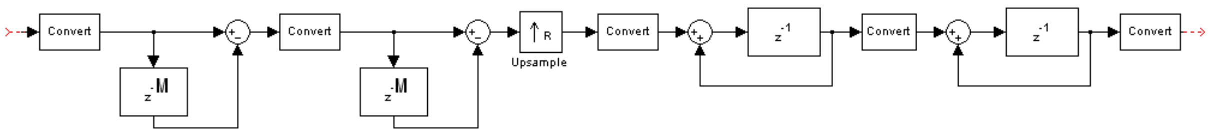
The CIC Interpolation block supports real and complex fixed-point inputs. Each channel of a complex input is treated as two real input channels.

CIC Filter Structure

The filter structures supported by the CIC Interpolation and CIC Decimation blocks exactly match those created by Filter Design Toolbox `mfilt` CIC objects. If you have the Filter Design Toolbox and

Fixed-Point Toolbox products installed, you can create an `mfilt` object in any workspace to specify in the **Multirate filter variable** parameter of this block. Otherwise, you can specify the CIC filter completely using only block dialog parameters.

This block can be used to create the following CIC filter structure. This interpolator has a latency of *N*, where *N* is the number of sections in either the comb or the integrator part of the filter.



Dialog Box

Coefficient Source

The CIC Interpolation block can operate in two different modes. Select the mode in the **Coefficient source** group box. If you select

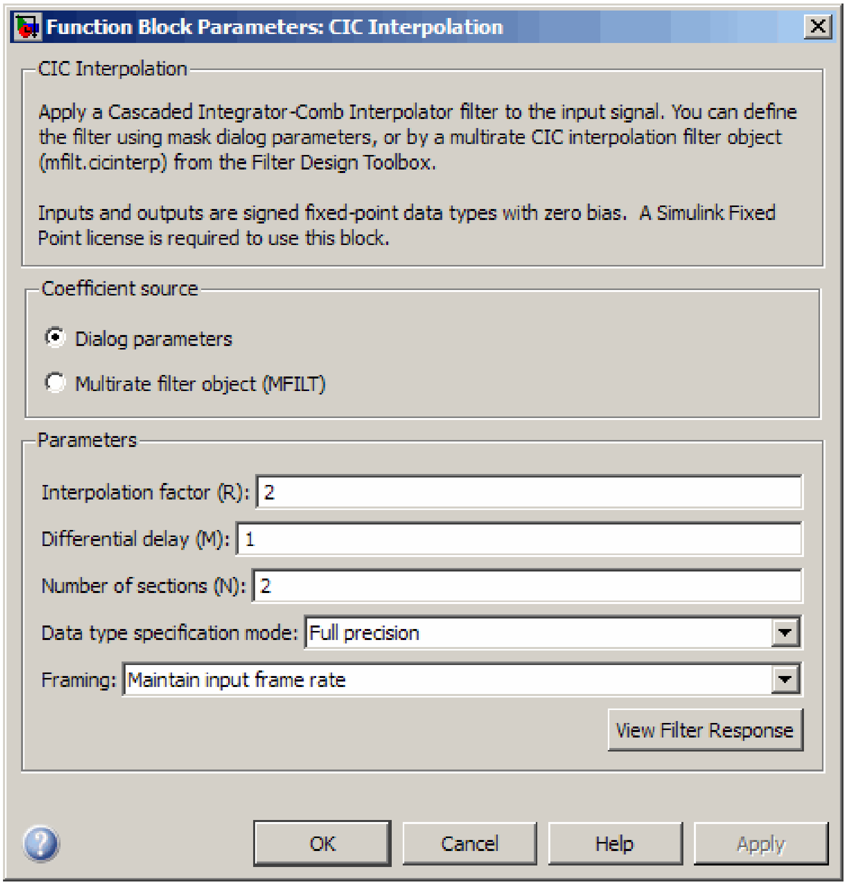
- **Dialog parameters**, you enter information about the filter such as structure and coefficients in the block mask
- **Multirate filter object (MFILT)**, you specify the filter using a Filter Design Toolbox `mfilt` object, if you have the Filter Design Toolbox and Fixed-Point Toolbox products installed.

Different items appear on the CIC Interpolation block dialog depending on whether you select **Dialog parameters** or **Multirate filter object (MFILT)** in the **Coefficient source** group box. See the following sections for details:

- "Specify Filter Characteristics in Dialog" on page 2-193
- "Specify Multirate Filter Object" on page 2-197

Specify Filter Characteristics in Dialog

The **Main** pane of the CIC Interpolation block dialog appears as follows when **Dialog parameters** is selected in the **Coefficient source** group box.



Interpolation factor (R)

Specify the interpolation factor of the filter.

Differential delay (M)

Specify the differential delay of the comb portion of the filter, `M`, as shown in the diagram in "CIC Filter Structure" on page 2-192.

Number of sections (N)

Specify the number of filter sections. This number is equal to the number of sections in either the comb part of the filter or in the integrator part of the filter. This value is not equal to the total number of sections in the comb and integrator parts combined.

Data type specification mode

Choose how you specify the fixed-point word length and fraction length of the filter sections and/or output.

- `Full precision` — In this mode, the word and fraction lengths of the filter sections and outputs are automatically selected for you. The output and last section word lengths are set to

  $$\text{word length} = \text{ceil}(\log_2(\frac{(R*M)^N}{R})) + I$$

  where

  - I = input word length
  - M = differential delay
  - N = number of sections
  - R = interpolation factor

  The other section word lengths are set is such a way as to accommodate the bit growth, as described in Hogenauer's paper [1]. All fraction lengths are set to the input fraction length.

- `Minimum section word lengths` — In this mode, you specify the word length of the filter output in the **Output word length** parameter. The word lengths of the filter sections are set in the same way as in `Full precision` mode.

  The section fraction lengths are set to the input fraction length. The output fraction length is set to the input fraction length minus the difference between the last section and output word lengths.

- Specify word lengths — In this mode you specify the word lengths of the filter sections and output in the **Section word lengths** and **Output word length** parameters. The fraction lengths of the filter sections are set such that the spread between word length and fraction length is the same as in full-precision mode. The output fraction length is set to the input fraction length minus the difference between the last section and output word lengths.
- Binary point scaling — In this mode you fully specify the word and fraction lengths of the filter sections and output in the **Section word lengths**, **Section fraction lengths**, **Output word length**, and **Output fraction length** parameters.

Section word lengths

Specify the word length, in bits, of the filter sections.

This parameter is only visible if Specify word lengths or Binary point scaling is selected for the **Data type specification mode** parameter.

Section fraction lengths

Specify the fraction length of the filter sections.

This parameter is only visible if Binary point scaling is selected for the **Data type specification mode** parameter.

Output word length

Specify the word length, in bits, of the filter output.

This parameter is only visible if Minimum section word lengths, Specify word lengths, or Binary point scaling is selected for the **Data type specification mode** parameter.

Output fraction length

Specify the fraction length of the filter output.

This parameter is only visible if Binary point scaling is selected for the **Data type specification mode** parameter.

Framing

For frame-based operation, specify the method by which to implement the interpolation; increase the output frame rate, or increase the output frame size. This parameter cannot be set to `Maintain input frame rate` for sample-based signals.

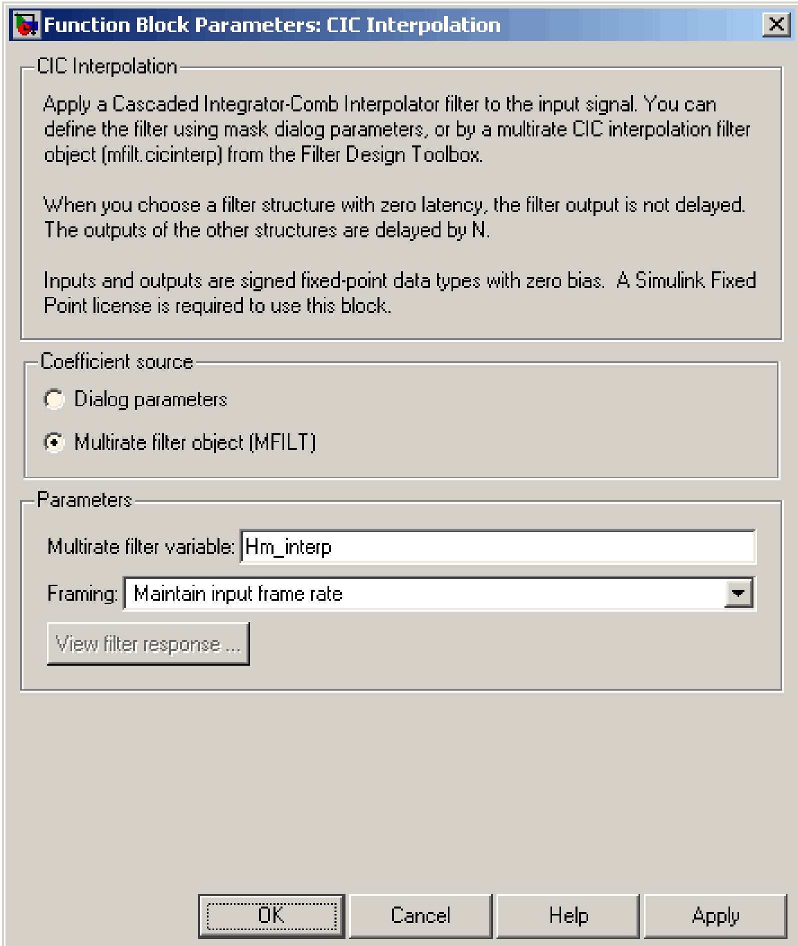
View filter response

This button opens the Filter Visualization Tool (`fvtool`) from the Signal Processing Toolbox product and displays the filter response of the `mfilt` object specified in the **Multirate filter variable** parameter. For more information on FVTool, see the Signal Processing Toolbox documentation.

Note This button is only available when the Filter Design Toolbox and Fixed-Point Toolbox products are installed. If you specify a filter in the **Multirate filter variable** parameter, you must apply the filter by clicking the **Apply** button before using the **View filter response** button.

Specify Multirate Filter Object

The **Main** pane of the CIC Interpolation block dialog appears as follows when **Multirate filter object (MFILT)** is specified in the **Coefficient source** group box.



Multirate filter variable

Specify the multirate filter object (`mfilt`) that you would like the block to implement. You can do this in one of three ways:

- You can fully specify the `mfilt` object in the block mask.

- You can enter the variable name of a `mfilt` object that is defined in any workspace.
- You can enter a variable name for a `mfilt` object that is not yet defined, as shown in the default value.

For more information on creating `mfilt` objects, see the `mfilt` function reference page in the Filter Design Toolbox documentation.

Framing

For frame-based operation, specify the method by which to implement the interpolation; increase the output frame rate, or increase the output frame size. This parameter cannot be set to `Maintain input frame rate` for sample-based signals.

View filter response

This button opens the Filter Visualization Tool (`fvtool`) from the Signal Processing Toolbox product and displays the filter response of the `mfilt` object specified in the **Multirate filter variable** parameter. For more information on FVTool, see the Signal Processing Toolbox documentation.

Note This button is only available when the Filter Design Toolbox and Fixed-Point Toolbox products are installed. If you specify a filter in the **Multirate filter variable** parameter, you must apply the filter by clicking the **Apply** button before using the **View filter response** button.

References

[1] Hogenauer, E.B., "An Economical Class of Digital Filters for Decimation and Interpolation," *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-29(2): pp. 155-162, 1981.

[2] Meyer-Baese, U., *Digital Signal Processing with Field Programmable Gate Arrays*, Springer Verlag, 2001.

[3] Harris, Fredric J., *Multirate Signal Processing for Communication Systems*, Prentice Hall PTR, 2004.

Supported Data Types

- Fixed point (signed only)
- 8-, 16-, and 32-bit signed integers

See Also

CIC Decimation	Signal Processing Blockset
FIR Decimation	Signal Processing Blockset
FIR Interpolation	Signal Processing Blockset
`filter`	Filter Design Toolbox
`mfilt.cicdecim`	Filter Design Toolbox
`mfilt.cicinterp`	Filter Design Toolbox

Purpose

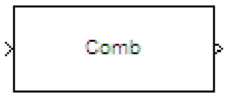
Design comb Filter

Library

Filtering / Filter Designs

`dspfdesign`

Description



This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. You must have a Filter Design Toolbox license to design filters with this block. However, you can run models containing this block without a license. This allows you to run a model sent to you by a colleague who has designed a filter using this block, even if you do not have the Filter Design Toolbox product.

Dialog Box

See "Comb Filter Design Dialog Box—Main Pane" in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Fixed point

Complex Cepstrum

Purpose

Compute complex cepstrum of input

Library

Transforms

`dspxfrm3`

Description

The Complex Cepstrum block computes the complex cepstrum of each channel in the real-valued M-by-N input matrix, u. For both sample-based and frame-based inputs, the block assumes that each input column is a frame containing M consecutive samples from an independent channel. The block does not accept complex-valued inputs.

The input is altered by the application of a linear phase term so that there is no phase discontinuity at ±π radians. That is, each input channel is independently zero padded and circularly shifted to have zero phase at π radians.

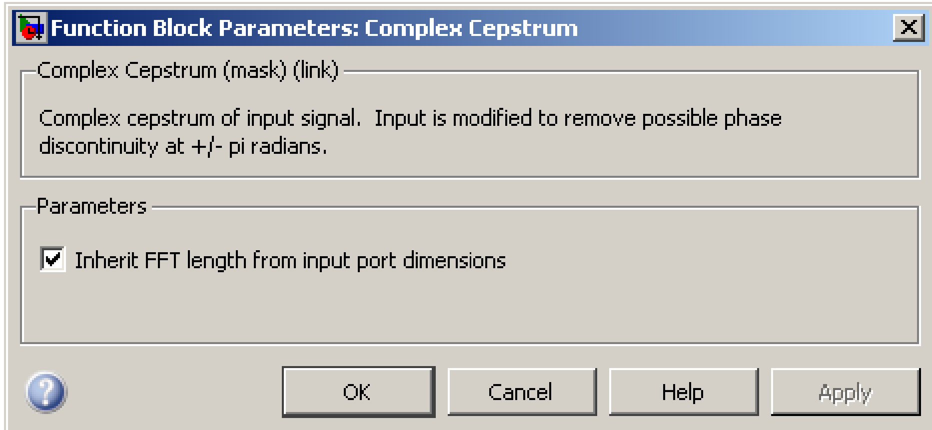
The output is a real M_o-by-N matrix, where M_o is specified by the **FFT length** parameter. Each output column contains the length-M_o complex cepstrum of the corresponding input column.

```
y = cceps(u,Mo)     % Equivalent MATLAB code
```

When you select the **Inherit FFT length from input port dimensions** check box, the output frame size matches the input frame size (M_o=M). In this case, the block processes *sample-based* length-M row vector inputs as a single channel (that is, as an M-by-1 column vector), and returns the result as a length-M column vector. The block *always* processes 1-D vector inputs as a single channel, and returns the result as a length-M column vector.

The output is always sample based, and the output port rate is the same as the input port rate.

Dialog Box



Inherit FFT length from input port dimensions
When you select this check box, the output frame size matches the input frame size.

FFT length
The number of frequency points at which to compute the FFT, which is also the output frame size, M_o. This parameter is visible only when you clear the **Inherit FFT length from input port dimensions** check box.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

DCT	Signal Processing Blockset
FFT	Signal Processing Blockset
Real Cepstrum	Signal Processing Blockset
`cceps`	Signal Processing Toolbox

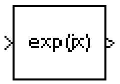
Complex Exponential

Purpose Compute complex exponential function

Library Math Functions / Math Operations

`dspmathops`

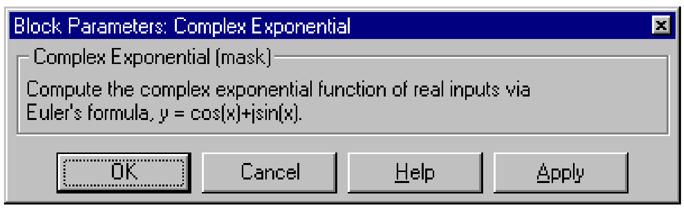
Description The Complex Exponential block computes the complex exponential function for each element of the real input, u.



$$y = e^{ju} = \cos u + j \sin u$$

where $j = \sqrt{-1}$. The output is complex, with the same size and frame status as the input.

Dialog Box



Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Math Function	Simulink
Sine Wave	Signal Processing Blockset
`exp`	MATLAB

Purpose

Generate constant value

Library

Signal Processing Sources

`dspsrcs4`

Description

The Constant block is an implementation of the Simulink Constant block. See Constant for more information.

Constant Diagonal Matrix

Purpose

Generate square, diagonal matrix

Library

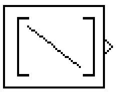
- Signal Processing Sources

  `dspsrcs4`

- Math Functions / Matrices and Linear Algebra / Matrix Operations

  `dspmtrx3`

Description

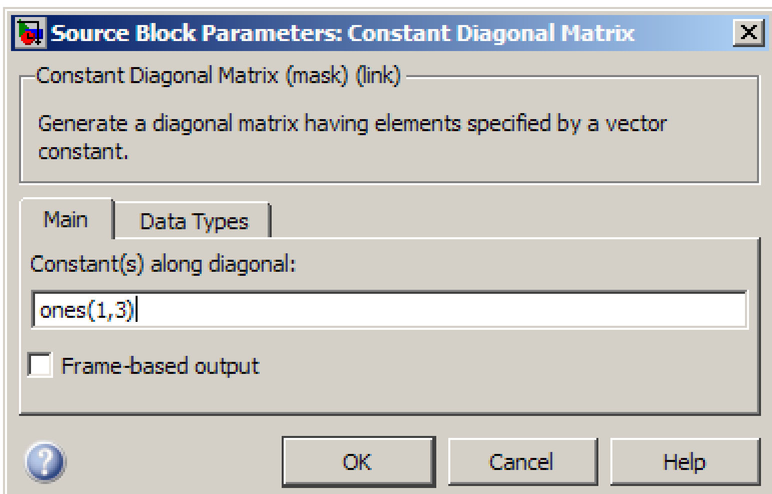


The Constant Diagonal Matrix block outputs a square diagonal matrix constant. The **Constant along diagonal** parameter determines the values along the matrix diagonal. This parameter can be a scalar to be repeated for all elements along the diagonal, or a vector containing the values of the diagonal elements. To generate the identity matrix, set the **Constant along diagonal** to `1`, or use the Identity Matrix block.

The output is frame based when you select the **Frame-based output** check box; otherwise, the output is sample based.

Dialog Box

The **Main** pane of the Constant Diagonal Matrix block dialog appears as follows.



Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters During Simulation" in the online Simulink documentation for details.

Constant(s) along diagonal

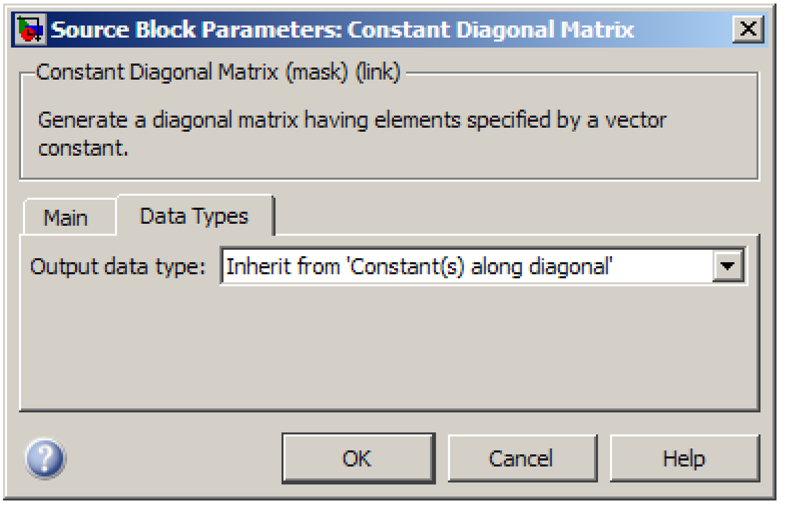
Specify the values of the elements along the diagonal. You can input a scalar or a vector. Tunable.

When you specify any data type information in this field, it is overridden by the value of the **Output data type** parameter on the **Data Types** pane, unless you select `Inherit from 'Constant(s) along diagonal'`.

Frame-based output

Select to cause the output of the block to be frame based. Otherwise, the output is sample based.

The **Data Types** pane of the Constant Diagonal Matrix block dialog appears as follows.



Output data type

Specify the output data type in one of the following ways:

- Choose one of the built-in data types from the list.
- Choose `Fixed-point` to specify the output data type and scaling in the **Signed**, **Word length**, **Set fraction length in output to,** and **Fraction length** parameters.
- Choose `User-defined` to specify the output data type and scaling in the **User-defined data type**, **Set fraction length in output to,** and **Fraction length** parameters.
- Choose `Inherit from 'Constant(s) along diagonal'` to set the output data type and scaling to match the values of the **Constant(s) along diagonal** parameter on the **Main** pane.
- Choose `Inherit via back propagation` to set the output data type and scaling to match the next block downstream.

The value of this parameter overrides any data type information specified in the **Constant(s) along diagonal** parameter on the **Main** pane, except when you select `Inherit from 'Constant(s) along diagonal'`.

Signed

Select to output a signed fixed-point signal. Otherwise, the signal is unsigned. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

Word length

Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

User-defined data type

Specify any built-in or fixed-point data type. You can specify fixed-point data types using the following Simulink® Fixed Point™ functions: `sfix`, `ufix`, `sint`, `uint`, `sfrac`, and `ufrac`. This parameter is only visible when you select `User-defined` for the **Output data type** parameter.

Set fraction length in output to

Specify the scaling of the fixed-point output by either of the following two methods:

- Choose `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.
- Choose `User-defined` to specify the output scaling in the **Fraction length** parameter.

This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter, or when you select `User-defined` and the specified output data type is a fixed-point data type.

Fraction length

For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Output data type** parameter and `User-defined` for the **Set fraction length in output to** parameter.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers

See Also

Create Diagonal Matrix	Signal Processing Blockset
Constant	Simulink

Identity Matrix	Signal Processing Blockset
`diag`	MATLAB

Purpose

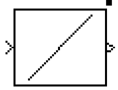
Generate ramp signal with length based on input dimensions

Library

Signal Operations

`dspsigops`

Description



The Constant Ramp block generates the constant ramp signal

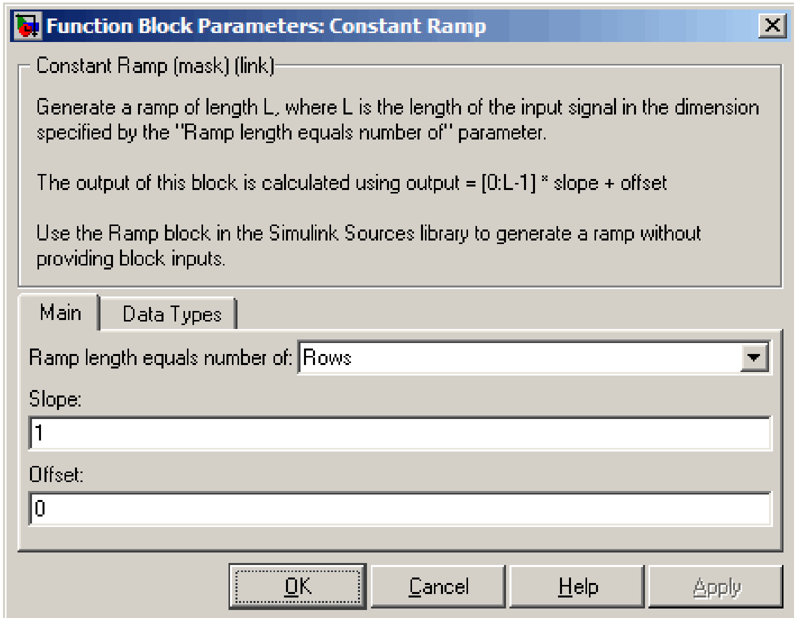
```
y = (0:L-1)*m + b
```

where m is the slope specified by the scalar **Slope** parameter, and b is the y-intercept specified by the scalar **Offset** parameter.

For a 1-D vector input, L is equal to the length of the input vector. For an N-D array input, the length L of the output ramp is equal to the length of the input in the dimension specified by the **Ramp length equals number of** or **Dimension** parameter. The output, y, is always a 1-D vector.

Dialog Box

The **Main** pane of the Constant Ramp block dialog appears as follows.



Ramp length equals number of
: Specify whether the length of the output ramp is the number of rows, number of columns, or the length of the specified dimension of the input.

Dimension
: Specify the one-based dimension of the input array that determines the length of the output ramp.

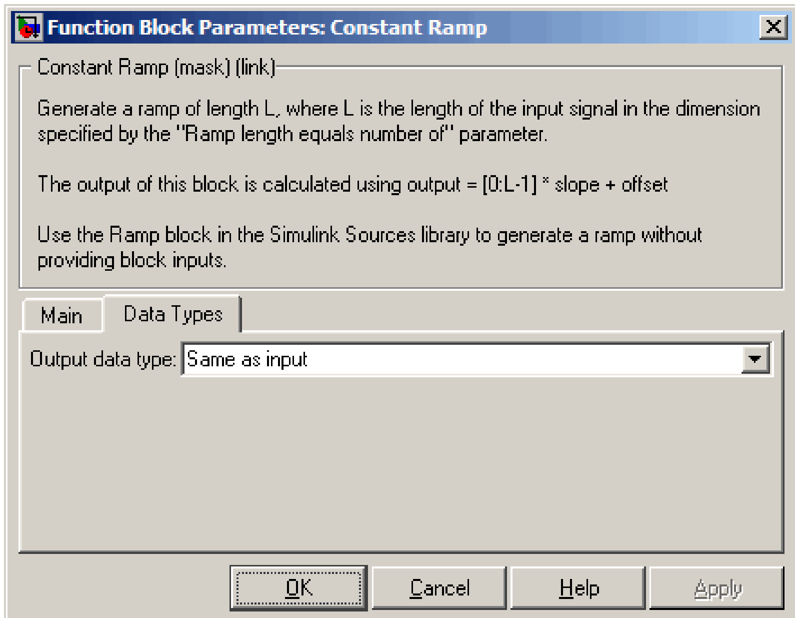
  This parameter is only visible when you select `Elements in specified dimension` for the **Ramp length equals number of** parameter.

Slope
: Specify the scalar slope of the ramp.

Offset

Specify the scalar *y*-intercept of the ramp.

The **Data Types** pane of the Constant Ramp block dialog appears as follows.



Output data type

Specify the output data type in one of the following ways:

- Select `Same as input` to force the data type of the output to be the same as the data type of the input to the block.
- Select one of the built-in data types from the list.
- Select `Fixed-point` to specify the output data type and scaling in the **Signed**, **Word length**, **Set fraction length in output to,** and **Fraction length** parameters.

- Select `User-defined` to specify the output data type and scaling in the **User-defined data type**, **Set fraction length in output to,** and **Fraction length** parameters.
- Select `Inherit via back propagation` to set the output data type and scaling to match the next block downstream.

This block differs from other Signal Processing Blockset blocks in that unless you choose `Same as input` for this parameter, the data types of the input and the output do not need to be the same.

Signed
Select to output a signed fixed-point signal. Otherwise, the signal is unsigned. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

Word length
Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

User-defined data type
Specify any built-in or fixed-point data type. You can specify fixed-point data types using the following Simulink Fixed Point functions: `sfix`, `ufix`, `sint`, `uint`, `sfrac`, and `ufrac`. This parameter is only visible when you select `User-defined` for the **Output data type** parameter.

Set fraction length in output to
Specify the scaling of the fixed-point output by either of the following two methods:

- Select `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.
- Select `User-defined` to specify the output scaling in the **Fraction length** parameter.

This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter, or when you select

`User-defined` and the specified output data type is a fixed-point data type.

Fraction length

For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Output data type** parameter and `User-defined` for the **Set fraction length in output to** parameter.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers

This block differs from other Signal Processing Blockset blocks in that unless you choose `Same as input` for the **Output data type** parameter, the data types of the input and the output do not need to be the same.

See Also

Create Diagonal Matrix	Signal Processing Blockset
Constant	Simulink
Identity Matrix	Signal Processing Blockset

Contiguous Copy (Obsolete)

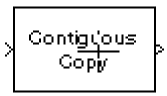
Purpose

Create discontiguous input in contiguous block of memory

Library

`dspobslib`

Description



Contiguous
Copy

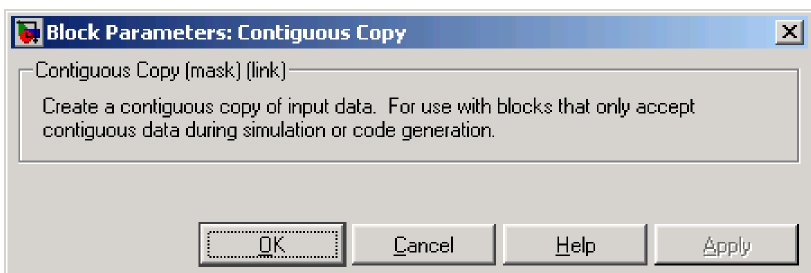
Note The Contiguous Copy block is still supported but is likely to be obsoleted in a future release.

The Contiguous Copy block copies the input to a contiguous block of memory, and passes this new copy to the output. The output is identical to the input, but is guaranteed to reside in a contiguous section of memory.

Because Simulink software employs an efficient copy-by-reference method for propagating data in a model, some operations produce outputs with discontiguous memory locations.

Although this does not present a problem during simulation, blocks linked to versions of DSP Blockset prior to 4.0 may require contiguous inputs for code generation with the Real-Time Workshop product. When such blocks are used in a model intended for code generation, they should be preceded by the Contiguous Copy block to ensure that their inputs are contiguous.

Dialog Box



Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point
- Boolean
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers

Convert 1-D to 2-D

Purpose

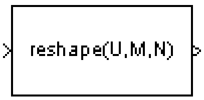
Reshape 1-D or 2-D input to 2-D matrix with specified dimensions

Library

Signal Management / Signal Attributes

`dspsigattribs`

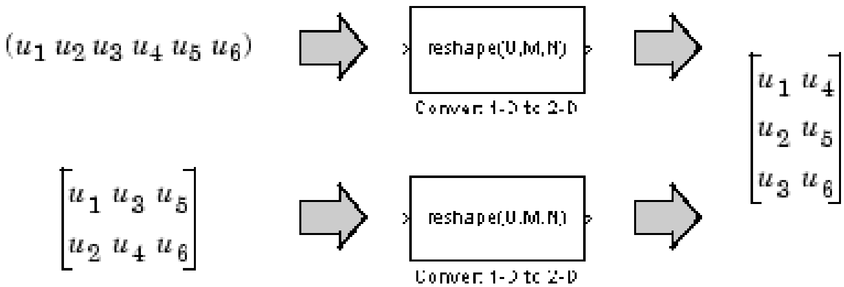
Description



The Convert 1-D to 2-D block reshapes a length-M_i 1-D vector or an M_i-by-N_i matrix to an M_o-by-N_o matrix, where M_o is specified by the **Number of output rows** parameter, and N_o is specified by the **Number of output columns** parameter.

```
y = reshape(u,Mo,No)       % Equivalent MATLAB code
```

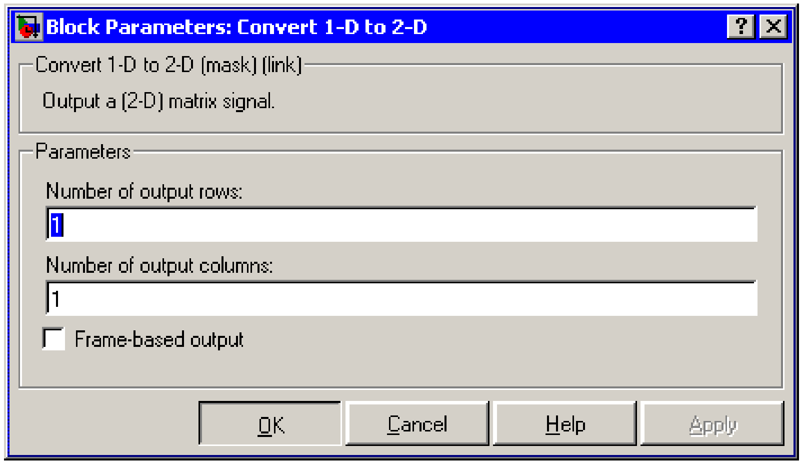
The input is reshaped *columnwise*, as shown in the two cases below. The length-6 vector and the 2-by-3 matrix are both reshaped to the same 3-by-2 output matrix.



An error is generated when $(M_o*N_o) \neq (M_i*N_i)$. That is, the total number of input elements must be conserved in the output.

The output is frame based when you select the **Frame-based output** check box; otherwise, the output is sample based.

Dialog Box



Number of output rows
The number of rows, M_o, in the output matrix.

Number of output columns
The number of rows, N_o, in the output matrix.

Frame-based output
Creates a frame-based output when selected.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Buffer	Signal Processing Blockset
Convert 2-D to 1-D	Signal Processing Blockset
Frame Status Conversion (Obsolete)	Signal Processing Blockset
Reshape	Simulink
Submatrix	Signal Processing Blockset

Purpose

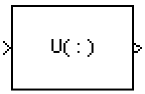
Convert 2-D matrix input to 1-D vector

Library

Signal Management / Signal Attributes

`dspsigattribs`

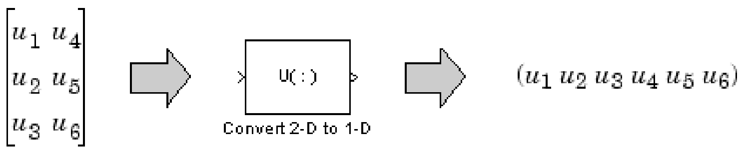
Description



The Convert 2-D to 1-D block reshapes an *M*-by-*N* matrix input to a 1-D vector with length *M***N*.

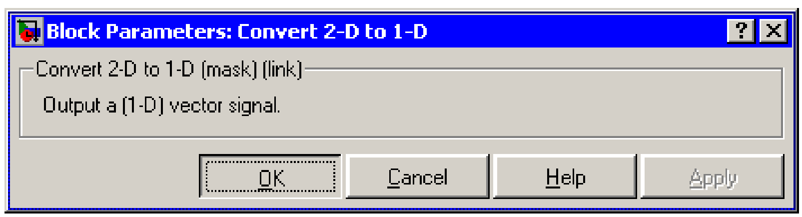
```
y = u(:)     % Equivalent MATLAB code
```

The input is reshaped *columnwise*, as shown below for a 3-by-2 matrix.



The output is always sample-based.

Dialog Box



Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated

See Also

Buffer	Signal Processing Blockset
Convert 1-D to 2-D	Signal Processing Blockset
Frame Status Conversion (Obsolete)	Signal Processing Blockset
Reshape	Simulink
Submatrix	Signal Processing Blockset

Purpose

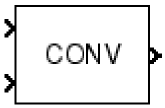
Compute convolution of two inputs

Library

Signal Operations

`dspsigops`

Description



The Convolution block convolves the first dimension of a sample-based N-D input array *u*, with the first dimension of a sample-based N-D input array *v*. The block can also independently convolve a sample-based vector with the first-dimension of an N-D input array. For frame-based inputs, the Convolution block convolves analogous columns of an M_u-by-N input matrix u and an M_v-by-N input matrix v. The Convolution block can also independently convolve a single-channel frame-based column vector with each column of a multiple-channel frame-based matrix.

The frame status of both inputs to the Convolution block must be the same. The output of the block is always sample-based.

The Convolution block accepts real and complex floating-point and fixed-point inputs except for complex unsigned fixed-point inputs. Fixed-point signals are not supported for the frequency domain.

Convolution with Signal Processing Blockset Blocks

The general equation for convolution is

$$y(k) = \sum_{n} u(n-k) * h(k)$$

There are two Signal Processing Blockset blocks that can be used for this purpose:

- Convolution
- Digital Filter

The Convolution block assumes that all of u and h are available at each Simulink time step, and computes the entire convolution at every one.

The Digital Filter block can be used for convolving signals in situations where all of h is available at each time step, but u is a sequence that comes in over the life of the simulation. When you use the Digital Filter block, the convolution is computed only once. To convolve inputs with the Digital Filter block, you must set the **Transfer function type** to `FIR (all zeros)`.

Use the following questions to help you determine which block best fits your needs:

Question	Answer	Recommended Block(s)
How many convolutions do you intend to perform?	Many convolutions, one at each time step	• Convolution block
	One convolution over the life of the simulation	• Convolution block • Digital Filter block (in FIR mode)
How long are your input sequences?	Both sequences have a finite length	• Convolution block • Digital Filter block (in FIR mode)
	One sequence has an infinite (not predetermined) length	• Digital Filter block (in FIR mode)
How many of the inputs are scalar sample-based streams?	None	• Convolution block • Digital Filter block (in FIR mode)
	One or both	• Buffer block followed by the Convolution block • Digital Filter block (in FIR mode)

Convolving Frame-Based Inputs

When the inputs to the Convolution block are a frame based M_u-by-N input matrix u and an M_v-by-N input matrix v, the output, y, is a sample-based (M_u+M_v-1)-by-N matrix whose jth column has elements

$$y_{i,j} = \sum_{k=0}^{\max(M_u,M_v)-1} u_{k,j}v_{(i-k),j} \quad 0 \le i \le (M_u + M_v - 2)$$

Inputs u and v are zero when indexed outside of their valid ranges. When both inputs are real, the output is real; when one or both inputs are complex, the output is complex.

When one input is a column vector (single channel) and the other is a matrix (multiple channels), the single-channel input is independently convolved with each channel of the multichannel input. For example, when u is a M_u-by-1 column vector and v is an M_v-by-N matrix, the output is an (M_u+M_v-1)-by-N matrix whose jth column has elements

$$y_{i,j} = \sum_{k=0}^{\max(M_u,M_v)-1} u_k v_{(i-k),j} \quad 0 \le i \le (M_u + M_v - 2)$$

Convolving Sample-Based Inputs

The Convolution block supports sample-based N-D input arrays. The convolution of N-D array input is always computed across the first dimension. If both inputs are N-D arrays, the size of their first dimension can differ, but the size of all other dimensions must be equal. For example, when u is an M_u-by-N-by-P array, and v is an M_v-by-N-by-P array, the output is an (M_u+M_v-1)-by-N-by-P array.

When one input is an N-D sample-based array and the other is a vector, the vector is independently convolved with the first dimension of the N-D input. For example, when u is a M_u-by-1 column vector and v is an M_v-by-N-by-P array, the output is an (M_u+M_v-1)-by-N-by-P array.

The Convolution block also accepts two vector inputs. When u and v are sample-based vectors with lengths M_u and M_v, the Convolution block performs the vector convolution

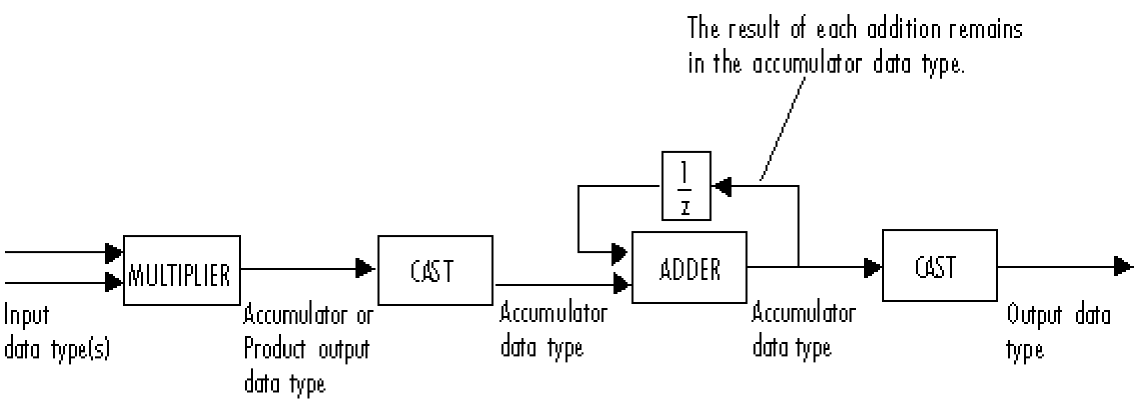
$$y_i = \sum_{k=0}^{\max(M_u, M_v)-1} u_k v_{(i-k)} \quad 0 \le i \le (M_u + M_v - 2)$$

The dimensions of the sample-based output vector are determined by the dimensions of the input vectors:

- When both inputs are row vectors, or when one input is a row vector and the other is a 1-D vector, the output is a 1-by-(M_u+M_v–1) row vector.
- When both inputs are column vectors, or when one input is a column vector and the other is a 1-D vector, the output is a (M_u+M_v–1)-by-1 column vector.
- When both inputs are 1-D vectors, the output is a 1-D vector of length M_u+M_v–1.

Fixed-Point Data Types

The following diagram shows the data types used within the Convolution block for fixed-point signals (time domain only).



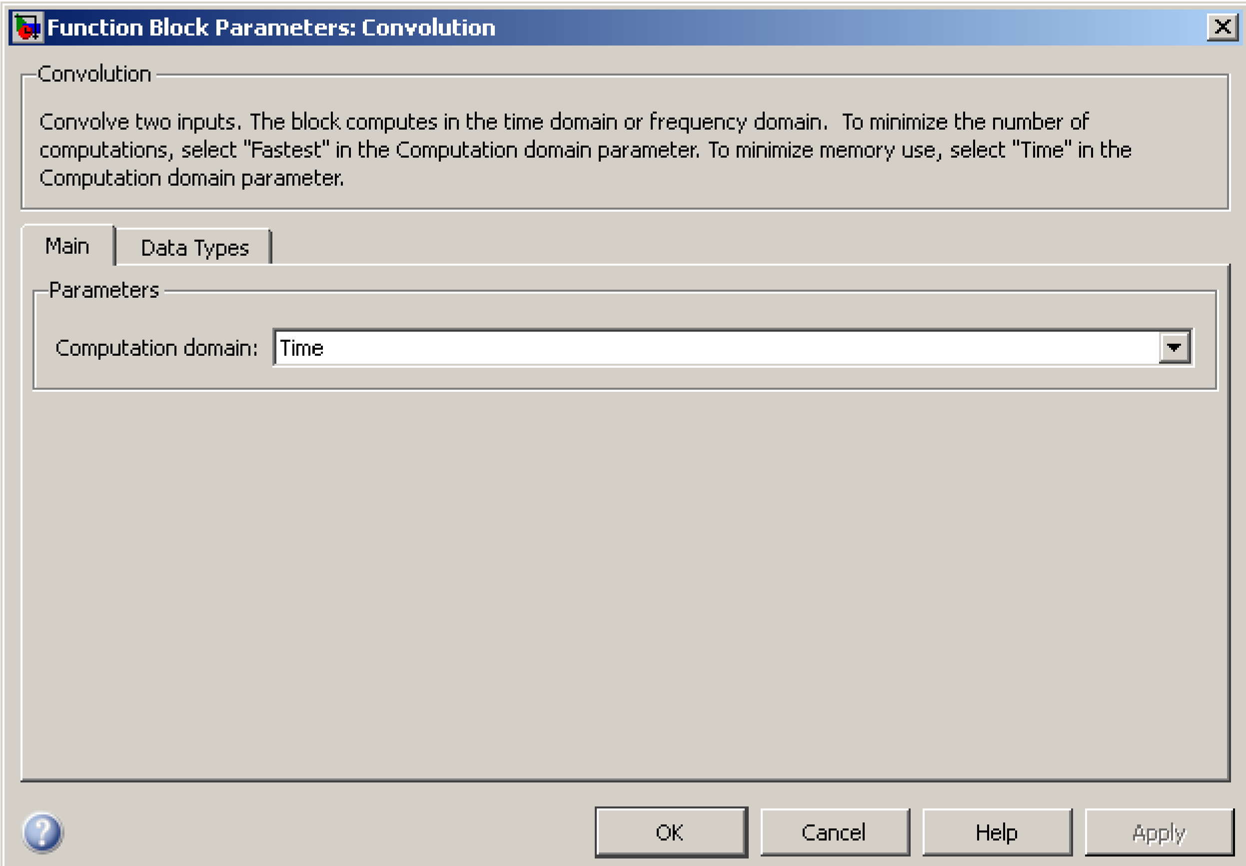
You can set the product output, accumulator, and output data types in the block dialog as discussed in the next section.

The output of the multiplier is in the product output data type when the input is real. When the input is complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see "Multiplication Data Types".

Note When one or both of the inputs are signed fixed-point signals, all internal block data types are signed fixed point. The internal block data types are unsigned fixed point only when *both* inputs are unsigned fixed-point signals.

Dialog Box

The **Main** pane of the Convolution block dialog appears as follows.



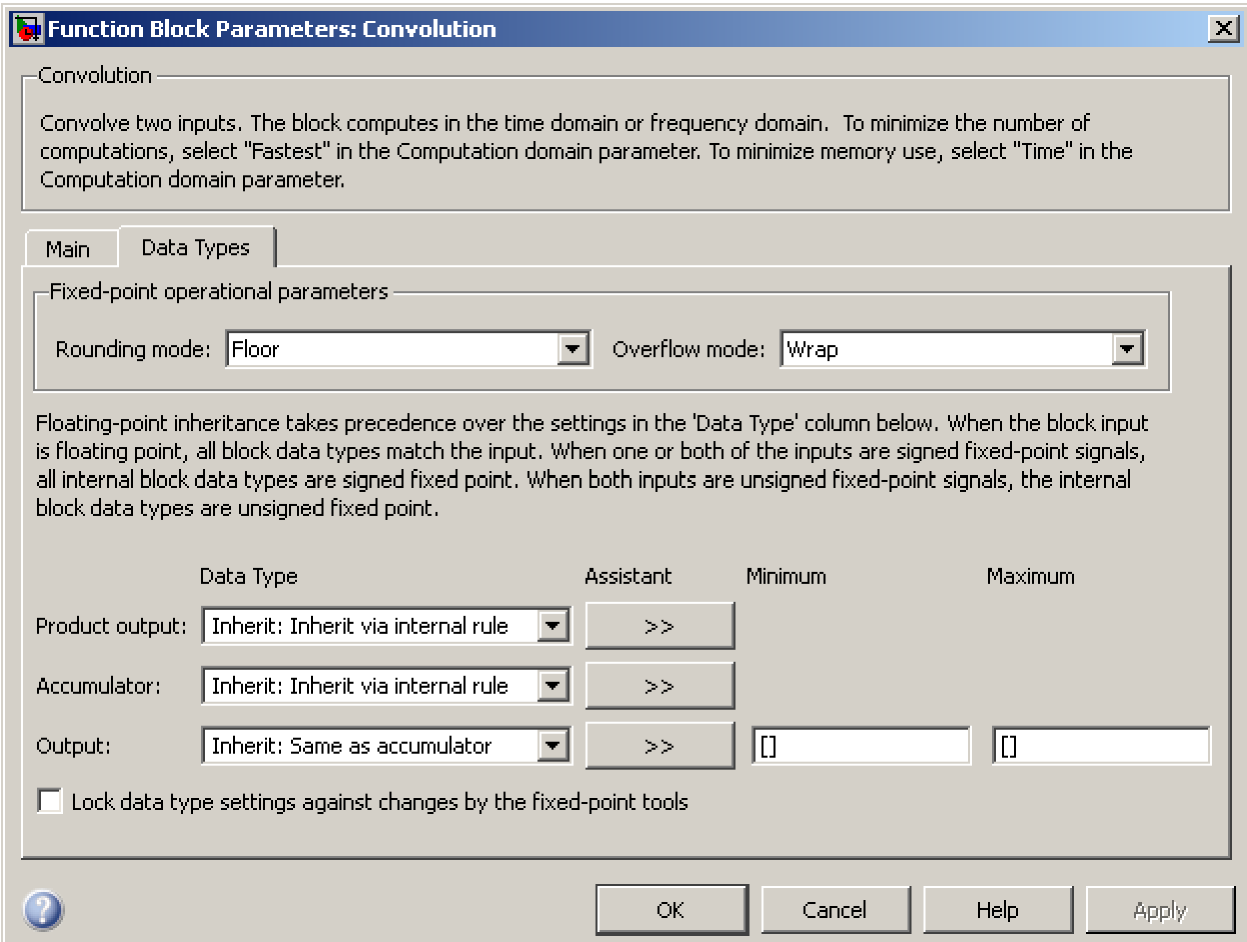
Computation domain

Set the domain in which the block computes convolutions:

- `Time` — The block computes in the time domain, which minimizes memory use.
- `Frequency` — The block computes in the frequency domain, which might require fewer computations than computing in the time domain, depending on the input length.

- `Fastest` — The block computes in the domain, which minimizes the number of computations.

The **Data Types** pane of the Convolution block dialog appears as follows.



Note Fixed-point signals are only supported for the time domain. To use the parameters on this pane, make sure `Time` is selected for the **Computation domain** parameter on the **Main** pane.

Rounding mode
Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode for fixed-point operations.

Product output data type
Specify the product output data type. See “Fixed-Point Data Types” on page 2-226 and “Multiplication Data Types” for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Accumulator data type
Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-226 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`

- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Output data type

Specify the output data type. See "Fixed-Point Data Types" on page 2-226 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as accumulator`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

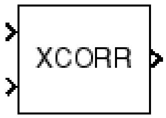
Correlation	Signal Processing Blockset
`conv`	MATLAB

Purpose Compute cross-correlation of two inputs

Library Statistics

`dspstat3`

Description



The Correlation block computes the cross-correlation of the first dimension of a sample-based N-D input array *u*, and the first dimension of a sample-based N-D input array *v*. The block can also independently cross-correlate a sample-based vector with the first-dimension of an N-D input array. For frame-based inputs, the Correlation block computes the cross-correlation of analogous columns of an M_u-by-N input matrix u and an M_v-by-N input matrix v. The Correlation block can also independently cross-correlate a single-channel frame-based column vector with each column of a multiple-channel frame-based matrix.

The frame status of both inputs to the Correlation block must be the same. The output of the block is always sample-based.

The Correlation block accepts real and complex floating-point and fixed-point inputs except for complex unsigned fixed-point inputs. Fixed-point signals are not supported for the frequency domain.

Correlating Frame-Based Inputs

When the inputs to the Correlation block are an M_u-by-N frame-based input matrix u and an M_v-by-N frame-based input matrix v, the output, y, is a sample-based (M_u+M_v-1)-by-N matrix whose jth column has elements

$$y_{uv(i,j)} = \sum_{k=0}^{\max(M_u,M_v)-1} u_{k,j} v^*_{(k-i),j} \qquad 0 \le i < M_v$$

$$y_{uv(i,j)} = y^*_{vu(-i,j)} \qquad -M_u < i < 0$$

where * denotes the complex conjugate. Inputs u and v are zero when indexed outside of their valid ranges. When both inputs are real, the output is real; when one or both inputs are complex, the output is complex.

When one input is a column vector (single channel) and the other is a matrix (multiple channels), the single-channel input is independently cross-correlated with each channel of the multichannel input. Each column of the input represents a separate channel. For example, when u is a M_u-by-1 column vector and v is an M_v-by-N matrix, the output is an (M_u+M_v-1)-by-N matrix whose jth column has elements

$$y_{uv(i,j)} = \sum_{k=0}^{\max(M_u,M_v)-1} u_k v^*_{(k-i),j} \qquad 0 \le i < M_v$$

$$y_{uv(i,j)} = y^*_{vu(-i,j)} \qquad -M_u < i < 0$$

Correlating Sample-Based Inputs

The Correlation block supports sample-based N-D array input. The cross-correlation for sample-based N-D inputs is always computed across the first dimension. If both inputs are N-D arrays, the size of their first dimensions can differ, but the size of all other dimensions must be equal. For example, when u is an M_u-by-N-by-P array and v is an M_v-by-N-by-P array, the output, y, is a sample-based (M_u+M_v-1)-by-N-by-P array.

When one input is an N-D sample-based array and the other is a vector, the vector is independently cross-correlated with each column of the N-D input. For example, when u is a M_u-by-1 column vector and v is an M_v-by-N-by-P array, the output is an $(M_u+M_v\text{-}1)$-by-N-by-P array.

The Correlation block also accepts two vector inputs. When u and v are sample-based column vectors with lengths M_u and M_v, the Correlation block performs the vector cross-correlation according to the following equation:

$$y_{uv(i)} = \sum_{k=0}^{\max(M_u, M_v)-1} u_k v^*_{(k-i)} \qquad 0 \le i < M_v$$

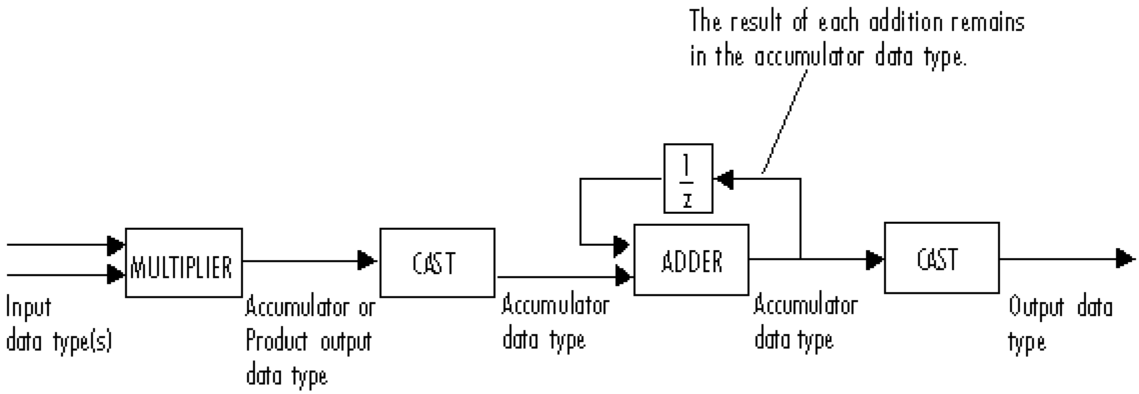
$$y_{uv(i)} = y^*_{vu(-i)} \qquad -M_u < i < 0$$

The dimensions of the sample-based output vector are determined by the dimensions of the input vectors:

- When both inputs are column vectors, or when one input is a column vector and the other is a 1-D vector, the output is a (M_u+M_v–1)-by-1 column vector.
- When both inputs are row vectors, or when one input is a row vector and the other is a 1-D vector, the output is a 1-by-(M_u+M_v–1) row vector.
- When both inputs are 1-D vectors, the output is a 1-D vector of length M_u+M_v–1.

Fixed-Point Data Types

The following diagram shows the data types used within the Correlation block for fixed-point signals (time domain only).



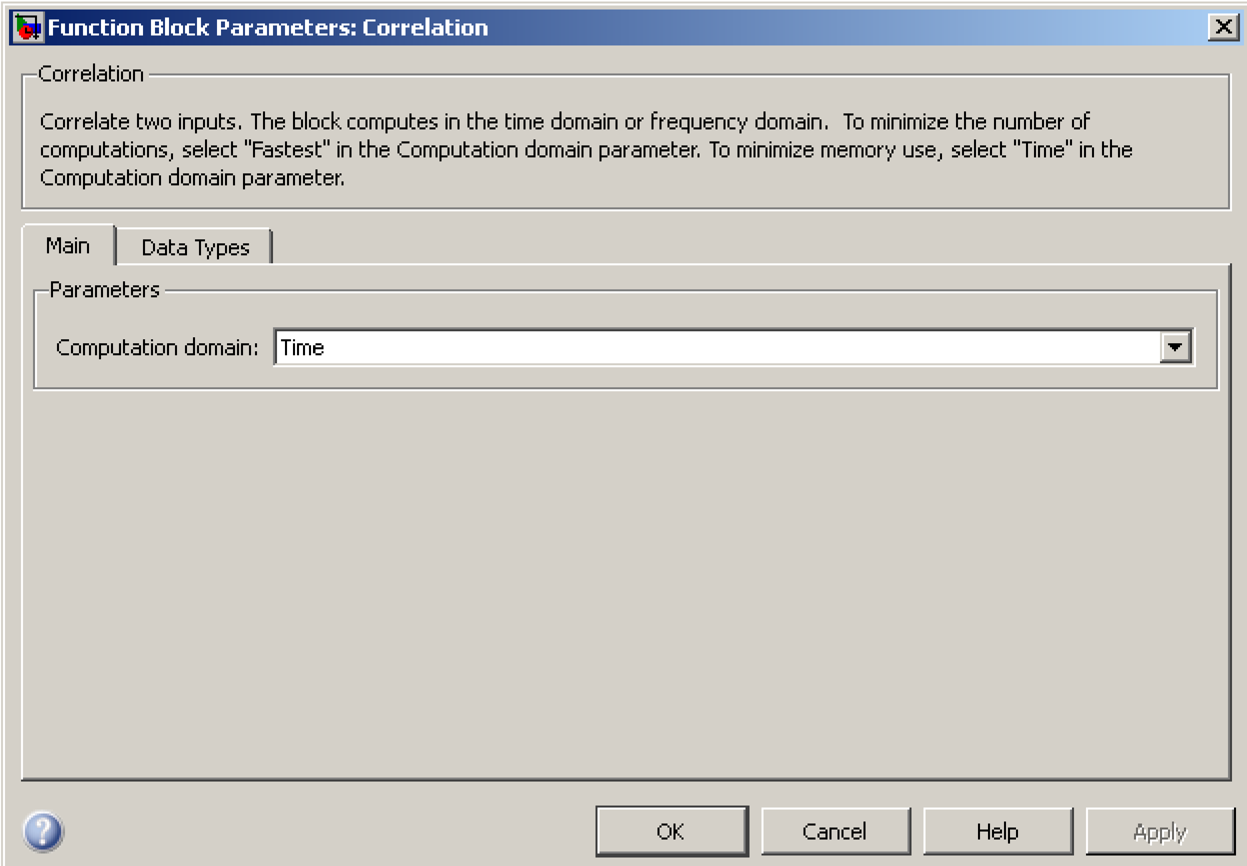
You can set the product output, accumulator, and output data types in the block dialog as discussed in the next section.

The output of the multiplier is in the product output data type when the input is real. When the input is complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see “Multiplication Data Types”.

Note When one or both of the inputs are signed fixed-point signals, all internal block data types are signed fixed point. The internal block data types are unsigned fixed point only when *both* inputs are unsigned fixed-point signals.

Dialog Box

The **Main** pane of the Correlation block dialog appears as follows.



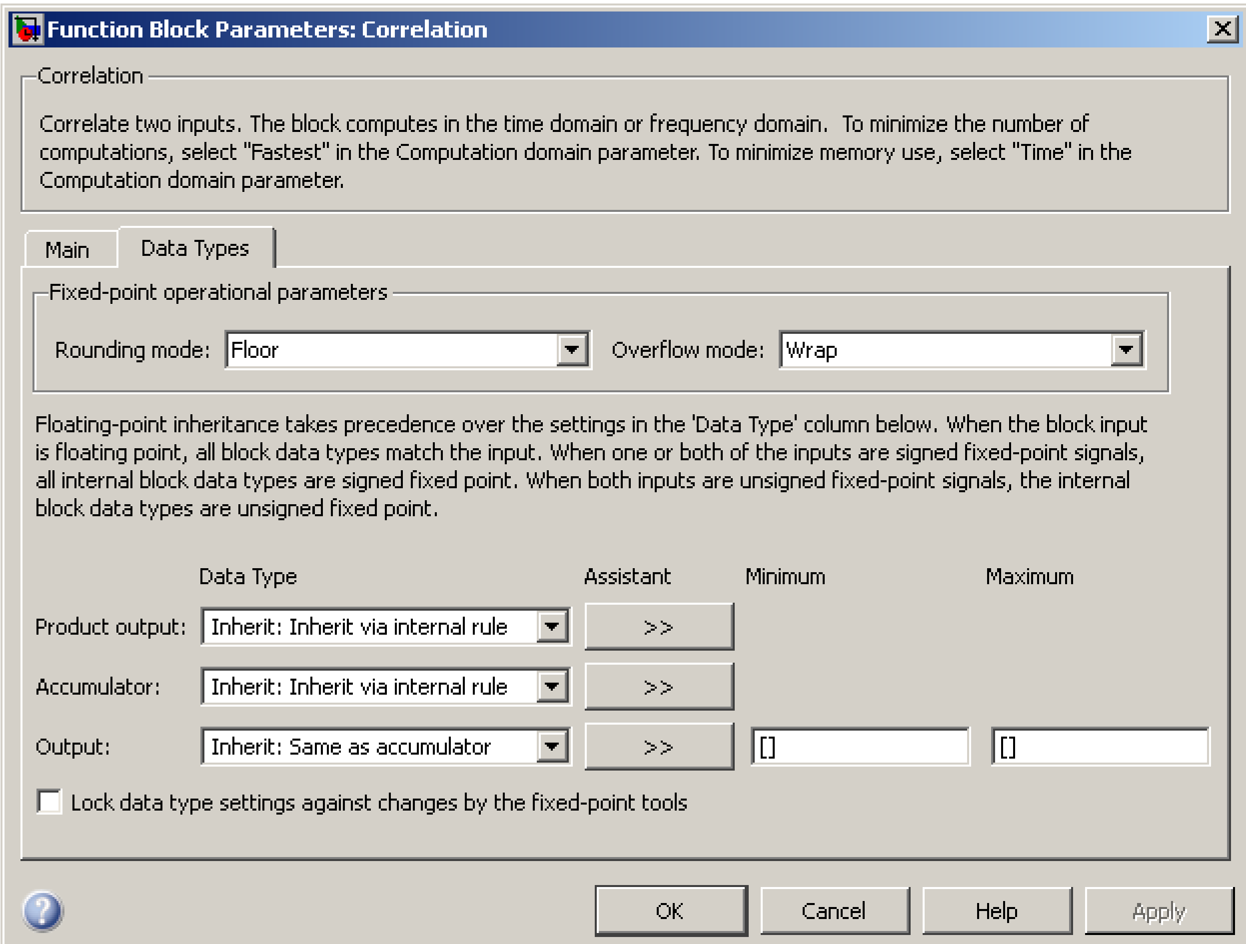
Computation domain

Set the domain in which the block computes correlations:

- `Time` — The block computes in the time domain, which minimizes memory use.
- `Frequency` — The block computes in the frequency domain, which might require fewer computations than computing in the time domain, depending on the input length.

- `Fastest` — The block computes in the domain, which minimizes the number of computations.

The **Data Types** pane of the Correlation block dialog appears as follows.



Note Fixed-point signals are only supported for the time domain. To use the parameters on this pane, make sure `Time` is selected for the **Computation domain** parameter on the **Main** pane.

Rounding mode
: Select the rounding mode for fixed-point operations.

Overflow mode
: Select the overflow mode for fixed-point operations.

Product output data type
: Specify the product output data type. See "Fixed-Point Data Types" on page 2-235 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set it to:

  - A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
  - An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

  Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

  See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Accumulator data type
: Specify the accumulator data type. See "Fixed-Point Data Types" on page 2-235 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

  - A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`

- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Output data type

Specify the output data type. See “Fixed-Point Data Types” on page 2-235 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as accumulator`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User’s Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see “Checking Signal Ranges”)
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Autocorrelation	Signal Processing Blockset
Convolution	Signal Processing Blockset
`xcorr`	Signal Processing Toolbox

Counter

Purpose

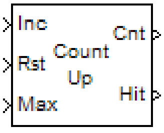
Count up or down through specified range of numbers

Library

Signal Management / Switches and Counters

`dspswit3`

Description



The Counter block increments or decrements an internal counter each time it receives a trigger event at the Inc/Dec port. A trigger event at the Rst port resets the counter to its initial state.

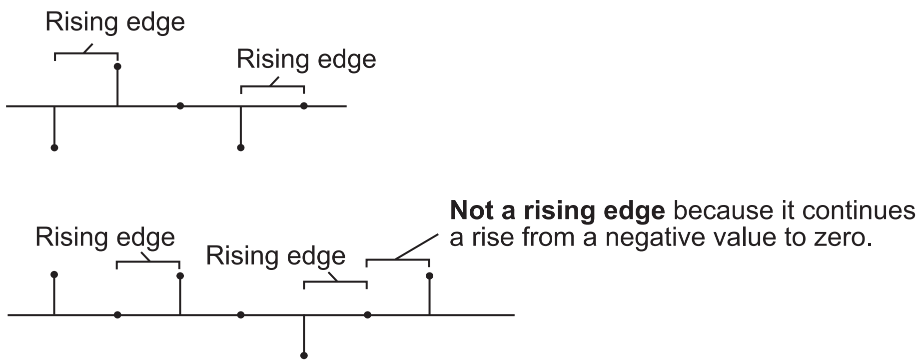
The input to the Rst port must be a real, sample-based scalar. The input to the Inc/Dec port can be a real, sample-based scalar, or a real, frame-based vector (that is, a single channel). When both inputs are sample based, they must have the same sample period. When the Inc/Dec input is frame based, the frame period must equal the sample period of the Rst input.

Sections of This Reference Page

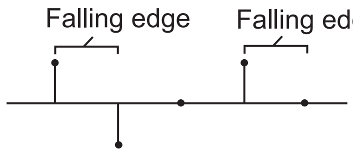
Setting the Count Event Parameter

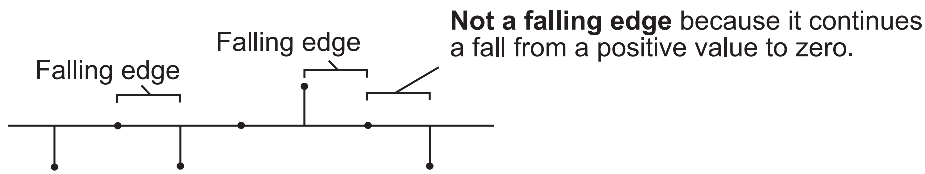
Specify the trigger event for both inputs through the **Count event** parameter. Use one of the following valid values:

- `Rising edge` — Triggers a count or reset operation when the Inc/Dec or Rst input behaves in one of the following ways:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)



- `Falling edge` — Triggers a count or reset operation when the Inc/Dec or Rst input behaves in one of the following ways:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the next figure)





- `Either edge` — Triggers a count or reset operation when the Inc/Dec or Rst input is a `Rising edge` or `Falling edge` (as described above).
- `Non-zero sample` — Triggers a count or reset operation at each sample time when the Inc/Dec or Rst input is not zero.
- `Free running` disables the Inc/Dec port and enables the **Samples per output frame** and **Sample time** parameters. The block increments or decrements the counter at a constant interval, T_s, specified by the **Sample time** parameter. For more information, see "Free-Running Operation" on page 2-247. The Rst port behaves as if the **Count event** parameter were set to `Non-zero sample`.

Note When running simulations in the Simulink `MultiTasking` mode, sample-based reset signals have a one-sample latency, and frame-based reset and clock signals have one frame of latency. Thus, there is a one-sample or one-frame delay between the time the block detects a trigger event at the Inc/Dec or Rst port and when it applies the trigger. For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

Setting the Counter Size and Initial Count Parameters

At the start of the simulation, the block sets the counter to the value specified by the **Initial count** parameter, which can be any unsigned integer in the range defined by the **Counter size** parameter. The **Counter size** parameter allows you to choose from three standard counter ranges, or specify an arbitrary counter limit in the dialog box or via input port:

- `8 bits` specifies a counter with a range of 0 to 255.
- `16 bits` specifies a counter with a range of 0 to 65535.
- `32 bits` specifies a counter with a range of 0 to 2^{32}-1.
- `User defined` enables the **Maximum count** parameter, which allows you to specify any arbitrary unsigned integer representable by the data type specified at the Max port as the upper-count limit. The range of the counter is then 0 to the **Maximum count** value. The data type at the Max port must not allow word lengths larger than the **Counter data type**.
- `Specify via input port` enables the Max port, which allows you to specify any arbitrary integer representable by the data type specified for the **Counter data type** parameter as the upper-count limit.

Sample-Based Operation

The block operates in sample-based mode when the Inc/Dec input is a sample-based scalar. Sample-based vectors and matrices are not accepted.

When the **Count direction** parameter is set to `Up`, a sample-based trigger event at the Inc (increment) input causes the block to increment the counter by one. The block continues incrementing the counter when triggered until the counter value reaches the upper-count limit. At the next Inc/Dec trigger event, the block resets the counter to 0 and resumes incrementing the counter with the subsequent Inc/Dec trigger event.

When the **Count direction** parameter is set to `Down`, a sample-based trigger event at the Dec (decrement) input causes the block to decrement the counter by one. The block continues decrementing the counter when triggered until the counter value reaches 0. At the next Inc/Dec trigger event, the block resets the counter to the upper-count limit and resumes decrementing the counter with the subsequent Inc/Dec trigger event.

Between triggering events the block holds the output at its most recent value. The block resets the counter to its initial state when the trigger event specified in the **Count event** menu is received at the optional Rst input. When the Inc/Dec and Rst ports receive trigger events simultaneously, the block first resets the counter, and then increments or decrements appropriately. If you do not need to reset the counter during the simulation, you can disable the Rst port by clearing the **Reset input** check box.

The **Output** parameter provides three options for the output port configuration of the block icon:

- `Count` configures the block icon to show a Cnt port, which produces the current value of the counter as a sample-based scalar with the same sample period as the inputs.
- `Hit` configures the block icon to show a Hit port, which produces zeros while the value of the counter does not equal the value of any of the integers specified for the **Hit values** parameter. You can specify an integer scalar or a vector of integers for the **Hit values** parameter. When the counter value *does* equal any of the values specified for the **Hit values** parameter, the block generates a value of `1` at the `Hit` port. This option produces sample-based outputs with the same sample period as the inputs.
- `Count and Hit` configures the block icon with both ports.

Frame-Based Operation

The Counter block operates in frame-based mode when the Inc/Dec input is a frame-based vector. The block does not accept multichannel frame-based inputs.

Frame-based operation is the same as sample-based operation, except that the block increments or decrements the counter by the total number of trigger events contained in the Inc/Dec input frame. Thus, the counter may change multiple times during the processing of a single Inc/Dec input frame.

When the block has a hit port, it outputs a value of `1` if any of the specified **Hit values** match any of the counter values during the processing of the Inc/Dec input frame.

When a trigger event splits across two consecutive frames, it is counted in the frame that contains the conclusion of the event. When the Rst port receives a trigger event first, the block first resets the counter, and then increments or decrements the counter by the number of trigger events contained in the Inc/Dec frame.

The Cnt and Hit outputs are sample-based scalars with sample period equal to the Inc/Dec input frame period.

Free-Running Operation

The block operates in free-running mode when you select `Free running` for the **Count event** parameter.

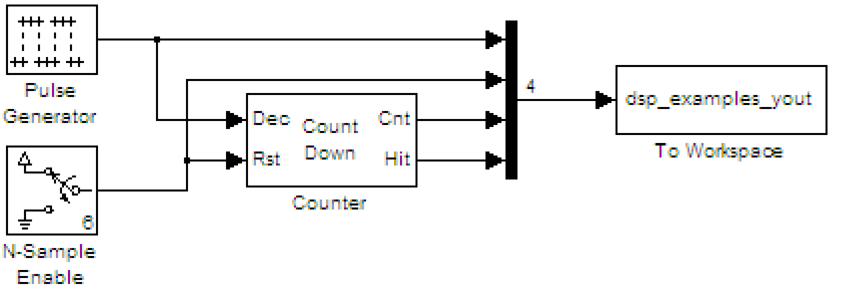
The Rst port behaves as if the **Count event** parameter were set to `Non-zero sample`: the block triggers a reset at each sample time that the Rst input is not zero.

The Inc/Dec input port is disabled in this mode, and the block simply increments or decrements the counter using the constant sample period specified by the **Sample time** parameter, T_s.

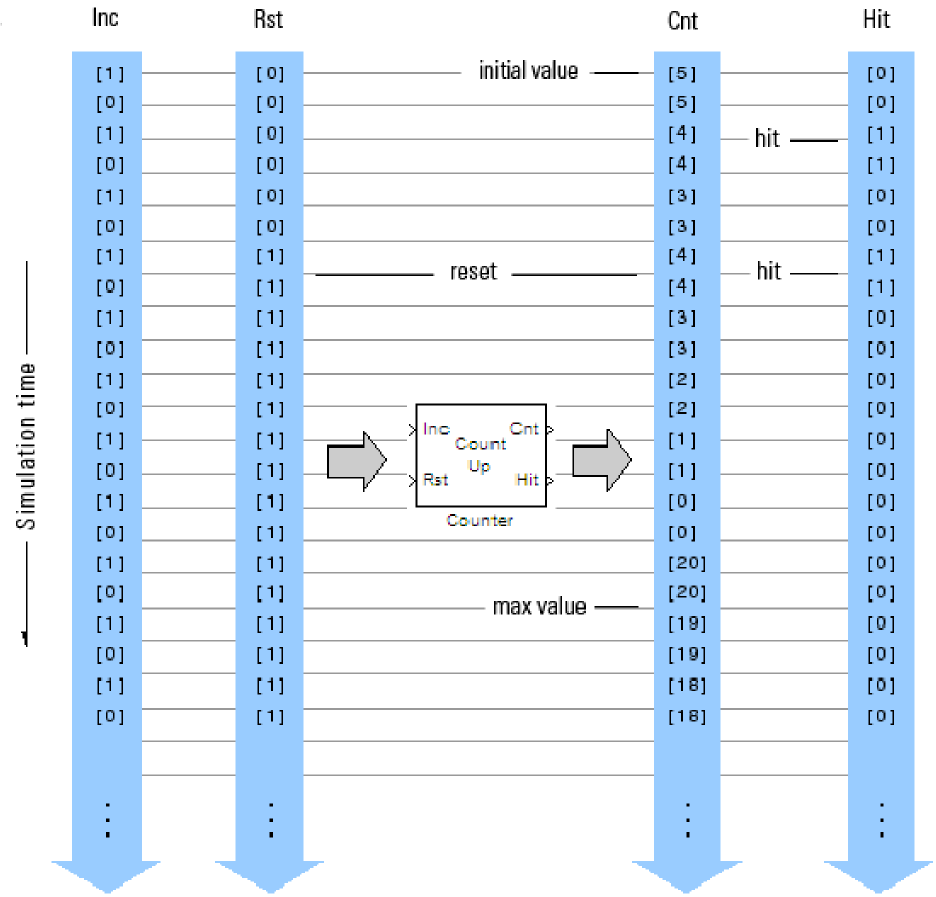
In this mode, the Cnt output is a frame-based M-by-1 matrix containing the count value at each of M consecutive sample times, where M is specified by the **Samples per output frame** parameter. The Hit output is a frame-based M-by-1 matrix containing the hit status (`0` or `1`) at each of those M consecutive sample times. Both outputs have a frame period of $M*T_s$.

Examples

In the following model, the Simulink Pulse Generator block drives the Dec port of the Counter block, and the N-Sample Enable block triggers the Rst port. All the Counter block's inputs and outputs are multiplexed into a single To Workspace block using a 4-port Mux block.

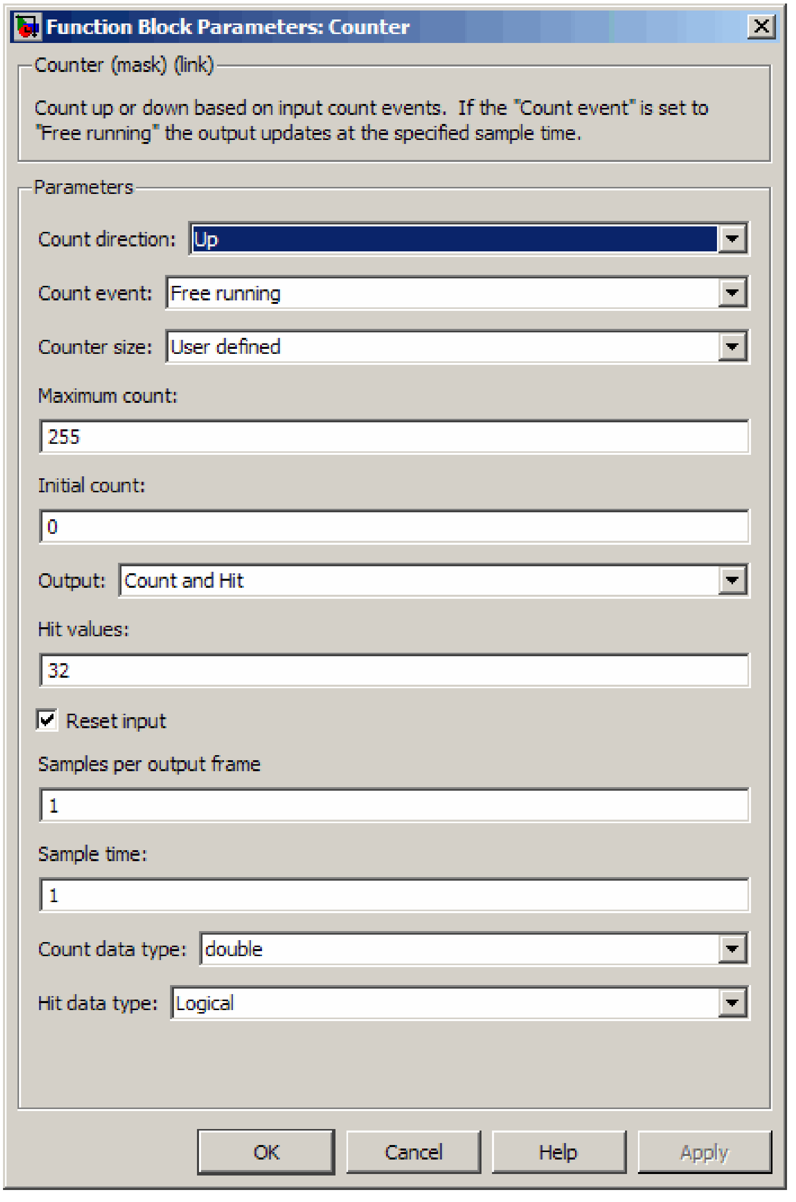


The following figure shows the first 22 samples of the model's four-column output, `yout`.



You can see that the seventh input samples to both the Dec and Rst ports of the Counter block represent trigger events (rising edges), so at this time step the block first resets the counter to its initial value of 5, and then immediately decrements the count to 4. When the counter reaches its minimum value of 0, it rolls over to its maximum value of 20 with the subsequent trigger event at the Cnt port.

Dialog Box


Function Block Parameters: Counter
Counter (mask) (link)
Count up or down based on input count events. If the "Count event" is set to "Free running" the output updates at the specified sample time.
Parameters
Count direction: Up
Count event: Free running
Counter size: User defined
Maximum count:
255
Initial count:
0
Output: Count and Hit
Hit values:
32
Reset input
Samples per output frame
1
Sample time:
1
Count data type: double
Hit data type: Logical
OK
Cancel
Help
Apply

Count direction
: Specify whether to count `Up` or `Down`. The port label on the block icon changes to Inc (increment) or Dec (decrement) based on the value of this parameter. Tunable in Simulink Normal mode.

Count event
: Specify the type of event that triggers the block to increment, decrement, or reset the counter when received at the Inc/Dec or Rst ports. `Free running` disables the Inc/Dec port, and counts continuously with the period specified by the **Sample time** parameter. For more information on all the possible settings, see "Setting the Count Event Parameter" on page 2-242.

Counter size
: Specify the range of integer values the block should count through before recycling to zero. For more information, see "Setting the Counter Size and Initial Count Parameters" on page 2-245.

Maximum count
: Specify the counter's maximum value when **Counter size** is set to `User defined`. You can set the **Maximum count** parameter to any integer representable by the data type specified for the **Counter data type** parameter. Tunable in Simulink Normal mode.

Initial count
: Specify the counter's initial value at the start of the simulation and after reset. Tunable.

Output
: Select the output ports to enable: Cnt, Hit, or both.

Hit values
: Specify a scalar or vector of integers whose occurrence in the count should be flagged by a `1` at the (optional) Hit output. This parameter is available when `Hit` or `Count and Hit` are selected in the **Output** menu. Tunable.

Reset input
: Select to enable the Rst input port.

Samples per output frame
Specify the number of samples, M, in each output frame. This parameter is available when you select `Free running` in the **Count event** menu.

Sample time
Specify the output sample period, T_s, in free-running mode. This parameter is available when you select `Free running` in the **Count event** menu.

Count data type
Specify the data type of the Cnt output. This parameter is available when the **Output** parameter is set to `Count` or `Count and Hit`.

Hit data type
Specify the data type of the Hit output. This parameter is available when the **Output** parameter is set to `Hit` or the **Output** parameter is set to `Count and Hit` and the **Count data type** parameter is set to `Double`.

Supported Data Types

Port	Supported Data Types
Inc/Dec	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Rst	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
Max	• 8-, 16-, and 32-bit unsigned integers
Cnt	• Double-precision floating point • Single-precision floating point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Hit	• Logical • Boolean — The block might output Boolean values from the `Hit` output port depending on the setting of the **Hit data type** parameter.

See Also

Edge Detector	Signal Processing Blockset
N-Sample Enable	Signal Processing Blockset
N-Sample Switch	Signal Processing Blockset

Covariance AR Estimator

Purpose

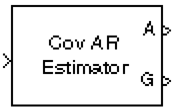
Compute estimate of autoregressive (AR) model parameters using covariance method

Library

Estimation / Parametric Estimation

`dspparest3`

Description



The Covariance AR Estimator block uses the covariance method to fit an autoregressive (AR) model to the input data. This method minimizes the forward prediction error in the least squares sense.

The input is a sample-based vector (row, column, or 1-D) or frame-based vector (column only) representing a frame of consecutive time samples from a single-channel signal, which is assumed to be the output of an AR system driven by white noise. The block computes the normalized estimate of the AR system parameters, $A(z)$, independently for each successive input frame.

$$H(z) = \frac{G}{A(z)} = \frac{G}{1 + a(2)z^{-1} + \ldots + a(p+1)z^{-p}}$$

The order, p, of the all-pole model is specified by the **Estimation order** parameter. To guarantee a valid output, you must set the **Estimation order** parameter to be less than or equal to half the input vector length.

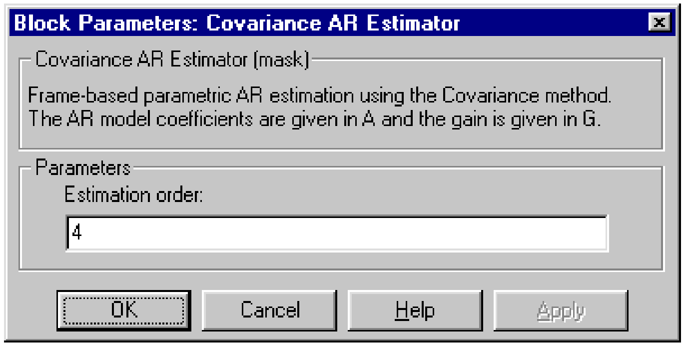
The top output, `A`, is a column vector of length p+1 with the same frame status as the input, and contains the normalized estimate of the AR model coefficients in descending powers of z.

```
[1 a(2) ... a(p+1)]
```

The scalar gain, G, is provided at the bottom output (`G`).

See the Burg AR Estimator block reference page for a comparison of the Burg AR Estimator, Covariance AR Estimator, Modified Covariance AR Estimator, and Yule-Walker AR Estimator blocks.

Dialog Box



Estimation order

The order of the AR model, p. To guarantee a nonsingular output, you must set p to be less than or equal to half the input length. Otherwise, the output might be singular.

References

Kay, S. M. *Modern Spectral Estimation: Theory and Application.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

Marple, S. L., Jr., *Digital Spectral Analysis with Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1987.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point
A	• Double-precision floating point • Single-precision floating point
G	• Double-precision floating point • Single-precision floating point

See Also

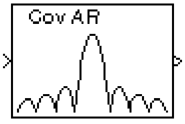
Burg AR Estimator	Signal Processing Blockset
Covariance Method	Signal Processing Blockset
Modified Covariance AR Estimator	Signal Processing Blockset
Yule-Walker AR Estimator	Signal Processing Blockset
`arcov`	Signal Processing Toolbox

Purpose Power spectral density estimate using covariance method

Library Estimation / Power Spectrum Estimation

`dspspect3`

Description



The Covariance Method block estimates the power spectral density (PSD) of the input using the covariance method. This method fits an autoregressive (AR) model to the signal by minimizing the forward prediction error in the least squares sense. The **Estimation order** parameter specifies the order of the all-pole model. The block computes the spectrum from the FFT of the estimated AR model parameters. To guarantee a valid output, the **Estimation order** parameter must be less than or equal to half the input vector length.

The input is a sample-based vector (row, column, or 1-D) or frame-based vector (column only). It represents a frame of consecutive time samples from a single-channel signal. The block outputs a column vector containing the estimate of the power spectral density of the signal at N_{fft} equally spaced frequency points. The frequency points are in the range $[0,F_s)$, where F_s is the sampling frequency of the signal.

Selecting **Inherit FFT length from estimation order**, specifies that N_{fft} is one greater than the estimation order. Clearing the **Inherit FFT length from estimation order** check box allows you to use the **FFT length** parameter to specify N_{fft} as a power of 2. The block zero-pads or wraps the input to N_{fft} before computing the FFT. The output is always sample based.

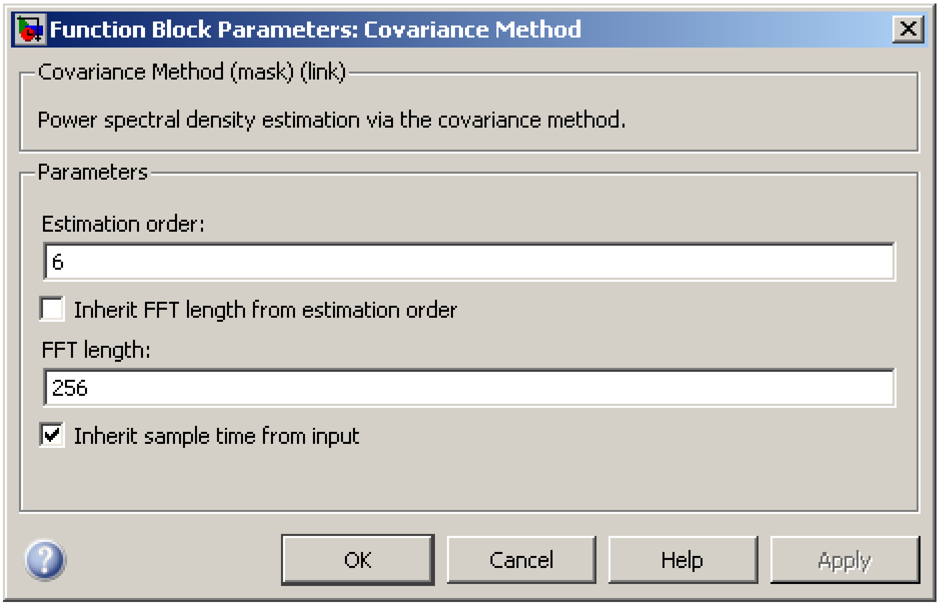
When you select the **Inherit sample time from input** check box, the block computes the frequency data from the sample period of the input signal. For the block to produce valid output, the following conditions must hold:

- The input to the block is the original signal, with no samples added or deleted (by insertion of zeros, for example).
- The sample period of the time-domain signal in the simulation equals the sample period of the original time series.

If these conditions do not hold, clear the **Inherit sample time from input** check box. You can then specify a sample time using the **Sample time of original time series** parameter.

See the Burg Method block reference for a comparison of the Burg Method, Covariance Method, Modified Covariance Method, and Yule-Walker Method blocks.

Dialog Box



Estimation order

The order of the AR model. To guarantee a nonsingular output, the value of this parameter must be less than or equal to half the input length.

Inherit FFT length from estimation order

When selected, this option specifies that the FFT length is one greater than the estimation order.

FFT length

Enter the number of data points on which to perform the FFT, N_{fft}. When N_{fft} is larger than the input frame size, the block zero-pads each frame as needed. When N_{fft} is smaller than the input frame size, the block wraps each frame as needed. This parameter becomes visible only when you clear the **Inherit FFT length from estimation order** check box.

Inherit sample time from input

When you select the **Inherit sample time from input** check box, the block computes the frequency data from the sample period of the input signal. For the block to produce valid output, the following conditions must hold:

- The input to the block is the original signal, with no samples added or deleted (by insertion of zeros, for example).
- The sample period of the time-domain signal in the simulation equals the sample period of the original time series.

If these conditions do not hold, clear the **Inherit sample time from input** check box. You can then specify a sample time using the **Sample time of original time series** parameter.

Sample time of original time series

Specify the sample time of the original time-domain signal. This parameter becomes visible only when you clear the **Inherit sample time from input** check box.

References

Kay, S. M. *Modern Spectral Estimation: Theory and Application.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

Marple, S. L. Jr., *Digital Spectral Analysis with Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1987.

Orfanidis, S. J. *Introduction to Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point
Output	• Double-precision floating point • Single-precision floating point

See Also

Burg Method	Signal Processing Blockset
Covariance AR Estimator	Signal Processing Blockset
Modified Covariance Method	Signal Processing Blockset
Short-Time FFT	Signal Processing Blockset
Yule-Walker Method	Signal Processing Blockset
`spectrum.cov`	Signal Processing Toolbox

See "Power Spectrum Estimation" for related information.

Purpose

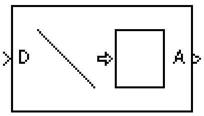
Create square diagonal matrix from diagonal elements

Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description

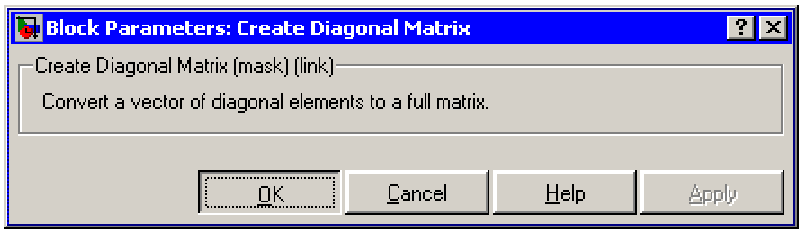


The Create Diagonal Matrix block populates the diagonal of the M-by-M matrix output with the elements contained in the length-M vector input, D. The elements off the diagonal are zero.

```
A = diag(D)     Equivalent MATLAB code
```

The output is always sample based.

Dialog Box



Supported Data Types

Port	Supported Data Types
D	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
A	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Constant Diagonal Matrix	Signal Processing Blockset
Extract Diagonal	Signal Processing Blockset
`diag`	MATLAB

Purpose

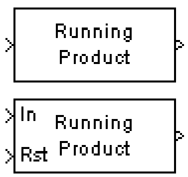
Compute cumulative product of channel, column, or row elements

Library

Math Functions / Math Operations

`dspmathops`

Description



The Cumulative Product block computes the cumulative product of elements in each channel, column, or row of the M-by-N input matrix.

The inputs can be sample-based or frame-based vectors and matrices. The output always has the same dimensions, rate, frame status, data type, and complexity as the input.

The Cumulative Product block accepts real and complex fixed-point and floating-point inputs except for complex unsigned fixed-point inputs.

Valid Input

The block computes the cumulative product of both sample- and frame-based vector and matrix inputs. Inputs can be real or complex. When multiplying along channels or columns, 1-D unoriented vectors are treated as column vectors. When multiplying along rows, 1-D vectors are treated as row vectors.

Valid Reset Signal

The optional reset port, `Rst`, accepts scalar values, which can be any built-in Simulink data type including `boolean`. The rate of the reset signal must be a positive integer multiple of the rate of the data signal input.

Output Characteristics

The output always has the same dimensions, rate, frame status, data type, and complexity as the data signal input.

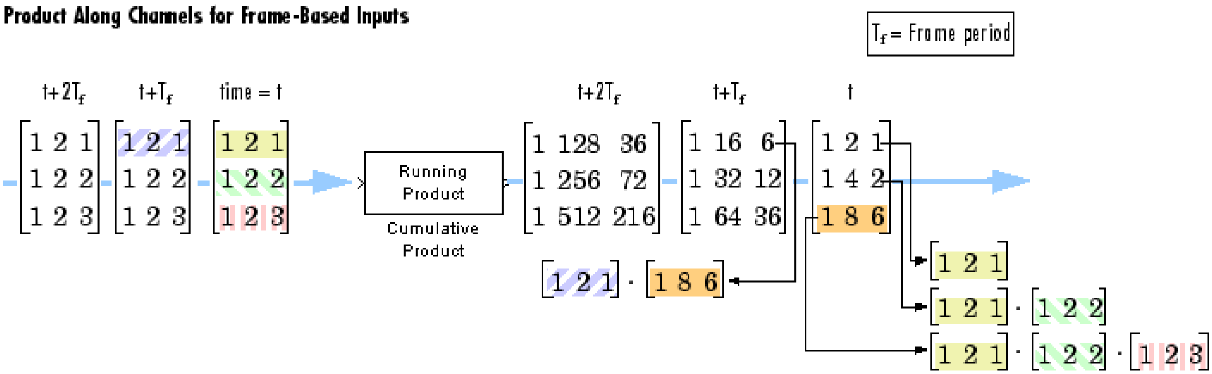
Multiplying Along Channels of Frame-Based Inputs

For frame-based inputs, the block treats each input column as an independent channel. As the following figure and equation illustrate, the output has the following characteristics:

- The first row of the first output is the same as the first row of the first input.
- The first row of each subsequent output is the element-wise product of the first row of the current input (time t), and the last row of the previous output (time t - T_f, where T_f is the frame period).
- The output has the same size, dimension, frame status, data type, and complexity as the input.

Given an M-by-N frame-based input, u, the output, y, is a frame-based M-by-N matrix whose first row has elements

$$y_{1,j}(t) = u_{1,j}(t) \cdot y_{M,j}\left(t - T_f\right)$$



Multiplying Along Channels of Sample-Based Inputs

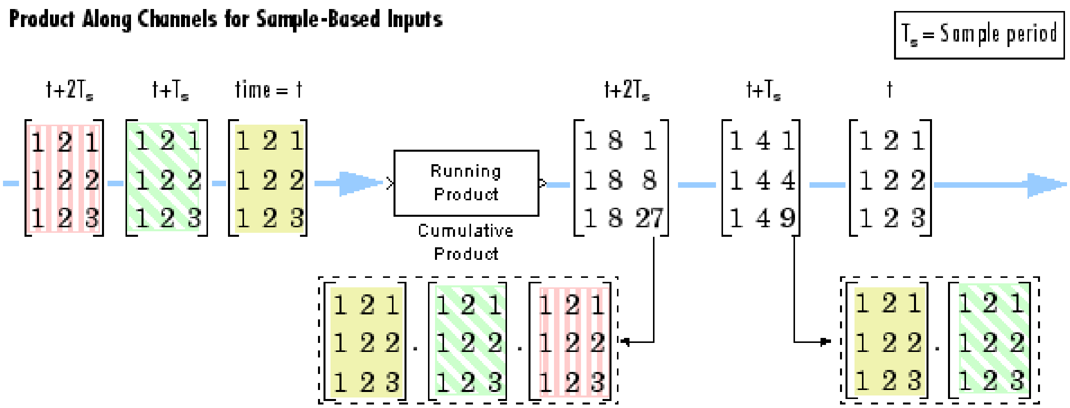
For sample-based inputs, the block treats each element of the input matrix as an independent channel. As the following figure and equation illustrate, the output has the following characteristics:

- The first output is the same as the first input.
- Each subsequent output is the element-wise product of the current input (time t) and the previous output (time t - T_s, where T_s is the sample period).
- The output has the same size, dimension, frame status, data type, and complexity as the input.

Given an M-by-N sample-based input, u, the output, y, is a sample-based M-by-N matrix with the elements

$$y_{i,j}(t) = u_{i,j}(t) \cdot y_{i,j}(t - T_s) \quad \begin{matrix} 1 \le i \le M \\ 1 \le j \le N \end{matrix}$$

For convenience, length-M 1-D vector inputs are treated as M-by-1 column vectors when multiplying along channels, and the output is a length-M 1-D vector.



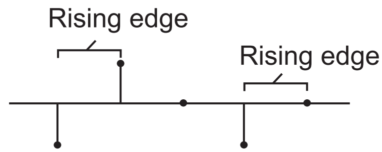
Resetting the Cumulative Product Along Channels

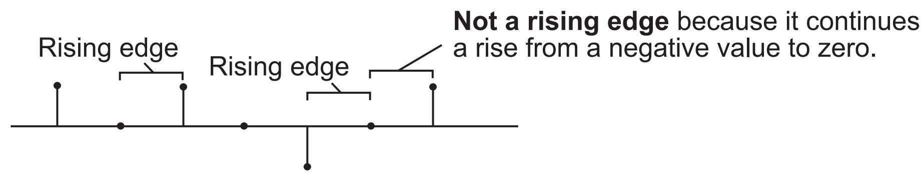
When you set the **Multiply input along** parameter to `Channels (running product)`, you can set the block to reset the running product whenever it detects a reset event at the optional `Rst` port. The reset sample time must be a positive integer multiple of the input sample time. The input to the `Rst` port can be of the Boolean data type.

When the block is reset for sample-based inputs, the block initializes the current output to the values of the current input. For frame-based inputs, the block initializes the first row of the current output to the values in the first row of the current input.

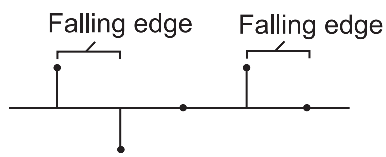
The **Reset port** parameter specifies the reset event, which can be one of the following:

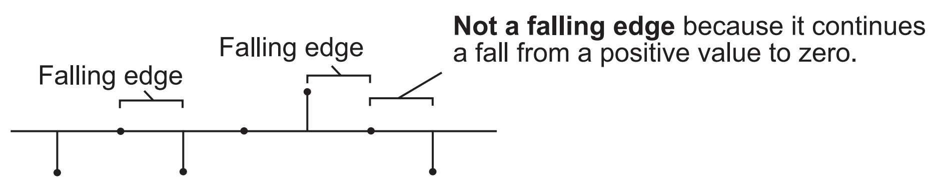
- `None` disables the `Rst` port.
- `Rising edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)





- `Falling edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)





- `Either edge` — Triggers a reset operation when the `Rst` input is a `Rising edge` or `Falling edge` (as described above)
- `Non-zero sample` — Triggers a reset operation at each sample time that the `Rst` input is not zero

---

Note When running simulations in the Simulink `MultiTasking` mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset. For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

Multiplying Along Columns

When the **Multiply input along** parameter is set to `Columns`, the block computes the cumulative product of each column of the input, where the current cumulative product is independent of the cumulative products of previous inputs.

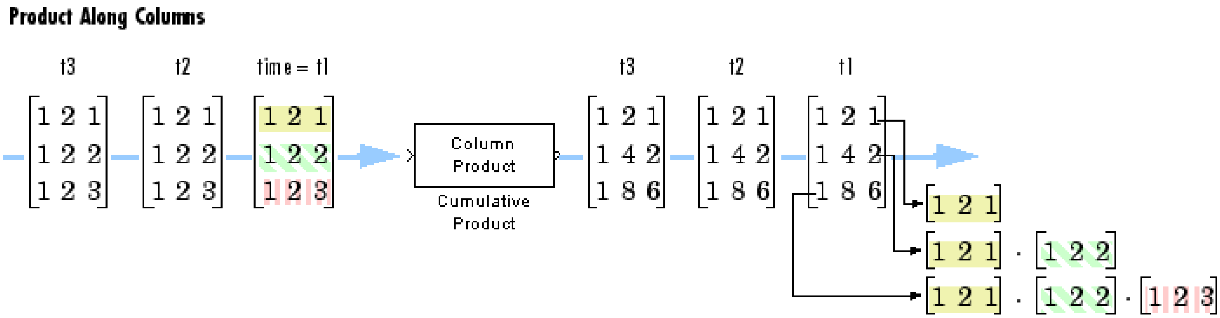
```
y = cumprod(u)      % Equivalent MATLAB code
```

The output has the same size, dimension, frame status, data type, and complexity as the input. The *m*th output row is the element-wise product of the first *m* input rows.

Given an M-by-N input, u, the output, y, is an M-by-N matrix whose jth column has elements

$$y_{i,j} = \prod_{k=1}^{i} u_{k,j} \qquad 1 \le i \le M$$

The block treats length-M 1-D vector inputs as M-by-1 column vectors when multiplying along columns.



Multiplying Along Rows

When the **Multiply input along** parameter is set to `Rows`, the block computes the cumulative product of the row elements, where the current cumulative product is independent of the cumulative products of previous inputs.

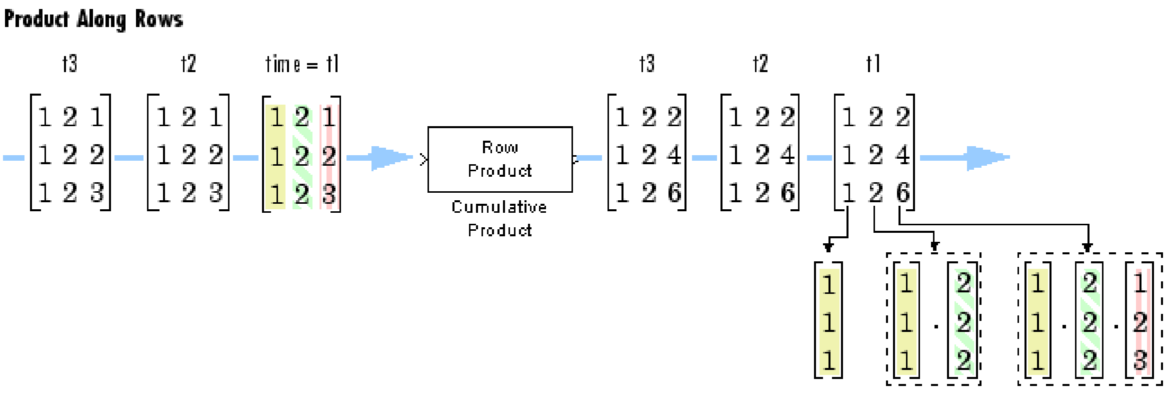
```
y = cumprod(u,2)      % Equivalent MATLAB code
```

The output has the same size, dimension, frame status, and data type as the input. The *n*th output column is the element-wise product of the first *n* input columns.

Given an *M*-by-*N* input, *u*, the output, *y*, is an *M*-by-*N* matrix whose *i*th row has elements

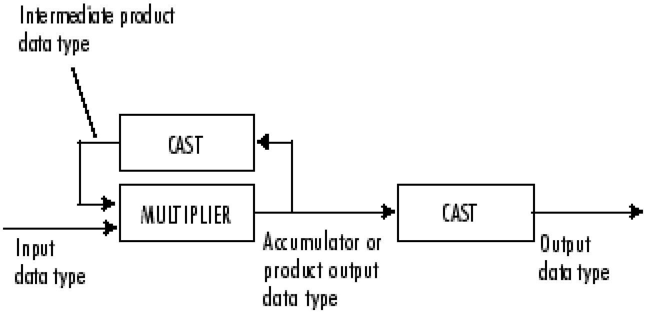
$$y_{i,j} = \prod_{k=1}^{j} u_{i,k} \qquad 1 \le j \le N$$

The block treats length-*N* 1-D vector inputs as 1-by-*N* row vectors when multiplying along rows.



Fixed-Point Data Types

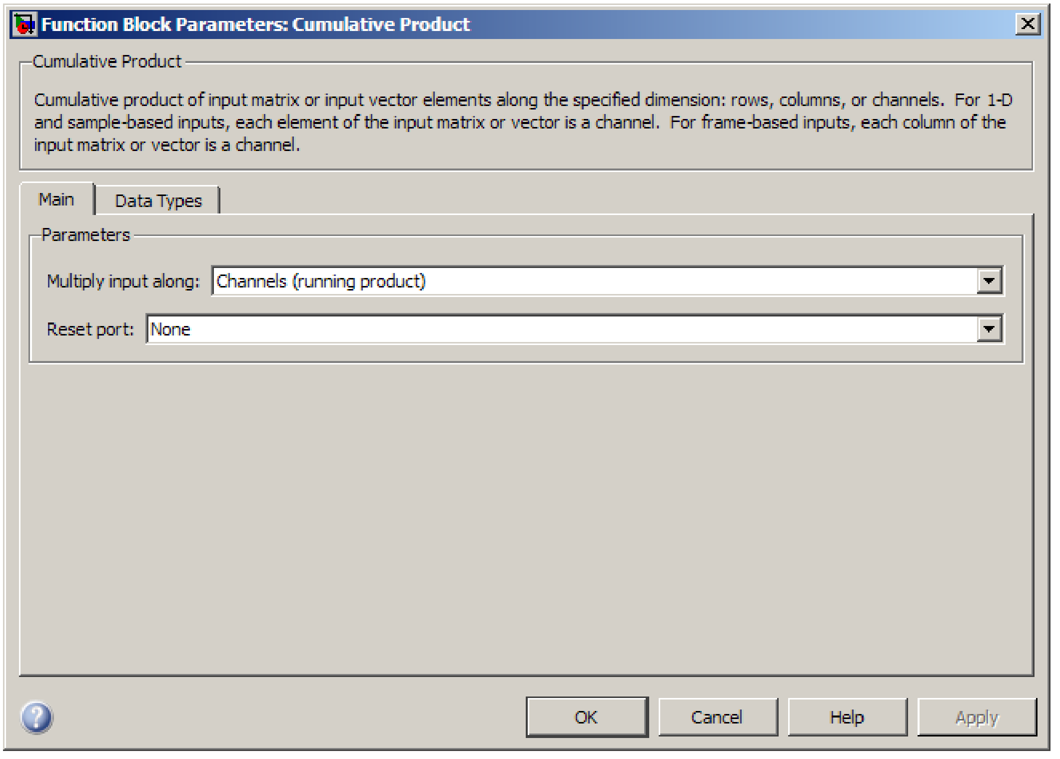
The following diagram shows the data types used within the Cumulative Product block for fixed-point signals.



The output of the multiplier is in the product output data type when at least one of the inputs to the multiplier is real. When both of the inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see "Multiplication Data Types". You can set the accumulator, product output, intermediate product, and output data types in the block dialog as discussed in "Dialog Box" on page 2-271.

Dialog Box

The **Main** pane of the Cumulative Product block dialog appears as follows.



Multiply input along

The dimension along which to compute the cumulative products. The options allow you to multiply along `Channels (running product)`, `Columns`, and `Rows`. For more information, see the following sections:

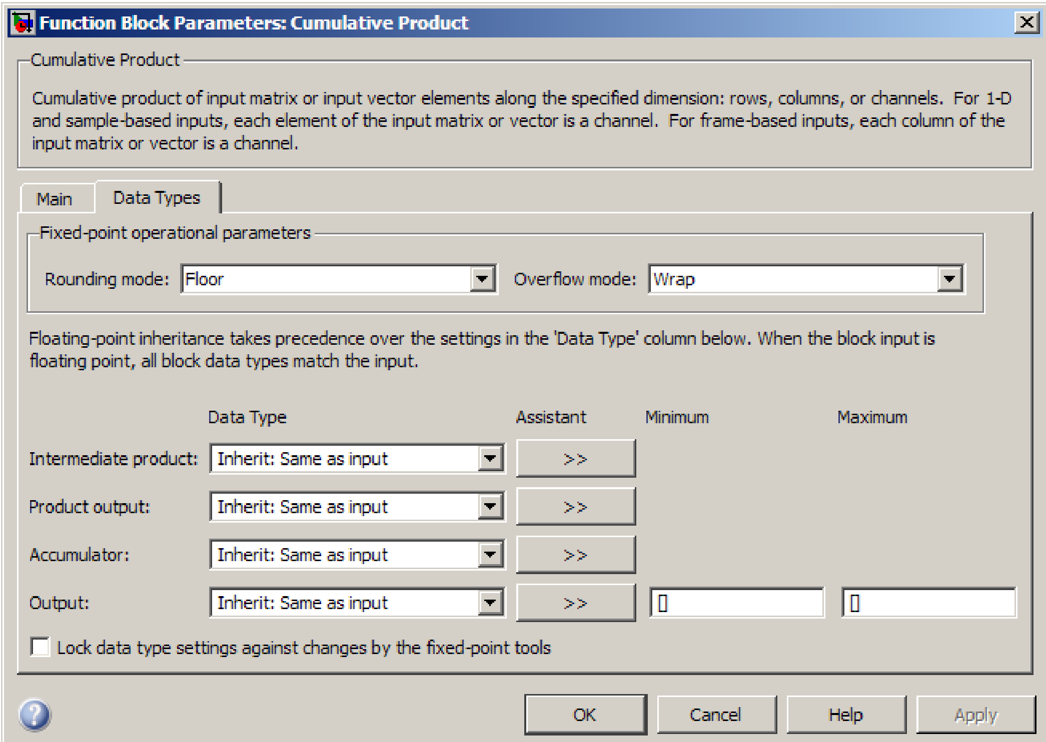
- "Multiplying Along Channels of Frame-Based Inputs" on page 2-264

- "Multiplying Along Channels of Sample-Based Inputs" on page 2-265
- "Multiplying Along Columns" on page 2-268
- "Multiplying Along Rows" on page 2-269

Reset port

Determines the reset event that causes the block to reset the product along channels. The rate of the reset signal must be a positive integer multiple of the rate of the data signal input. This parameter is enabled only when you set the **Multiply input along** parameter to `Channels (running product)`. For more information, see "Resetting the Cumulative Product Along Channels" on page 2-266.

The **Data Types** pane of the Cumulative Product block dialog appears as follows.



Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Rounding mode
Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode for fixed-point operations.

Intermediate product

Specify the intermediate product data type. As shown in "Fixed-Point Data Types" on page 2-270, the output of the multiplier is cast to the intermediate product data type before the next element of the input is multiplied into it. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Product output data type

Specify the product output data type. See "Fixed-Point Data Types" on page 2-270 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Accumulator data type

Specify the accumulator data type. See "Fixed-Point Data Types" on page 2-270 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Output data type

Specify the output data type. See "Fixed-Point Data Types" on page 2-270 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Input and Output Ports	Supported Data Types
Data input port, `In`	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Reset input port, `Rst`	All built-in Simulink data types: • Double-precision floating point • Single-precision floating point • Boolean

Input and Output Ports	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output port	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Cumulative Sum	Signal Processing Blockset
Matrix Product	Signal Processing Blockset
`cumprod`	MATLAB

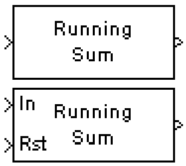
Cumulative Sum

Purpose Compute cumulative sum of channel, column, or row elements

Library Math Functions / Math Operations

`dspmathops`

Description



The Cumulative Sum block computes the cumulative sum of the elements in each channel, column, or row of the M-by-N input matrix.

The inputs can be sample-based or frame-based vectors and matrices. The output always has the same dimensions, rate, frame status, data type, and complexity as the input.

The Cumulative Sum block accepts real and complex fixed-point and floating-point inputs except for complex unsigned fixed-point inputs.

Sections of This Reference Page

Input and Output Characteristics

Valid Input

The block computes the cumulative sum of both sample- and frame-based vector and matrix inputs. Inputs can be real or complex. When summing along channels or columns, 1-D unoriented vectors are treated as column vectors. When summing along rows, 1-D vectors are treated as row vectors.

Valid Reset Signal

The optional reset port, `Rst`, accepts scalar values, which can be any built-in Simulink data type including `boolean`. The rate of the reset signal must be a positive integer multiple of the rate of the data signal input.

Output Characteristics

The output always has the same dimensions, rate, frame status, data type, and complexity as the data signal input.

Summing Along Channels

When the **Sum input along** parameter is set to `Channels (running sum)`, the block computes the cumulative sum of the elements in each input channel. The running sum of the current input takes into account the running sum of all previous inputs. See the following sections for more information:

- "Summing Along Channels of Frame-Based Inputs" on page 2-279
- "Summing Along Channels of Sample-Based Inputs" on page 2-280
- "Resetting the Cumulative Sum Along Channels" on page 2-281

Summing Along Channels of Frame-Based Inputs

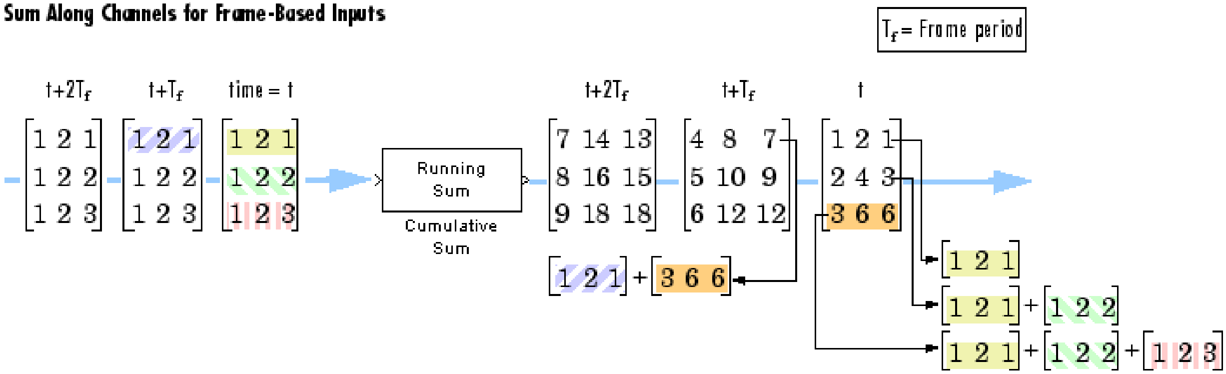
For frame-based inputs, the block treats each input column as an independent channel. As the following figure and equation illustrate, the output has the following characteristics:

- The first row of the first output is the same as the first row of the first input.
- The first row of each subsequent output is the sum of the first row of the current input (time t), and the last row of the previous output (time t - T_f, where T_f is the frame period).
- The output has the same size, dimension, frame status, data type, and complexity as the input.

Given an M-by-N frame-based input, u, the output, y, is a frame-based M-by-N matrix whose first row has elements

$$y_{1,j}(t) = u_{1,j}(t) + y_{M,j}(t - T_f)$$

Sum Along Channels for Frame-Based Inputs



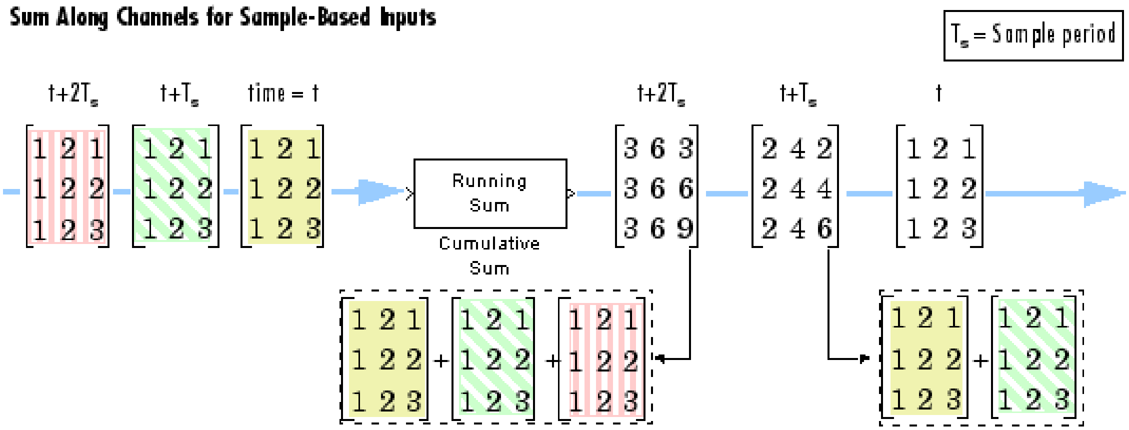
Summing Along Channels of Sample-Based Inputs

For sample-based inputs, the block treats each element of the input matrix as an independent channel. As the following figure and equation illustrate, the output has the following characteristics:

- The first output is the same as the first input.
- Each subsequent output is the sum of the current input (time t) and the previous output (time t - T_s, where T_s is the sample period).
- The output has the same size, dimension, frame status, data type, and complexity as the input.

Given an M-by-N sample-based input, u, the output, y, is a sample-based M-by-N matrix with the elements

$$y_{i,j}(t) = u_{i,j}(t) + y_{i,j}(t - T_s) \qquad \begin{array}{l} 1 \le i \le M \\ 1 \le j \le N \end{array}$$



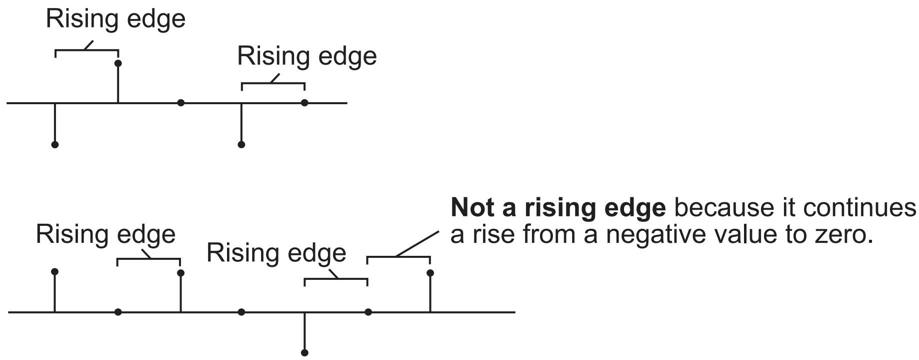
Resetting the Cumulative Sum Along Channels

When you set the **Sum input along** parameter to `Channels (running sum)`, you can set the block to reset the running sum whenever it detects a reset event at the optional `Rst` port. The reset sample time must be a positive integer multiple of the input sample time. The input to the `Rst` port can be of the `boolean` data type.

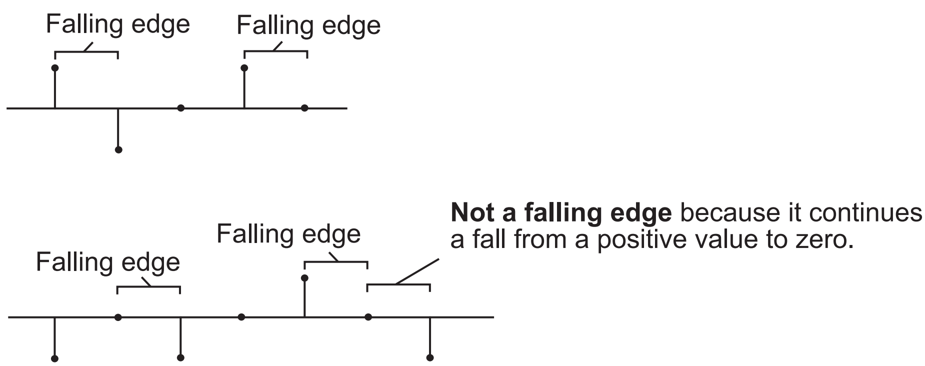
When the block is reset for sample-based inputs, the block initializes the current output to the values of the current input. For frame-based inputs, the block initializes the first row of the current output to the values in the first row of the current input.

The **Reset port** parameter specifies the reset event, which can be one of the following:

- `None` disables the `Rst` port.
- `Rising edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)



- `Falling edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)



- `Either edge` — Triggers a reset operation when the `Rst` input is a `Rising edge` or `Falling edge` (as described above)
- `Non-zero sample` — Triggers a reset operation at each sample time that the `Rst` input is not zero

Note When running simulations in the Simulink `MultiTasking` mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset. For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and the topic on models with multiple sample rates in the Real-Time Workshop documentation.

Summing Along Columns

When the **Sum input along** parameter is set to `Columns`, the block computes the cumulative sum of each column of the input, where the current cumulative sum is independent of the cumulative sums of previous inputs.

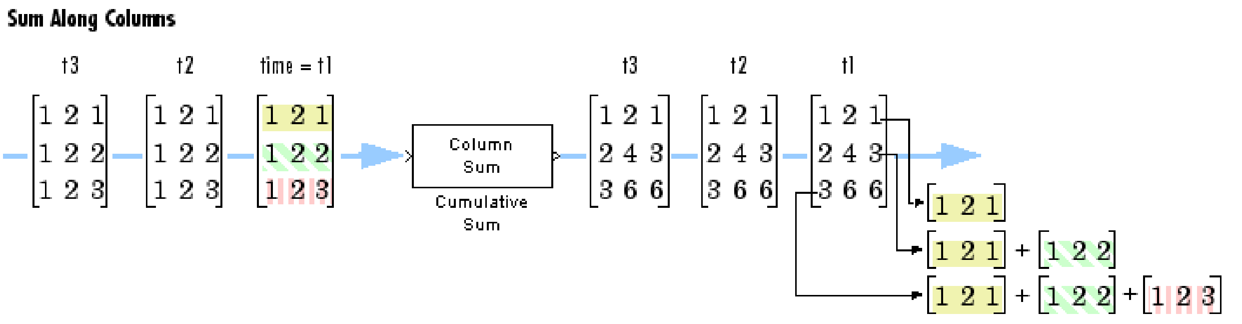
```
y = cumsum(u)      % Equivalent MATLAB code
```

The output has the same size, dimension, frame status, data type, and complexity as the input. The *m*th output row is the sum of the first *m* input rows.

Given an M-by-N input, u, the output, y, is an M-by-N matrix whose jth column has elements

$$y_{i,j} = \sum_{k=1}^{j} u_{k,j} \qquad 1 \le i \le M$$

The block treats length-M 1-D vector inputs as M-by-1 column vectors when summing along columns.



Summing Along Rows

When the **Sum input along** parameter is set to `Rows`, the block computes the cumulative sum of the row elements, where the current cumulative sum is independent of the cumulative sums of previous inputs.

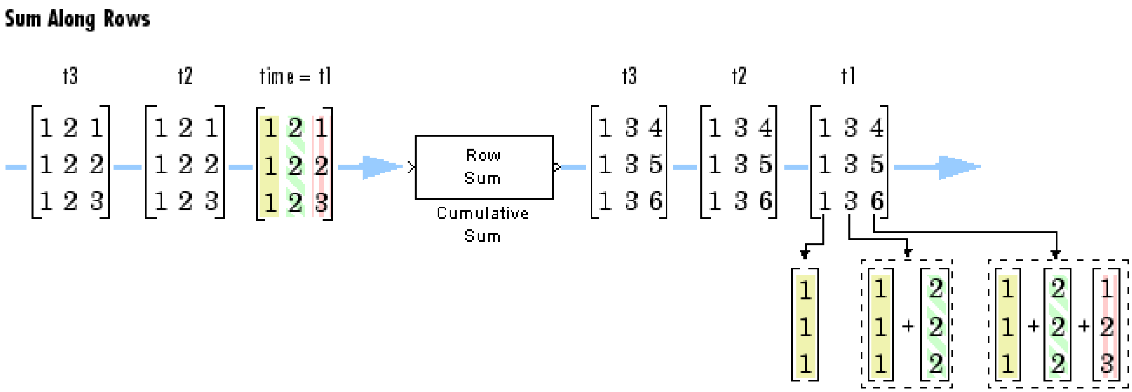
```
y = cumsum(u,2)      % Equivalent MATLAB code
```

The output has the same size, dimension, frame status, and data type as the input. The *n*th output column is the sum of the first *n* input columns.

Given an M-by-N input, u, the output, y, is an M-by-N matrix whose ith row has elements

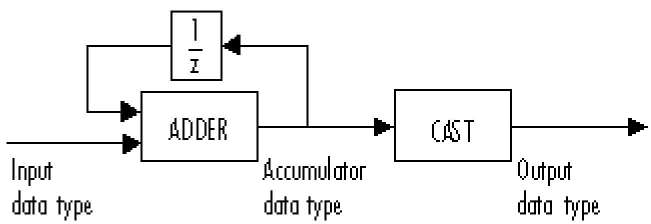
$$y_{i,j} = \sum_{k=1}^{j} u_{i,k} \qquad 1 \le j \le N$$

The block treats length-N 1-D vector inputs as 1-by-N row vectors when summing along rows.



Fixed-Point Data Types

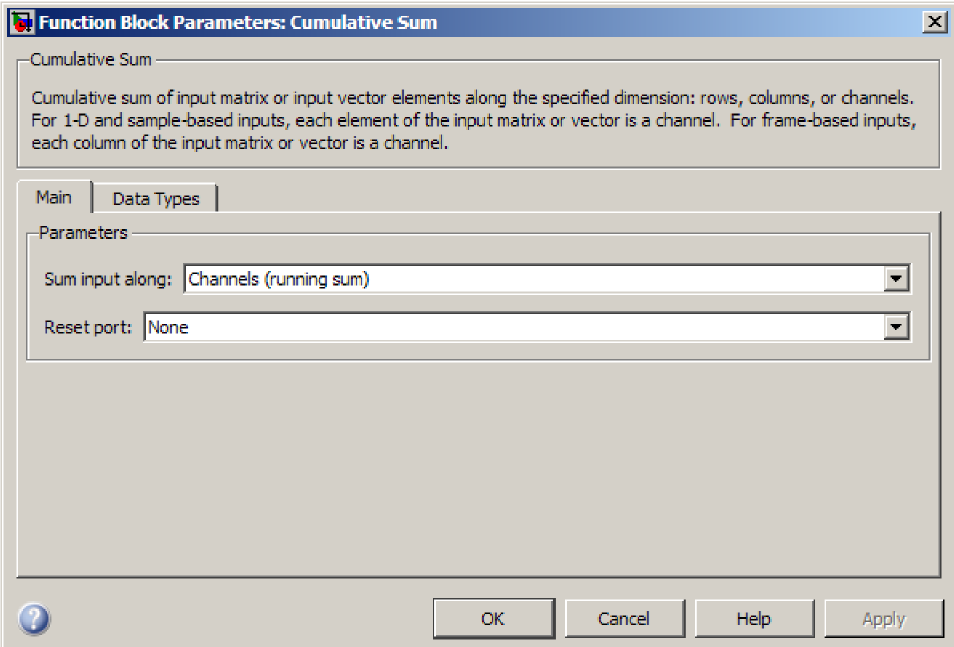
The following diagram shows the data types used within the Cumulative Sum block for fixed-point signals.



You can set the accumulator and output data types in the block dialog as discussed in "Dialog Box" on page 2-285.

Dialog Box

The **Main** pane of the Cumulative Sum block dialog appears as follows.



Sum input along

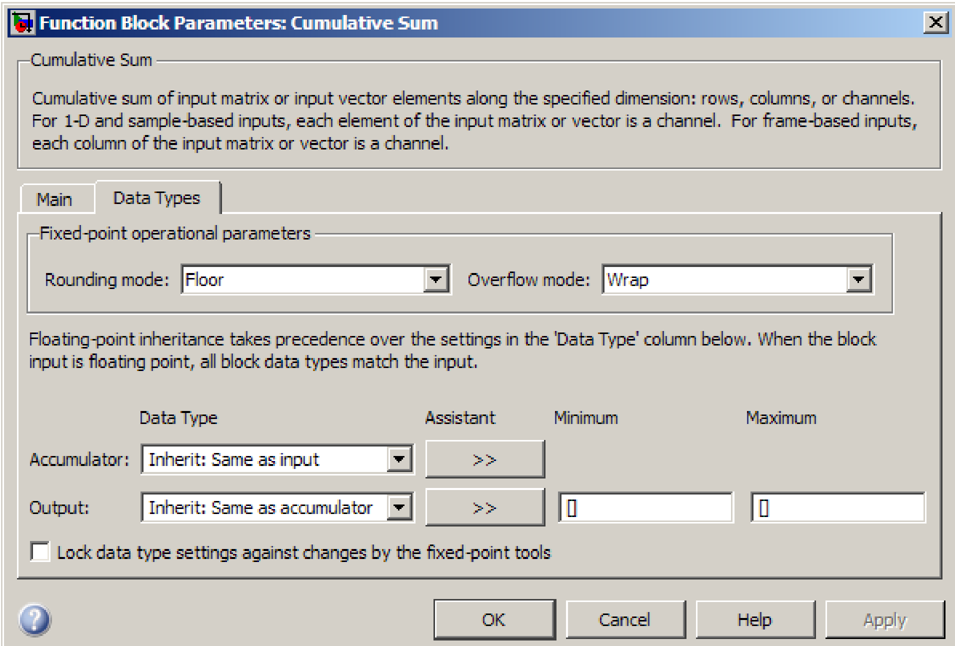
The dimension along which to compute the cumulative summations. The options allow you to sum along `Channels (running sum)`, `Columns`, and `Rows`. For more information, see the following sections:

- "Summing Along Channels" on page 2-279
- "Summing Along Columns" on page 2-283
- "Summing Along Rows" on page 2-284

Reset port

Determines the reset event that causes the block to reset the sum along channels. The rate of the reset signal must be a positive integer multiple of the rate of the data signal input. This parameter is enabled only when you set the **Sum input along** parameter to `Channels (running sum)`. For more information, see “Resetting the Cumulative Sum Along Channels” on page 2-281.

The **Data Types** pane of the Cumulative Sum block dialog appears as follows.



Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode for fixed-point operations.

Accumulator data type
Specify the accumulator data type. See "Fixed-Point Data Types" on page 2-285 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Output data type
Specify the output data type. See "Fixed-Point Data Types" on page 2-285 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as accumulator`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Input and Output Ports	Supported Data Types
Data input port, `In`	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Reset input port, `Rst`	All built-in Simulink data types: • Double-precision floating point • Single-precision floating point • Boolean

Input and Output Ports	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output port	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Cumulative Product	Signal Processing Blockset
Difference	Signal Processing Blockset
Matrix Sum	Signal Processing Blockset
`cumsum`	MATLAB

Purpose Convert input signal to specified data type

Library Signal Management / Signal Attributes

`dspsigattribs`

Description The Data Type Conversion block is an implementation of the Simulink Data Type Conversion block. See Data Type Conversion for more information.

dB Conversion

Purpose

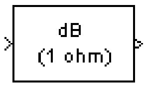
Convert magnitude data to decibels (dB or dBm)

Library

Math Functions / Math Operations

`dspmathops`

Description



The dB Conversion block converts a linearly scaled power or amplitude input to dB or dBm. The reference power is 1 Watt for conversions to dB and 1 mWatt for conversions to dBm. The **Input signal** parameter specifies whether the input is a power signal or a voltage signal, and the **Convert to** parameter controls the scaling of the output. When selected, the **Add eps to input to protect against "log(0) = -inf"** parameter adds a value of `eps` to all power and voltage inputs. When this option is not enabled, zero-valued inputs produce `-inf` at the output.

The size and frame status of the output are the same as the input.

Power Inputs

Select `Power` as the **Input signal** parameter when the input, `u`, is a real, nonnegative, power signal (units of watts). When the **Convert to** parameter is set to `dB`, the block performs the dB conversion

```
y = 10*log10(u)       % Equivalent MATLAB code
```

When the **Convert to** parameter is set to `dBm`, the block performs the dBm conversion

```
y = 10*log10(u) + 30
```

The dBm conversion is equivalent to performing the dB operation *after* converting the input to milliwatts.

Voltage Inputs

Select `Amplitude` as the **Input signal** parameter when the input, `u`, is a real voltage signal (units of volts). The block uses the scale factor specified in ohms by the **Load resistance** parameter, `R`, to convert the voltage input to units of power (watts) before converting to dB or dBm.

When the **Convert to** parameter is set to `dB`, the block performs the dB conversion

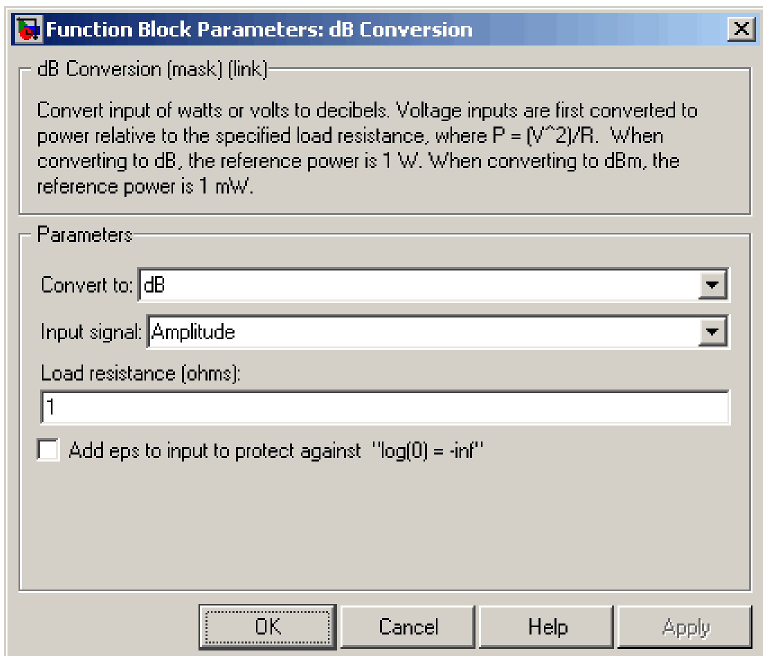
```
y = 10*log10(abs(u)^2/R)
```

When the **Convert to** parameter is set to `dBm`, the block performs the dBm conversion

```
y = 10*log10(abs(u)^2/R) + 30
```

The dBm conversion is equivalent to performing the dB operation *after* converting the `(abs(u)^2/R)` result to milliwatts.

Dialog Box



Convert to
The logarithmic scaling to which the input is converted, `dB` or `dBm`. The reference power is 1 W for conversions to dB and 1 mW for conversions to dBm. Tunable.

Input signal
The type of input signal, `Power` or `Amplitude`.

Load resistance
The scale factor used to convert voltage inputs to units of power. Tunable.

Add eps to input to protect against "log(0) = -inf"
When selected, adds `eps` to all input values (power or voltage). Tunable.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

dB Gain	Signal Processing Blockset
Math Function	Simulink
`log10`	MATLAB

Purpose

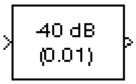
Apply decibel gain

Library

Math Functions / Math Operations

`dspmathops`

Description



The dB Gain block multiplies the input by the decibel values specified in the **Gain** parameter. For an M-by-N input matrix u with elements u_{ij}, the **Gain** parameter can be a real M-by-N matrix with elements g_{ij} to be multiplied element-wise with the input, or a real scalar.

$$y_{ij} = 10u_{ij}^{(g_{ij}/k)}$$

The value of k is 10 for power signals (select `Power` as the **Input signal** parameter) and 20 for voltage signals (select `Amplitude` as the **Input signal** parameter).

The value of the equivalent linear gain

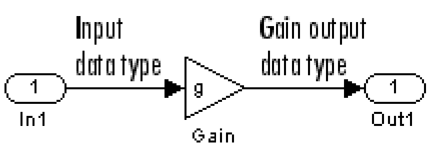
$$g_{ij}^{lin} = 10^{(g_{ij}/k)}$$

is displayed in the block icon below the dB gain value. The size and frame status of the output are the same as the input.

The dB Gain block supports real and complex floating-point and fixed-point data types.

Fixed-Point Data Types

The following diagram shows the data types used within the dB Gain subsystem block for fixed-point signals.

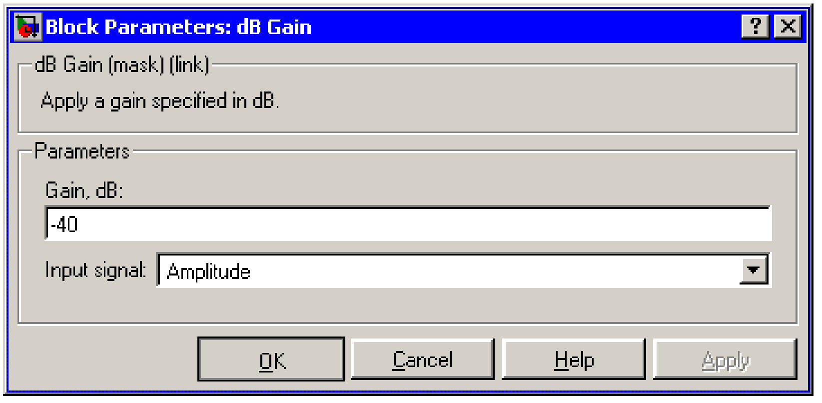


The settings for the fixed-point parameters of the Gain block in the diagram above are as follows:

- Integer rounding mode: `Floor`
- Saturate on integer overflow — unselected
- Parameter data type mode — `Inherit via internal rule`
- Output data type mode — `Inherit via internal rule`

See the Gain reference page for more information.

Dialog Box



Gain
The dB gain to apply to the input, a scalar or a real M-by-N matrix. Tunable.

Input signal
The type of input signal: `Power` or `Amplitude`. Tunable.

Note This block does not support tunability in generated code.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed and unsigned)

- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers

See Also

dB Conversion	Signal Processing Blockset
Math Function	Simulink
`log10`	MATLAB

Purpose

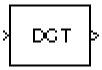
Compute discrete cosine transform (DCT) of input

Library

Transforms

`dspxfrm3`

Description



The DCT block computes the unitary discrete cosine transform (DCT) of each channel in the M-by-N input matrix, u.

```
y = dct(u)     % Equivalent MATLAB code
```

When the input is a sample-based row vector, the DCT block computes the discrete cosine transform across the vector dimension of the input. For all other sample-based N-D arrays, the block computes the DCT across the first dimension of the input.

For both sample-based and frame-based inputs, the block assumes that each input column is a frame containing M consecutive samples from an independent channel. The frame size, M, must be a power of two. To work with other frame sizes, use the Pad block to pad or truncate the frame size to a power-of-two length.

When the input is an M-by-N matrix, the DCT block outputs an M-by-N matrix whose lth column contains the length-M DCT of the corresponding input column.

$$y(k,l) = w(k)\sum_{m=1}^{M} u(m,l)\cos\frac{\pi(2m-1)(k-1)}{2M}, \quad k = 1,\ldots,M$$

where

$$w(k) = \begin{cases} \dfrac{1}{\sqrt{M}}, & k = 1 \\ \sqrt{\dfrac{2}{M}}, & 2 \le k \le M \end{cases}$$

The output is always sample based, and the output port rate and data type (real/complex) are the same as those of the input port.

For convenience, length-M 1-D vector inputs and *sample-based* length-M row vector inputs are processed as single channels (that is, as M-by-1 column vectors), and the output has the same dimension as the input.

The **Sine and cosine computation** parameter determines how the block computes the necessary sine and cosine values. This parameter has two settings, each with its advantages and disadvantages, as described in the following table.

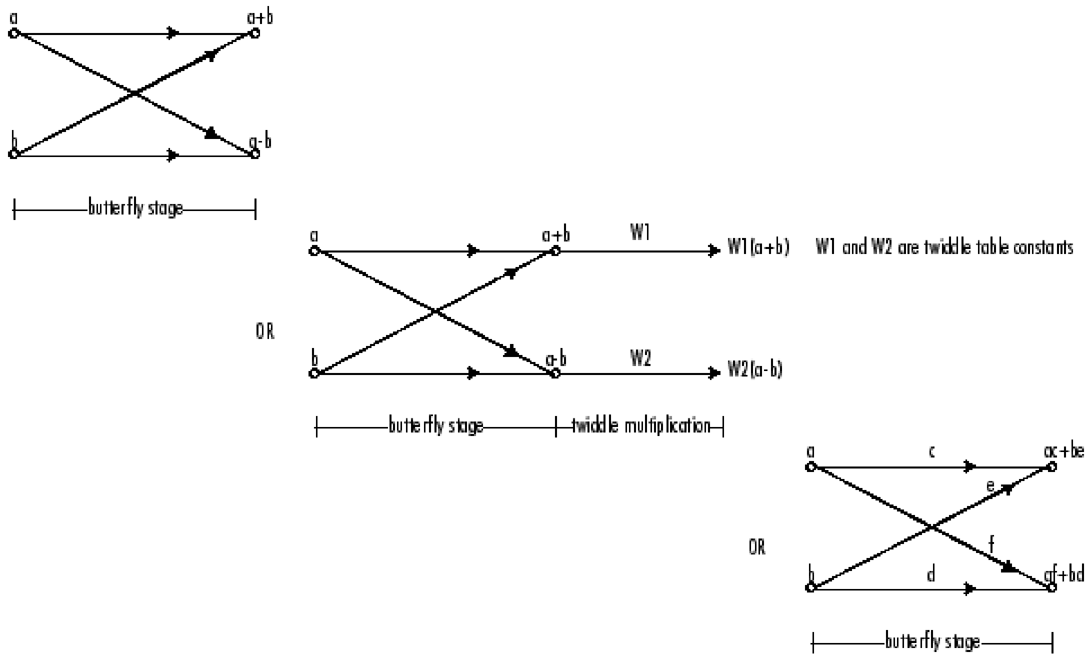
Sine and Cosine Computation Parameter Setting	Sine and Cosine Computation Method	Effect on Block Performance
`Table lookup`	The block computes and stores the trigonometric values before the simulation starts, and retrieves them during the simulation. When you generate code from the block, the processor running the generated code stores the trigonometric values computed by the block in a speed-optimized table, and retrieves the values during code execution.	The block usually runs much more quickly, but requires extra memory for storing the precomputed trigonometric values.
`Trigonometric fcn`	The block computes sine and cosine values during the simulation. When you generate code from the block, the processor running the generated code computes the sine and cosine values while the code runs.	The block usually runs more slowly, but does not need extra data memory. For code generation, the block requires a support library to emulate the trigonometric functions, increasing the size of the generated code.

This block supports Simulink virtual buses.

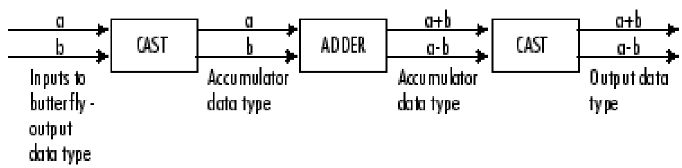
Fixed-Point Data Types

The following diagrams show the data types used within the DCT block for fixed-point signals. You can set the sine table, accumulator, product output, and output data types displayed in the diagrams in the DCT block dialog as discussed in "Dialog Box" on page 2-303.

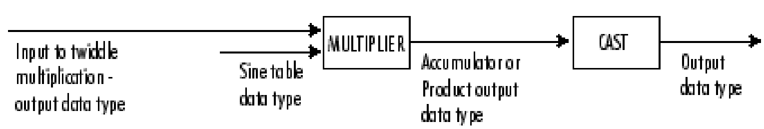
Inputs to the DCT block are first cast to the output data type and stored in the output buffer. Each butterfly stage processes signals in the accumulator data type, with the final output of the butterfly being cast back into the output data type.



Butterfly Stage Data Types



Twiddle Multiplication Data Types



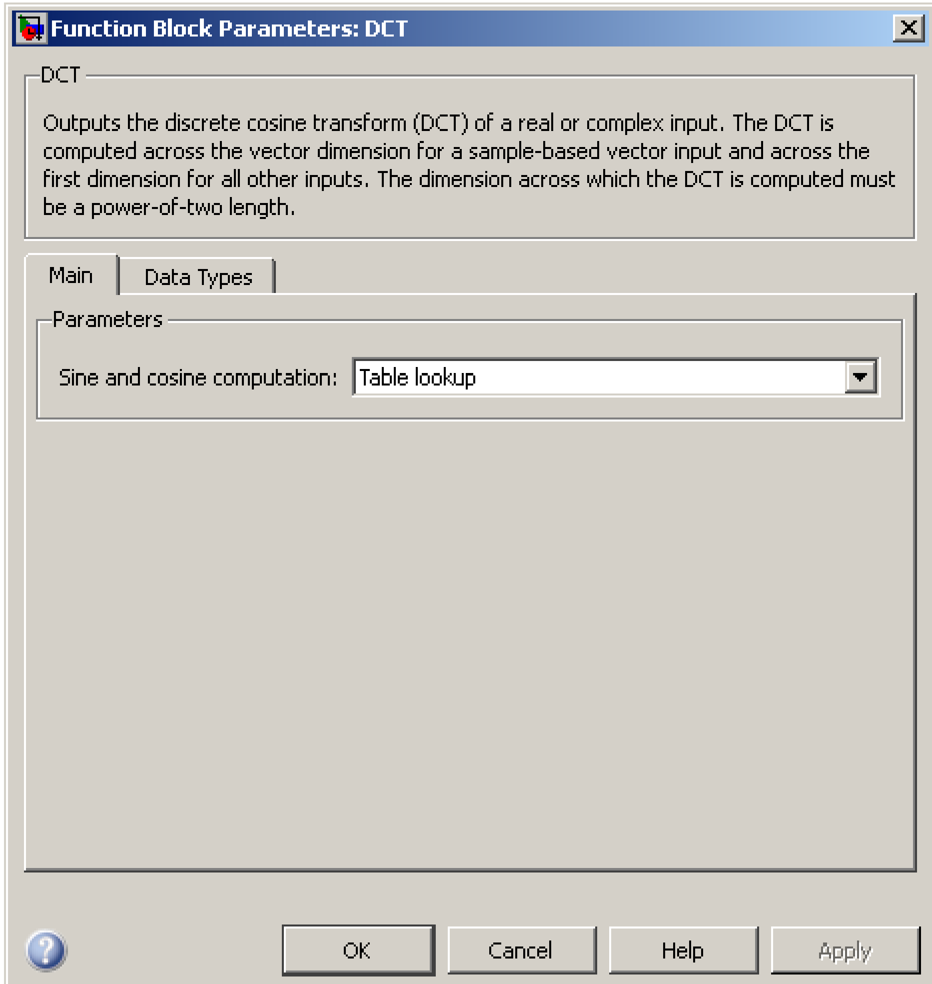
The output of the multiplier is in the product output data type when at least one of the inputs to the multiplier is real. When both of the

inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see "Multiplication Data Types".

Note When the block input is fixed point, all internal data types are signed fixed point.

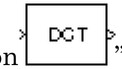
Dialog Box

The **Main** pane of the DCT block dialog appears as follows.



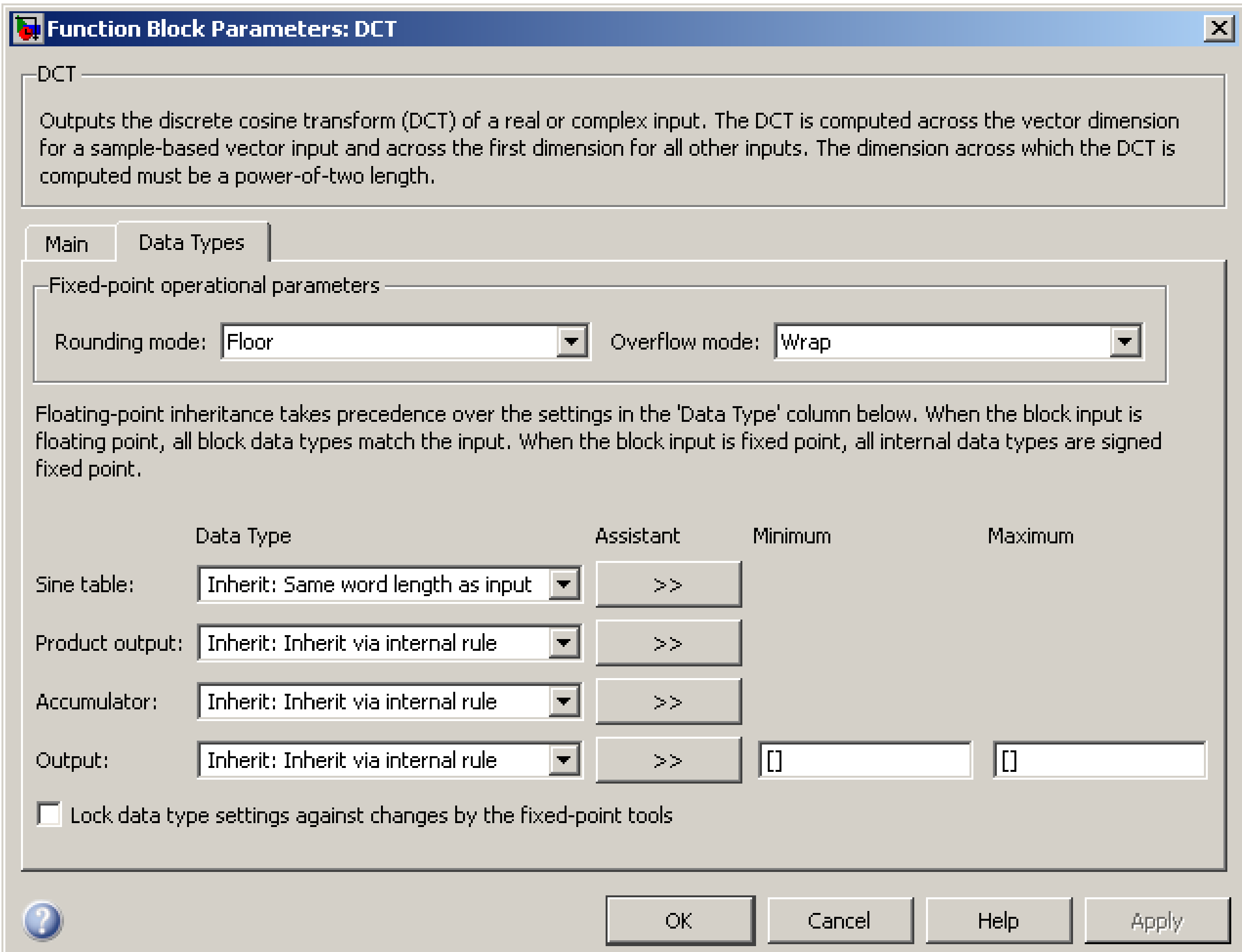
Sine and cosine computation

Sets the block to compute sines and cosines by either looking up sine and cosine values in a speed-optimized table (`Table lookup`), or by making sine and cosine function calls (`Trigonometric fcn`).

See the table in the "Description 

" on page 2-298 section.

The **Data Types** pane of the DCT block dialog appears as follows.



Rounding mode

Select the rounding mode for fixed-point operations. The sine table values do not obey this parameter; they always round to `Nearest`.

Overflow mode

Select the overflow mode for fixed-point operations. The sine table values do not obey this parameter; instead, they are always saturated.

Sine table data type

Choose how you specify the word length of the values of the sine table. The fraction length of the sine table values always equals the word length minus one. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same word length as input`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16)`

The sine table values do not obey the **Rounding mode** and **Overflow mode** parameters; instead, they are always saturated and rounded to `Nearest`.

Product output data type

Specify the product output data type. See “Fixed-Point Data Types” on page 2-300 and “Multiplication Data Types” for illustrations depicting the use of the product output data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

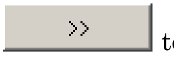
Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Accumulator data type

Specify the accumulator data type. See "Fixed-Point Data Types" on page 2-300 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Output data type

Specify the output data type. See "Fixed-Point Data Types" on page 2-300 for illustrations depicting the use of the output data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`.

  When you select `Inherit:  Inherit via internal rule`, the block calculates the output word length and fraction length automatically. The internal rule first calculates an ideal output word length and fraction length using the following equations:

  $$WL_{ideal\ output} = WL_{input} + floor(\log_2(DCT\ length - 1)) + 1$$

  $$FL_{ideal\ output} = FL_{input}$$

Using these ideal results, the internal rule then selects word lengths and fraction lengths that are appropriate for your hardware. For more information, see "Inherit via Internal Rule".

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

Complex Cepstrum	Signal Processing Blockset
FFT	Signal Processing Blockset
IDCT	Signal Processing Blockset
Real Cepstrum	Signal Processing Blockset
`dct`	Signal Processing Toolbox

Purpose

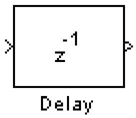
Delay discrete-time input by specified number of samples or frames

Library

Signal Operations

`dspsigops`

Description



The Delay block delays a discrete-time input by the number of samples or frames specified in the **Delay units** and **Delay** parameters. The **Delay** value must be an integer value greater than or equal to zero. When you enter a value of zero for the **Delay** parameter, any initial conditions you might have entered have no effect on the output.

The Delay block allows you to set the initial conditions of the signal that is being delayed. The initial conditions must be numeric.

This block reference contains the following topics:

- "Sample-Based Operation" on page 2-309 — Use the Delay block with a sample-based input signal
- "Frame-Based Operation" on page 2-310 — Use the Delay block with a frame-based input signal

Sample-Based Operation

When the input is a sample-based N-D array, each sample of the input is treated as an independent channel. Thus, the total number of channels is equal to the product of the input dimensions. The dimension of the output is the same as that of the input.

When the input is a sample-based N-D array, the **Delay** parameter can be a scalar integer by which to equally delay all channels or an N-D array of the same dimensions as the input array, containing nonnegative integers that specify the number of sample intervals to delay each channel of the input.

There are four different choices for initial conditions. The initial conditions can be the same or different for each channel. They can also be the same or different within a channel. See the "Sample-Based Operation Examples" on page 2-313 section for more information.

Frame-Based Operation

When the input is a frame-based M-by-N matrix, the block treats each of the N columns as an independent channel, and delays each channel as specified by the **Delay** parameter.

When the input is frame based, the **Delay** parameter can be a scalar integer by which to equally delay all channels or a vector whose length is equal to the number of channels.

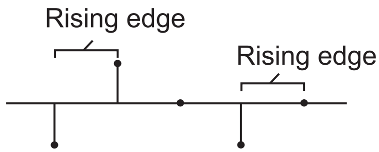
There are four different choices for initial conditions. The initial conditions can be the same or different for each channel. They can also be constant or varying along each channel. See the "Frame-Based Operation Examples" on page 2-318 section for more information.

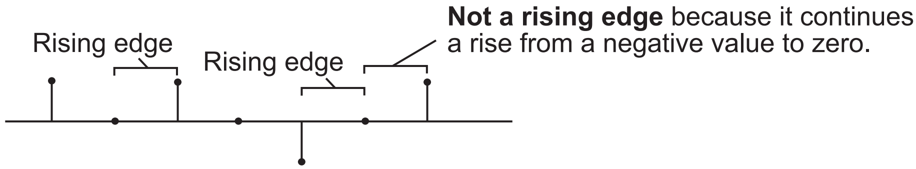
Resetting the Delay

The Delay block resets the delay whenever it detects a reset event at the optional `Rst` port. The reset sample time must be a positive integer multiple of the input sample time.

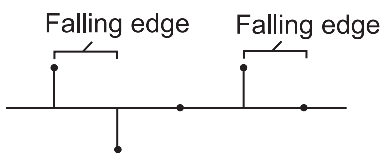
The reset event is specified by the **Reset port** parameter, and can be one of the following:

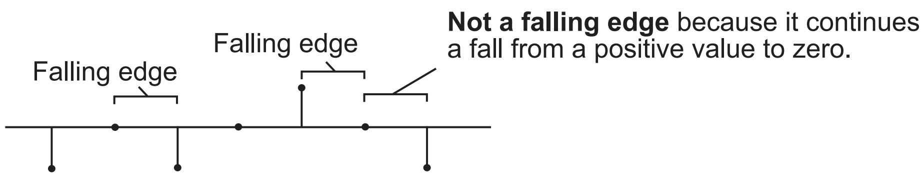
- `None` disables the `Rst` port.
- `Rising edge` triggers a reset operation when the `Rst` input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)





- `Falling edge` triggers a reset operation when the `Rst` input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)



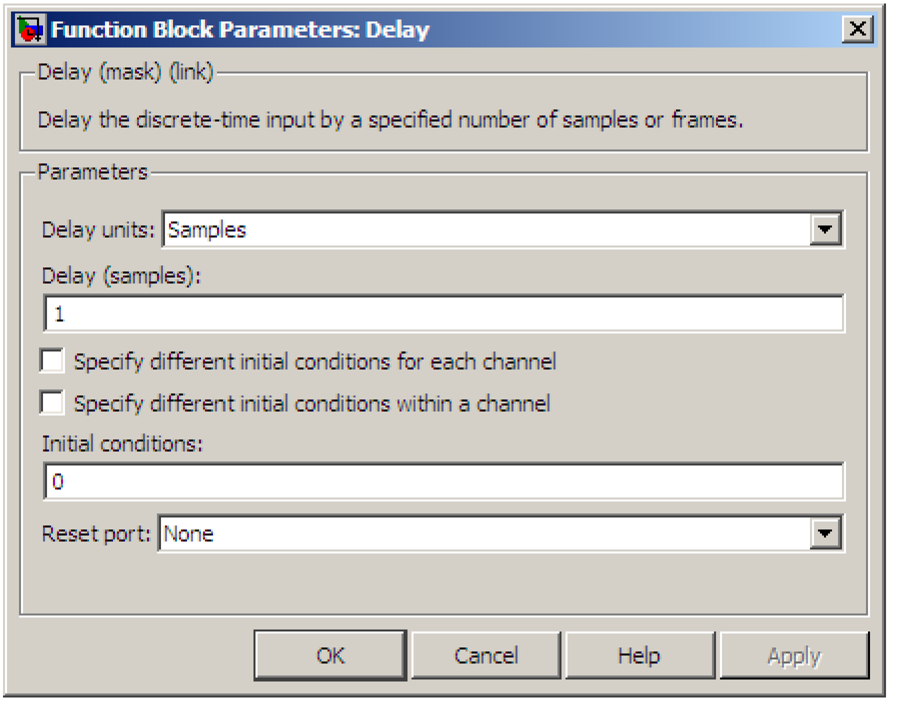


- `Either edge` triggers a reset operation when the `Rst` input is `Rising edge` or `Falling edge` (as described earlier).
- `Non-zero sample` triggers a reset operation at each sample time that the `Rst` input is not zero.

Note When running simulations in the Simulink MultiTasking mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset. For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

This block supports Simulink virtual buses.

Dialog Box



Delay units

Select whether you want to delay your input by a specified number of `Samples` or `Frames`. You can choose to delay your signal by a certain number of samples or frames, regardless of whether your input is sample or frame based.

Delay (samples) or Delay (frames)

See "Sample-Based Operation" on page 2-309 and "Frame-Based Operation" on page 2-310 for a description of what format to use for each configuration of the block dialog.

Specify different initial conditions for each channel

Select this check box when you want the initial conditions to vary across the channels. When you do not select this check box, the initial conditions are the same across the channels.

Specify different initial conditions within a channel

Select this check box when you want the initial conditions to vary within the channels. When you do not select this check box, the initial conditions are the same within the channels.

Initial conditions

Enter a scalar, vector, matrix, or cell array of initial condition values, depending on your choice for the **Specify different initial conditions for each channel** and **Specify different initial conditions within a channel** check boxes. See "Sample-Based Operation" on page 2-309 and "Frame-Based Operation" on page 2-310 for a description of what format to use for each configuration of the block dialog.

Reset port

Determines the reset event that causes the block to reset the delay. For more information, see "Resetting the Delay" on page 2-310.

Examples

Sample-Based Operation Examples

There are four different choices for initial conditions. The initial conditions can be the same or different for each channel. They can also be the same or different along each channel. The next sections describe the behavior of the block for each of these four cases:

- "Case 1 — Use the Same Initial Conditions for Each Channel and Within a Channel" on page 2-314
- "Case 2 — Use Different Initial Conditions for Each Channel and the Same Initial Conditions Within a Channel" on page 2-315
- "Case 3 — Use the Same Initial Conditions for Each Channel and Different Initial Conditions Within a Channel" on page 2-316
- "Case 4 — Use Different Initial Conditions for Each Channel and Within a Channel" on page 2-317

Case 1 — Use the Same Initial Conditions for Each Channel and Within a Channel

Enter a scalar value for the initial conditions. This value is used as the constant initial condition value for each of the channels.

For example, suppose your input is a sample-based matrix.

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}, \ldots$$

You want the initial conditions of your four-channel signal to be identical and zero for the first two samples:

1. For the **Delay (samples)** parameter, type `2`.
2. Clear the **Specify different initial conditions for each channel** and **Specify different initial conditions within a channel** check boxes.
3. For the **Initial conditions** parameter, specify a scalar value of `0`.

   The output of the delay block is

$$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix},\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix},\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix},\begin{bmatrix}2 & 2\\2 & 2\end{bmatrix},\begin{bmatrix}3 & 3\\3 & 3\end{bmatrix},\ldots$$

`0`, the scalar initial condition value, is used for each channel and within the channels. It is the output at sample time zero and sample time one.

Case 2 — Use Different Initial Conditions for Each Channel and the Same Initial Conditions Within a Channel

The initial conditions must be an N-D array for N-D input. The initial conditions must have the same dimensions as the input data. These initial condition values are used as the constant initial condition value for each of the channels.

For example, suppose your input is a sample-based matrix.

$$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix},\begin{bmatrix}2 & 2\\2 & 2\end{bmatrix},\begin{bmatrix}3 & 3\\3 & 3\end{bmatrix},\ldots$$

You want the initial conditions of your four-channel signal to be

$$\begin{bmatrix}7 & 9\\11 & 13\end{bmatrix}$$

for the first two samples:

1. For the **Delay (samples)** parameter, type `2`.
2. Select the **Specify different initial conditions for each channel** check box.
3. Clear the **Specify different initial conditions within a channel** check box.
4. For the **Initial conditions** parameter, type `[7 9; 11 13]`.

   The output of the delay block is

$$\begin{bmatrix} 7 & 9 \\ 11 & 13 \end{bmatrix}, \begin{bmatrix} 7 & 9 \\ 11 & 13 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}, \ldots$$

The initial condition matrix is the output at sample time zero and sample time one. Different initial conditions are used for each channel; the same initial condition value is used within a channel.

Case 3 — Use the Same Initial Conditions for Each Channel and Different Initial Conditions Within a Channel

In this case, for N-D sample-based inputs, the initial conditions parameter must be a vector whose length is equal to the delay value, specified by the **Delay** parameter. The values in this vector are used as the initial condition values along each of the channels to be delayed.

For example, suppose your input is a sample-based matrix.

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}, \ldots$$

You want the initial conditions of your four channel signal to be the same along each of the channels to be delayed:

1. For the **Delay (samples)** parameter, type `2`.
2. Clear the **Specify different initial conditions for each channel** check box.
3. Select the **Specify different initial conditions within a channel** check box.
4. For the **Initial conditions** parameter, type `[10 20]`.

   The output of the delay block is

$$\begin{bmatrix} 10 & 10 \\ 10 & 10 \end{bmatrix}, \begin{bmatrix} 20 & 20 \\ 20 & 20 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}, \ldots$$

The first element of the initial conditions vector is the output, for all channels, at sample time zero. The second element of the initial conditions vector is the output, for all channels, at sample time one. The same initial conditions are used for each channel, but different initial condition values are used within a channel.

Case 4 — Use Different Initial Conditions for Each Channel and Within a Channel

Enter a cell array for your initial condition values. The cell array must be the same size as your input signal. Each cell of the cell array represents the delay values for one channel, and must be a vector of size equal to the delay value. If you have a vector or scalar input and a scalar delay value, you can enter the initial conditions as a matrix.

For example, suppose your input is a sample-based vector.

$$[1\quad 1],\quad [2\quad 2],\quad [3\quad 3],\ldots$$

You want the initial conditions of your two channel signal to be different for each channel and along each channel:

1. For the **Delay (samples)** parameter, type `2`.
2. Select the **Specify different initial conditions for each channel** and **Specify different initial conditions within a channel** check boxes.
3. For the **Initial conditions** parameter, type `[10 20; 30 40]`

   The output of the delay block is

   $$[10\quad 20],\quad [30\quad 40],\quad [1\quad 1],\quad [2\quad 2]\ldots$$

   The first row of the initial conditions vector is the output at sample time zero. The second row of the initial conditions vector is the output at sample time one. Different initial conditions are used for each channel and within the channels.

In addition, suppose your input is a sample-based matrix.

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}, \ldots$$

You want the initial conditions of your two-channel signal to be different for each channel and along each channel:

1 For the **Delay (samples)** parameter, type `2`.

2 Select the **Specify different initial conditions for each channel** and the **Specify different initial conditions within a channel** check boxes.

3 For the **Initial conditions** parameter, type `{[11 15] [12 16]; [13 17] [14 18]}`. The dimensions of the cell array match the dimensions of the input. Also, each element of the cell array represents the initial conditions within one channel.

   The output of the delay block is

$$\begin{bmatrix} 11 & 12 \\ 13 & 14 \end{bmatrix}, \begin{bmatrix} 15 & 16 \\ 17 & 18 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \ldots$$

   Each element of the cell array represents the initial conditions within a channel. The first element, a vector, represents the initial conditions within channel 1. The second element, a vector, represents the initial conditions within channel 2, and so on. Different initial conditions are used for each channel and within the channels.

Frame-Based Operation Examples

There are four different choices for initial conditions. The initial conditions can be the same or different for each channel. They can also be constant or varying along each channel. The next sections describe the behavior of the block for each of these four cases:

- "Case 1 — Use the Same Initial Conditions for Each Channel and Within a Channel" on page 2-319
- "Case 2 — Use Different Initial Conditions for Each Channel and the Same Initial Conditions Within a Channel" on page 2-320
- "Case 3 — Use the Same Initial Conditions for Each Channel and Different Initial Conditions Within a Channel" on page 2-321
- "Case 4 — Use Different Initial Conditions for Each Channel and Within a Channel" on page 2-322

Case 1 — Use the Same Initial Conditions for Each Channel and Within a Channel

Enter a scalar value for the initial conditions. This value is used as the constant initial condition value for each of the channels.

For example, suppose your input is a frame-based matrix.

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 & 4 \\ 5 & 5 & 5 \\ 6 & 6 & 6 \end{bmatrix}, \begin{bmatrix} 7 & 7 & 7 \\ 8 & 8 & 8 \\ 9 & 9 & 9 \end{bmatrix}, \ldots$$

You want the initial conditions of your three-channel signal to be identical and zero for the first frame:

1. For the **Delay (frames)** parameter, type `1`.

2. Clear the **Specify different initial conditions for each channel** and the **Specify different initial conditions within a channel** check boxes.

3. For the **Initial conditions** parameter, specify a scalar value of `0`.

   The output of the delay block is

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 & 4 \\ 5 & 5 & 5 \\ 6 & 6 & 6 \end{bmatrix}, \begin{bmatrix} 7 & 7 & 7 \\ 8 & 8 & 8 \\ 9 & 9 & 9 \end{bmatrix}, \ldots$$

`0`, the scalar initial condition value, is used across the channels and within the channels for the first frame. This frame is the output at sample time zero.

Case 2 — Use Different Initial Conditions for Each Channel and the Same Initial Conditions Within a Channel

The initial conditions must be a vector of length N, where $N \geq 1$. N is also equal to the number of channels in your signal. These initial condition values are used as the constant initial condition value for each of the channels.

For example, suppose your input is a frame-based matrix.

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 & 4 \\ 5 & 5 & 5 \\ 6 & 6 & 6 \end{bmatrix}, \begin{bmatrix} 7 & 7 & 7 \\ 8 & 8 & 8 \\ 9 & 9 & 9 \end{bmatrix}, \ldots$$

You want the initial conditions of your three-channel signal to be `[0 10 20]` for the first frame:

1. For the **Delay (frames)** parameter, type `1`.

2. Select the **Specify different initial conditions for each channel** check box.

3. Clear the **Specify different initial conditions within a channel** check box.

4. For the **Initial conditions** parameter, type `[0 10 20]`.

   The output of the delay block is

$$\begin{bmatrix} 0 & 10 & 20 \\ 0 & 10 & 20 \\ 0 & 10 & 20 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 & 4 \\ 5 & 5 & 5 \\ 6 & 6 & 6 \end{bmatrix}, \begin{bmatrix} 7 & 7 & 7 \\ 8 & 8 & 8 \\ 9 & 9 & 9 \end{bmatrix}, \ldots$$

The initial condition vector expands to create the frame that is output at sample time zero. Different initial conditions are used for each channel, but the same initial condition value is used with a channel.

Case 3 — Use the Same Initial Conditions for Each Channel and Different Initial Conditions Within a Channel

In this case, the **Delay** parameter can be a scalar integer by which to equally delay all channels or a vector whose length is equal to the number of channels. All the values of this vector must be equal.

Enter the initial conditions as a vector. These values are used as the initial condition value along each of the channels to be delayed. The initial condition vector must have length equal to the value of the **Delay (frames)** parameter multiplied by the frame length. For example, if you want to delay your signal by two frames with frame length two and an initial condition value of 3, enter your initial condition vector as `[3 3 3 3]`.

For example, suppose your input is a frame-based matrix.

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 & 4 \\ 5 & 5 & 5 \\ 6 & 6 & 6 \end{bmatrix}, \begin{bmatrix} 7 & 7 & 7 \\ 8 & 8 & 8 \\ 9 & 9 & 9 \end{bmatrix}, \ldots$$

You want the initial conditions of your three-channel signal to be the same along each of the channels to be delayed:

1 For the **Delay (frame)** parameter, type `1`.

2 Clear the **Specify different initial conditions for each channel** check box.

3 Select the **Specify different initial conditions within a channel** check box.

4 For the **Initial conditions** parameter, type `[10 20 30]`.

The output of the delay block is

$$\begin{bmatrix}10 & 10 & 10\\20 & 20 & 20\\30 & 30 & 30\end{bmatrix},\begin{bmatrix}1 & 1 & 1\\2 & 2 & 2\\3 & 3 & 3\end{bmatrix},\begin{bmatrix}4 & 4 & 4\\5 & 5 & 5\\6 & 6 & 6\end{bmatrix},\begin{bmatrix}7 & 7 & 7\\8 & 8 & 8\\9 & 9 & 9\end{bmatrix},\ldots$$

The initial condition vector defines the initial condition values within each of the three channels. The same initial conditions are used for each channel, but different initial condition values are used with a channel.

Case 4 — Use Different Initial Conditions for Each Channel and Within a Channel

Enter a cell array for your initial condition values. Or, when you have a scalar delay value, you can enter the initial conditions as a matrix.

For example, suppose your input is a frame-based matrix.

$$\begin{bmatrix}1 & 1 & 1\\2 & 2 & 2\\3 & 3 & 3\end{bmatrix},\begin{bmatrix}4 & 4 & 4\\5 & 5 & 5\\6 & 6 & 6\end{bmatrix},\begin{bmatrix}7 & 7 & 7\\8 & 8 & 8\\9 & 9 & 9\end{bmatrix},\ldots$$

You want the initial conditions of your three-channel signal to be different for each channel and along each channel.

1 For the **Delay (frames)** parameter, type `1`.

2 Select the **Specify different initial conditions for each channel** and the **Specify different initial conditions within a channel** check boxes.

3 For the **Initial conditions** parameter, type either `[10 20 30; 40 50 60; 70 80 90]` or `{[10 40 70];[20 50 80];[30 60 90]}`. Each cell of the cell array represents the delay along one channel.

Regardless of whether you use a matrix or cell array, the output of the delay block is

$$\begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 & 4 \\ 5 & 5 & 5 \\ 6 & 6 & 6 \end{bmatrix}, \begin{bmatrix} 7 & 7 & 7 \\ 8 & 8 & 8 \\ 9 & 9 & 9 \end{bmatrix} \ldots$$

The initial condition matrix is the output at sample time zero. The elements of the initial condition cell array define the initial condition values within each channel. The first element, a vector, represents the initial conditions within channel 1. The second element, a vector, represents the initial conditions within channel 2, and so on. Different initial conditions are used for each channel and within the channels.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed and unsigned)
- Boolean
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers

See Also

Unit Delay	Simulink
Variable Fractional Delay	Signal Processing Blockset
Variable Integer Delay	Signal Processing Blockset

Delay Line

Purpose

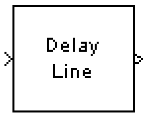
Rebuffer sequence of inputs with one-sample shift

Library

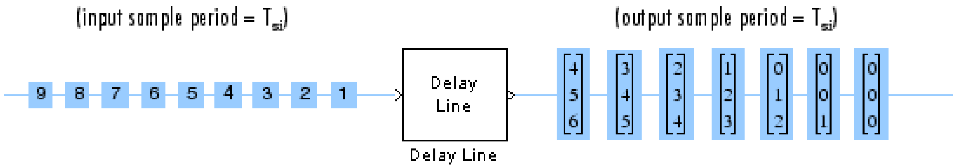
Signal Management / Buffers

`dspbuff3`

Description



The Delay Line block buffers the input samples into a sequence of overlapping or underlapping matrix outputs. In the most typical use (sample-based inputs), each output differs from the preceding output by only one sample, as illustrated below for scalar input.



Note that the first output of the block in the example above is all zeros; this is because the **Initial Conditions** parameter is set to zero. Due to the latency of the Delay Line block, all outputs are delayed by one frame, the entries of which are defined by the **Initial Conditions** parameter.

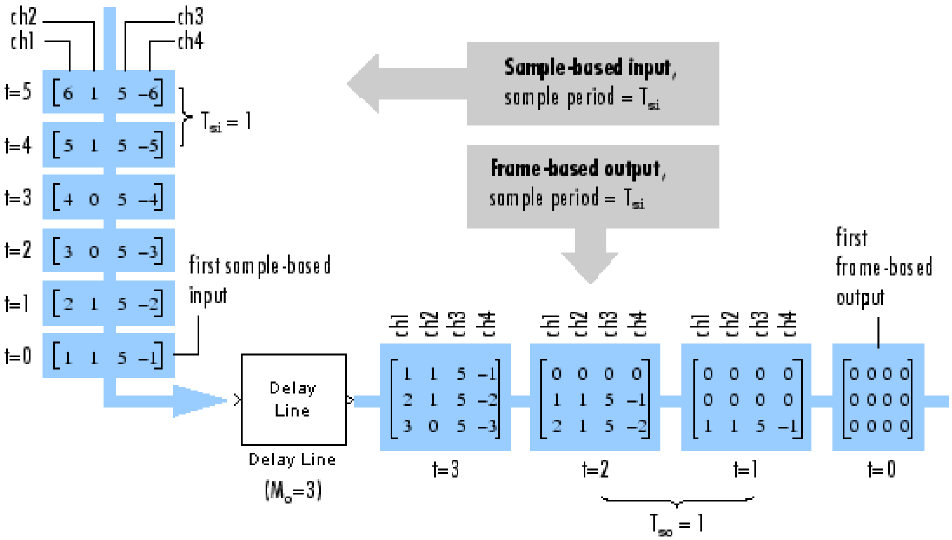
Sample-Based Operation

In sample-based operation, the Delay Line block buffers a sequence of sample-based length-N vector inputs (1-D, row, or column) into a sequence of overlapping frame-based M_o-by-N matrix outputs, where M_o is specified by the **Delay line size** parameter (M_o>1). That is, each input vector becomes a *row* in the frame-based output matrix.

At each sample time the new input vector is added in the last row of the output, so each output overlaps the previous output by M_o-1 samples. Therefore, the output sample period and frame period is the same as the input sample period (T_{so}=T_{si}, and T_{fo}=T_{si}). When M_o=1, the input is simply passed through to the output and retains the same dimension, but becomes frame based. The latency of the block always causes an initial delay in the output; the value of the first output is specified by the **Initial conditions** parameter (see “Initial Conditions” on page 2-327). Sample-based full-dimension matrix inputs are not accepted.

The Delay Line block's sample-based operation is similar to that of a Buffer block with **Buffer size** equal to M_o and **Buffer overlap** equal to M_o-1, except that the Buffer block has a different latency.

In the following model, the block operates on a sample-based input with a **Delay line size** of 3.



The input vectors in the example above do not begin appearing at the output until the second row of the second matrix due to the block's latency (see "Initial Conditions" on page 2-327). The first output matrix (all zeros in this example) reflects the block's **Initial conditions** setting. As for any sample-based input, the output frame rate and output sample rate are both equal to the input sample rate.

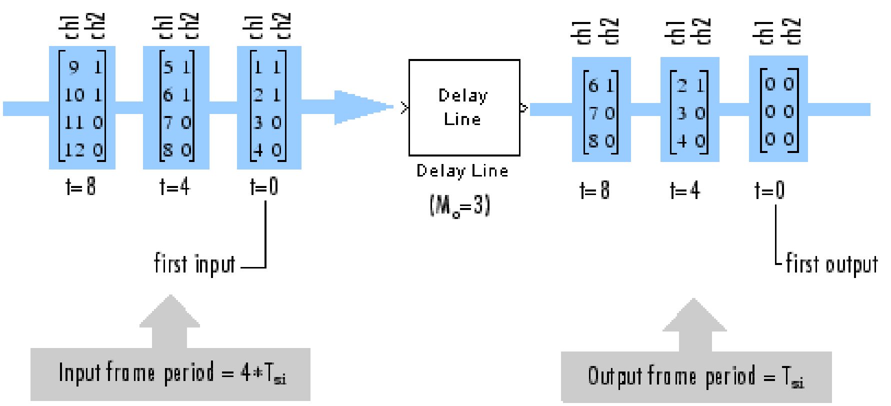
Frame-Based Operation

In frame-based operation, the Delay Line block rebuffers a sequence of frame-based M_i-by-N matrix inputs into a sequence of frame-based M_o-by-N matrix outputs, where M_o is the output frame size specified by the **Delay line size** parameter. Depending on whether M_o is greater than, less than, or equal to the input frame size, M_i, the output frames can be underlapped or overlapped. Each of the N input channels is rebuffered independently.

When $M_o > M_i$, the output frame overlap is the difference between the output and input frame size, M_o-M_i. When $M_o < M_i$, the output is underlapped; the Delay Line block discards the first M_i-M_o samples of each input frame so that only the last M_o samples are buffered into the corresponding output frame. When $M_o = M_i$, the output data is identical to the input data, but is delayed by the latency of the block. Due to the block's latency, the outputs are always delayed by one frame, the entries of which are specified by the **Initial conditions** (see "Initial Conditions" on page 2-327).

The output frame period is equal to the input frame period (T_{fo}=T_{fi}). The output sample period, T_{so}, is therefore equal to T_{fi}/M_o, or equivalently, $T_{si}(M_i/M_o)$

In the following model, the block rebuffers a two-channel frame-based input with a **Delay line size** of 3.

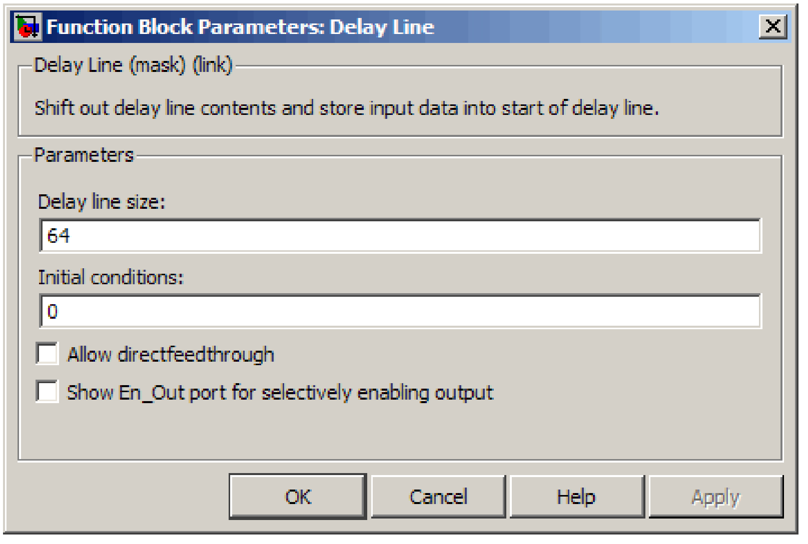


The first output frame in the example is a product of the latency of the Delay Line block; it is all zeros because the **Initial conditions** is set to be zero. Since the input frame size, 4, is larger than the output frame size, 3, only the last three samples in each input frame are propagated to the corresponding output frame. The frame periods of the input and output are the same, and the output sample period is $T_{si}(M_i/M_o)$, or 4/3 the input sample period.

Initial Conditions

The Delay Line block's buffer is initialized to the value specified by the **Initial condition** parameter. The block outputs this buffer at the first simulation step (t=0). When the block's output is a vector, the **Initial condition** can be a vector of the same size, or a scalar value to be repeated across all elements of the initial output. When the block's output is a matrix, the **Initial condition** can be a matrix of the same size, a vector (of length equal to the number of matrix rows) to be repeated across all columns of the initial output, or a scalar to be repeated across all elements of the initial output.

Dialog Box



Delay line size
The number of rows in output matrix, M_o.

Initial conditions
The value of the block's initial output, a scalar, vector, or matrix.

Allow direct feedthrough

When you select this check box, the input data is not delayed by an extra frame before it is available at the output buffer. Instead, the input data is available immediately at the output port of the block.

Show En_Out port for selectively enabling output

When you select this check box, the En_Out port appears on the block icon. This block uses a circular buffer internally even though the output is linear. This means that for valid output, data from the circular buffer has to be linearized. The En_Out port determines whether or not a valid output needs to be computed based on the value of its `Boolean` input. If the input value to the En_Out port is `1`, the block output is linearized, and thus is valid. Otherwise, the output is not linearized, and is invalid. This allows the block to be more efficient when the tapped Delay Line's output is not required at each sample time.

Note that when the input value to the En_Out port is `0`, the block can give different results depending on the state of the model. The results can appear to match valid results or can be invalid, and they cannot be predicted. You should ignore the block output in all cases when the input to the En_Out port is `0`.

Hold previous value when the output is disabled

This parameter only appears and applies when the **Show En_Out port for selectively enabling output** parameter is selected. Use this parameter to specify the block output at those time steps when the internal state buffer is not being linearized to output valid data.

When you do not select this check box, the block memory is free to be used by other parts of the model, and the signal on the output port is invalid. When you select this check box, the most recent valid value is held on the output port, and slightly more memory is used by the block.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Buffer	Signal Processing Blockset
Triggered Delay Line (Obsolete)	Signal Processing Blockset

Detrend

Purpose

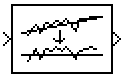
Remove linear trend from vectors

Library

Statistics

`dspstat3`

Description



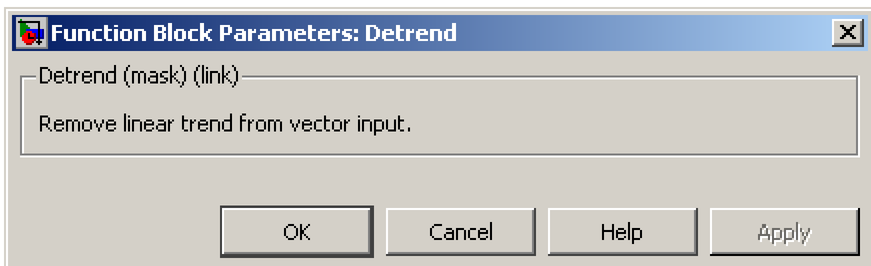
The Detrend block removes a linear trend from the length-M input vector, u, by subtracting the straight line that best fits the data in the least squares sense.

The least squares line, $\hat{u} = ax + b$, is the line with parameters a and b that minimizes the quantity

$$\sum_{i=1}^{M} (u_i - u_i)^2$$

for M evenly-spaced values of x, where u_i is the ith element in the input vector. The output, $y = u\text{-}\hat{u}$, is an M-by-1 column vector (regardless of the input vector dimension) with the same frame status as the input.

Dialog Box



Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Cumulative Sum	Signal Processing Blockset
Difference	Signal Processing Blockset
Least Squares Polynomial Fit	Signal Processing Blockset
Unwrap	Signal Processing Blockset
`detrend`	MATLAB

Difference

Purpose

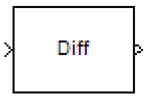
Compute element-to-element difference along specified dimension of input

Library

Math Functions / Math Operations

`dspmathops`

Description



The Difference block computes the difference between adjacent elements in rows, columns, or a specified dimension of the input array `u`. This block accepts real and complex fixed-point and floating-point inputs, except for complex unsigned fixed-point inputs.

Columnwise Differencing

When the **Difference along** parameter is set to `Columns`, the block computes differences between adjacent elements along each column.

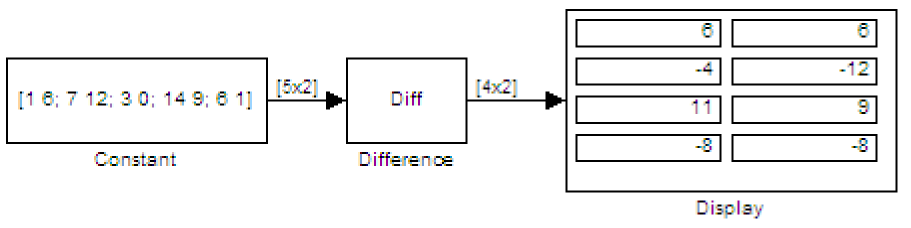
```
y = diff(u)      % Equivalent MATLAB code
```

For sample-based inputs, the output is a sample-based (M-1)-by-N matrix whose jth column has elements

$$y_{i,j} = u_{i+1,j} - u_{i,j} \quad 1 \le i \le (M-1)$$

For convenience, length-M 1-D vector inputs are treated as M-by-1 column vectors for columnwise differencing, and the output is 1-D.

For example, the following figure shows the block output for sample-based inputs:



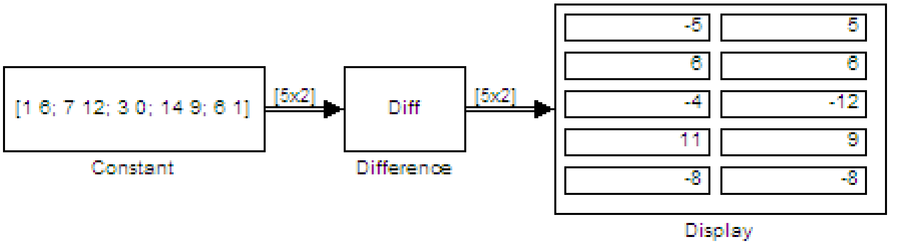
For frame-based inputs, the output is a frame-based *M*-by-*N* matrix whose *j*th column has elements

$$y_{i,j} = u_{i+1,j} - u_{i,j} \quad 2 \le i \le (M-1)$$

The first element of the output for each column is the first input element minus the last input element of the previous frame. For the first frame, zero is subtracted from the first input element.

$$y_{1,j}(t) = u_{1,j}(t) - u_{M,j}(t - T_f)$$

For example, the following figure shows the second frame of the block output for a frame-based input:



Rowwise Differencing

When the **Difference along** parameter is set to `Rows`, the block computes differences between adjacent elements along each row. The result is the same regardless of the frame status of the input signal.

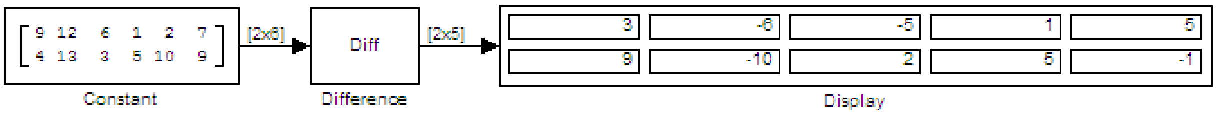
```
y = diff(u,[],2)      % Equivalent MATLAB code
```

The output is an *M*-by-(*N*-1) matrix whose *i*th row has elements

$$y_{i,j} = u_{i,j+1} - u_{i,j} \quad 1 \le j \le (N-1)$$

The frame status of the output is the same as the input. For convenience, length-*N* 1-D vector inputs are treated as 1-by-*N* row vectors for rowwise differencing, and the output is 1-D.

For example, the following figure shows the block output for sample-based inputs. The output is the same for frame-based inputs:



Differencing Along Arbitrary Dimensions

When the **Difference along** parameter is set to `Specified dimension`, the behavior of the block is an extension of the rowwise differencing described earlier. The block computes differences between adjacent elements along the dimension specified by the **Dimension** parameter.

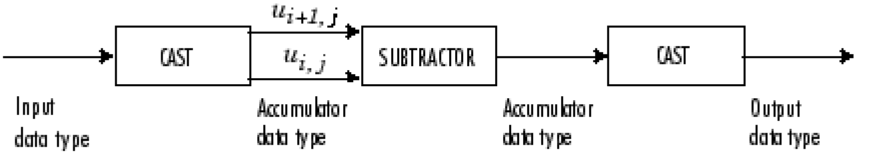
```
y = diff(u,[],d) % Equivalent MATLAB code where d is the dimension
```

The output is an array whose length in the specified dimension is one less than that of the input, and whose lengths in other dimensions are unchanged. For example, consider an M-by-N-by-P-by-R input array with elements *u(i,j,k,l)* and assume that **Dimension** is `3`. The output of the block is an M-by-N-by-(P–1)-by-R array with elements

$$y_{i,j,k,l} = u_{i,j,k+1,l} - u_{i,j,k,l} \quad 1 \le k \le (P-1)$$

Fixed-Point Data Types

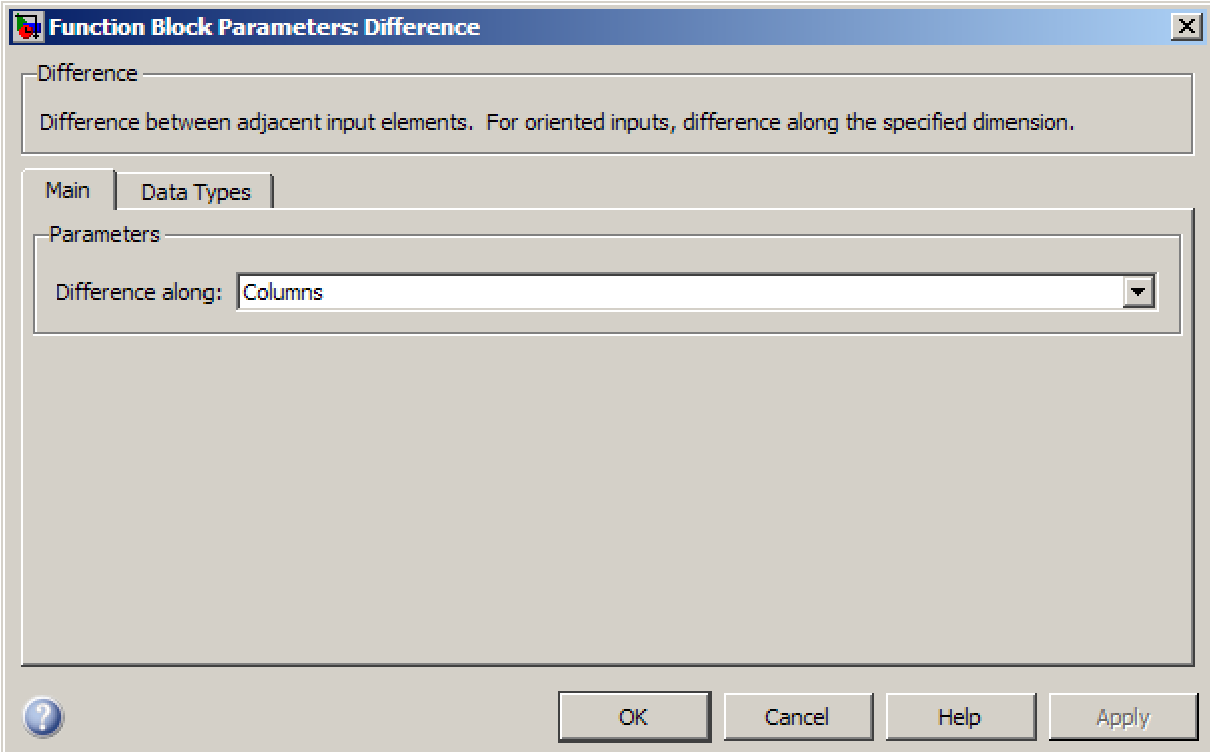
The following diagram shows the data types used within the Difference block for fixed-point signals.



You can set the accumulator and output data types in the block dialog as discussed in “Dialog Box” on page 2-335 .

Dialog Box

The **Main** pane of the Difference block appears as follows.



Difference along

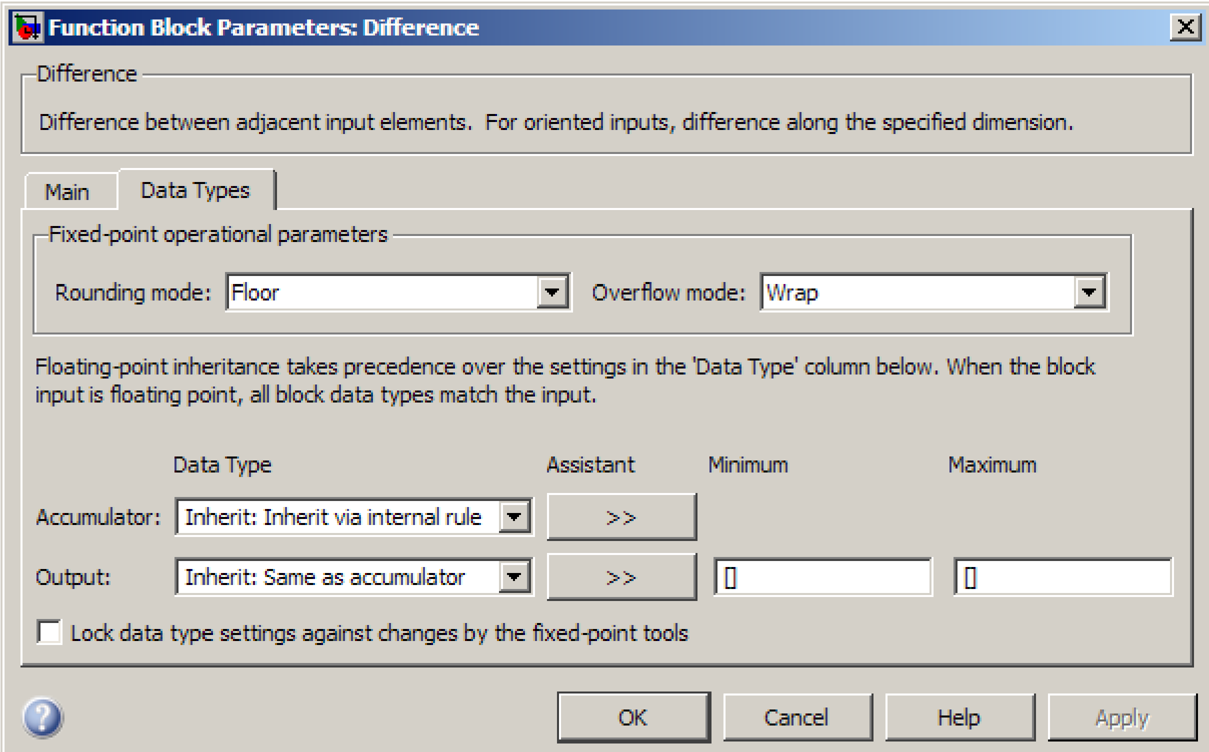
Specify whether the block performs columnwise differencing, rowwise differencing, or differencing along a specified dimension.

Dimension

Specify the one-based dimension along which to compute element-to-element differences.

This parameter is only visible when you select `Specified dimension` for the **Difference along** parameter.

The **Data Types** pane of the Difference block appears as follows.



Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Rounding mode
Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode for fixed-point operations.

Accumulator data type
Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-334 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Output data type
Specify the output data type. See “Fixed-Point Data Types” on page 2-334 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as accumulator`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
Output	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Cumulative Sum	Signal Processing Blockset
`diff`	MATLAB

Purpose Design differentiator filter

Library Filtering / Filter Designs

`dspfdesign`

Description

This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. Without a Filter Design Toolbox license, you can run models that contain this block, and can edit some, but not all, block parameters. To enable the full filter design functionality of this block, you must have a Filter Design Toolbox license.

Dialog Box

See "Differentiator Filter Design Dialog Box — Main Pane" in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Purpose

Filter each channel of input over time using static or time-varying digital filter implementations

Library

Filtering / Filter Implementations

`dsparch4`

Description

Note Use this block to efficiently implement a floating-point or fixed-point filter for which you know the coefficients, or that is already defined in a Signal Processing Toolbox `dfilt` object or a Filter Design Toolbox `dfilt` object. The following Signal Processing Blockset blocks also implement digital filters, but serve slightly different purposes:

- Digital Filter Design — Use to design, analyze, and then efficiently implement floating-point filters. This block provides the same filter implementation as the Digital Filter block for floating-point signals.
- Filter Realization Wizard — Use to implement floating-point or fixed-point filters built from Sum, Gain, and Unit Delay blocks. You can either design the filter using block filter design and analysis parameters, or import the coefficients of a filter that you designed elsewhere.

The Digital Filter block independently filters each channel of the input signal with a specified digital IIR or FIR filter. The block can implement *static filters* with fixed coefficients, as well as *time-varying filters* with coefficients that change over time. You can tune the coefficients of a static filter during simulation.

This block filters each channel of the input signal independently over time. The output frame status and dimensions are always the same as those of the input signal that is filtered. When inputs are frame based, the block treats each column as an independent channel; the block filters each column. When inputs are sample based, the block treats each element of the input as an individual channel.

The outputs of this block numerically match the outputs of the Digital Filter Design block and of the `dfilt` function in Signal Processing Toolbox software or Filter Design Toolbox software.

Note The Digital Filter block has direct feedthrough, so if you connect the output of this block back to its input you get an algebraic loop. For more information on direct feedthrough and algebraic loops, see "Algebraic Loops" in the Simulink documentation.

Sections of This Reference Page

Coefficient Source

The Digital Filter block can operate in three different modes. Select the mode in the **Coefficient source** group box. If you select

- **Dialog parameters**, you enter information about the filter such as structure and coefficients in the block mask.
- **Input port(s)**, you enter the filter structure in the block mask, and the filter coefficients come in through one or more block ports. This mode is useful for specifying time-varying filters.

- **Discrete-time filter object (DFILT)**, you specify the filter using a `dfilt` object from the Signal Processing Toolbox product or the Filter Design Toolbox product.

Supported Filter Structures

When you select **Discrete-time filter object (DFILT)**, the following `dfilt` structures are supported:

- `dfilt.df1`
- `dfilt.df1t`
- `dfilt.df2`
- `dfilt.df2t`
- `dfilt.df1sos`
- `dfilt.df1tsos`
- `dfilt.df2sos`
- `dfilt.df2tsos`
- `dfilt.dffir`
- `dfilt.dffirt`
- `dfilt.dfsymfir`
- `dfilt.dfasymfir`
- `dfilt.latticear`
- `dfilt.latticemamin`

When you select **Dialog parameters** or **Input port(s)**, the list of filter structures offered in the **Filter structure** parameter depends on whether you set the **Transfer function type** to `IIR (poles & zeros)`, `IIR (all poles)`, or `FIR (all zeros)`, as summarized in the following table.

Note Each structure listed in the table below supports both fixed-point and floating-point signals.

The table also shows the vector or matrix of filter coefficients you must provide for each filter structure. For more information on how to specify filter coefficients for various filter structures, see "Specifying Static Filters" and "Specifying Time-Varying Filters".

Filter Structures and Filter Coefficients

Transfer Function Type	Supported Filter Structures	Filter Coefficient Specification

IIR (poles & zeros)	`Direct form I`
	`Direct form I transposed`
	`Direct form II`
	`Direct form II transposed`
		• Numerator coefficients vector [b0, b1, b2, ..., bn]
		• Denominator coefficients vector [a0, a1, a2, ..., am]
		
		See Special Consideration for the Leading Denominator Coefficient.

	`Biquadratic direct form I (SOS)`
	`Biquadratic direct form I transposed (SOS)`
	`Biquadratic direct form II (SOS)`
	`Biquadratic direct form II transposed (SOS)`
		• *M*-by-6 second-order section (SOS) matrix.
		• Scale values
		
		See "Specifying the SOS Matrix (Biquadratic Filter Coefficients)".

Filter Structures and Filter Coefficients (Continued)

Transfer Function Type	Supported Filter Structures	Filter Coefficient Specification
IIR (all poles)	`Direct form` `Direct form transposed`	Denominator coefficients vector [a0, a1, a2, ..., am] See Special Consideration for the Leading Denominator Coefficient.
	`Lattice AR`	Reflection coefficients vector [k1, k2, ..., kn]

FIR (all zeros)	`Direct form`
	`Direct form symmetric`
	`Direct form antisymmetric`
	`Direct form transposed`
		Numerator coefficients vector [b0, b1, b2, ..., bn]

	`Lattice MA`	Reflection coefficients vector [k1, k2, ..., kn]

Special Considerations for the Leading Denominator Coefficient

In some cases, the Digital Filter block requires the leading denominator coefficient (a_0) to be 1. This requirement applies under the following conditions:

- The Digital Filter block is operating in a fixed-point mode. The block operates in a fixed-point mode when at least one of the following statements is true:
  - The input to the Digital Filter block has a fixed-point or integer data type.
  - The **Fixed-point instrumentation mode** parameter in the Fixed-Point Tool has a setting of `Minimums, maximums and overflows`.

- The **Coefficient source** has a setting of `Dialog` or `Input port(s)`.

> **Note** If you are working in one of the fixed-point situations described in the previous bullet, and the **Coefficient source** is set to `Input port(s)`, you must select the **First denominator coefficient = 1, remove a0 term in the structure** check box.

- The **Transfer function type** and **Filter structure** parameters are set to one of the combinations described in the following table.

Transfer function type	Filter structure
IIR (poles & zeros)	`Direct form I`
	`Direct form I transposed`
	`Direct form II`
	`Direct form II transposed`
IIR (all poles)	`Direct form`
	`Direct form transposed`

The Digital Filter block produces an error if you use it in one of the these configurations and your leading denominator coefficient (a_0) does not equal 1. To resolve the error, set your leading denominator coefficient to 1 by scaling all numerator and denominator coefficients by a factor of a_0.

Specifying Initial Conditions

In **Dialog parameters** and **Input port(s)** modes, the block initializes the internal filter states to zero by default, which is equivalent to assuming past inputs and outputs are zero. You can optionally use the **Initial conditions** parameter to specify nonzero initial conditions for the filter delays.

To determine the number of initial condition values you must specify, and how to specify them, see the following table on Valid Initial Conditions and Number of Delay Elements (Filter States) on page 2-350. The **Initial conditions** parameter can take one of four forms as described in the following table.

Valid Initial Conditions

Initial Condition	Examples	Description
Scalar	5 Each delay element for each channel is set to 5.	The block initializes all delay elements in the filter to the scalar value.
Vector (for applying the same delay elements to each channel)	For a filter with two delay elements: $[d_1\ d_2]$ The delay elements for all channels are d1 and d2.	Each vector element specifies a unique initial condition for a corresponding delay element. The block applies the same vector of initial conditions to each channel of the input signal. The vector length must equal the number of delay elements in the filter (specified in the table Number of Delay Elements (Filter States) on page 2-350).
Vector or matrix (for applying different delay elements to each channel)	For a 3-channel input signal and a filter with two delay elements: $[d_1\ d_2\ D_1\ D_2\ d_1\ d_2]$ or $\begin{bmatrix} d_1 & D_1 & d_1 \\ d_2 & D_2 & d_2 \end{bmatrix}$ • The delay elements for channel 1 are d_1 and d_2. • The delay elements for channel 2 are D_1 and D_2.	Each vector or matrix element specifies a unique initial condition for a corresponding delay element in a corresponding channel: • The vector length must be equal to the product of the number of input channels and the number of delay elements in the filter (specified in the table Number of Delay Elements (Filter States) on page 2-350). • The matrix must have the same number of rows as the number of

Valid Initial Conditions (Continued)

Initial Condition	Examples	Description
	• The delay elements for channel 3 are $d_1$and d_2.	delay elements in the filter (specified in the table Number of Delay Elements (Filter States) on page 2-350), and must have one column for each channel of the input signal.
Empty matrix	`[ ]` Each delay element for each channel is set to `0`.	The empty matrix, `[]`, is equivalent to setting the **Initial conditions** parameter to the scalar value `0`.

The number of delay elements (filter states) per input channel depends on the filter structure, as indicated in the following table.

Number of Delay Elements (Filter States)

Filter Structure	Number of Delay Elements per Channel

`Direct form`
`Direct form transposed`
`Direct form symmetric`
`Direct form antisymmetric`
	`#_of_filter_coeffs-1`

`Direct form I` `Direct form I transposed`	• `#_of_zeros-1` • `#_of_poles-1`
`Direct form II` `Direct form II transposed`	`max(#_of_zeros, #_of_poles)-1`

`Biquadratic direct form I (SOS)`
`Biquadratic direct form I transposed (SOS)`
`Biquadratic direct form II (SOS)`
`Biquadratic direct form II transposed (SOS)`
	`2 * #_of_filter_sections`

`Lattice AR` `Lattice MA`	`#_of_reflection_coeffs`

State Logging

Simulink enables you to log the states in your model to the MATLAB workspace. The following table indicates which filter structures of the Digital Filter block support the Simulink state logging feature. See "States" in the Simulink User's Guide documentation for more information.

Transfer Function Type	Filter Structure	State Logging Supported
IIR (poles & zeros)	`Direct form I`	No
	`Direct form I transposed`	Yes
	`Direct form II`	No
	`Direct form II transposed`	Yes
	`Biquadratic direct form I (SOS)`	Yes
	`Biquadratic direct form I transposed (SOS)`	Yes
	`Biquadratic direct form II (SOS)`	Yes
	`Biquadratic direct form II transposed (SOS)`	Yes
IIR (all poles)	`Direct form`	No
	`Direct form transposed`	Yes
	`Lattice AR`	Yes
FIR (all zeros)	`Direct form`	No
	`Direct form symmetric`	No
	`Direct form antisymmetric`	No
	`Direct form transposed`	Yes
	`Lattice MA`	Yes

Fixed-Point Data Types

All structures supported by the Digital Filter block support fixed-point data types. You can specify intermediate fixed-point data types for quantities such as the coefficients, accumulator, and product output for each filter structure. See "Filter Structure Diagrams" on page 2-368 for diagrams depicting the use of these intermediate fixed-point data types in each filter structure.

Dialog Box

Coefficient Source

The Digital Filter block can operate in three different modes. Select the mode in the **Coefficient source** group box. If you select

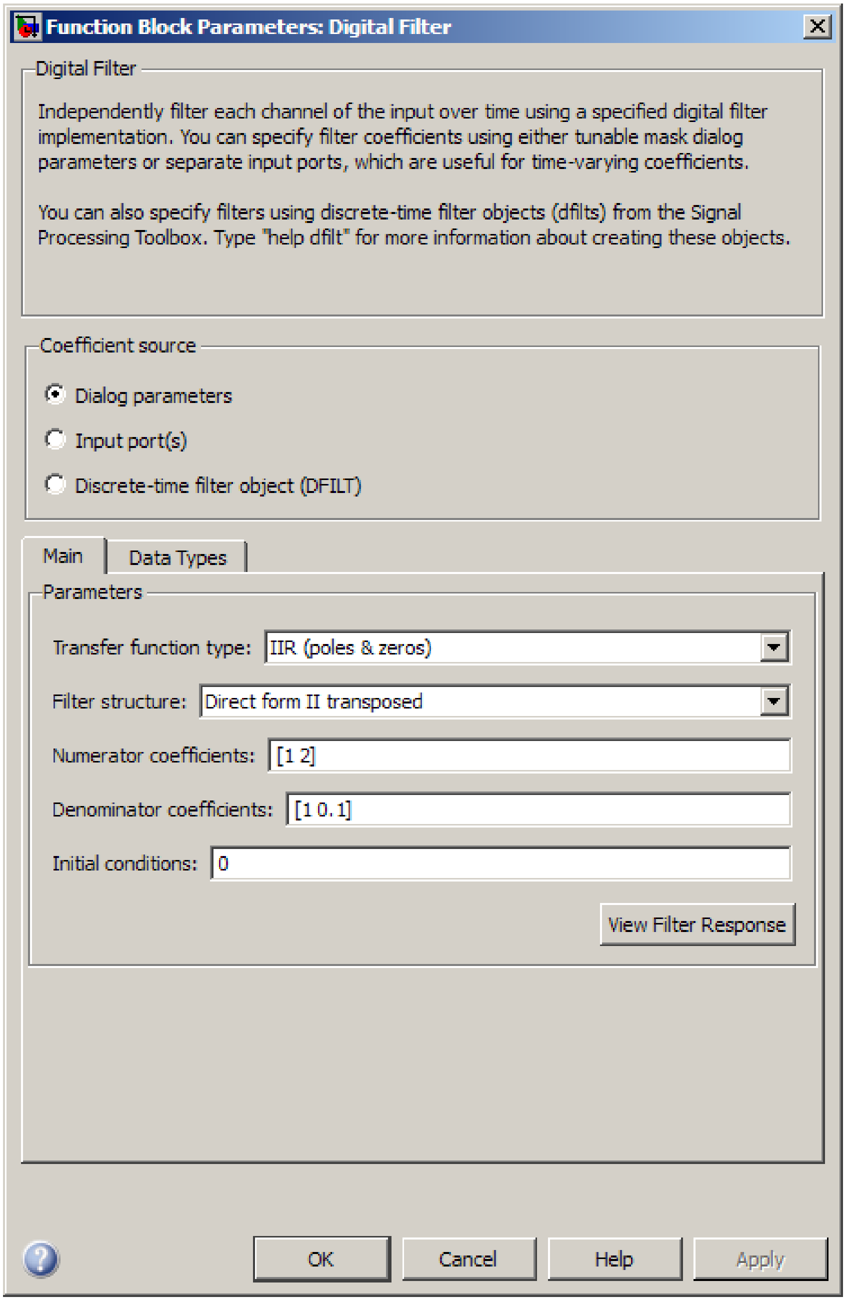
- **Dialog parameters**, you enter information about the filter such as structure and coefficients in the block mask.
- **Input port(s)**, you enter the filter structure in the block mask, and the filter coefficients come in through one or more block ports. This mode is useful for specifying time-varying filters.
- **Discrete-time filter object (DFILT)**, you specify the filter using a `dfilt` object from the Signal Processing Toolbox product or the Filter Design Toolbox product.

Different items appear on the Digital Filter block dialog depending on whether you select **Dialog parameters**, **Input port(s)**, or **Discrete-time filter object (DFILT)** in the **Coefficient source** group box. See the following sections for details:

- "Specify Filter Characteristics in Dialog and/or Through Input Ports" on page 2-352
- "Specify Discrete-Time Filter Object" on page 2-364

Specify Filter Characteristics in Dialog and/or Through Input Ports

The **Main** pane of the Digital Filter block dialog appears as follows when **Dialog parameters** is specified in the **Coefficient source** group box. The parameters below can appear when **Dialog parameters** or **Input port(s)** is selected, as noted.


Function Block Parameters: Digital Filter
Digital Filter
Independently filter each channel of the input over time using a specified digital filter implementation. You can specify filter coefficients using either tunable mask dialog parameters or separate input ports, which are useful for time-varying coefficients.
You can also specify filters using discrete-time filter objects (dfilts) from the Signal Processing Toolbox. Type "help dfilt" for more information about creating these objects.
Coefficient source
Dialog parameters
Input port(s)
Discrete-time filter object (DFILT)
Main
Data Types
Parameters
Transfer function type:
IIR (poles & zeros)
Filter structure:
Direct form II transposed
Numerator coefficients:
[1 2]
Denominator coefficients:
[1 0.1]
Initial conditions:
0
View Filter Response
OK
Cancel
Help
Apply

Transfer function type

Select the type of transfer function of the filter; `IIR (poles & zeros)`, `IIR (all poles)`, or `FIR (all zeros)`. See "Supported Filter Structures" on page 2-344 for more information.

Filter structure

Select the filter structure. The selection of available structures varies depending the setting of the **Transfer function type** parameter. See "Supported Filter Structures" on page 2-344 for more information.

Numerator coefficients

Specify the vector of numerator coefficients of the filter's transfer function.

This parameter is only visible when **Dialog parameters** is selected *and* when the selected filter structure lends itself to specification with numerator coefficients. Tunable.

Denominator coefficients

Specify the vector of denominator coefficients of the filter's transfer function.

In some cases, the leading denominator coefficient (a0) must be 1. See Special Consideration for the Leading Denominator Coefficient for more information.

This parameter is only visible when **Dialog parameters** is selected *and* when the selected filter structure lends itself to specification with denominator coefficients. Tunable.

Reflection coefficients

Specify the vector of reflection coefficients of the filter's transfer function.

This parameter is only visible when **Dialog parameters** is selected *and* when the selected filter structure lends itself to specification with reflection coefficients. Tunable.

SOS matrix (Mx6)

Specify an *M*-by-6 *SOS matrix* containing coefficients of a second-order section (SOS) filter, where *M* is the number of sections. You can use the `ss2sos` and `tf2sos` functions from Signal Processing Toolbox software to check whether your SOS matrix is valid. For more on the requirements of the SOS matrix, see "Specifying the SOS Matrix (Biquadratic Filter Coefficients)".

This parameter is only visible when **Dialog parameters** is selected *and* when the selected filter structure is biquadratic. Tunable.

Scale values

Specify the scale values to be applied before and after each section of a biquadratic filter.

- If you specify a scalar, that value is applied before the first filter section. The rest of the scale values are set to 1.
- You can also specify a vector with $M + 1$ elements, assigning a different value to each scale. See "Filter Structure Diagrams" on page 2-368 for diagrams depicting the use of scale values in biquadratic filter structures.

This parameter is only visible when **Dialog parameters** is selected *and* when the selected filter structure is biquadratic. Tunable.

First denominator coefficient = 1, remove a0 term in the structure

Select this parameter to reduce the number of computations the block must make to produce the output by omitting the $1 / a_0$ term in the filter structure. The block output is invalid if you select this parameter when the first denominator filter coefficient is *not* always 1 for your time-varying filter.

This parameter is only enabled when the **Input port(s)** is selected *and* when the selected filter structure lends itself to this specification. See “Removing the a0 Term in the Filter Structure” for a diagram and details.

Coefficient update rate

Specify how often the block updates time-varying filters; once per sample or once per frame. This parameter only affects the output when the input signal is frame based.

This parameter is only visible when the **Input port(s)** is selected *and* when the selected filter structure lends itself to this specification. For more information, see “Specifying Time-Varying Filters”.

Initial conditions

Specify the initial conditions of the filter states. To learn how to specify initial conditions, see “Specifying Initial Conditions” on page 2-347.

Initial conditions on zeros side

(Not shown in dialog above.) Specify the initial conditions for the filter states on the side of the filter structure with the zeros (b_0, b_1,b_2, ...); see the diagram below.

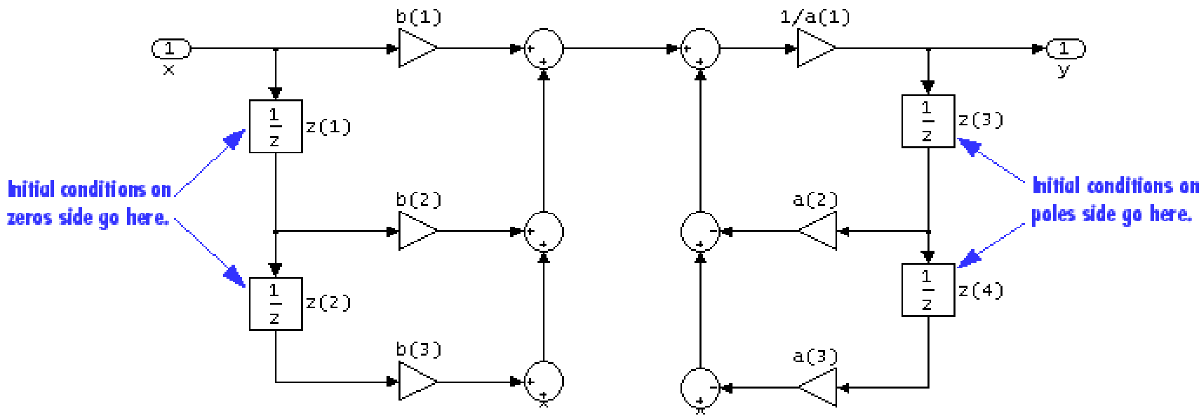
This parameter is enabled only when the filter has both poles and zeros, *and* when you select a structure such as direct form I, which has separate filter states corresponding to the poles (a_k) and zeros (b_k). To learn how to specify initial conditions, see “Specifying Initial Conditions” on page 2-347.

Initial conditions on poles side

(Not shown in dialog above). Specify the initial conditions for the filter states on the side of the filter structure with the poles (a_0, a_1,a_2, ...); see the diagram below.

This parameter is enabled only when the filter has both poles and zeros, *and* when you select a structure such as direct form I, which has separate filter states corresponding to the poles (a_k) and zeros (b_k). To learn how to specify initial conditions, see “Specifying Initial Conditions” on page 2-347.

Digital Filter

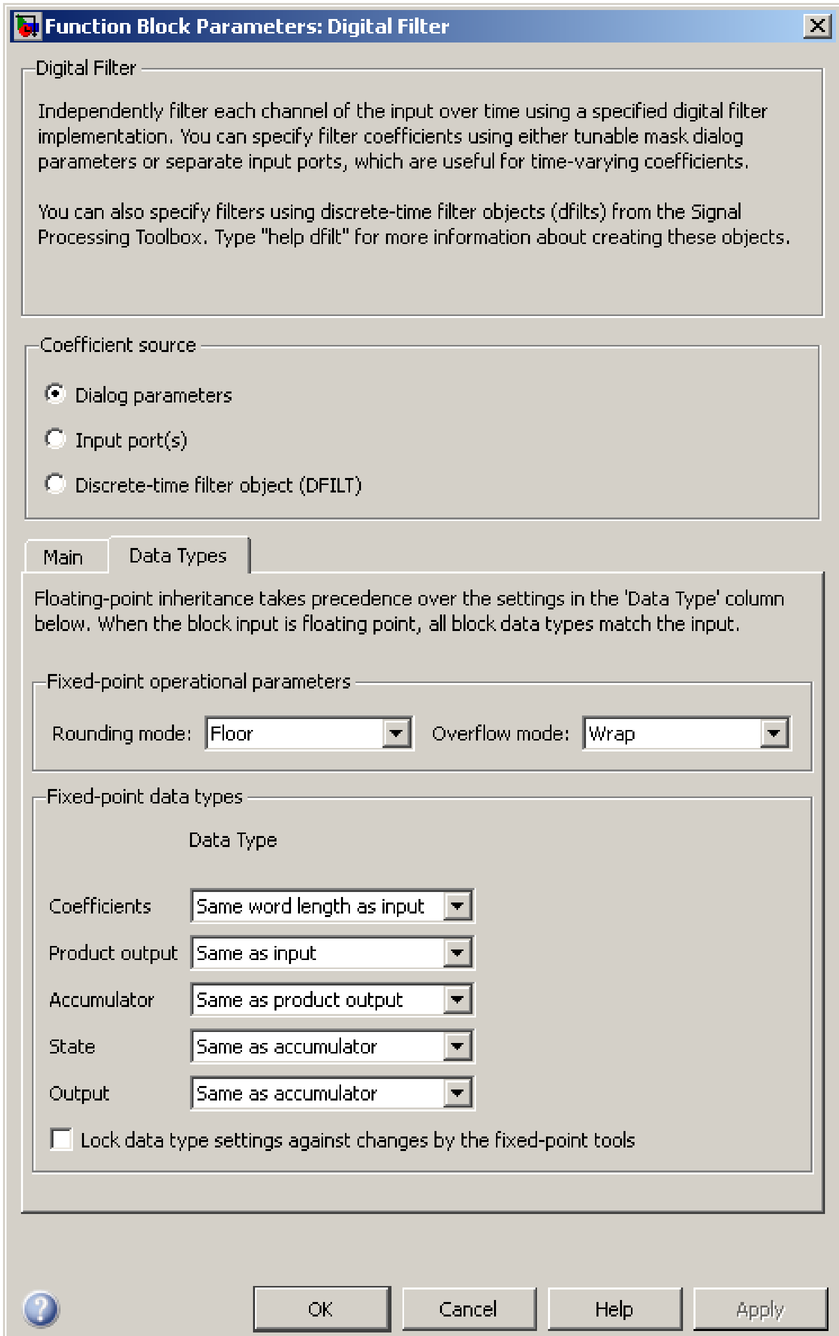


View filter response

This button opens the Filter Visualization Tool (`fvtool`) from the Signal Processing Toolbox product and displays the filter response of the filter defined by the block. For more information on FVTool, see the Signal Processing Toolbox documentation.

Note This button is only available when the Filter Design Toolbox product is installed. If you specify a filter in the **Filter** parameter, you must apply the filter by clicking the **Apply** button before using the **View filter response** button.

The **Data Types** pane of the Digital Filter block dialog appears as follows when **Dialog parameters** is specified in the **Coefficient source** group box. The parameters below can appear when **Dialog parameters** or **Input port(s)** is selected, depending on the filter structure and whether the coefficients are being entered via ports or on the block mask.


Function Block Parameters: Digital Filter
Digital Filter
Independently filter each channel of the input over time using a specified digital filter implementation. You can specify filter coefficients using either tunable mask dialog parameters or separate input ports, which are useful for time-varying coefficients.
You can also specify filters using discrete-time filter objects (dfilts) from the Signal Processing Toolbox. Type "help dfilt" for more information about creating these objects.
Coefficient source
Dialog parameters
Input port(s)
Discrete-time filter object (DFILT)
Main
Data Types
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input.
Fixed-point operational parameters
Rounding mode: Floor
Overflow mode: Wrap
Fixed-point data types
Data Type
Coefficients Same word length as input
Product output Same as input
Accumulator Same as product output
State Same as accumulator
Output Same as accumulator
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Rounding mode

Select the rounding mode for fixed-point operations. The filter coefficients do not obey this parameter; they always round to `Nearest`.

Overflow mode

Select the overflow mode for fixed-point operations. The filter coefficients do not obey this parameter; they are always saturated.

Section I/O

Choose how you specify the word length and the fraction length of the fixed-point data type going into and coming out of each section of a biquadratic filter. See "Filter Structure Diagrams" on page 2-368 for illustrations depicting the use of the section I/O data type in this block.

This parameter is only visible when the selected filter structure is biquadratic:

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word and fraction lengths of the section input and output, in bits.
- When you select `Slope and bias scaling`, you can enter the word lengths, in bits, and the slopes of the section input and output. This block requires power-of-two slope and a bias of zero.

Tap sum

Choose how you specify the word length and the fraction length of the tap sum data type of a direct form symmetric or direct form antisymmetric filter. See "Filter Structure Diagrams" on page 2-368 for illustrations depicting the use of the tap sum data type in this block.

This parameter is only visible when the selected filter structure is either `Direct form symmetric` or `Direct form antisymmetric`:

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the tap sum accumulator, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the tap sum accumulator. This block requires power-of-two slope and a bias of zero.

Multiplicand

Choose how you specify the word length and the fraction length of the multiplicand data type of a direct form I transposed or biquadratic direct form I transposed filter. See "Filter Structure Diagrams" on page 2-368 for illustrations depicting the use of the multiplicand data type in this block.

This parameter is only visible when the selected filter structure is either `Direct form I transposed` or `Biquad direct form I transposed (SOS)`:

- When you select `Same as output`, these characteristics match those of the output to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the multiplicand data type, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the multiplicand data type. This block requires power-of-two slope and a bias of zero.

Coefficients

Choose how you specify the word length and the fraction length of the filter coefficients (numerator and/or denominator). See "Filter Structure Diagrams" on page 2-368 for illustrations depicting the use of the coefficient data types in this block:

- When you select `Same word length as input`, the word length of the filter coefficients match that of the input to the block. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

- When you select `Specify word length`, you can enter the word length of the coefficients, in bits. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the coefficients, in bits. If applicable, you can enter separate fraction lengths for the numerator and denominator coefficients.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the coefficients. If applicable, you can enter separate slopes for the numerator and denominator coefficients. This block requires power-of-two slope and a bias of zero.
- The filter coefficients do not obey the **Rounding mode** and the **Overflow mode** parameters; they are always saturated and rounded to `Nearest`.

Product output

Use this parameter to specify how you would like to designate the product output word and fraction lengths. See "Filter Structure Diagrams" on page 2-368 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator

Use this parameter to specify how you would like to designate the accumulator word and fraction lengths. See "Filter Structure Diagrams" on page 2-368 and "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Same as product output`, these characteristics match those of the product output.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

State

Use this parameter to specify how you would like to designate the state word and fraction lengths. See "Filter Structure Diagrams" on page 2-368 for illustrations depicting the use of the state data type in this block.

This parameter is not visible for direct form and direct form I filter structures.

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Same as accumulator`, these characteristics match those of the accumulator.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

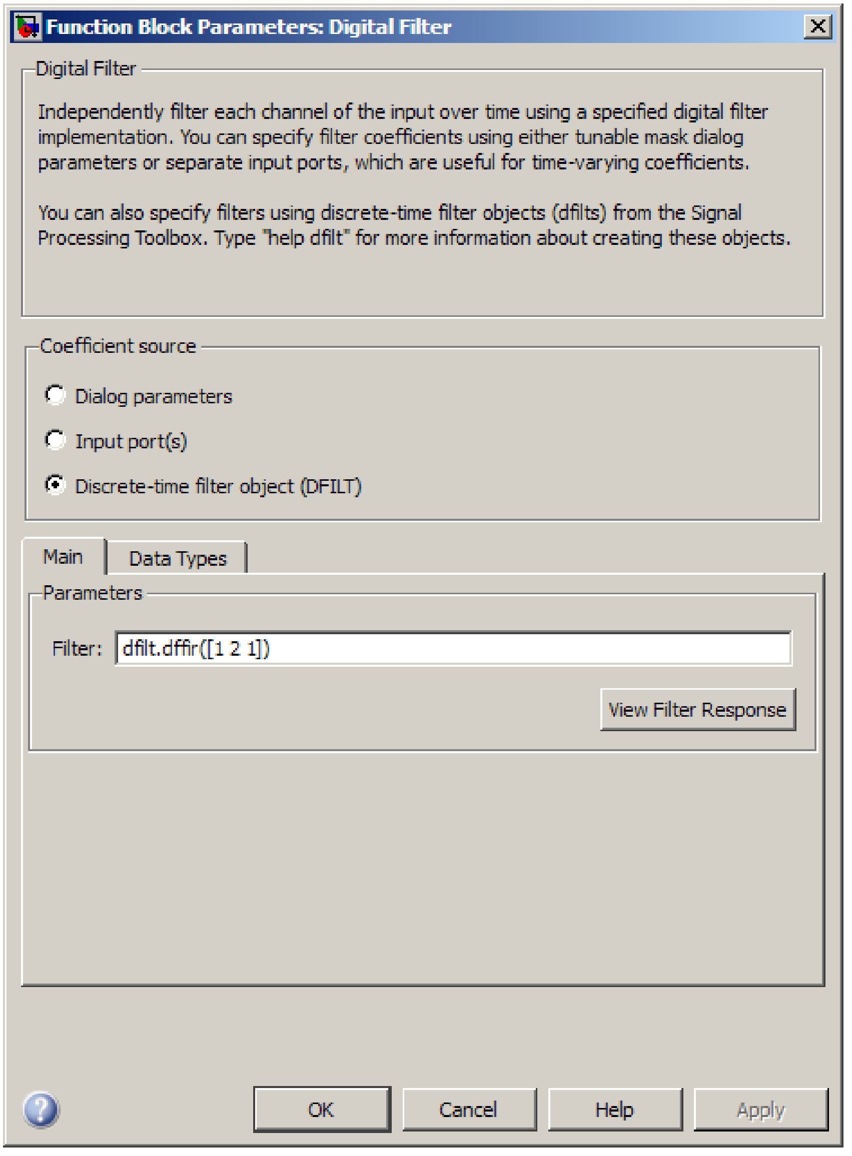
Output
Choose how you specify the output word length and fraction length:

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Same as accumulator`, these characteristics match those of the accumulator.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Specify Discrete-Time Filter Object

The **Main** pane of the Digital Filter block dialog appears as follows when **Discrete-time filter object (DFILT)** is specified in the **Coefficient source** group box.


Function Block Parameters: Digital Filter
Digital Filter
Independently filter each channel of the input over time using a specified digital filter implementation. You can specify filter coefficients using either tunable mask dialog parameters or separate input ports, which are useful for time-varying coefficients.
You can also specify filters using discrete-time filter objects (dfilts) from the Signal Processing Toolbox. Type "help dfilt" for more information about creating these objects.
Coefficient source
Dialog parameters
Input port(s)
Discrete-time filter object (DFILT)
Main
Data Types
Parameters
Filter:
dfilt.dffir([1 2 1])
View Filter Response
OK
Cancel
Help
Apply

Filter

Specify the discrete-time filter object (`dfilt`) that you would like the block to implement. You can do this in one of three ways:

- You can fully specify the `dfilt` object in the block mask, as shown in the default value.
- You can enter the variable name of a `dfilt` object that is defined in any workspace.
- You can enter a variable name for a `dfilt` object that is not yet defined.

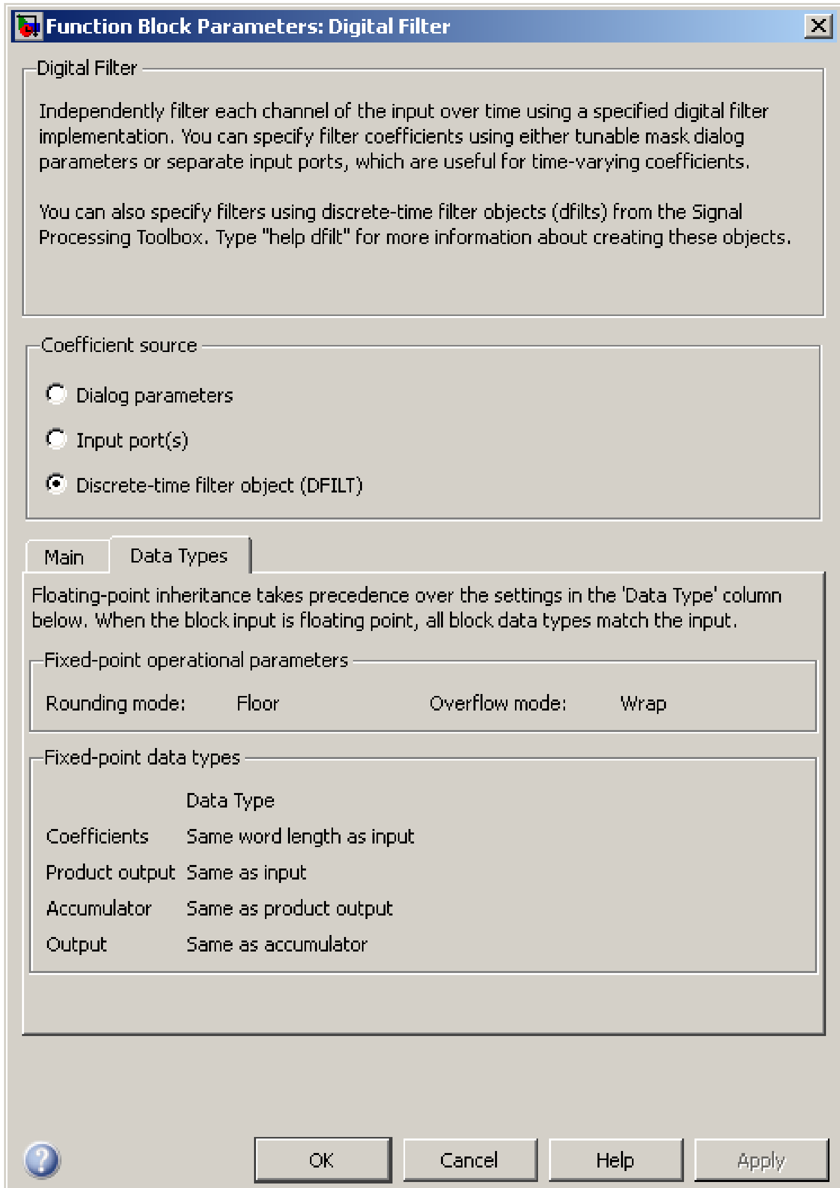
For more information on creating `dfilt` objects, see the `dfilt` function reference page in the Signal Processing Toolbox documentation or the Filter Design Toolbox documentation.

View filter response

This button opens the Filter Visualization Tool (`fvtool`) from the Signal Processing Toolbox product and displays the filter response of the `dfilt` object specified in the **Filter** parameter. For more information on FVTool, see the Signal Processing Toolbox documentation.

Note This button is only available when the Filter Design Toolbox product is installed. If you specify a filter in the **Filter** parameter, you must apply the filter by clicking the **Apply** button before using the **View filter response** button.

The **Data Types** pane of the Digital Filter block dialog appears as follows when **Discrete-time filter object (DFILT)** is specified in the **Coefficient source** group box.


Function Block Parameters: Digital Filter
Digital Filter
Independently filter each channel of the input over time using a specified digital filter implementation. You can specify filter coefficients using either tunable mask dialog parameters or separate input ports, which are useful for time-varying coefficients.
You can also specify filters using discrete-time filter objects (dfilts) from the Signal Processing Toolbox. Type "help dfilt" for more information about creating these objects.
Coefficient source
Dialog parameters
Input port(s)
Discrete-time filter object (DFILT)
Main
Data Types
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input.
Fixed-point operational parameters
Rounding mode: Floor
Overflow mode: Wrap
Fixed-point data types
Data Type
Coefficients Same word length as input
Product output Same as input
Accumulator Same as product output
Output Same as accumulator
OK
Cancel
Help
Apply

The fixed-point settings of the filter object specified on the **Main** pane are displayed on the **Data Types** pane. You cannot change these settings directly on the block mask. To change the fixed-point settings you must edit the filter object directly.

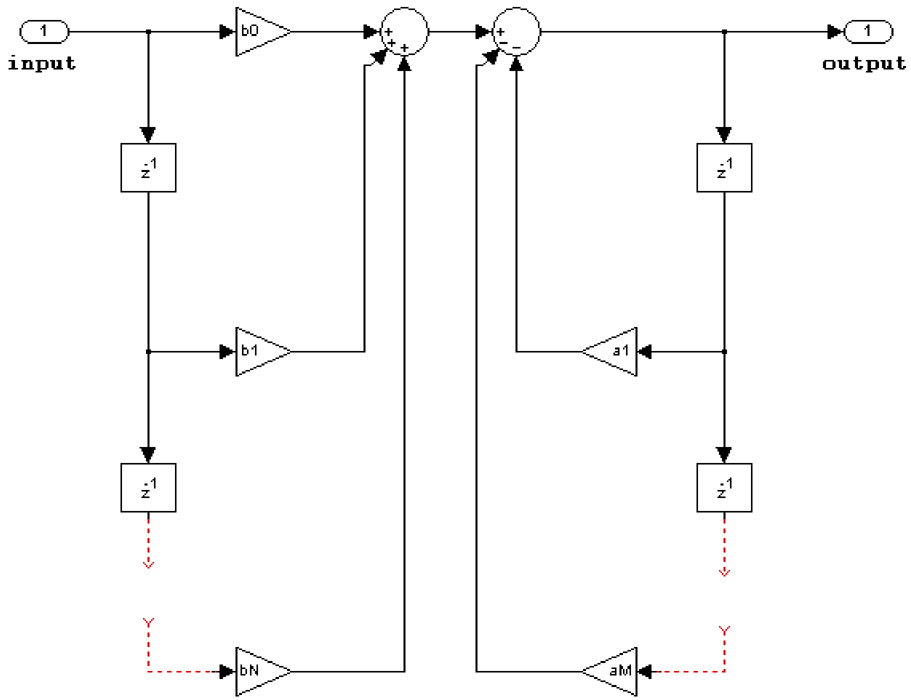
For more information on discrete-time filter objects, see the `dfilt` function reference page in the Signal Processing Toolbox documentation or the Filter Design Toolbox documentation.

Filter Structure Diagrams

The diagrams in the following sections show the filter structures supported by the Digital Filter block. They also show the data types used in the filter structures for fixed-point signals. You can set the coefficient, output, accumulator, product output, and state data types shown in these diagrams in the block dialog. This is discussed in "Dialog Box" on page 2-352.

- "FIR (all zeros) direct form transposed" on page 2-401
- "FIR (all zeros) lattice MA" on page 2-403

IIR direct form I

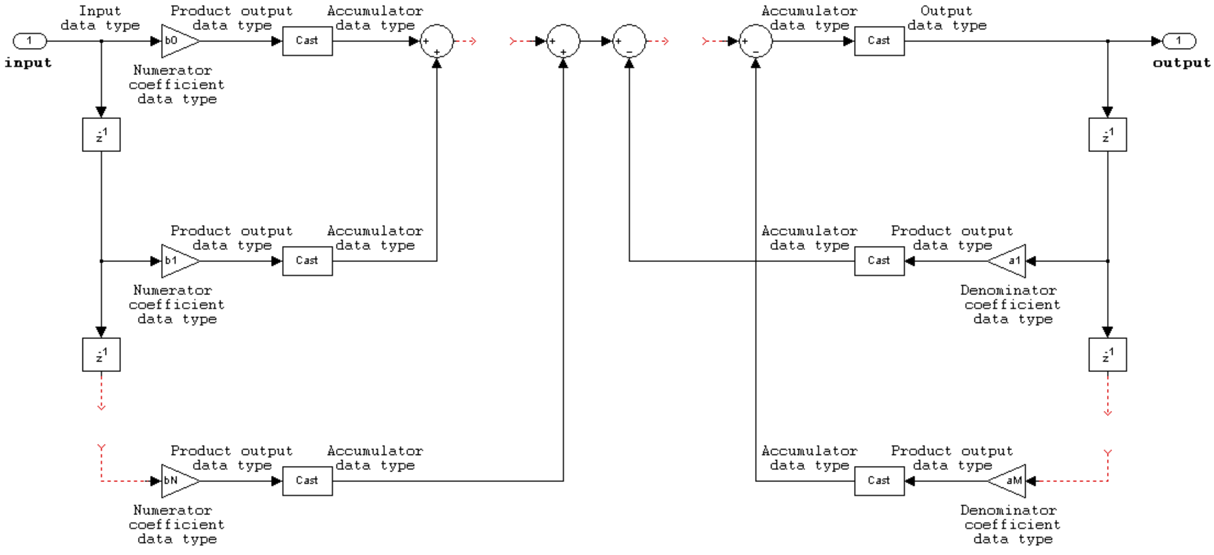


The following constraints are applicable when processing a fixed-point signal with this filter structure:

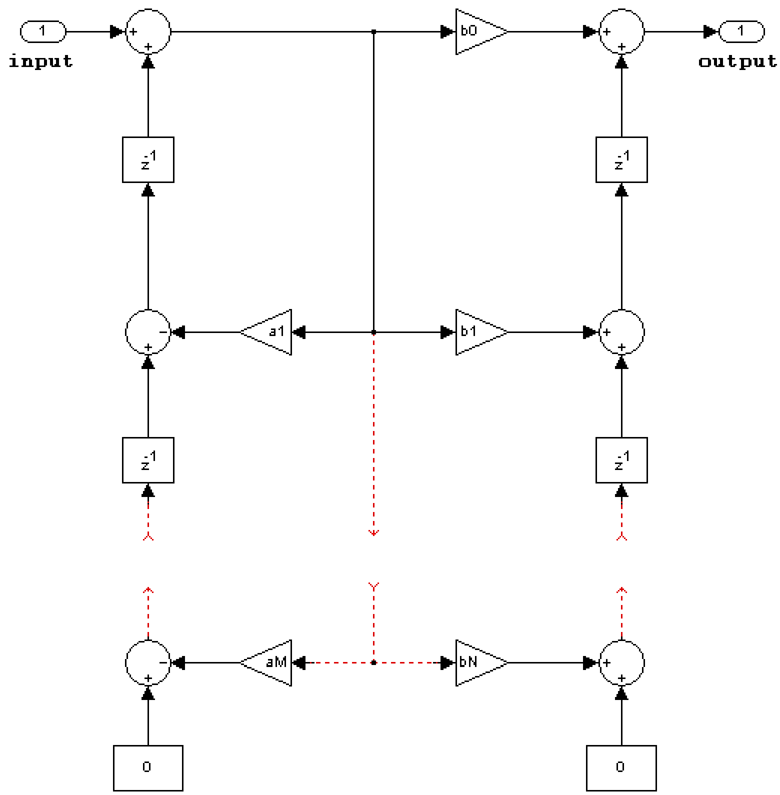
- Inputs can be real or complex.
- Numerator and denominator coefficients can be real or complex.
- Numerator and denominator coefficients must be the same complexity as each other.

  - When the numerator and denominator coefficients are specified via input ports and have different complexities from each other, you get an error.
  - When the numerator and denominator coefficients are specified in the dialog and have different complexities from each other, the block does not error, but instead processes the filter as if two sets of complex coefficients are provided. The coefficient set that is real-valued is treated as if it is a complex vector with zero-valued imaginary parts.

- Numerator and denominator coefficients must have the same word length. They can have different fraction lengths.
- The State data type cannot be specified on the block mask for this structure, because the input and output states have the same data types as the input and output buffers.



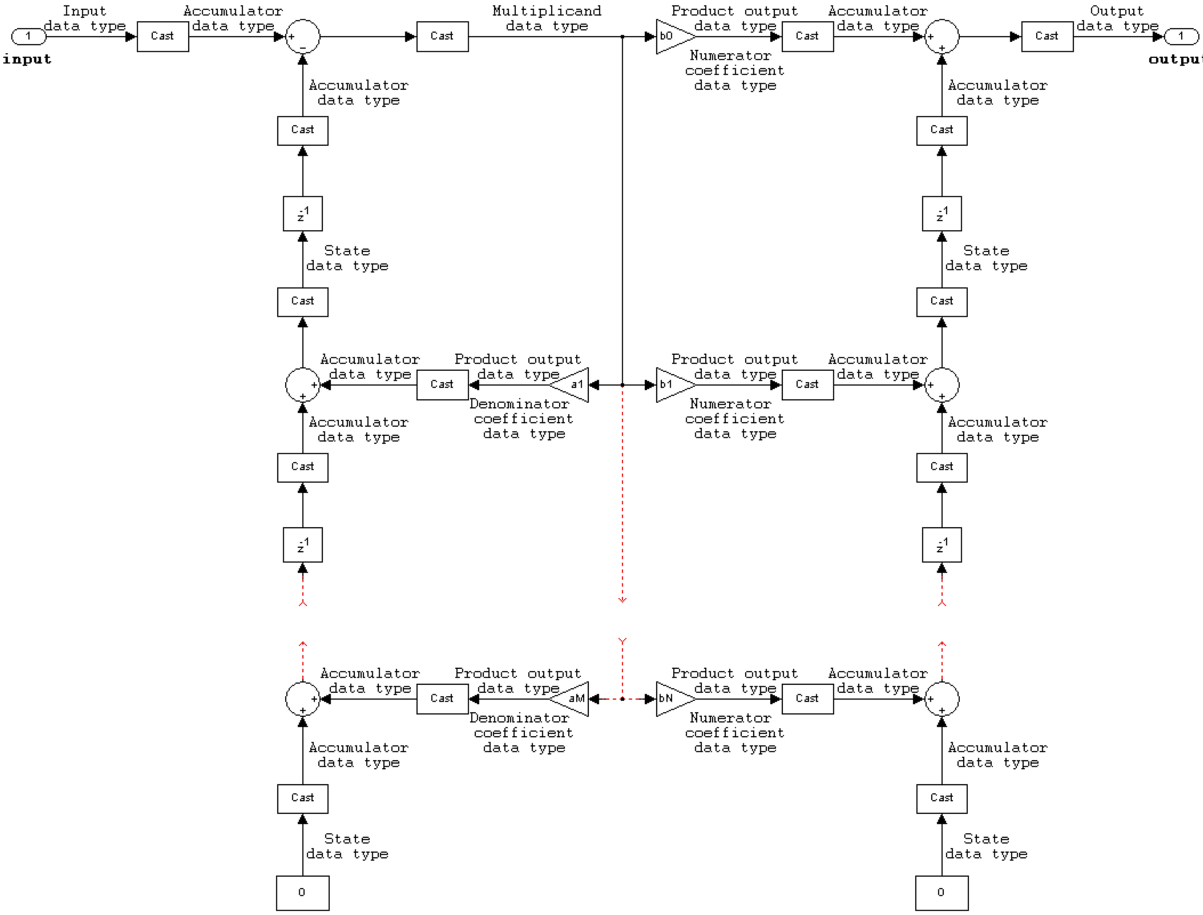
IIR direct form I transposed



The following constraints are applicable when processing a fixed-point signal with this filter structure:

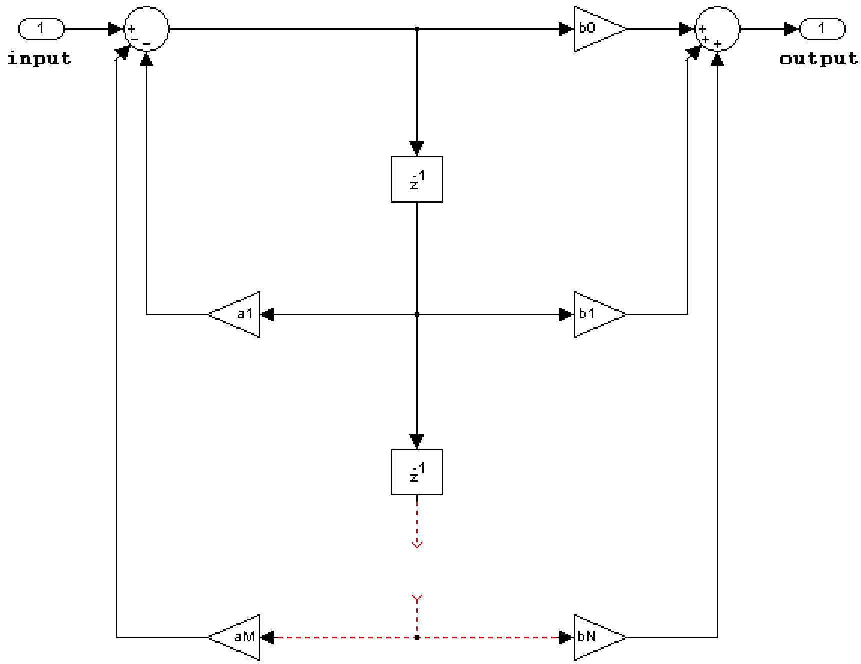
- Inputs can be real or complex.
- Numerator and denominator coefficients can be real or complex.
- Numerator and denominator coefficients must be the same complexity as each other.

  - When the numerator and denominator coefficients are specified via input ports and have different complexities from each other, you get an error.
  - When the numerator and denominator coefficients are specified in the dialog and have different complexities from each other, the block does not error, but instead processes the filter as if two sets of complex coefficients are provided. The coefficient set that is real-valued is treated as if it is a complex vector with zero-valued imaginary parts.
- States are complex when either the input or the coefficients are complex.
- Numerator and denominator coefficients must have the same word length. They can have different fraction lengths.



Input data type
Accumulator data type
Multiplicand data type
Product output data type
Accumulator data type
Output data type
input
output
Cast
Numerator coefficient data type
Accumulator data type
State data type
Denominator coefficient data type
b0
b1
bN
a1
aM
0

IIR direct form II

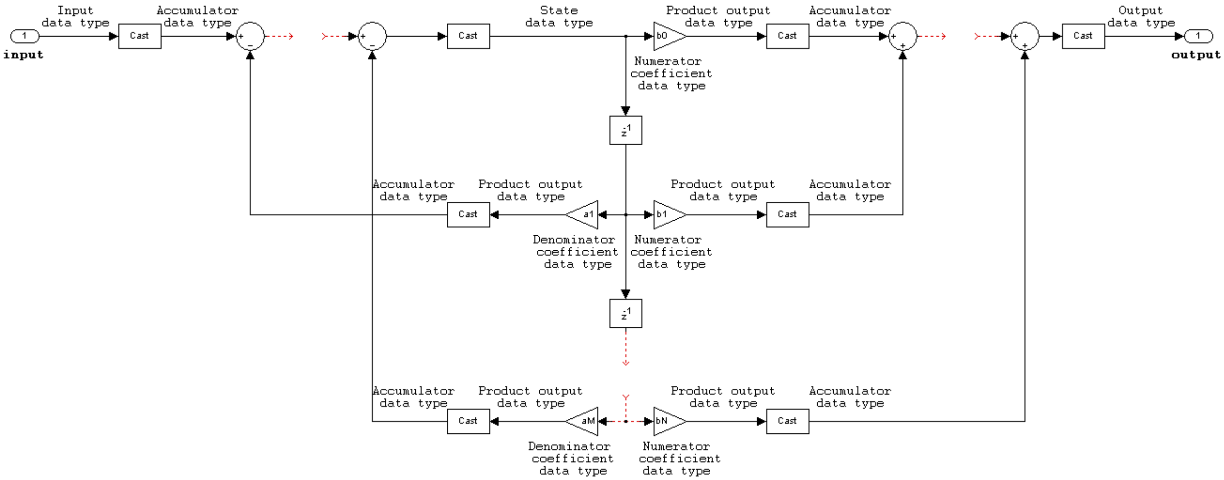


The following constraints are applicable when processing a fixed-point signal with this filter structure:

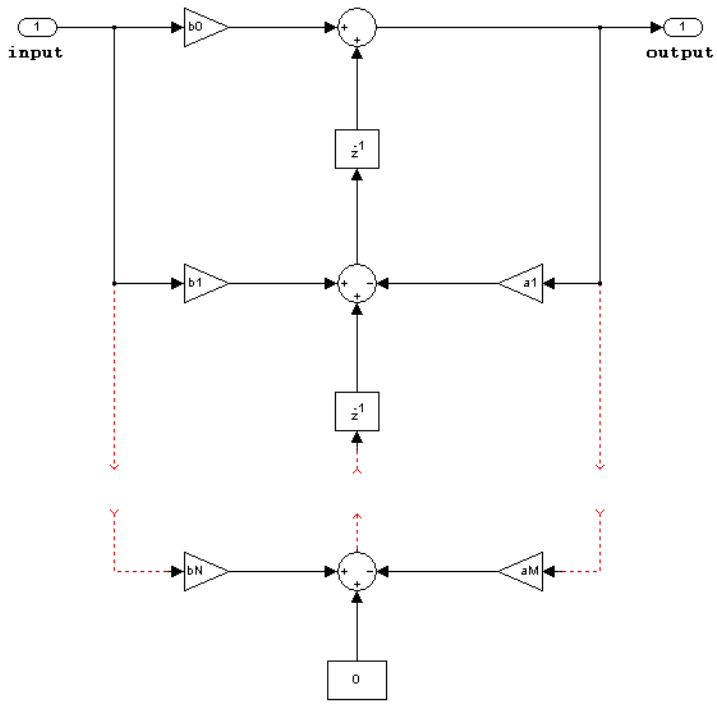
- Inputs can be real or complex.
- Numerator and denominator coefficients can be real or complex.
- Numerator and denominator coefficients must be the same complexity as each other.
  - When the numerator and denominator coefficients are specified via input ports and have different complexities from each other, you get an error.

- When the numerator and denominator coefficients are specified in the dialog and have different complexities from each other, the block does not error, but instead processes the filter as if two sets of complex coefficients are provided. The coefficient set that is real-valued is treated as if it is a complex vector with zero-valued imaginary parts.

- States are complex when either the inputs or the coefficients are complex.
- Numerator and denominator coefficients must have the same word length. They can have different fraction lengths.



IIR direct form II transposed

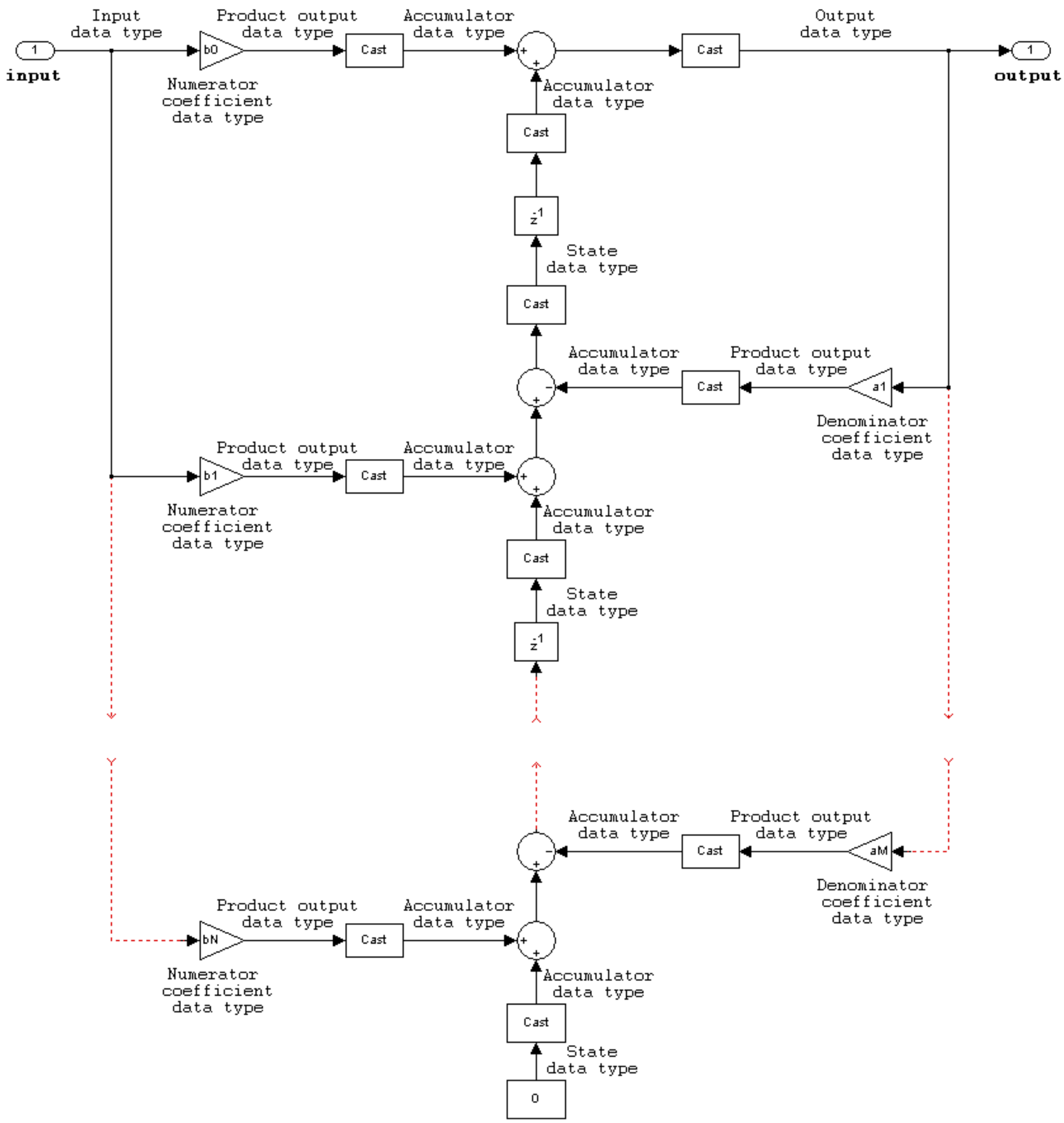


The following constraints are applicable when processing a fixed-point signal with this filter structure:

- Inputs can be real or complex.
- Numerator and denominator coefficients can be real or complex.
- Numerator and denominator coefficients must be the same complexity as each other.
  - When the numerator and denominator coefficients are specified via input ports and have different complexities from each other, you get an error.

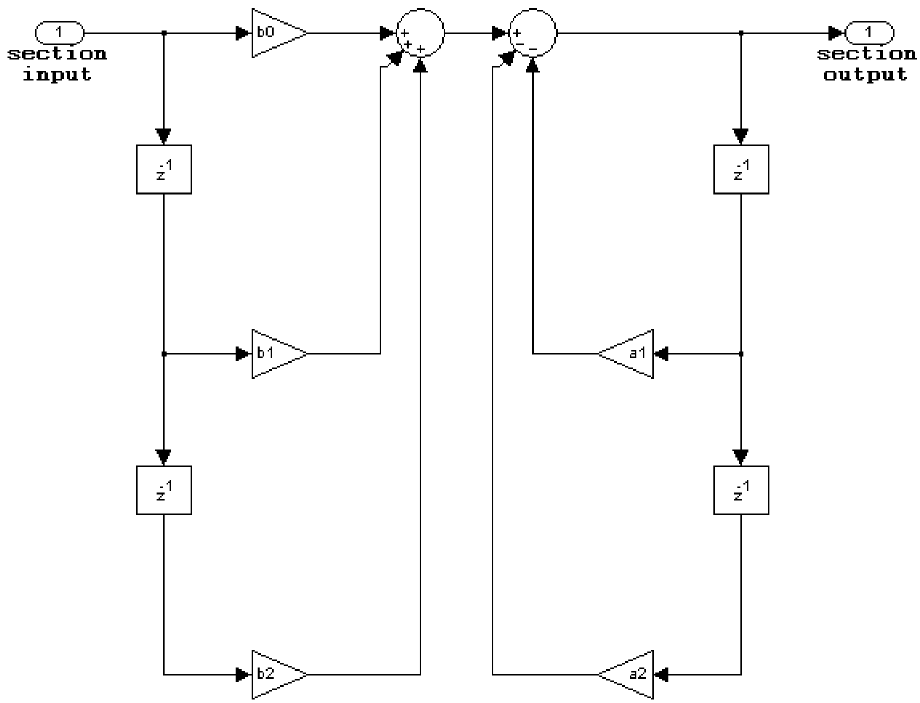
  - When the numerator and denominator coefficients are specified in the dialog and have different complexities from each other, the block does not error, but instead processes the filter as if two sets of complex coefficients are provided. The coefficient set that is real-valued is treated as if it is a complex vector with zero-valued imaginary parts.

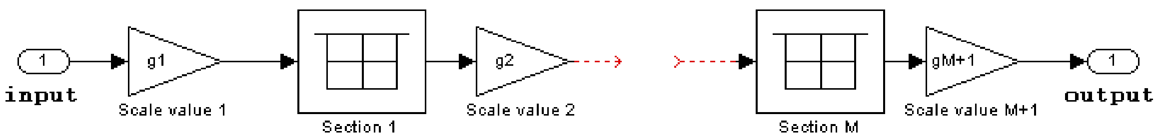
- States are complex when either the inputs or the coefficients are complex.
- Numerator and denominator coefficients must have the same word length. They can have different fraction lengths.



Input
data type
Product output
data type
Accumulator
data type
Cast
Output
data type
1
input
b0
Numerator
coefficient
data type
output
Accumulator
data type
z-1
State
data type
Accumulator
data type
Product output
data type
a1
Denominator
coefficient
data type
b1
bN
aM
0

IIR biquadratic direct form I



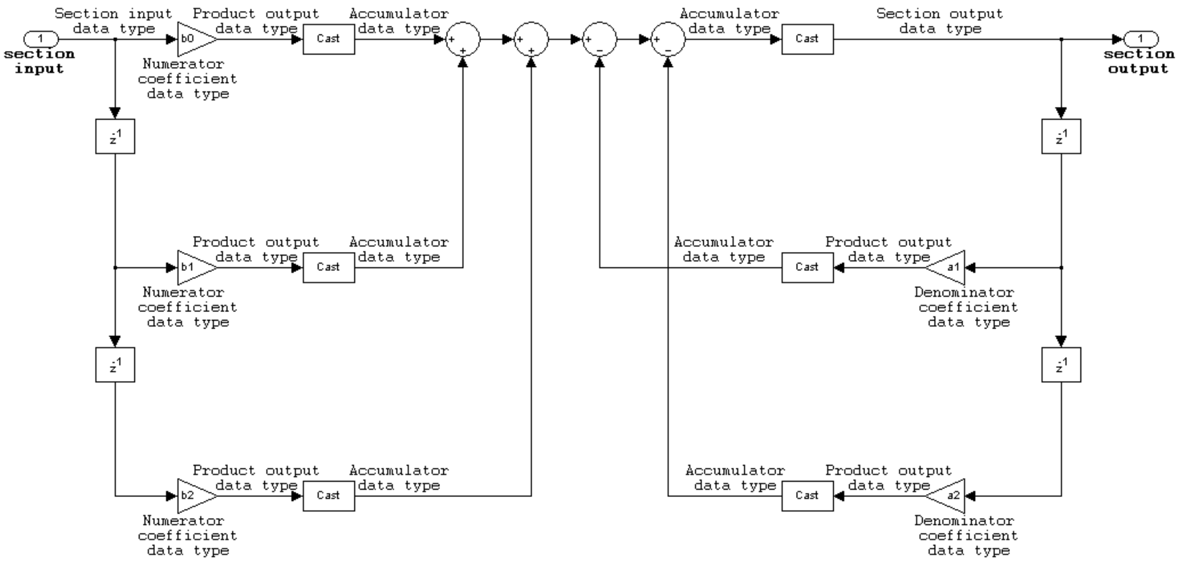


The following constraints are applicable when processing a fixed-point signal with this filter structure:

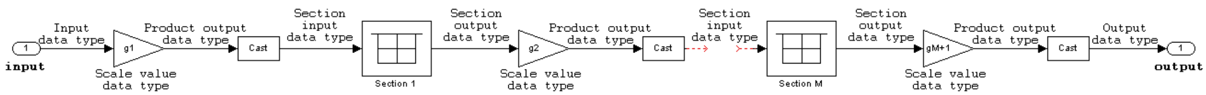
- Inputs and coefficients can be real or complex.
- Numerator and denominator coefficients can be real or complex.

- Specify the coefficients by a M-by-6 matrix in the block mask. You cannot specify coefficients by input ports for this filter structure.
- When the a0 element of any row is not equal to one, that row is normalized by a0 prior to filtering.
- States are complex when either the inputs or the coefficients are complex.
- You cannot specify the state data type on the block mask for this structure, because the input and output states have the same data types as the input.
- Scale values must have the same complexity as the coefficient SOS matrix.
- The scale value parameter must be a scalar or a vector of length M+1, where M is the number of sections.
- The **Section I/O** parameter determines the data type for the section input and output data types. The section input and stage output data type must have the same word length but can have different fraction lengths.

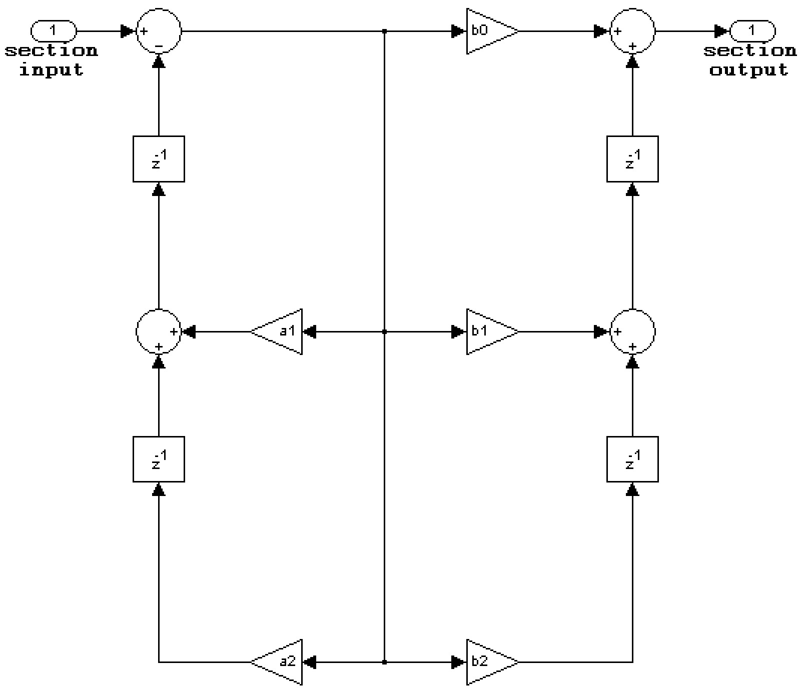
The following diagram shows the data types for one section of the filter.

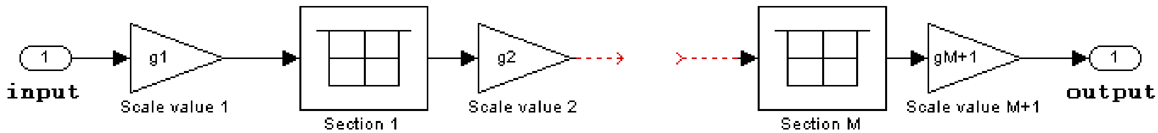


The following diagram shows the data types between filter sections.



IIR biquadratic direct form I transposed



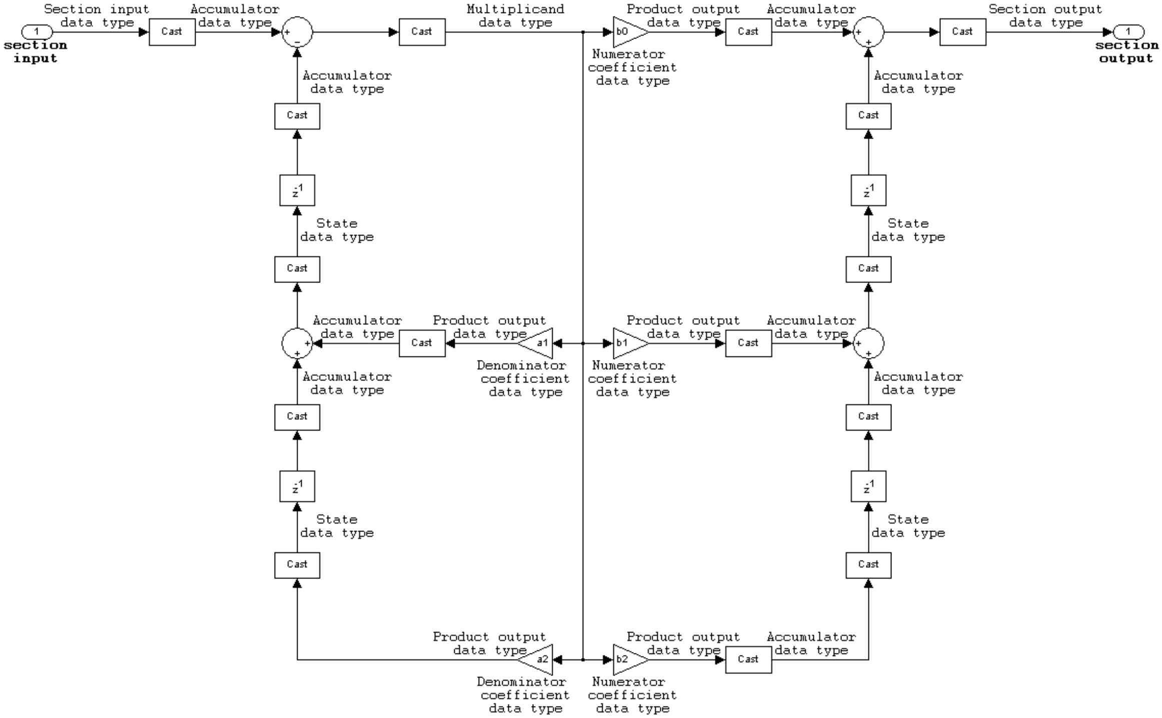


The following constraints are applicable when processing a fixed-point signal with this filter structure:

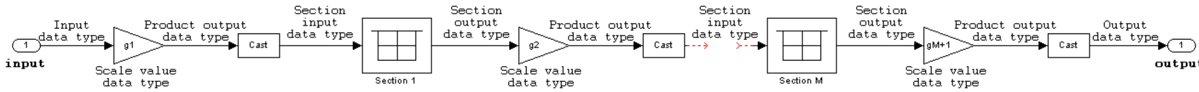
- Inputs and coefficients can be real or complex.
- Numerator and denominator coefficients can be real or complex.

- Specify the coefficients by a *M*-by-6 matrix in the block mask. You cannot specify coefficients by input ports for this filter structure.
- When the a0 element of any row is not equal to one, that row is normalized by a0 prior to filtering.
- States are complex when either the inputs or the coefficients are complex.
- Scale values must have the same complexity as the coefficient SOS matrix.
- The scale value parameter must be a scalar or a vector of length *M*+1, where *M* is the number of sections.
- The **Section I/O** parameter determines the data type for the section input and output data types. The section input and section output data type must have the same word length but can have different fraction lengths.

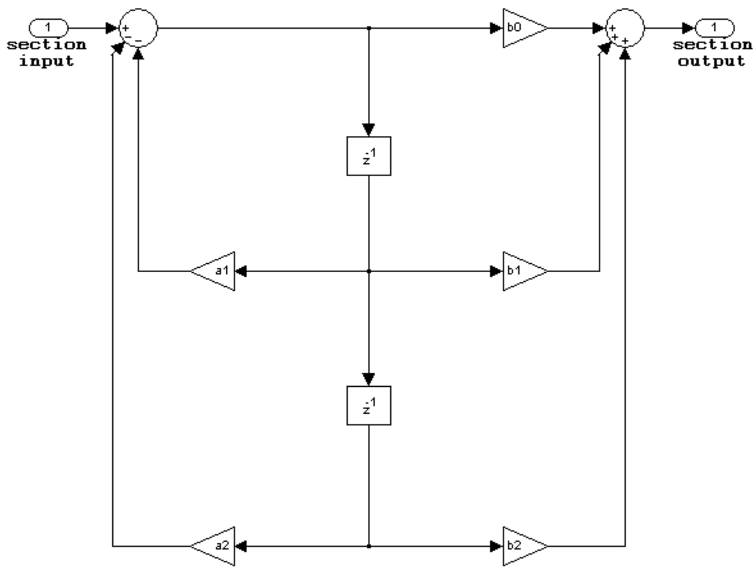
The following diagram shows the data types for one section of the filter.

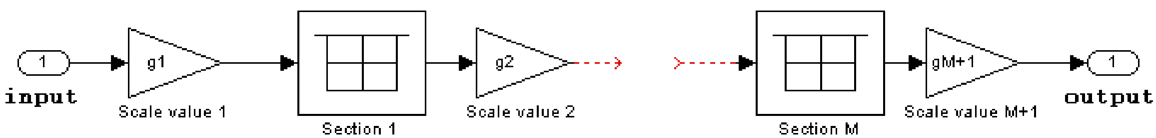


The following diagram shows the data types between filter sections.



IIR biquadratic direct form II



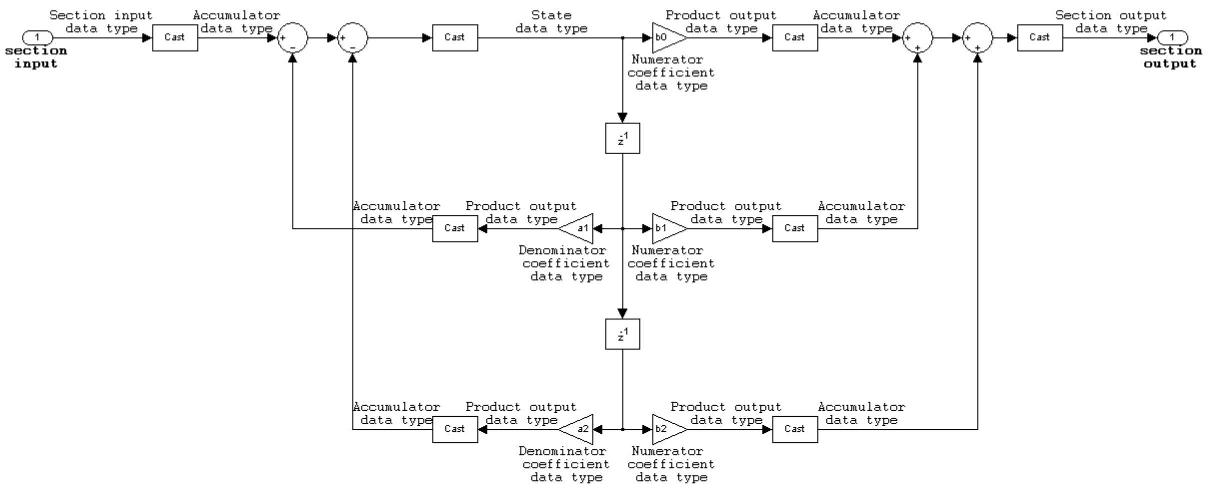


The following constraints are applicable when processing a fixed-point signal with this filter structure:

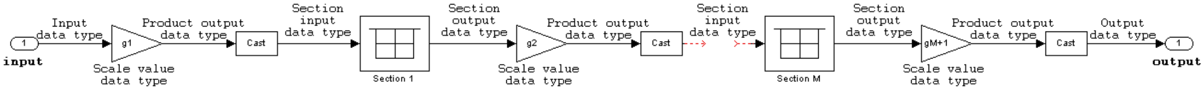
- Inputs and coefficients can be real or complex.
- Numerator and denominator coefficients can be real or complex.

- Specify the coefficients by a M-by-6 matrix in the block mask. You cannot specify coefficients by input ports for this filter structure.
- When the a0 element of any row is not equal to one, that row is normalized by a0 prior to filtering.
- States are complex when either the inputs or the coefficients are complex.
- Scale values must have the same complexity as the coefficient SOS matrix.
- The scale value parameter must be a scalar or a vector of length M+1, where M is the number of sections.
- The **Section I/O** parameter determines the data type for the section input and output data types. The section input and section output data type must have the same word length but can have different fraction lengths.

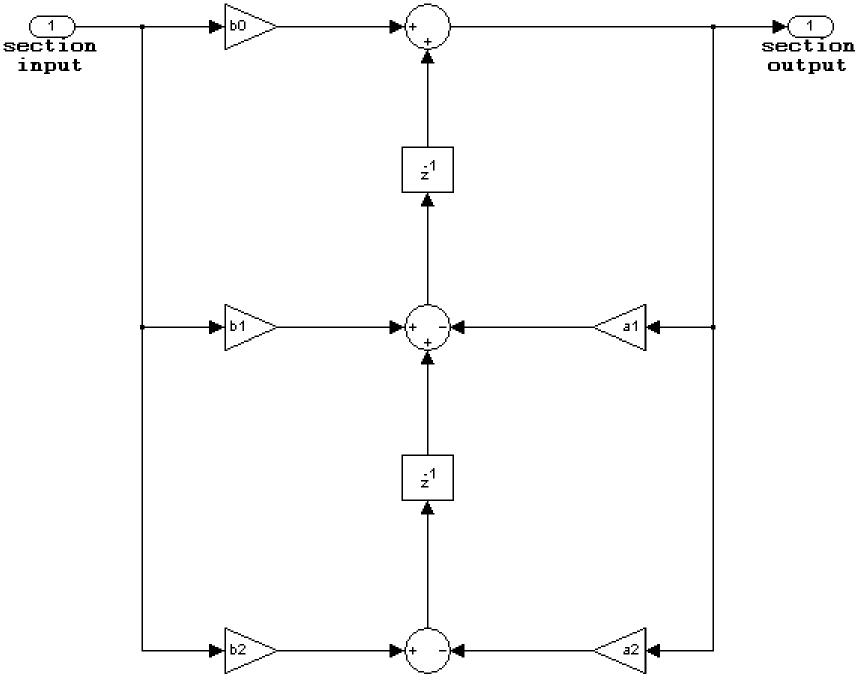
The following diagram shows the data types for one section of the filter.

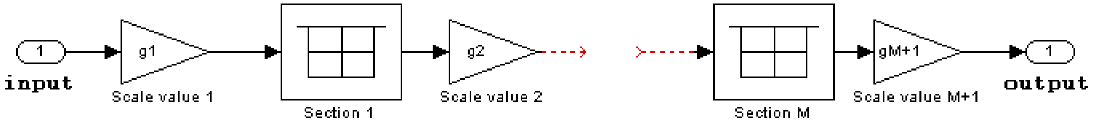


The following diagram shows the data types between filter sections.



IIR biquadratic direct form II transposed

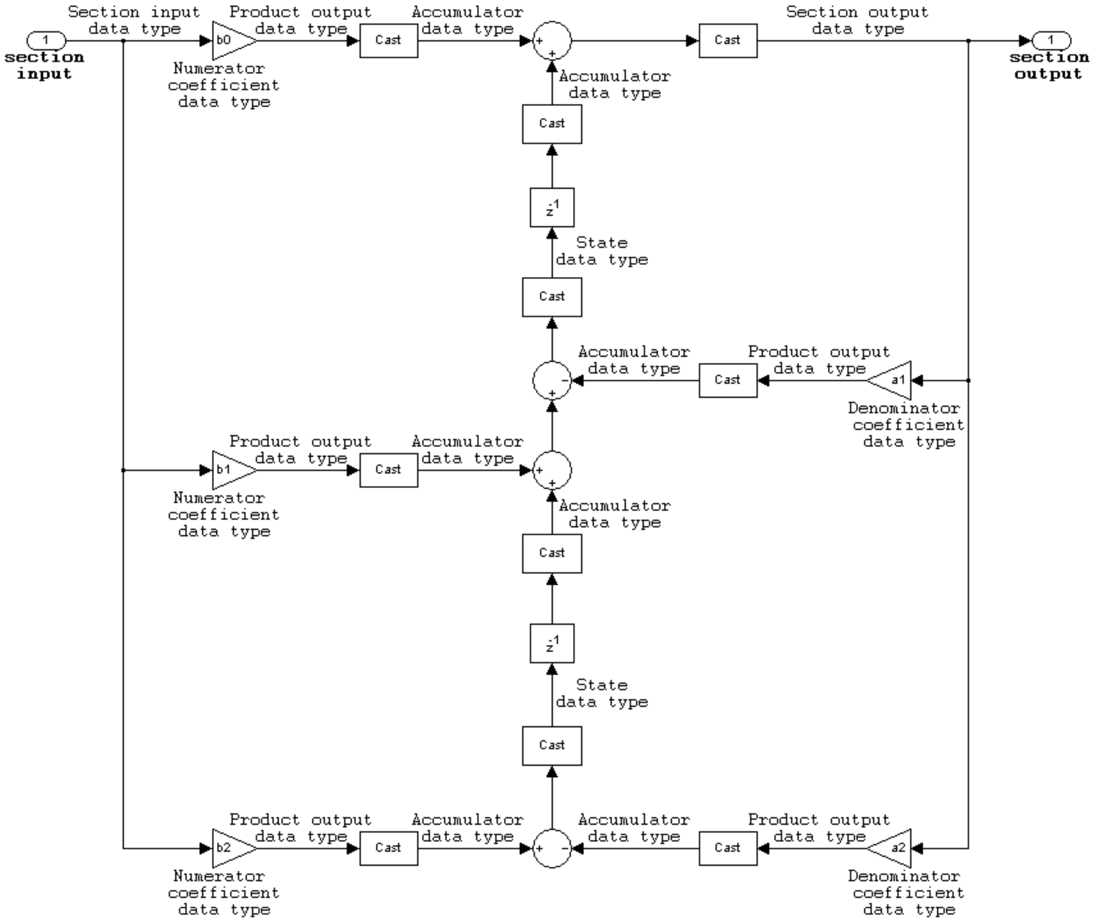




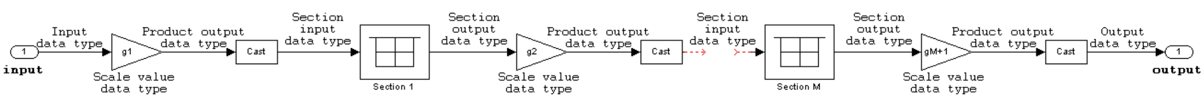
The following constraints are applicable when processing a fixed-point signal with this filter structure:

- Inputs and coefficients can be real or complex.
- Numerator and denominator coefficients can be real or complex.
- Specify the coefficients by a *M*-by-6 matrix in the block mask. You cannot specify coefficients by input ports for this filter structure.
- When the a0 element of any row is not equal to one, that row is normalized by a0 prior to filtering.
- States are complex when either the inputs or the coefficients are complex.
- Scale values must have the same complexity as the coefficient SOS matrix.
- The scale value parameter must be a scalar or a vector of length *M*+1, where *M* is the number of sections.
- The **Section I/O** parameter determines the data type for the section input and output data types. The section input and section output data type must have the same word length but can have different fraction lengths.

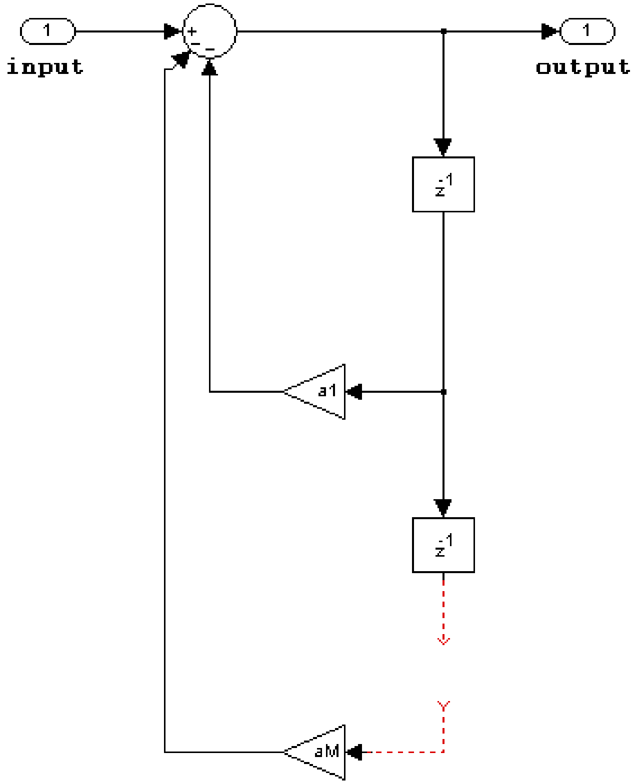
The following diagram shows the data types for one section of the filter.



The following diagram shows the data types between filter sections.

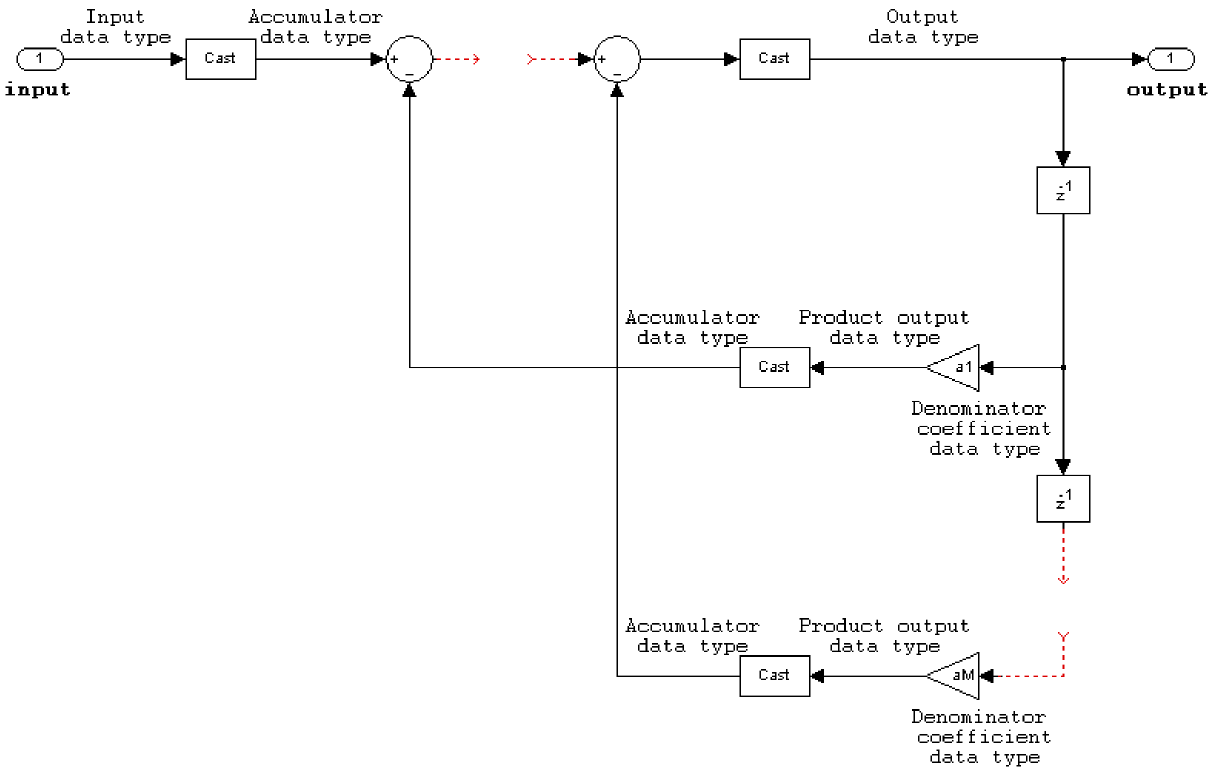


IIR (all poles) direct form

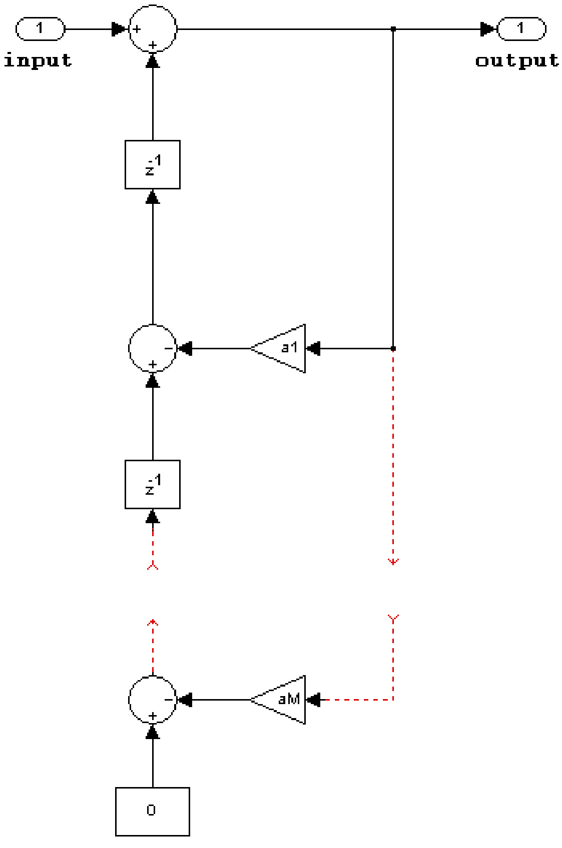


The following constraints are applicable when processing a fixed-point signal with this filter structure:

- Inputs and coefficients can be real or complex.
- Denominator coefficients can be real or complex.
- You cannot specify the state data type on the block mask for this structure, because the input and output states have the same data types as the input.

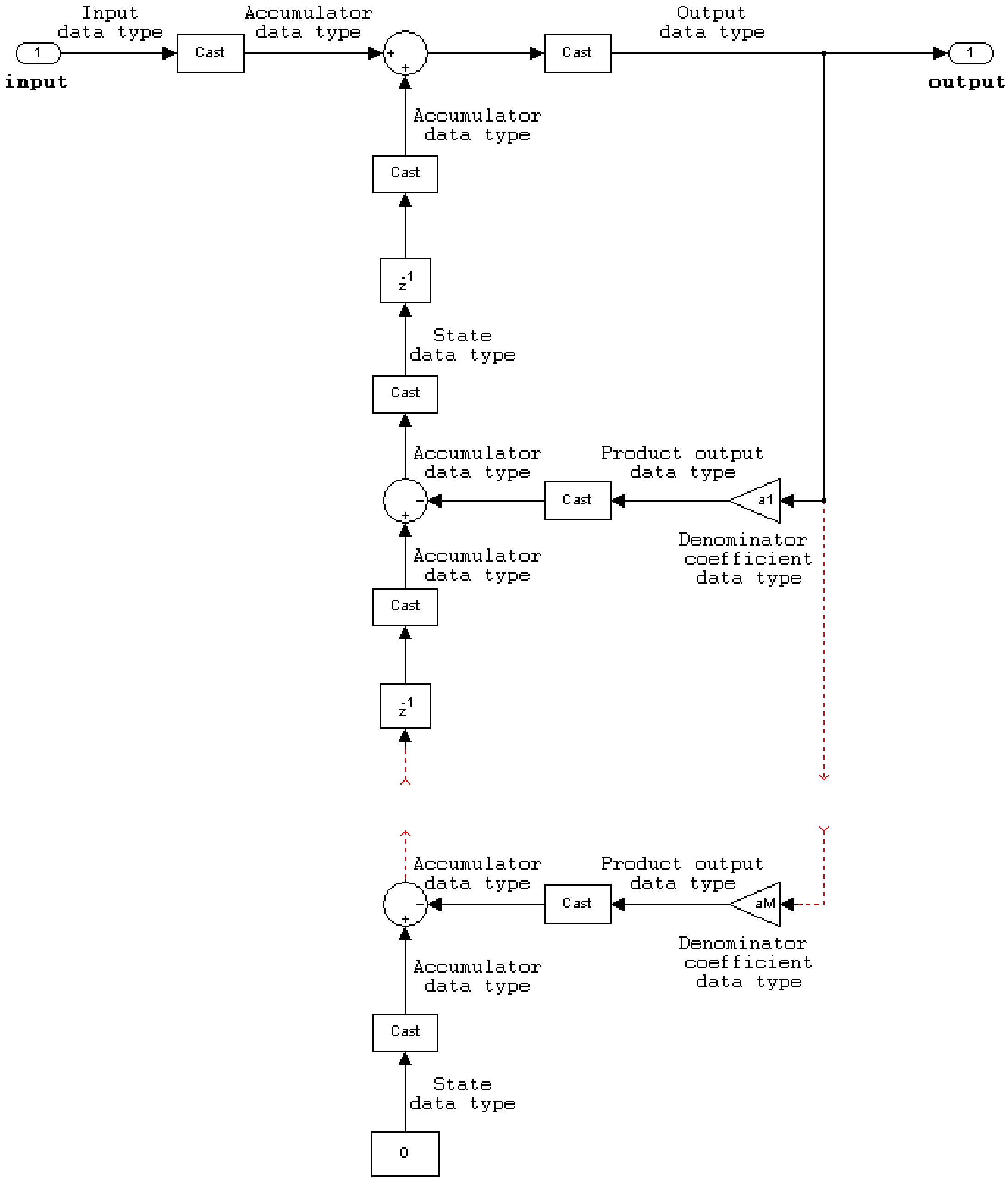

Input
data type
Cast
Accumulator
data type
Cast
Output
data type
1
input
1
output
z-1
Accumulator
data type
Cast
Product output
data type
a1
Denominator
coefficient
data type
z-1
Accumulator
data type
Cast
Product output
data type
aM
Denominator
coefficient
data type

IIR (all poles) direct form transposed

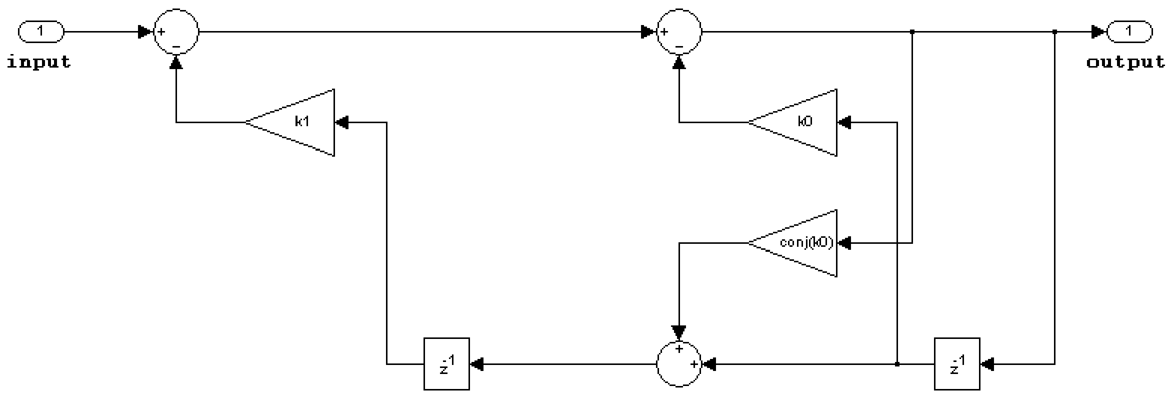


The following constraints are applicable when processing a fixed-point signal with this filter structure:

- Inputs and coefficients can be real or complex.
- Denominator coefficients can be real or complex.

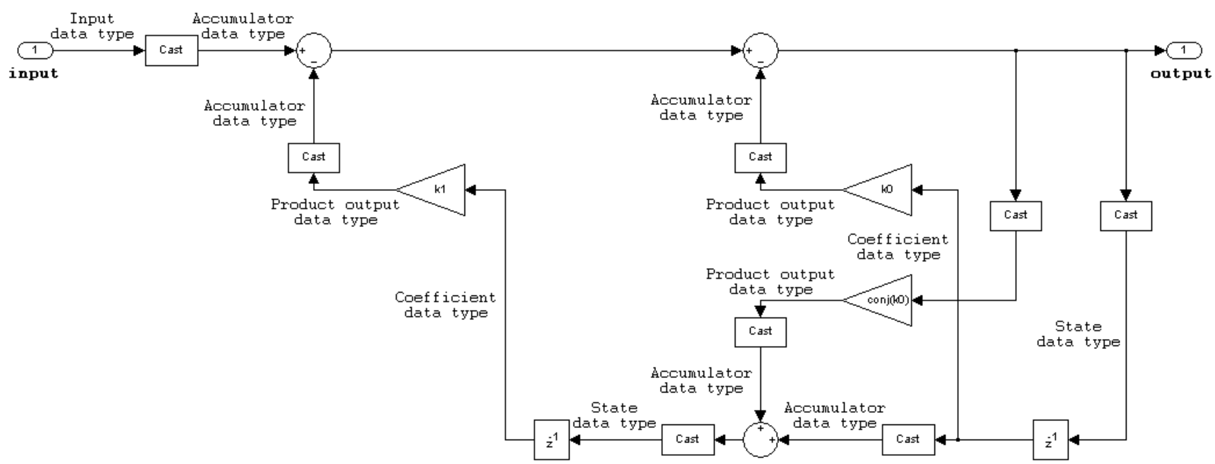

Input
data type
Accumulator
data type
Output
data type
1
input
Cast
Cast
1
output
Accumulator
data type
Cast
z-1
State
data type
Cast
Accumulator
data type
Product output
data type
Cast
a1
Denominator
coefficient
data type
Accumulator
data type
Cast
z-1
Accumulator
data type
Product output
data type
Cast
aM
Denominator
coefficient
data type
Accumulator
data type
Cast
State
data type
0

IIR (all poles) direct form lattice AR

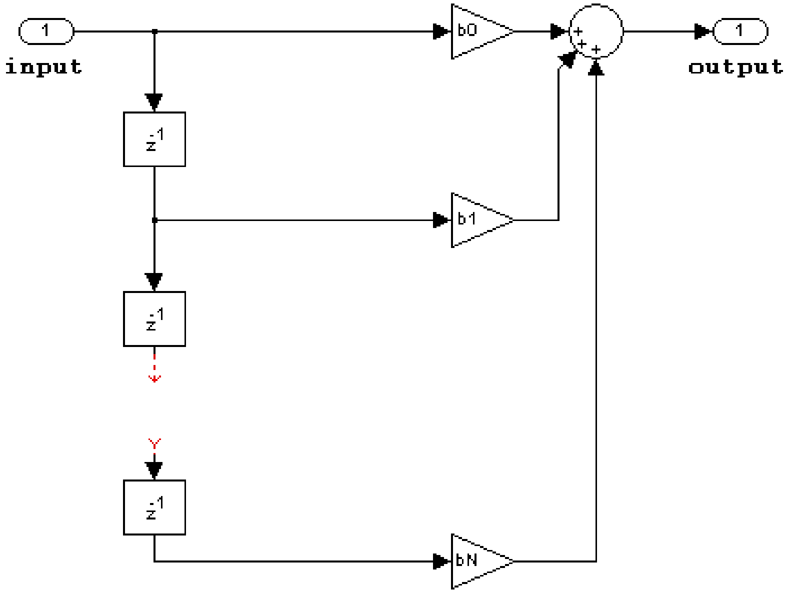


The following constraints are applicable when processing a fixed-point signal with this filter structure:

- Inputs and coefficients can be real or complex.
- Coefficients can be real or complex.

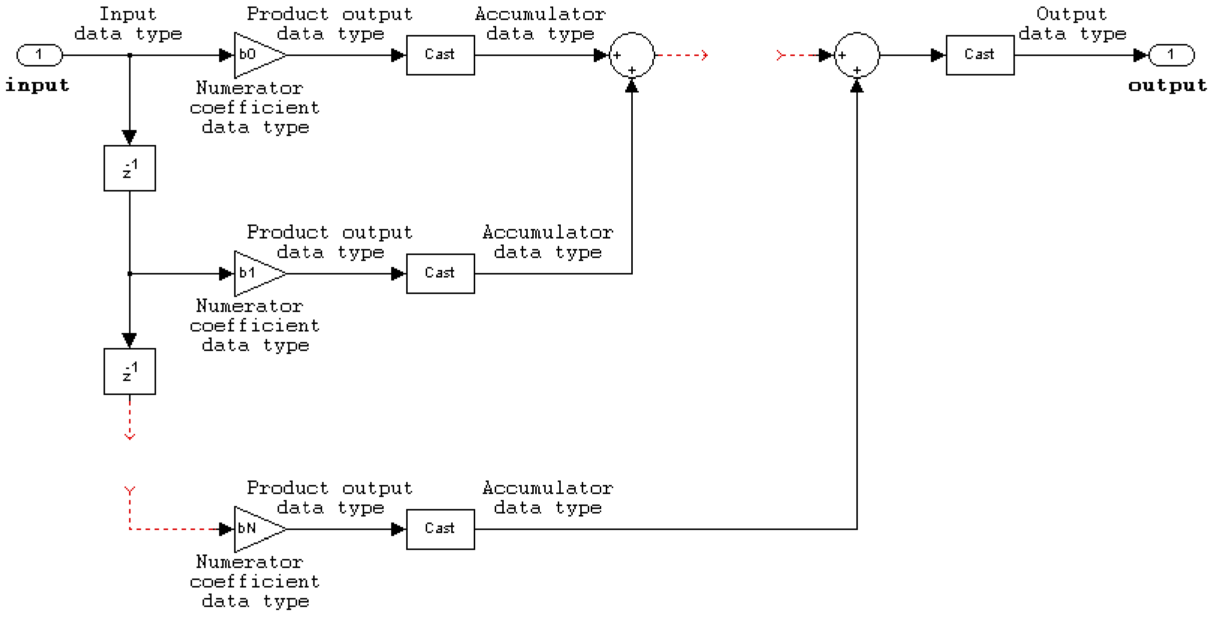


FIR (all zeros) direct form

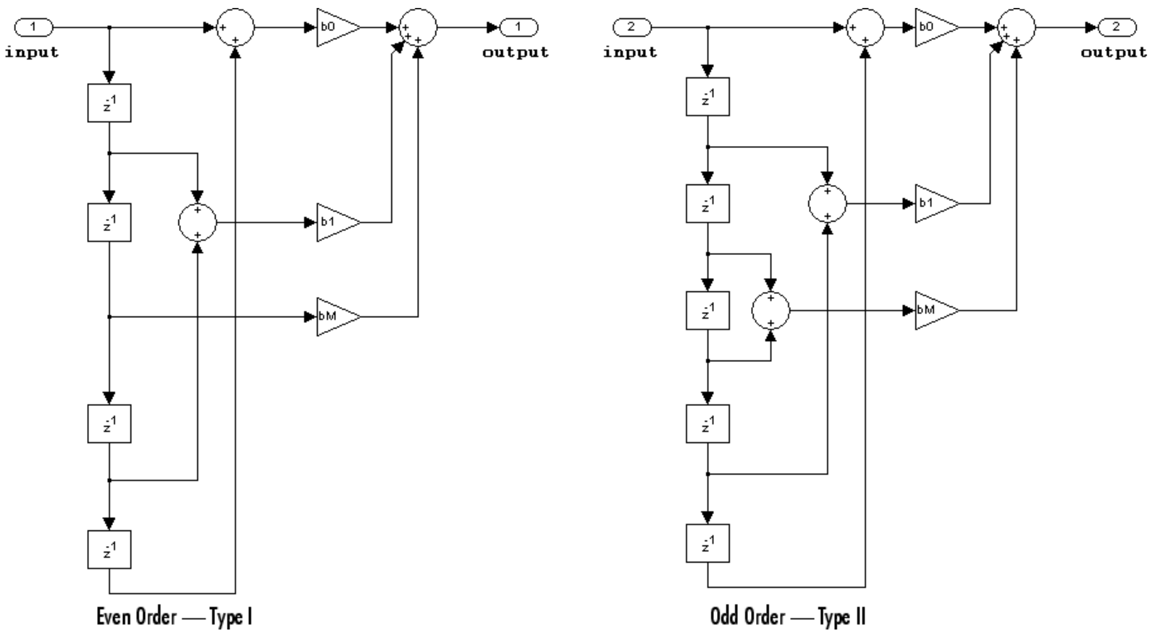


The following constraints are applicable when processing a fixed-point signal with this filter structure:

- Inputs can be real or complex.
- Numerator coefficients can be real or complex.
- You cannot specify the state data type on the block mask for this structure, because the input and output states have the same data types as the input.

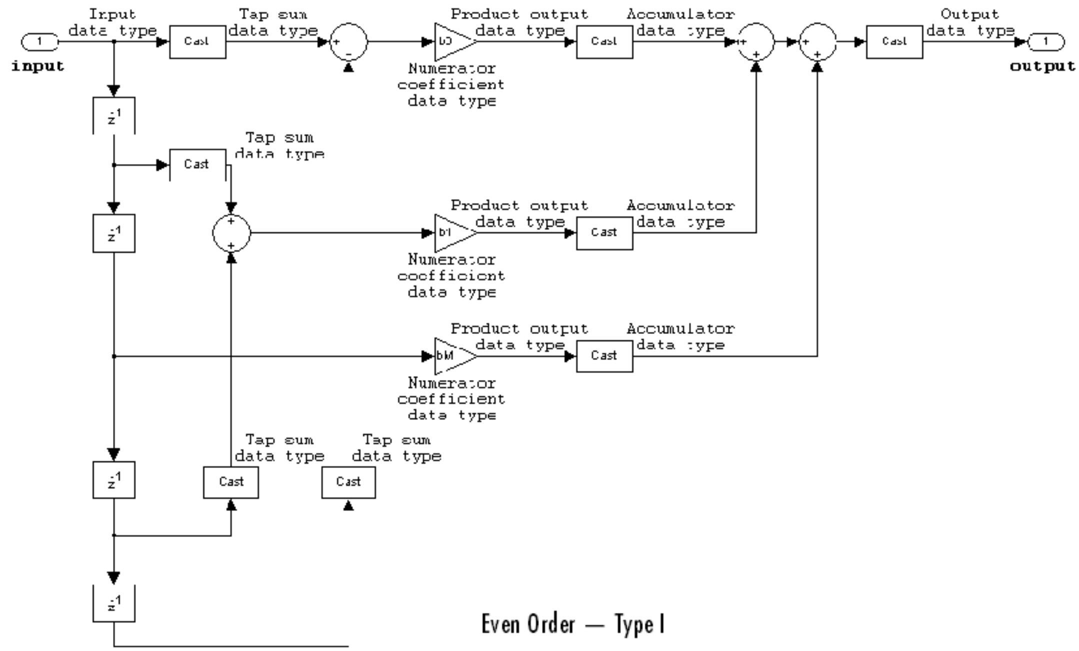

Input
data type
Product output
data type
Accumulator
data type
Output
data type
1
input
b0
Cast
1
output
Numerator
coefficient
data type
z-1
b1
Numerator
coefficient
data type
bN

FIR (all zeros) direct form symmetric

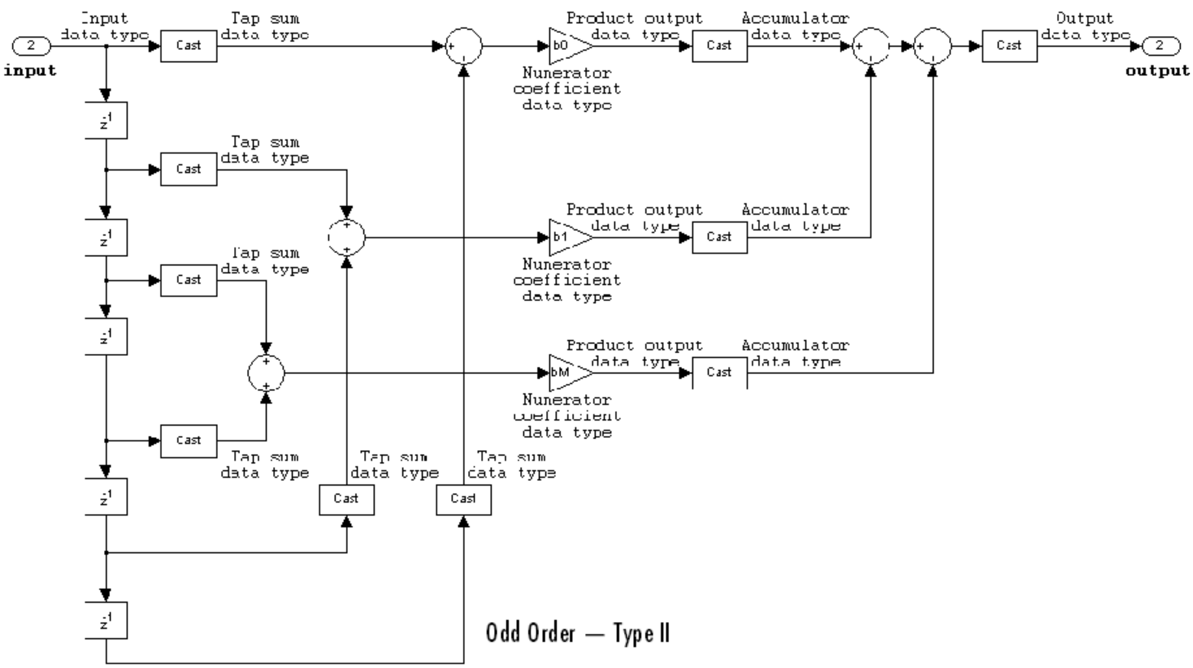


The following constraints are applicable when processing a fixed-point signal with this filter structure:

- Inputs can be real or complex.
- Numerator coefficients can be real or complex.
- You cannot specify the state data type on the block mask for this structure, because the input and output states have the same data types as the input.
- It is assumed that the filter coefficients are symmetric. Only the first half of the coefficients are used for filtering.
- The **Tap Sum** parameter determines the data type the filter uses when it sums the inputs prior to multiplication by the coefficients.

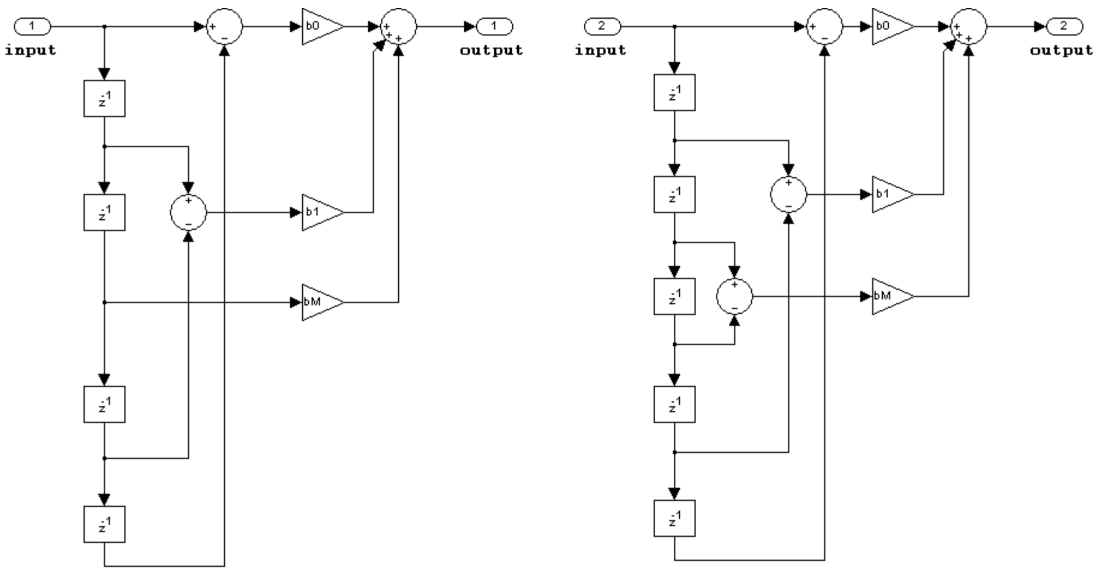


Even Order — Type I



Odd Order — Type II

FIR (all zeros) direct form antisymmetric

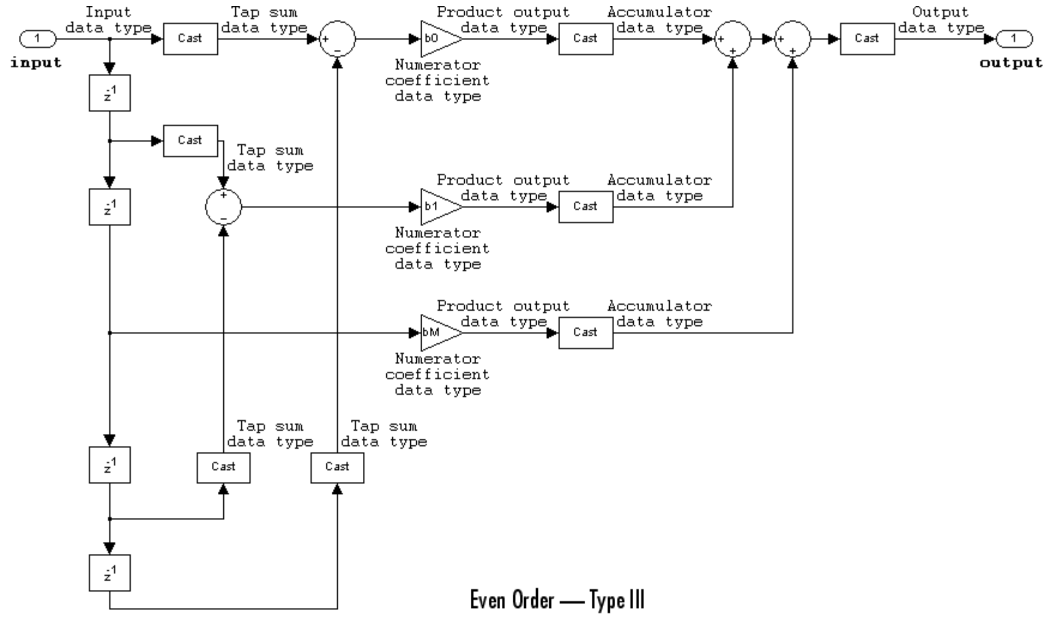


Even Order — Type III

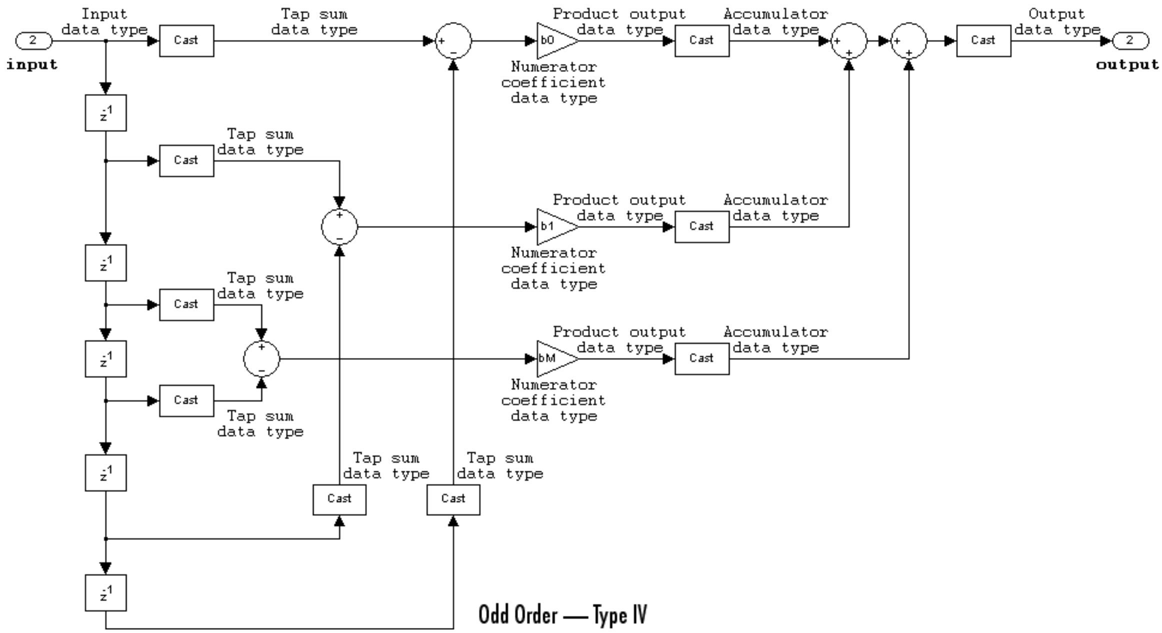
Odd Order — Type IV

The following constraints are applicable when processing a fixed-point signal with this filter structure:

- Inputs can be real or complex.
- Numerator coefficients can be real or complex.
- You cannot specify the state data type on the block mask for this structure, because the input and output states have the same data types as the input.
- It is assumed that the filter coefficients are antisymmetric. Only the first half of the coefficients are used for filtering.
- The **Tap Sum** parameter determines the data type the filter uses when it sums the inputs prior to multiplication by the coefficients.

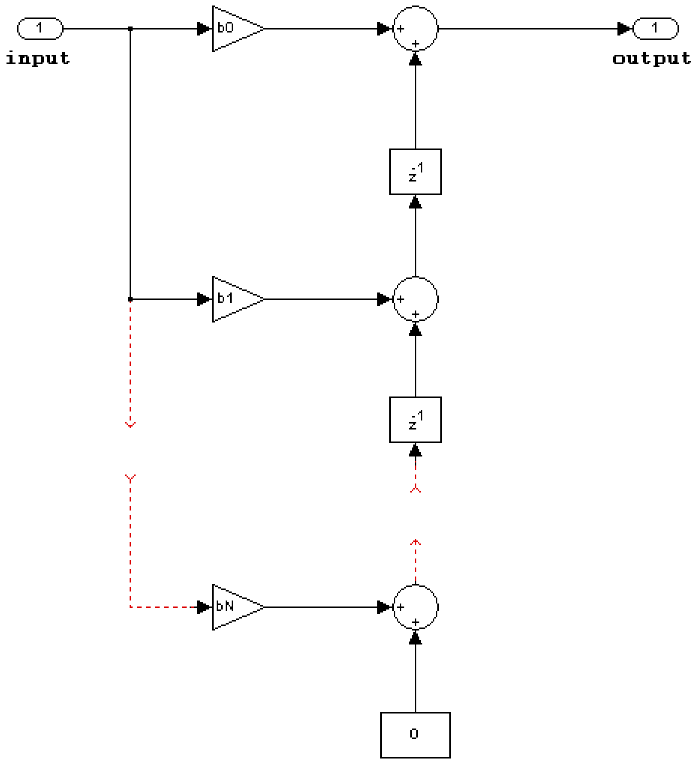


Even Order — Type III



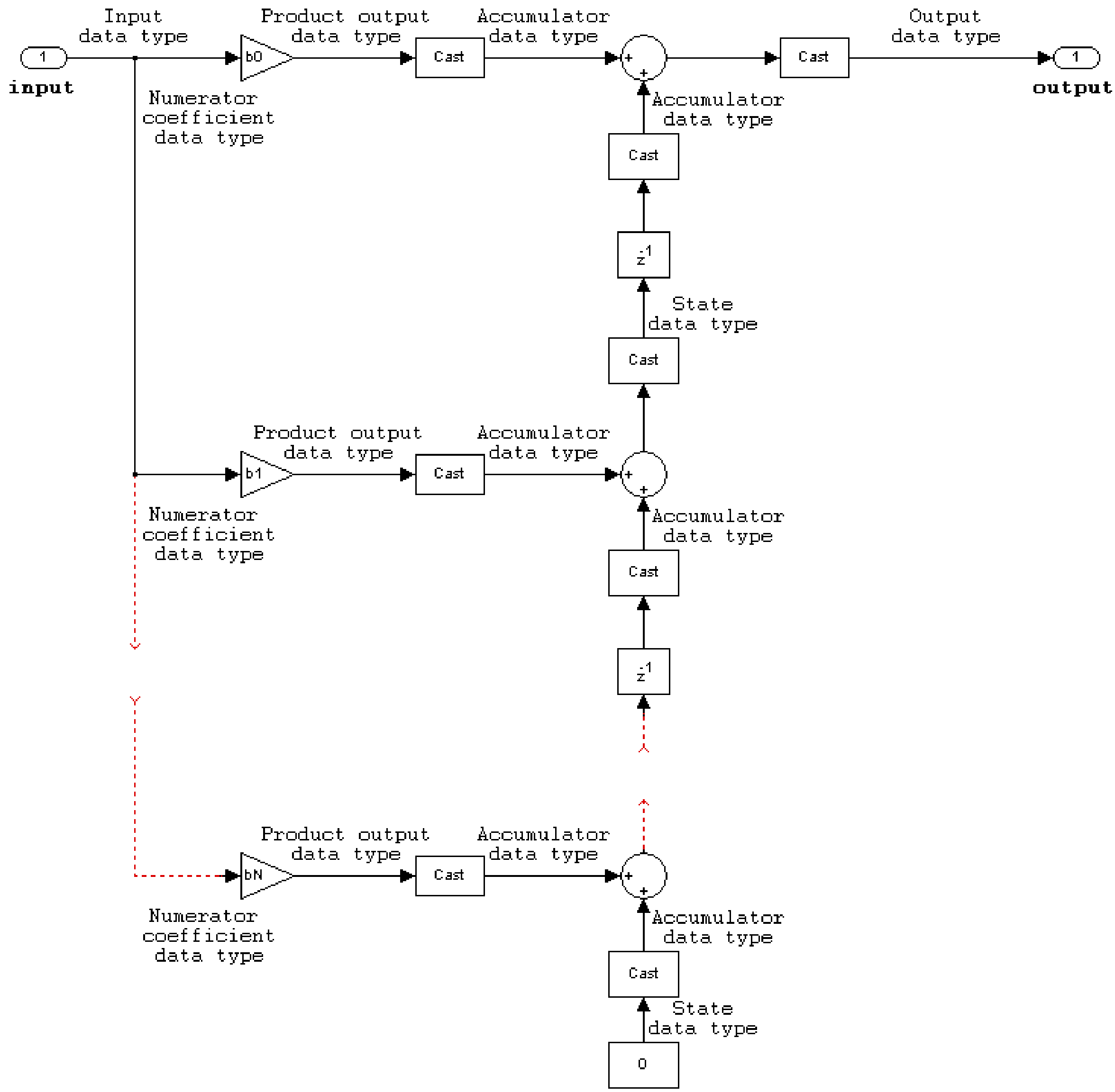
Odd Order — Type IV

FIR (all zeros) direct form transposed

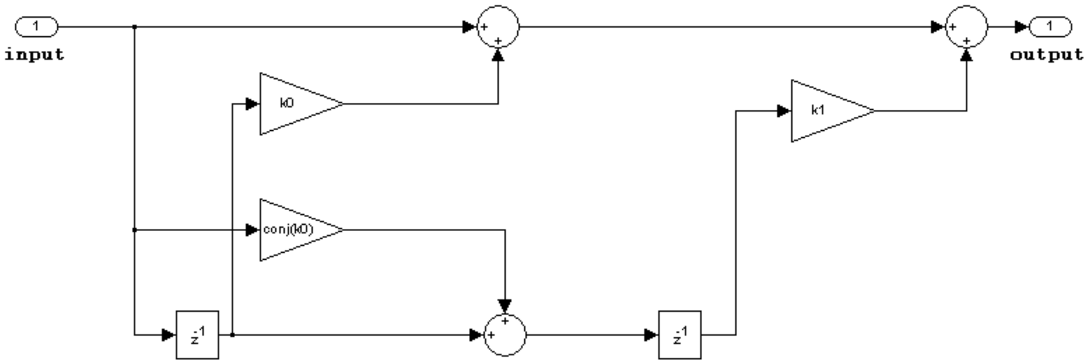


The following constraints are applicable when processing a fixed-point signal with this filter structure:

- Inputs can be real or complex.
- Coefficients can be real or complex.
- States are complex when either the inputs or the coefficients are complex.

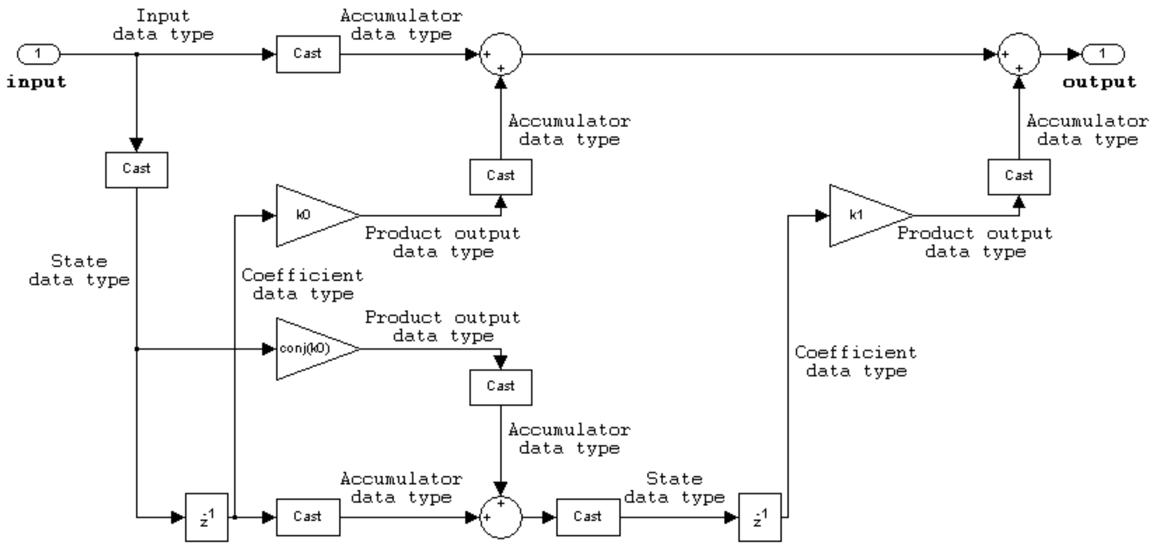

Input
data type
Product output
data type
Accumulator
data type
Output
data type
1
input
b0
Cast
Cast
1
output
Numerator
coefficient
data type
Accumulator
data type
Cast
z-1
State
data type
Cast
Product output
data type
Accumulator
data type
b1
Cast
Numerator
coefficient
data type
Accumulator
data type
Cast
z-1
Product output
data type
Accumulator
data type
bN
Cast
Numerator
coefficient
data type
Accumulator
data type
Cast
State
data type
0

FIR (all zeros) lattice MA



The following constraints are applicable when processing a fixed-point signal with this filter structure:

- Inputs and coefficients can be real or complex.
- Coefficients can be real or complex.



Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed only)
- 8-, 16-, and 32-bit signed integers

See Also

Digital Filter Design	Signal Processing Blockset
Filter Realization Wizard	Signal Processing Blockset
`dfilt`	Signal Processing Toolbox
`fdatool`	Signal Processing Toolbox
`fvtool`	Signal Processing Toolbox
`sptool`	Signal Processing Toolbox

Purpose

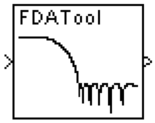
Design and implement digital FIR and IIR filters

Library

Filtering / Filter Implementations

`dsparch4`

Description



Note Use this block to design, analyze, and then efficiently implement floating-point filters. The following blocks also implement digital filters, but serve slightly different purposes:

- Digital Filter — Use to efficiently implement floating-point or fixed-point filters that you have already designed. This block provides the same exact filter implementation as the Digital Filter Design block.
- Filter Realization Wizard — Use to implement floating-point or fixed-point filters built from Sum, Gain, and Unit Delay blocks. You can either design the filter within this block, or import the coefficients of a filter that you designed elsewhere.

The Digital Filter Design block implements a digital FIR or IIR filter that you design using the Filter Design and Analysis Tool (`fdatool`) GUI. This block provides the same exact filter implementation as the Digital Filter block.

The block applies the specified filter to each channel of a discrete-time input signal, and outputs the result. The outputs of the block numerically match the outputs of the Digital Filter block, the MATLAB `filter` function, and the Filter Design Toolbox `filter` function.

The sampling frequency, Fs, that you specify in the FDATool GUI should be identical to the sampling frequency of the Digital Filter Design block's input block. When the sampling frequencies of these blocks do not match, the Digital Filter Design block returns a warning message and inherits the sampling frequency of the input block.

Valid Inputs and Corresponding Outputs

The block accepts inputs that are sample-based or frame-based vectors and matrices. The block filters each input channel independently over time, where

- Each *column* of a frame-based vector or matrix is an independent channel.
- Each *element* of a sample-based vector or matrix is an independent channel.

The output has the same dimensions and frame status as the input.

Designing the Filter

Double-click the Digital Filter Design block to open FDATool. Use FDATool to design or import a digital FIR or IIR filter. To learn how to design filters with this block and FDATool, see the following topics:

- "Digital Filter Design Block"
- `fdatool` reference page in the Signal Processing Toolbox documentation

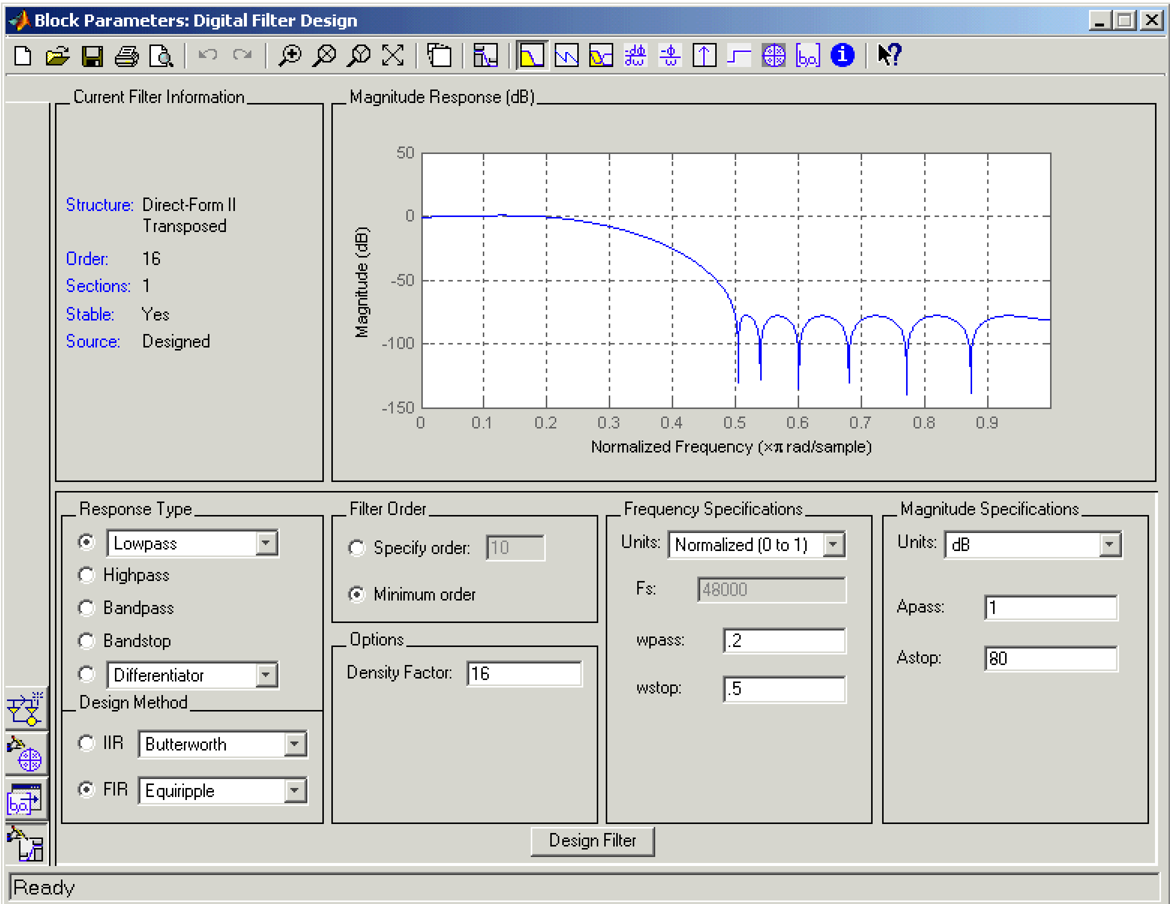
Tuning the Filter During Simulation

You can tune the filter specifications in FDATool during simulations as long as your changes do not modify the filter length or filter order. The block's filter updates as soon as you apply any filter changes in FDATool.

Examples

See the "Digital Filter Design Block" section in the *Signal Processing Blockset User's Guide*.

Dialog Box



The FDATool GUI Opened from the Digital Filter Design Block

To get the **Transform Filter** button , install the Filter Design Toolbox product. To get the **Targets** menu, install the Target Support Package™ product.

To learn how to use the FDATool GUI, see "Designing the Filter" on page 2-406.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Analog Filter Design	Signal Processing Blockset
Window Function	Signal Processing Blockset
`fdatool`	Signal Processing Toolbox
`filter`	Signal Processing Toolbox
`fvtool`	Signal Processing Toolbox
`sptool`	Signal Processing Toolbox
`filter`	Filter Design Toolbox

To learn how to use this block and FDATool, see the following:

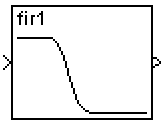
- "Filters"
- "Digital Filter Design Block"
- `fdatool` reference page in the Signal Processing Toolbox documentation

Digital FIR Filter Design (Obsolete)

Purpose Design and implement a variety of FIR filters

Library `dspobslib`

Description



Note The Digital FIR Filter Design block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the Digital Filter block.

The Digital FIR Filter Design block designs a discrete-time (digital) FIR filter in one of several different band configurations using a window method. Most of these filters are designed using the Signal Processing Toolbox `fir1` function, and are real with linear phase response. The block applies the filter to a discrete-time input using the Direct-Form II Transpose Filter (Obsolete) block.

An M-by-N sample-based matrix input is treated as $M*N$ independent channels, and an M-by-N frame-based matrix input is treated as N independent channels. In both cases, the block filters each channel independently over time, and the output has the same size and frame status as the input.

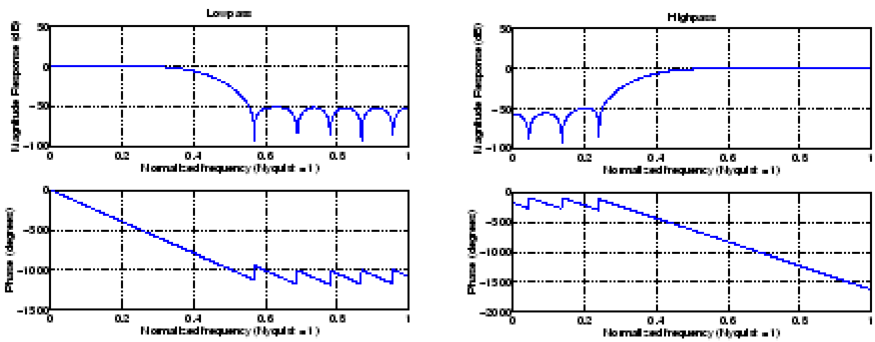
For complete details on the classical FIR filter design algorithm, see the description of the `fir1` and `fir2` functions in the Signal Processing Toolbox documentation.

Band Configurations

The band configuration for the filter is set from the **Filter type** pop-up menu. The band configuration parameters below this pop-up menu adapt appropriately to match the **Filter type** selection.

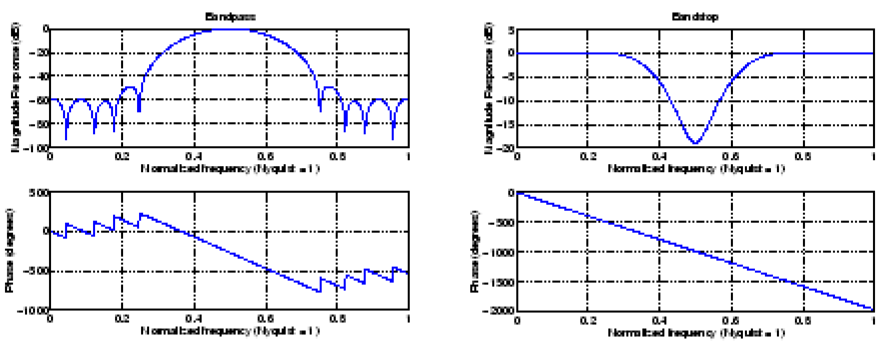
- **Lowpass** and **Highpass**

  In lowpass and highpass configurations, the **Filter order** and **Cutoff frequency** parameters specify the filter design. Frequencies are normalized to half the sample frequency. The figure below shows the frequency response of the default order-22 filter with cutoff at 0.4.



- **Bandpass** and **Bandstop**

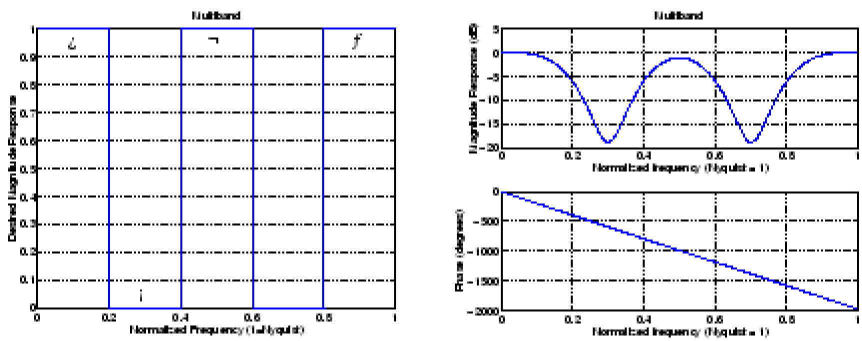
  In bandpass and bandstop configurations, the **Filter order**, **Lower cutoff frequency**, and **Upper cutoff frequency** parameters specify the filter design. Frequencies are normalized to half the sample frequency, and the actual filter order is twice the **Filter order** parameter value. The figure below shows the frequency response of the default order-22 filter with lower cutoff at 0.4, and upper cutoff at 0.6.



- **Multiband**

  In the multiband configuration, the **Filter order**, **Cutoff frequency vector**, and **Gain in the first band** parameters specify the filter design. The **Cutoff frequency vector** contains frequency points in the range 0 to 1, where 1 corresponds to half the sample frequency. Frequency points must appear in ascending order. The **Gain in the first band** parameter specifies the gain in the first band: 0 indicates a stopband, and 1 indicates a passband. Additional bands alternate between passband and stopband. The figure below shows the frequency response of the default order-22 filter with five bands, the first a passband.

  Cutoff frequency vector = `[0.2 0.4 0.6 0.8]`



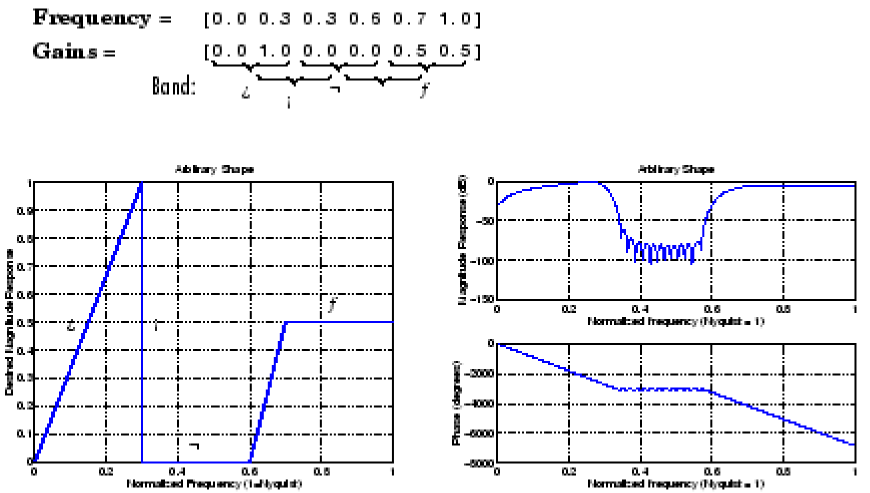
- **Arbitrary shape**

  In the arbitrary shape configuration, the **Filter order**, **Frequency vector**, and **Gains at these frequencies** parameters specify the filter design. The **Frequency vector**, `fn`, contains frequency points in the range 0 to 1 (inclusive) in ascending order, where 1 corresponds to half the sample frequency. The **Gains at these frequencies** parameter, `mn`, is a vector containing the desired magnitude response at the corresponding points in the **Frequency vector**. (Note that the specifications for the **Arbitrary shape** configuration are similar to those for the Yule-Walker IIR Filter Design block. Arbitrary-shape filters are designed using the Signal Processing Toolbox `fir2` function.)

The desired magnitude response of the design can be displayed by typing

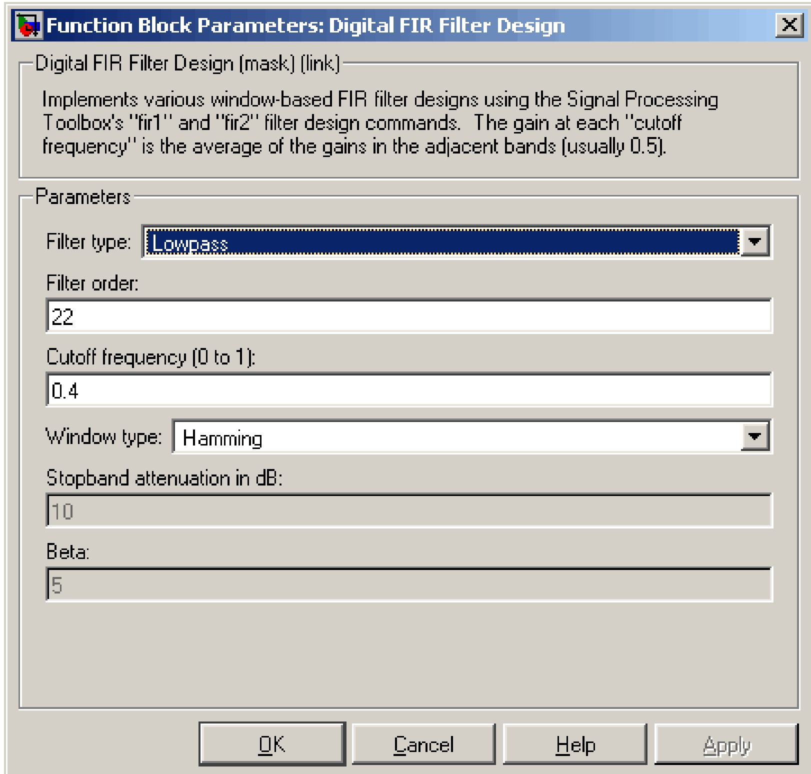
```
plot(fn,mn)
```

Duplicate frequencies can be used to specify a step in the response (such as band 2 below). The figure shows an order-100 filter with five bands.



The **Window type** parameter allows you to select from a variety of different windows. See the Window Function block reference for a complete description of the available options.

Dialog Box



The parameters displayed in the dialog box vary for different design/band combinations. Only some of the parameters listed below are visible in the dialog box at any one time.

Filter type

The type of filter to design: **Lowpass**, **Highpass**, **Bandpass**, **Bandstop**, **Multiband**, or **Arbitrary Shape**. Tunable.

Filter order

The order of the filter. The filter length is one more than this value. For the **Bandpass** and **Bandstop** configurations, the order of the final filter is twice this value.

Cutoff frequency
The normalized cutoff frequency for the **Highpass** and **Lowpass** filter configurations. A value of 1 specifies half the sample frequency. Tunable.

Lower cutoff frequency
The lower passband or stopband frequency for the **Bandpass** and **Bandstop** filter configurations. A value of 1 specifies half the sample frequency. Tunable.

Upper cutoff frequency
The upper passband or stopband frequency for the **Bandpass** and **Bandstop** filter configurations. A value of 1 specifies half the sample frequency. Tunable.

Cutoff frequency vector
A vector of ascending frequency points defining the cutoff edges for the **Multiband** filter. A value of 1 specifies half the sample frequency. Tunable.

Gain in the first band
The gain in the first band of the **Multiband** filter: 0 specifies a stopband, 1 specifies a passband. Additional bands alternate between passband and stopband. Tunable.

Frequency vector
A vector of ascending frequency points defining the frequency bands of the **Arbitrary shape** filter. The frequency range is 0 to 1 including the endpoints, where 1 corresponds to half the sample frequency. Tunable.

Gains at these frequencies
A vector containing the desired magnitude response for the **Arbitrary shape** filter at the corresponding points in the **Frequency vector**. Tunable.

Window type
The type of window to apply. See the Window Function block reference. Tunable.

Stopband ripple

The level (dB) of stopband ripple, R_s, for the **Chebyshev** window. Tunable.

Beta

The **Kaiser** window β parameter. Increasing **Beta** widens the mainlobe and decreases the amplitude of the window sidelobes in the window's frequency magnitude response. Tunable.

References

Antoniou, A. *Digital Filters: Analysis, Design, and Applications*. 2nd ed. New York, NY: McGraw-Hill, 1993.

Oppenheim, A. V. and R. W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

Proakis, J. and D. Manolakis. *Digital Signal Processing*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

Digital FIR Raised Cosine Filter Design (Obsolete)

Purpose

Design and implement raised cosine FIR filter

Library

`dspobslib`

Description

Note The Digital FIR Raised Cosine Filter Design block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the Digital Filter block.

The Digital FIR Raised Cosine Filter Design block uses the Signal Processing Toolbox `firrcos` function to design a lowpass, linear-phase, digital FIR filter with a raised cosine transition band. The block applies the filter to a discrete-time input using the Direct-Form II Transpose Filter (Obsolete) block.

An M-by-N sample-based matrix input is treated as $M*N$ independent channels, and an M-by-N frame-based matrix input is treated as N independent channels. In both cases, the block filters each channel independently over time, and the output has the same size and frame status as the input.

The frequency response of the raised cosine filter is

$$H(f) = \begin{cases} \dfrac{1}{2f_{n0}} & 0 \le |f| \le (1-R)f_{n0} \\ \left[\dfrac{1+\cos\dfrac{\pi}{2Rf_{n0}}|f| - (1-R)f_{n0}}{4f_{n0}}\right] & (1-R)f_{n0} \le |f| \le (1+R)f_{n0} \\ 0 & (1+R)f_{n0} \le |f| \le 1 \end{cases}$$

where $H(f)$ is the magnitude response at frequency f, f_{n0} is the normalized cutoff frequency (-6 dB) specified by the **Upper cutoff frequency** parameter, and R is a rolloff factor in the range `[0,1]` determining the passband-to-stopband transition width.

The **Square-root raised cosine filter** option designs a filter with magnitude response $\sqrt{H(f)}$. This is useful when the filter is part of a pair of matched filters.

When the **Design method** parameter is set to **Rolloff factor**, the secondary **Rolloff factor** parameter is enabled, and R can be directly specified. When **Design method** is set to **Transition bandwidth**, the secondary **Transition bandwidth** parameter is enabled, and the transition region bandwidth, Δf, can be specified in place of R. The transition region is centered on f_{n0} and must be sufficiently narrow to satisfy

$$0 < \left(f_{n0} \pm \frac{\Delta f}{2}\right) < 1$$

The **Upper cutoff frequency** and **Transition bandwidth** parameter values are normalized to half the sample frequency.

The **Window type** parameter allows you to apply a variety of different windows to the raised cosine filter. See the Window Function block reference for a complete description of the available options.

Algorithm

The filter output is computed by convolving the input with a truncated, delayed, windowed version of the filter's impulse response. The impulse response for the raised cosine filter is

$$h(kT_s) = \frac{1}{F_s}\operatorname{sinc}(2\pi kT_s f_{n0})\frac{\cos(2\pi RkT_s f_{n0})}{1-(4RkT_s f_{n0})^2} \quad -\infty < k < \infty$$

which has limits

$$h(0) = \frac{1}{F_s}$$

and

$$h\left(\pm\frac{1}{4Rf_{n0}}\right) = \frac{R}{2F_s}\sin\left(\frac{\pi}{2R}\right)$$

The impulse response for the square-root raised cosine filter is

$$h(kT_s) = \frac{4R\cos((1+R)2\pi kT_s f_{n0}) + \dfrac{\sin((1-R)2\pi kT_s f_{n0})}{8RkT_s f_{n0}}}{\pi F_s \sqrt{\dfrac{1}{2f_{n0}}}((8RkT_s f_{n0})^2 - 1)} \quad -\infty < k < \infty$$

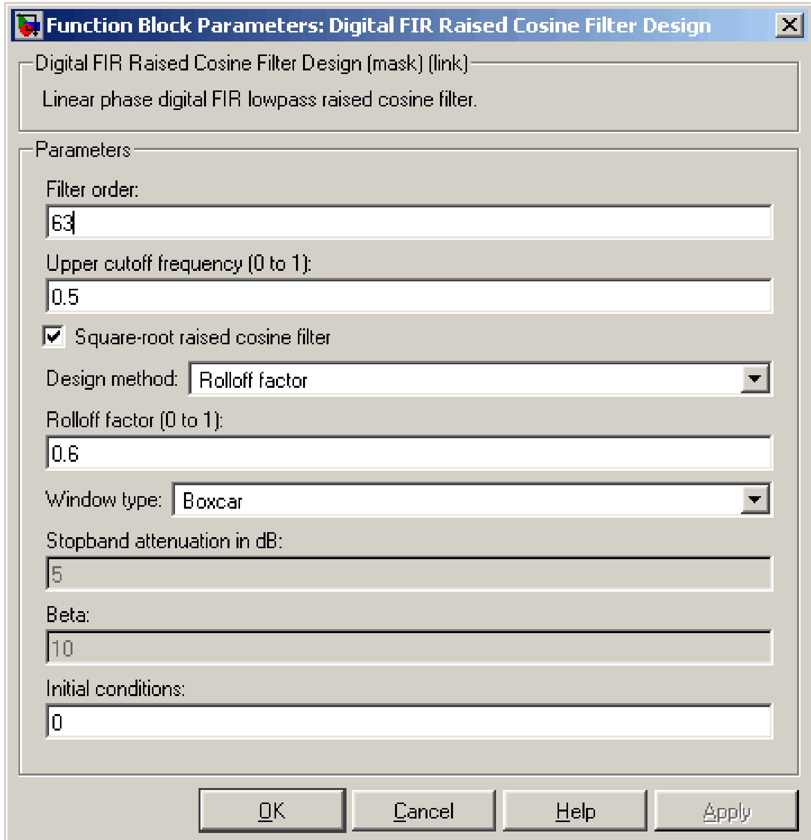
which has limits

$$h(0) = (-4R - \pi + \pi R)\frac{\sqrt{2f_{n0}}}{\pi F_s}$$

and

$$\left(\pm\frac{1}{8Rf_{n0}}\right) = \frac{\sqrt{2f_{n0}}}{2\pi F_s}\left[\pi(1+R)\sin\left(\frac{\pi(1+R)}{4R}\right) - 4R\sin\left(\frac{\pi(R-1)}{4R}\right)\right.$$
$$\left. + \pi(R-1)\cos\left(\frac{\pi(R-1)}{4R}\right)\right]$$

Dialog Box



Filter order

The order of the filter. The filter length is one more than this value.

Upper cutoff frequency

The normalized cutoff frequency, f_{n0}. A value of 1 specifies half the sample frequency. Tunable.

Square-root raised cosine filter
: Selects the square-root filter option, which designs a filter with magnitude response $\sqrt{H(f)}$. Tunable.

Design method
: The method used to design the transition region of the filter, **Rolloff factor** or **Transition bandwidth**. Tunable.

Rolloff factor
: The rolloff factor, R, enabled when **Rolloff factor** is selected in the **Design method** parameter. Tunable.

Transition bandwidth
: The transition bandwidth, Δf, enabled when **Transition bandwidth** is selected in the **Design method** parameter. Tunable.

Window type
: The type of window to apply. See the Window Function block reference. Tunable.

Stopband attenuation in dB
: The level (dB) of stopband attenuation, R_s, for the Chebyshev window. Tunable.

Beta
: The **Kaiser** window β parameter. Increasing β widens the mainlobe and decreases the amplitude of the window sidelobes in the window's frequency magnitude response. Tunable.

Initial conditions
: The filter's initial conditions, a scalar, vector, or matrix. See the Direct-Form II Transpose Filter (Obsolete) block reference for complete syntax information.

References

Proakis, J. G. *Digital Communications.* Third ed. New York, NY: McGraw-Hill, 1995.

Proakis, J. G. and M. Salehi. *Contemporary Communication Systems Using MATLAB.* Boston, MA: PWS Publishing, 1998.

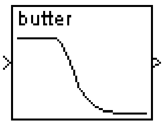
Purpose

Design and implement IIR filter

Library

`dspobslib`

Description



Note The Digital IIR Filter Design block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the Digital Filter block.

The Digital IIR Filter Design block designs a discrete-time (digital) IIR filter in a lowpass, highpass, bandpass, or bandstop configuration, and applies it to the input using the Direct-Form II Transpose Filter (Obsolete) block.

An M-by-N sample-based matrix input is treated as $M*N$ independent channels, and an M-by-N frame-based matrix input is treated as N independent channels. In both cases, the block filters each channel independently over time, and the output has the same size and frame status as the input.

The **Design method** parameter allows you to specify Butterworth, Chebyshev type I, Chebyshev type II, and elliptic filter designs. Note that for the bandpass and bandstop configurations, the actual filter length is twice the **Filter order** parameter value.

Filter Design	Description
Butterworth	The magnitude response of a Butterworth filter is maximally flat in the passband and monotonic overall.
Chebyshev type I	The magnitude response of a Chebyshev type I filter is equiripple in the passband and monotonic in the stopband.

Filter Design	Description
Chebyshev type II	The magnitude response of a Chebyshev type II filter is monotonic in the passband and equiripple in the stopband.
Elliptic	The magnitude response of an elliptic filter is equiripple in both the passband and the stopband.

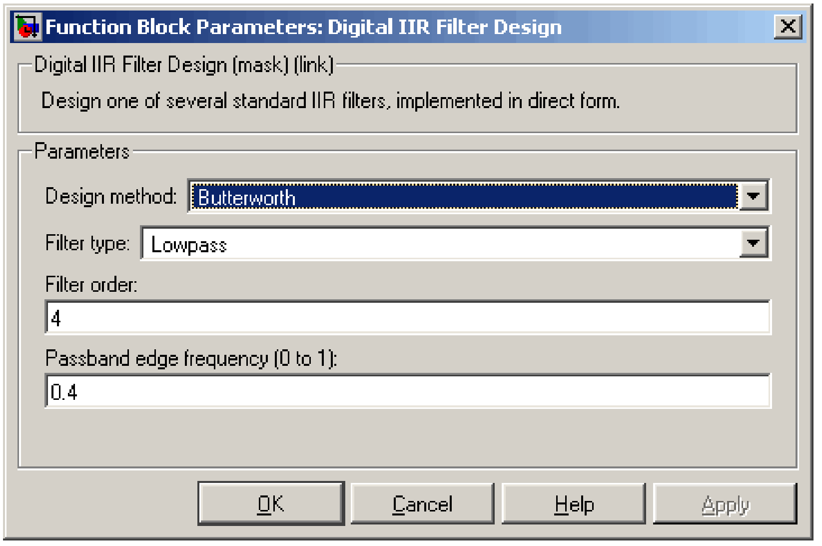
The design and band configuration of the filter are selected from the **Design method** and **Filter type** pop-up menus in the dialog box. For each combination of design method and band configuration, an appropriate set of secondary parameters is displayed.

The table below lists the available parameters for each design/band combination. For lowpass and highpass band configurations, these parameters include the passband edge frequency f_{np}, the stopband edge frequency f_{ns}, the passband ripple R_p, and the stopband attenuation R_s. For bandpass and bandstop configurations, the parameters include the lower and upper passband edge frequencies, f_{np1} and f_{np2}, the lower and upper stopband edge frequencies, f_{ns1} and f_{ns2}, the passband ripple R_p, and the stopband attenuation R_s. Frequency values are normalized to half the sample frequency, and ripple and attenuation values are in dB.

	Lowpass	Highpass	Bandpass	Bandstop
Butterworth	Order, f_{np}	Order, f_{np}	Order, f_{np1}, f_{np2}	Order, f_{np1}, f_{np2}
Chebyshev Type I	Order, f_{np}, R_p	Order, f_{np}, R_p	Order, f_{np1}, f_{np2}, $\mathrm{R_p}$	Order, f_{np1}, f_{np2}, R_p
Chebyshev Type II	Order, f_{ns}, R_s	Order, f_{ns}, R_s	Order, f_{ns1}, f_{ns2}, R_s	Order, f_{ns1}, f_{ns2}, R_s
Elliptic	Order, f_{np}, R_p, R_s	Order, f_{np}, R_p, R_s	Order, f_{np1}, f_{np2}, R_p, R_s	Order, f_{np1}, f_{np2}, R_p, R_s

The digital filters are designed using Signal Processing Toolbox software's filter design commands `butter`, `cheby1`, `cheby2`, and `ellip`.

Dialog Box



The parameters displayed in the dialog box vary for different design/band combinations. Only some of the parameters listed below are visible in the dialog box at any one time.

Design method
: The filter design method: `Butterworth`, `Chebyshev type I`, `Chebyshev type II`, or `Elliptic`. Tunable.

Filter type
: The type of filter to design: `Lowpass`, `Highpass`, `Bandpass`, or `Bandstop`. Tunable.

Filter order
: The order of the filter for lowpass and highpass configurations. For bandpass and bandstop configurations, the length of the final filter is twice this value.

Passband edge frequency
: The normalized passband edge frequency for the highpass and lowpass configurations of the Butterworth, Chebyshev type I, and elliptic designs. Tunable.

Lower passband edge frequency
: The normalized lower passband frequency for the bandpass and bandstop configurations of the Butterworth, Chebyshev type I, and elliptic designs. Tunable.

Upper passband edge frequency
: The normalized upper passband frequency for the bandpass and bandstop configurations of the Butterworth, Chebyshev type I, or elliptic designs. Tunable.

Stopband edge frequency
: The normalized stopband edge frequency for the highpass and lowpass band configurations of the Chebyshev type II design. Tunable.

Lower stopband edge frequency
: The normalized lower stopband frequency for the bandpass and bandstop configurations of the Chebyshev type II design. Tunable.

Upper stopband edge frequency
: The normalized upper stopband frequency for the bandpass and bandstop filter configurations of the Chebyshev type II design. Tunable.

Passband ripple in dB
: The passband ripple, in dB, for the Chebyshev type I and elliptic designs. Tunable.

Stopband attenuation in dB
: The stopband attenuation, in dB, for the Chebyshev type II and elliptic designs. Tunable.

References

Antoniou, A. *Digital Filters: Analysis, Design, and Applications*. 2nd ed. New York, NY: McGraw-Hill, 1993.

Oppenheim, A. V. and R. W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

Proakis, J. and D. Manolakis. *Digital Signal Processing*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

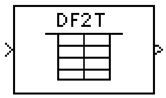
Purpose

Apply IIR filter to input

Library

`dspobslib`

Description



Direct-Form II Transpose Filter

Note This block is now just an implementation of the Digital Filter block.

Discrete Impulse

Purpose

Generate discrete impulse

Library

Signal Processing Sources

`dspsrcs4`

Description

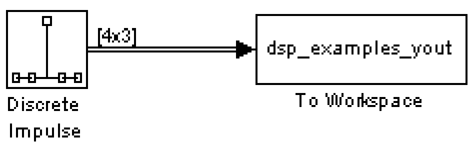
The Discrete Impulse block generates an impulse (the value `1`) at output sample `D+1`, where `D` is specified by the **Delay** parameter (`D` ≥`0`). All output samples preceding and following sample `D+1` are zero.

When `D` is a length-N vector, the block generates an M-by-N matrix output representing N distinct channels, where frame size M is specified by the **Samples per frame** parameter. The impulse for the `i`th channel appears at sample `D(i)+1`. For M=1, the output is sample based; otherwise, the output is frame based.

The **Sample time** parameter value, T_s, specifies the output signal sample period. The resulting frame period is M^*T_s.

Examples

Construct the model below.



Configure the Discrete Impulse block to generate a frame-based three-channel output of type `double`, with impulses at samples 1, 4, and 6 of channels 1, 2, and 3, respectively. Use a sample period of 0.25 and a frame size of 4. The corresponding settings should be as follows:

- **Delay** = `[0 3 5]`
- **Sample time** = `0.25`
- **Samples per frame** = `4`
- **Output data type** = `double`

Run the model and look at the output, `dsp_examples_yout`. The first few samples of each channel are shown below.

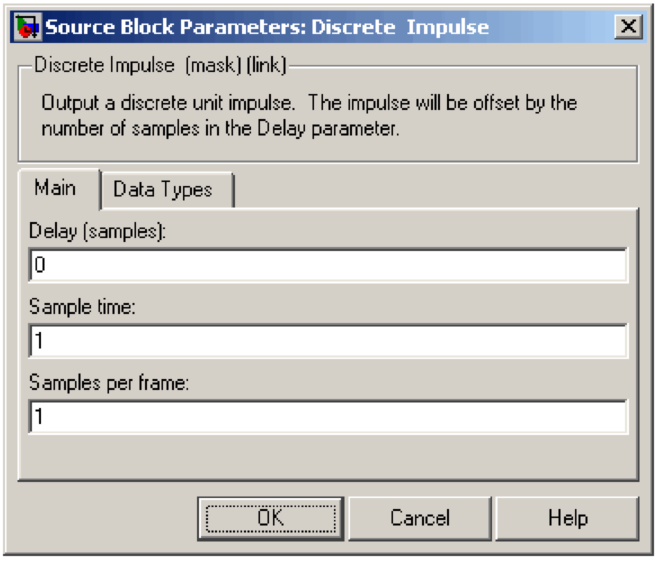
```
dsp_examples_yout(1:10,:)

ans =
     1     0     0
     0     0     0
     0     0     0
     0     1     0
     0     0     0
     0     0     1
     0     0     0
     0     0     0
     0     0     0
     0     0     0
```

The block generates an impulse at sample 1 of channel 1 (first column), at sample 4 of channel 2 (second column), and at sample 6 of channel 3 (third column).

Dialog Box

The **Main** pane of the Discrete Impulse block dialog appears as follows.



Delay

The number of zero-valued output samples, D, preceding the impulse. A length-N vector specifies an N-channel output.

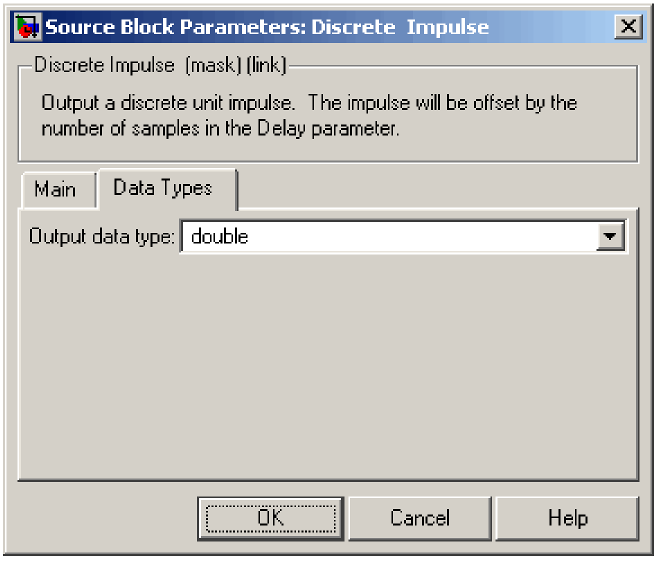
Sample time

The sample period, T_s, of the output signal. The output frame period is $M*T_s$.

Samples per frame

The number of samples, M, in each output frame. When the value of this parameter is 1, the block outputs a sample-based signal.

The **Data Types** pane of the Discrete Impulse block dialog appears as follows.



Output data type

Specify the output data type in one of the following ways:

- Choose one of the built-in data types from the list.
- Choose `Fixed-point` to specify the output data type and scaling in the **Signed**, **Word length**, **Set fraction length in output to**, and **Fraction length** parameters.
- Choose `User-defined` to specify the output data type and scaling in the **User-defined data type**, **Set fraction length in output to**, and **Fraction length** parameters.
- Choose `Inherit via back propagation` to set the output data type and scaling to match the next block downstream.

Signed

Select to output a signed fixed-point signal. Otherwise, the signal is unsigned. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

Word length

Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

User-defined data type

Specify any built-in or fixed-point data type. You can specify fixed-point data types using the following Simulink Fixed Point functions: `sfix`, `ufix`, `sint`, `uint`, `sfrac`, and `ufrac`. This parameter is only visible when you select `User-defined` for the **Output data type** parameter.

Set fraction length in output to

Specify the scaling of the fixed-point output by either of the following two methods:

- Choose `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.
- Choose `User-defined` to specify the output scaling in the **Fraction length** parameter.

This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter, or when you select `User-defined` and the specified output data type is a fixed-point data type.

Fraction length

For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Output data type** parameter and `User-defined` for the **Set fraction length in output to** parameter.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point

- Boolean
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers

See Also

Data Type Conversion	Simulink
Constant	Simulink
Multiphase Clock	Signal Processing Blockset
N-Sample Enable	Signal Processing Blockset
Signal From Workspace	Signal Processing Blockset
`impz`	Signal Processing Toolbox

Display

Purpose Show value of input

Library Signal Processing Sinks

`dspsnks4`

Description The Display block is an implementation of the Simulink Display block. See Display for more information.

Purpose

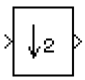
Resample input at lower rate by deleting samples

Library

Signal Operations

`dspsigops`

Description



The Downsample block decreases the sampling rate of the input by deleting samples. When the input is a frame-based M_i-by-N matrix, the block resamples each channel of the input independently. When the input is a sample-based N-D array, the Downsample block resamples the input array across time. The resample rate is K times lower than the input sample rate, where K is an integer specified by the **Downsample factor** parameter. The Downsample block resamples the input by discarding K–1 consecutive samples following each sample that is passed through to the output.

The **Sample offset** parameter delays the output samples by an integer number of sample periods, D, where $0 \leq D < (K–1)$, so that any of the K possible output phases can be selected. For example, when you downsample the sequence 1, 2, 3, ... by a factor of 4, you can select from the following four phases.

Input Sequence	Sample Offset, D	Output Sequence (K=4)
`1,2,3,...`	0	`1,5,9,13,17,21,25,29,...`
`1,2,3,...`	1	`0,2,6,10,14,18,22,26,...`
`1,2,3,...`	2	`0,3,7,11,15,19,23,27,...`
`1,2,3,...`	3	`0,4,8,12,16,20,24,28,...`

The initial zero in each of the latter three output sequences in the table is a result of the default zero **Initial condition** parameter setting for this example. See “Latency” on page 2-436 for more information on the **Initial condition** parameter.

This block supports triggered subsystems if, for **Sample-based mode**, you select `Force single-rate` and, for **Frame-based mode**, you select `Maintain input frame rate`.

Sample-Based Operation

When the input is a sample-based N-D array, the Downsample block resamples the input across time. The block downsamples the input array by discarding K–1 samples following each sample that it passes through to the output. The input and output sizes of the Downsample block are identical.

The **Sample-based mode** parameter determines how the block represents the new rate at the output. There are two available options:

- `Allow multirate`

  When you select `Allow multirate`, the sample period of the sample-based output is K times longer than the input sample period ($T_{so} = KT_{si}$). The block is therefore multirate.

- `Force single rate`

  When you select `Force single rate`, the block forces the output sample rate to match the input sample rate ($T_{so} = T_{si}$) by repeating every Kth input sample K times at the output. The block is therefore single rate. (The block's operation when you select `Force single rate` is similar to the operation of a Sample and Hold block with a repeating trigger event of period KT_{si}.)

The setting of the **Frame-based mode** parameter does not affect sample-based inputs.

Frame-Based Inputs

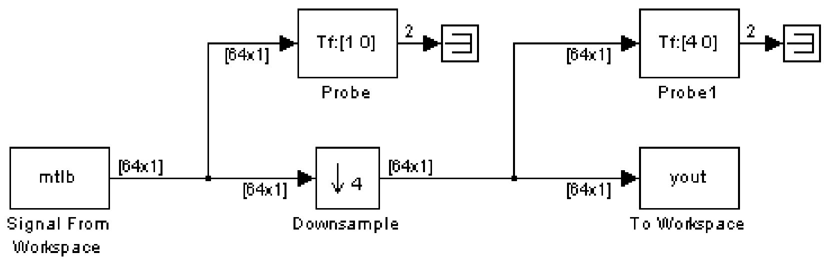
When the input is frame based, the block treats each of the N input columns as a frame containing M_i sequential time samples from an independent channel. The block downsamples each channel independently by discarding K–1 rows of the input matrix following each row that it passes through to the output.

The **Frame-based mode** parameter determines how the block adjusts the rate at the output to accommodate the reduced number of samples. There are two available options:

- `Maintain input frame size`

  The block generates the output at the slower (downsampled) rate by using a proportionally longer frame *period* at the output port than at the input port. For downsampling by a factor of K, the output frame period is K times longer than the input frame period ($T_{fo} = KT_{fi}$), but the input and output frame sizes are equal.

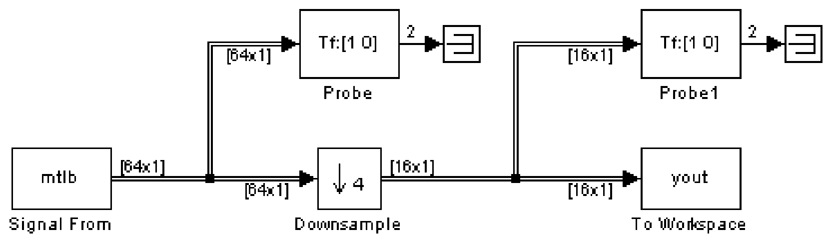
  The following model shows a single-channel input with a frame period of 1 second being downsampled by a factor of 4 to a frame period of 4 seconds. The input and output frame sizes are identical.



- `Maintain input frame rate`

  The block generates the output at the slower (downsampled) rate using a proportionally smaller frame *size* than the input. For downsampling by a factor of K, the output frame size is K times smaller than the input frame size ($M_o = M_i/K$), but the input and output frame rates are equal.

  The following model shows a single-channel input of frame size 64 being downsampled by a factor of 4 to a frame size of 16. The input and output frame rates are identical.



The setting of the **Sample-based mode** pop-up menu does not affect frame-based inputs.

Latency

The Downsample block has *zero-tasking latency* in the following cases:

- The **Downsample factor** parameter, *K*, is 1
- The block input is a frame-based signal, and the **Maintain input frame rate** parameter is selected
- The block input is a sample-based signal, and **Sample offset** parameter, *D*, is 0

Zero-tasking latency means that the block propagates input sample D+1 (received at t=0) as the first output sample, followed by input sample D+1+K, input sample D+1+2K, and so on. The **Initial condition** parameter value is not used.

In all other cases, the latency is nonzero:

- For sample-based input, the latency is one sample.
- For frame-based input, the latency is one frame.

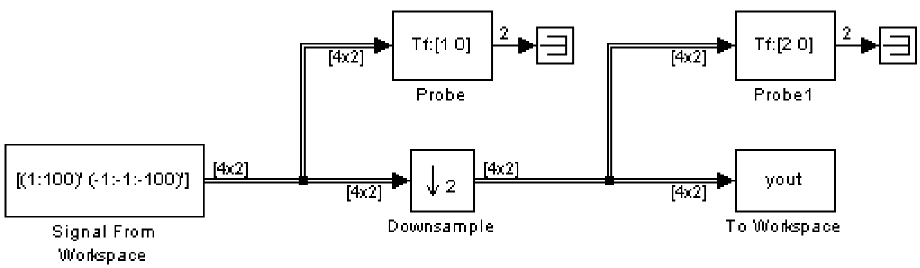
In all cases of *one-sample latency*, the initial condition for each channel appears as the first output sample. Input sample D+1 appears as the second output sample for each channel, followed by input sample D+1+K, input sample D+1+2K, and so on. The **Initial condition** parameter can be an array of the same size as the input, or a scalar to be applied to all signal channels.

In all cases of *one-frame latency*, the M_i rows of the initial condition matrix appear in sequence as the first M_i output rows. Input sample D+1 (i.e, row D+1 of the input matrix) appears in the output as sample M_i+1, followed by input sample D+1+K, input sample D+1+2K, and so on. The **Initial condition** value can be an M_i-by-N matrix containing one value for each channel, or a scalar to be repeated across all elements of the M_i-by-N matrix. See the following example for an illustration of this case.

Note For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

Examples

Construct this frame-based model.



Adjust the block parameters as follows:

- Configure the Signal From Workspace block to generate a two-channel signal with frame size of 4 and sample period of 0.25 second. This represents an output frame period of 1 second (0.25*4). The first channel should contain the positive ramp signal 1, 2, ..., 100, and the second channel should contain the negative ramp signal –1, –2, ..., –100. The settings are:
  - **Signal** = `[(1:100)' (-1:-1:-100)']`
  - **Sample time** = `0.25`

- **Samples per frame** = `4`

- Configure the Downsample block to downsample the two-channel input by decreasing the output frame rate by a factor of 2 relative to the input frame rate. Set a sample offset of 1, and a 4-by-2 initial condition matrix of

$$\begin{bmatrix} 11 & -11 \\ 12 & -12 \\ 13 & -13 \\ 14 & -14 \end{bmatrix}$$

  - **Downsample factor** = `2`
  - **Sample offset** = `1`
  - **Initial condition** = `[11 -11;12 -12;13 -13;14 -14]`
  - **Frame-based mode** = `Maintain input frame size`

- Configure the Probe blocks by selecting only the **Probe sample time** check box.

This model is multirate because there are at least two distinct frame rates, as shown by the two Probe blocks. To run this model in the Simulink multitasking mode, open the Configuration Parameters dialog box. From the list on the left side of the dialog box, click **Solver**. From the **Type** list, select `Fixed-step`, and from the **Solver** list, select `Discrete (no continuous states)`. From the **Tasking mode for periodic sample times** list, select `MultiTasking`. Additionally, set the **Stop time** parameter to `30`.

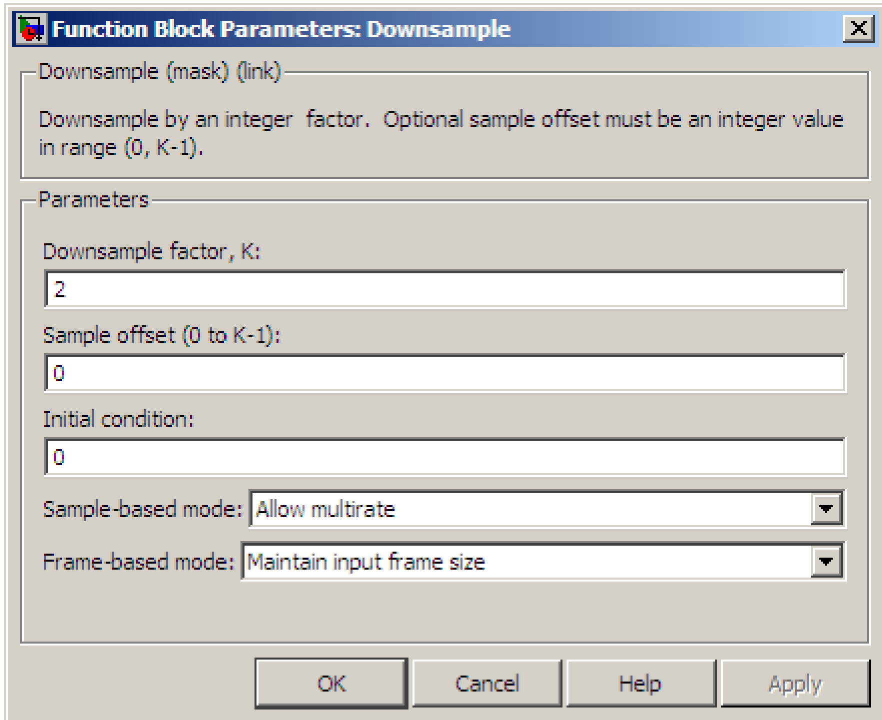
Run the model and look at the output, `yout`. The first few samples of each channel are shown:

```
yout =
    11    -11
    12    -12
    13    -13
    14    -14
```

```
 2    -2
 4    -4
 6    -6
 8    -8
10   -10
12   -12
14   -14
```

Since you ran this frame-based multirate model in multitasking mode, the first row of the initial condition matrix appears as the first output sample, followed by the other three initial condition rows. The second row of the first input matrix (row D+1, where D is the **Sample offset**) appears in the output as sample 5 (sample M_i+1, where M_i is the input frame size).

Dialog Box



Downsample factor

The integer factor, *K*, by which to decrease the input sample rate.

Sample offset

The sample offset, *D*, which must be an integer in the range [0, *K*–1].

Initial condition

The value with which the block is initialized for cases of nonzero latency; a scalar value or an array of the same size as the input.

Sample-based mode

The method by which to implement downsampling for sample-based inputs: `Allow multirate` (decrease the output sample rate), or `Force single-rate` (force the output sample rate to match the input sample rate by repeating every Kth input sample K times at the output).

Frame-based mode

The method by which to implement downsampling for frame-based inputs: `Maintain input frame size` (decrease the frame rate), or `Maintain input frame rate` (decrease the frame size).

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

FIR Decimation	Signal Processing Blockset
FIR Rate Conversion	Signal Processing Blockset
Repeat	Signal Processing Blockset
Sample and Hold	Signal Processing Blockset
Upsample	Signal Processing Blockset

DSP Constant (Obsolete)

Purpose

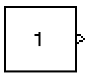
Generate discrete- or continuous-time constant signal

Library

Signal Processing Sources

`dspobslib`

Description



Note The DSP Constant block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the Constant block.

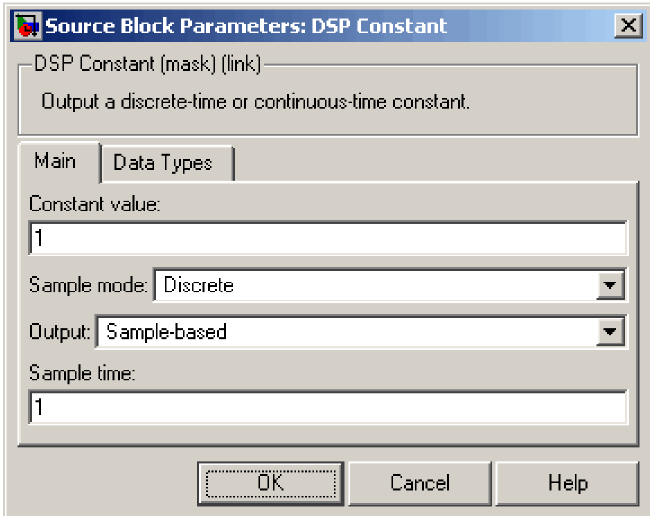
The DSP Constant block generates a signal whose value remains constant throughout the simulation. The **Constant value** parameter specifies the constant to output, and can be any valid MATLAB expression that evaluates to a scalar, vector, or matrix.

When **Sample mode** is set to `Continuous`, the output is a continuous-time signal. When **Sample mode** is set to `Discrete`, the **Sample time** parameter is visible, and the signal has the discrete output period specified by the **Sample time** parameter.

You can set the output signal to `Frame-based`, `Sample-based`, or `Sample-based (interpret vectors as 1-D)` with the **Output** parameter.

Dialog Box

The **Main** pane of the DSP Constant block dialog box appears as follows.



Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters During Simulation" in the online Simulink documentation for details.

Constant value

Specify the constant to generate. This parameter is tunable; values entered here can be tuned, but their dimensions must remain fixed.

When you specify any data type information in this field, it is overridden by the value of the **Output data type** parameter in the **Data Types** pane, unless you select `Inherit from 'Constant value'`.

Sample mode

Specify the sample mode of the output, `Discrete` for a discrete-time signal or `Continuous` for a continuous-time signal.

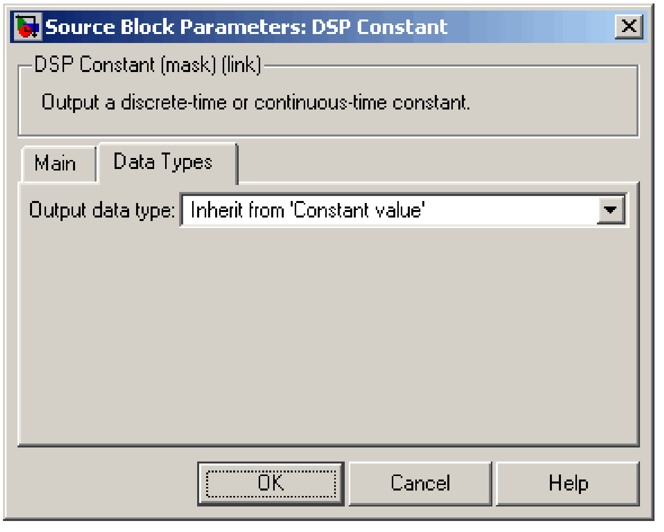
Output

Specify whether the output is `Sample-based (interpret vectors as 1-D)`, `Sample-based`, or `Frame-based`. When you select `Sample-based` and the output is a vector, its dimension is constrained to match the **Constant value** dimension (row or column). When you select `Sample-based (interpret vectors as 1-D)`, however, the output has no specified dimensionality.

Sample time

Specify the discrete sample period for sample-based outputs. When you select `Frame-based` for the **Output** parameter, this parameter is named **Frame period**, and is the discrete frame period for the frame-based output. This parameter is only visible when you select `Discrete` for the **Sample mode** parameter.

The **Data Types** pane of the DSP Constant block dialog box appears as follows.



Output data type

Specify the output data type in one of the following ways:

- Choose one of the built-in data types from the list.
- Choose `Fixed-point` to specify the output data type and scaling in the **Signed**, **Word length**, **Set fraction length in output to**, and **Fraction length** parameters.
- Choose `User-defined` to specify the output data type and scaling in the **User-defined data type**, **Set fraction length in output to,** and **Fraction length** parameters.
- Choose `Inherit from 'Constant value'` to set the output data type and scaling to match the values of the **Constant value** parameter in the **Main** pane.
- Choose `Inherit via back propagation` to set the output data type and scaling to match the following block.

The value of this parameter overrides any data type information specified in the **Constant value** parameter in the **Main** pane, except when you select `Inherit from 'Constant value'`.

Signed
Select to output a signed fixed-point signal. Otherwise, the signal is unsigned. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

Word length
Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

User-defined data type
Specify any built-in or fixed-point data type. You can specify fixed-point data types using the following Simulink Fixed Point functions: `sfix`, `ufix`, `sint`, `uint`, `sfrac`, and `ufrac`. This parameter is only visible when you select `User-defined` for the **Output data type** parameter.

Set fraction length in output to
Specify the scaling of the fixed-point output by either of the following two methods:

- Choose `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.
- Choose `User-defined` to specify the output scaling in the **Fraction length** parameter.

This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter, or when you select `User-defined` and the specified output data type is a fixed-point data type.

Fraction length

For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Output data type** parameter and `User-defined` for the **Set fraction length in output to** parameter.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point
- Boolean
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers

See Also

Constant	Simulink
Signal From Workspace	Signal Processing Blockset

Purpose

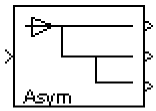
Discrete wavelet transform (DWT) of input or decompose signals into subbands with smaller bandwidths and slower sample rates

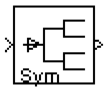
Library

Transforms

`dspxfrm3`

Description





The DWT block is the same as the Dyadic Analysis Filter Bank block in the Multirate Filters library, but with different default settings. See the Dyadic Analysis Filter Bank block reference page for more information.

Purpose

Decompose signals into subbands with smaller bandwidths and slower sample rates or compute discrete wavelet transform (DWT)

Library

Filtering / Multirate Filters

`dspmlti4`

Description

Note This block always interprets input signals as frames. The frame-size of the input signal must be a multiple of 2^n, where `n` is the value of the **Number of levels** parameter. The block decomposes the input signal into either n+1 or 2^n subbands. To decompose sample- or frame-based signals of different sizes, use the Two-Channel Analysis Subband Filter block. (You can connect multiple copies of the Two-Channel Analysis Subband Filter block to create a multilevel dyadic analysis filter bank.)

You can configure this block to compute the Discrete Wavelet Transform (DWT) or decompose a broadband signal into a collection of subbands with smaller bandwidths and slower sample rates. The block uses a series of highpass and lowpass FIR filters to repeatedly divide the input frequency range, as illustrated in the figure "n-Level Asymmetric Dyadic Analysis Filter Bank".

You can specify the filter bank's highpass and lowpass filters by providing vectors of filter coefficients. You can do so directly on the block mask, or, if you have a Wavelet Toolbox™ license, you can specify wavelet-based filters by selecting a wavelet from the **Filter** parameter. You must set the filter bank structure to asymmetric or symmetric, and specify the number of levels in the filter bank.

For the same input, the DWT configuration of this block does not produce the same results as the Wavelet Toolbox `dwt` function. Because the blockset is designed for real-time implementation and the toolbox is designed for analysis, the products handle boundary conditions and filter states differently. To make the output of the `dwt` function match the DWT output of this block, complete the following steps:

1. Set the boundary condition of the `dwt` function to zero-padding. To do so, type `dwtmode('zpd')` at the MATLAB command line.

2. To match the latency of the block (implemented using FIR filters), add zeros to the input of the `dwt` function. The number of zeros you add must be equal to the half-length of the filter.

Input Requirements

- Input must be a vector or matrix.
- The input frame size must be a multiple of 2^n, where n is the number of filter bank levels. For example, a frame size of 16 would be appropriate for a three-level tree (16 is a multiple of 2^3).
- The block always treats input signals as frames and operates along the columns.

For an illustration of why the above input requirements exist, see the figure Outputs of a 3-Level Asymmetric Dyadic Analysis Filter Bank on page 2-450.

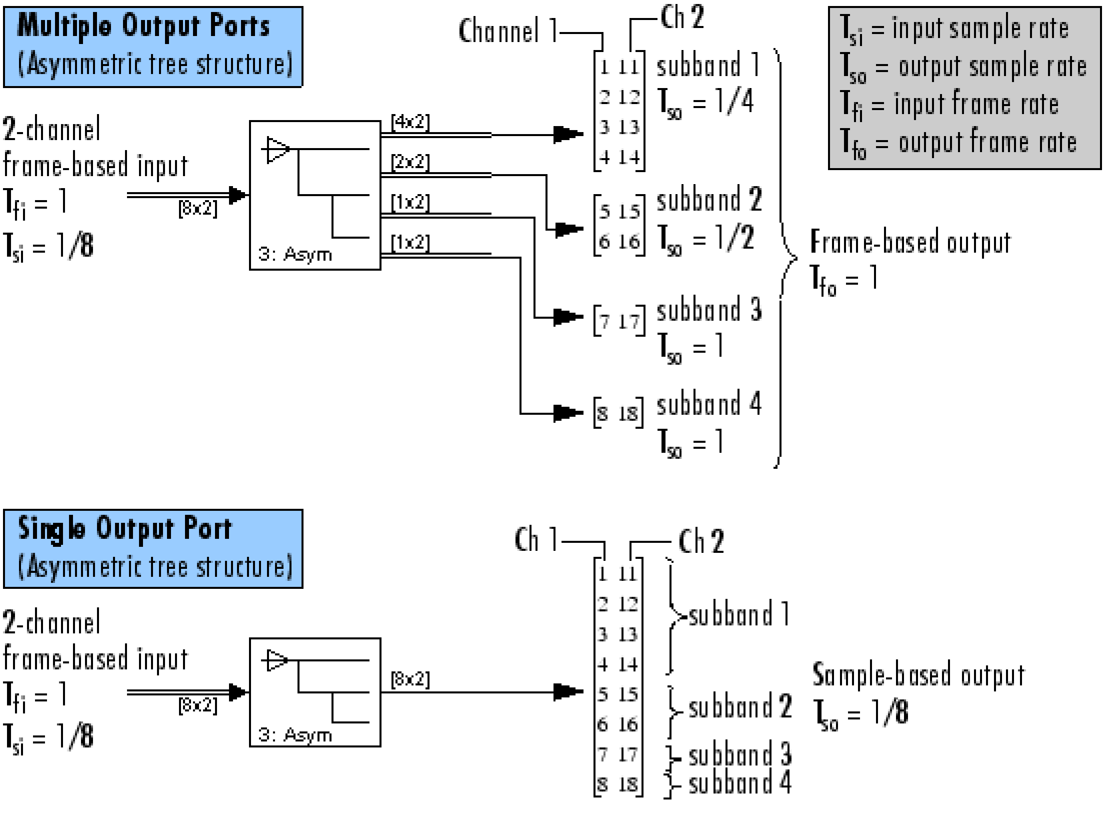
Output Characteristics

The output characteristics vary depending on the block's parameter settings, as summarized in the following list and figure:

- **Number of levels** parameter set to n
- **Tree structure** parameter setting:
  - `Asymmetric` — Block produces n+1 output subbands
  - `Symmetric` — Block produces 2^n output subbands
- **Output** parameter setting can be `Multiple ports` or `Single port`. When you set the **Output** parameter to `Single port`, the block outputs one vector or matrix of concatenated subbands. The following figure illustrates the difference between the two settings for a 3-level asymmetric dyadic analysis filter bank. For an explanation of the illustrated output characteristics, see the table Output

Characteristics for an n-Level Dyadic Analysis Filter Bank on page 2-451.

For more information about the filter bank levels and structures, see "Dyadic Analysis Filter Banks".



Outputs of a 3-Level Asymmetric Dyadic Analysis Filter Bank

The following table summarizes the different output characteristics of the block when it is set to output from single or multiple ports.

Output Characteristics for an n-Level Dyadic Analysis Filter Bank

	Single Output Port	Multiple Output Ports
Output Description	Block concatenates all the subbands into one vector or matrix, and outputs the concatenated subbands from a single output port. Each output column contains subbands of the corresponding input channel.	Block outputs each subband from a separate output port. The topmost port outputs the subband with the highest frequencies. Each output column contains a subband for the corresponding input channel.
Output Frame Status	Sample based	Frame based
Output Frame Rate	*Not applicable*	Same as input frame rate (However, the output frame sizes can vary, so the output sample rates can vary.)
Output Dimensions (Frame Size)	Same number of rows and columns as the input.	The output has the same number of columns as the input. The number of output rows is the output frame size. For an input with frame size M_i output y_k has frame size $M_{o,k}$: • `Symmetric` — All outputs have the frame size, M_i / 2^n. • `Asymmetric` — The frame size of each output (except the last) is half that of the output from the previous level. The outputs from the last two output ports have the same frame size since they

Output Characteristics for an n-Level Dyadic Analysis Filter Bank (Continued)

	Single Output Port	Multiple Output Ports
		originate from the same level in the filter bank. $$M_{o,k} = \begin{cases} M_i / 2^k & (1 \le k \le n) \\ M_i / 2^n & (k = n+1) \end{cases}$$
Output Sample Rate	Same as input sample rate.	Though the outputs have the same frame rate as the input, they have different frame sizes than the input. Thus, the output sample rates, F_{so}, k, are different from the input sample rate, F_{si}: • `Symmetric` — All outputs have the sample rate F_{si} / 2^n. • `Asymmetric` — $$F_{so,k} = \begin{cases} F_{si} / 2^k & (1 \le k \le n) \\ F_{si} / 2^n & (k = n+1) \end{cases}$$

Filter Bank Filters

You must specify the highpass and lowpass filters in the filter bank by setting the **Filter** parameter to one of the following options:

- `User defined` — Allows you to explicitly specify the filters with two vectors of filter coefficients in the **Lowpass FIR filter coefficients** and **Highpass FIR filter coefficients** parameters. The block uses the same lowpass and highpass filters throughout the filter bank. The two filters should be halfband filters, where each filter passes the frequency band that the other filter stops.

- Wavelet such as `Biorthogonal` or `Daubechies` — The block uses the specified wavelet to construct the lowpass and highpass filters using the Wavelet Toolbox `wfilters` function. Depending on the wavelet, the block might enable either the **Wavelet order** or **Filter order [synthesis / analysis]** parameter. (The latter parameter allows you to specify different wavelet orders for the analysis and synthesis filter stages.) You must have a Wavelet Toolbox license to use wavelets.

Specifying Filters with the Filter Parameter and Related Parameters

Filter	Sample Setting for Related Filter Specification Parameters	Corresponding Wavelet Function Syntax
User-defined	Filters based on Daubechies wavelets with wavelet order 3: • **Lowpass FIR filter coefficients =** `[0.0352 -0.0854 -0.1350 0.4599 0.8069 0.3327]` • **Highpass FIR filter coefficients =** `[-0.3327 0.8069 -0.4599 -0.1350 0.0854 0.0352]`	None
Haar	None	`wfilters('haar')`
Daubechies	**Wavelet order =** 4	`wfilters('db4')`
Symlets	**Wavelet order =** 3	`wfilters('sym3')`
Coiflets	**Wavelet order =** 1	`wfilters('coif1')`
Biorthogonal	**Filter order [synthesis / analysis] =** `[3/1]`	`wfilters('bior3.1')`
Reverse Biorthogonal	**Filter order [synthesis / analysis] =** `[3/1]`	`wfilters('rbio3.1')`
Discrete Meyer	None	`wfilters('dmey')`

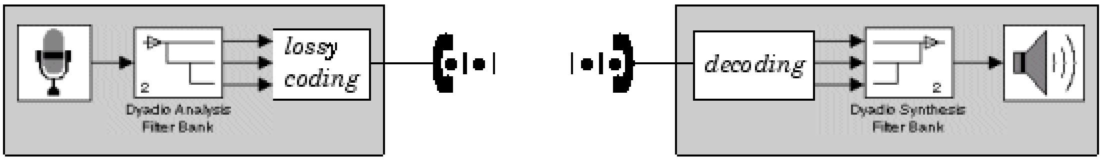
Dyadic Analysis Filter Bank

Examples

Wavelets

The primary application for dyadic analysis filter banks and dyadic synthesis filter banks is coding for data compression using wavelets.

At the transmitting end, the output of the dyadic analysis filter bank is fed to a lossy compression scheme, which typically assigns the number of bits for each filter bank output in proportion to the relative energy in that frequency band. This represents the more powerful signal components by a greater number of bits than the less powerful signal components.



At the receiving end, the transmission is decoded and fed to a dyadic synthesis filter bank to reconstruct the original signal. The filter coefficients of the complementary analysis and synthesis stages are designed to cancel aliasing introduced by the filtering and resampling.

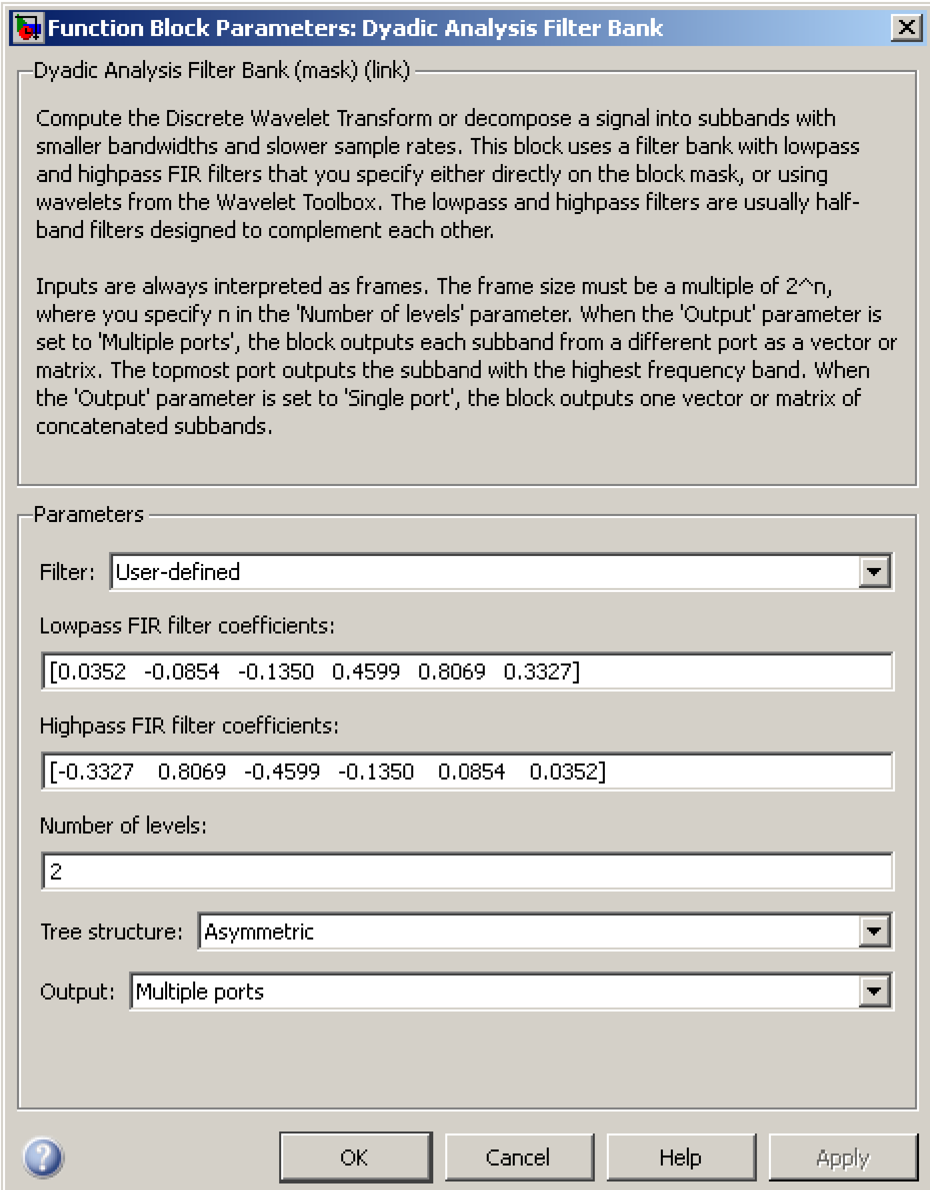
See “Calculating the Channel Latencies Required for Wavelet Reconstruction” for an example using the Dyadic Analysis and Dyadic Synthesis Filter Bank blocks.

Demos

See the floating-point frame-based version of the Signal Processing Blockset Wavelet Reconstruction and Noise Reduction demo, which uses the Dyadic Analysis Filter Bank and Dyadic Synthesis Filter Bank blocks.

Dialog Box

The parameters displayed in the block dialog vary depending on the setting of the **Filter** parameter. Only some of the parameters described below are visible in the dialog box at any one time.


Function Block Parameters: Dyadic Analysis Filter Bank
Dyadic Analysis Filter Bank (mask) (link)
Compute the Discrete Wavelet Transform or decompose a signal into subbands with smaller bandwidths and slower sample rates. This block uses a filter bank with lowpass and highpass FIR filters that you specify either directly on the block mask, or using wavelets from the Wavelet Toolbox. The lowpass and highpass filters are usually half-band filters designed to complement each other.
Inputs are always interpreted as frames. The frame size must be a multiple of 2^n, where you specify n in the 'Number of levels' parameter. When the 'Output' parameter is set to 'Multiple ports', the block outputs each subband from a different port as a vector or matrix. The topmost port outputs the subband with the highest frequency band. When the 'Output' parameter is set to 'Single port', the block outputs one vector or matrix of concatenated subbands.
Parameters
Filter: User-defined
Lowpass FIR filter coefficients:
[0.0352 -0.0854 -0.1350 0.4599 0.8069 0.3327]
Highpass FIR filter coefficients:
[-0.3327 0.8069 -0.4599 -0.1350 0.0854 0.0352]
Number of levels:
2
Tree structure: Asymmetric
Output: Multiple ports
OK
Cancel
Help
Apply

Filter

The type of filter used to determine the high- and low-pass FIR filters in the filter bank:

Select `User defined` to explicitly specify the filter coefficients in the **Lowpass FIR filter coefficients** and **Highpass FIR filter coefficients** parameters.

Select a wavelet such as `Biorthogonal` or `Daubechies` to specify a wavelet-based filter. The block uses the Wavelet Toolbox `wfilters` function to construct the filters. Extra parameters such as **Wavelet order** or **Filter order [synthesis / analysis]** might become enabled. For a list of the supported wavelets, see Specifying Filters with the Filter Parameter and Related Parameters on page 2-453.

Lowpass FIR filter coefficients

A vector of filter coefficients (descending powers of *z*) that specifies coefficients used by all the lowpass filters in the filter bank. This parameter is enabled when you set **Filter** to `User defined`. The lowpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the **Highpass FIR filter coefficients** parameter. The default values of this parameter specify a filter based on a Daubechies wavelet with wavelet order `3`.

Highpass FIR filter coefficients

A vector of filter coefficients (descending powers of *z*) that specifies coefficients used by all the highpass filters in the filter bank. This parameter is enabled when you set **Filter** to `User defined`. The highpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the **Lowpass FIR filter coefficients** parameter. The default values of this parameter specify a filter based on a Daubechies wavelet with wavelet order `3`.

Wavelet order

The order of the wavelet selected in the **Filter** parameter. This parameter is enabled only when you set **Filter** to certain types of wavelets, as shown in the Specifying Filters with the Filter Parameter and Related Parameters on page 2-453 table.

Filter order [synthesis / analysis]

The order of the wavelet for the synthesis and analysis filter stages. For example, when you set the **Filter** parameter to `Biorthogonal` and set the **Filter order [synthesis / analysis]** parameter to `[2 / 6]`, the block calls the `wfilters` function with input argument `'bior2.6'`. This parameter is enabled only when you set **Filter** to certain types of wavelets, as shown in Specifying Filters with the Filter Parameter and Related Parameters on page 2-453.

Number of levels

The number of filter bank levels. An n-level asymmetric structure has n+1 outputs, and an n-level symmetric structure has 2^n outputs, as shown in the figures “n-Level Asymmetric Dyadic Analysis Filter Bank” and “n-Level Symmetric Dyadic Analysis Filter Bank”. The block’s icon changes depending on the value of this parameter.

The default setting of this parameter is `2`.

Tree structure

The structure of the filter bank: `Asymmetric`, or `Symmetric`. See the figures “n-Level Asymmetric Dyadic Analysis Filter Bank” and “n-Level Symmetric Dyadic Analysis Filter Bank”.

The default setting of this parameter is `Asymmetric` for the Dyadic Analysis Filter Bank block, and `Symmetric` for the DWT block.

Output

Set to `Multiple ports` to output each output subband on a separate port (the topmost port outputs the subband with the highest frequency band). Set to `Single port` to concatenate the subbands into one vector or matrix and output the concatenated subbands on a single port. For more information, see "Output Characteristics" on page 2-449.

The default setting of this parameter is `Multiple ports` for the Dyadic Analysis Filter Bank block, and `Single port` for the DWT block.

References

Fliege, N. J. *Multirate Digital Signal Processing: Multirate Systems, Filter Banks, Wavelets*. West Sussex, England: John Wiley & Sons, 1994.

Strang, G. and T. Nguyen. *Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1996.

Vaidyanathan, P. P. *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice Hall, 1993.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point
Output	• Double-precision floating point • Single-precision floating point

See Also

DWT	Signal Processing Blockset
Dyadic Synthesis Filter Bank	Signal Processing Blockset
Two-Channel Analysis Subband Filter	Signal Processing Blockset

Purpose

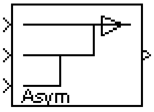
Reconstruct signals from subbands with smaller bandwidths and slower sample rates or compute inverse discrete wavelet transform (IDWT)

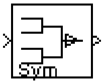
Library

Filtering / Multirate Filters

`dspmlti4`

Description





Note This block always outputs frame-based signals, and its inputs must be of certain sizes. To get sample-based outputs or to use input subbands that do not fit the criteria of this block, use the Two-Channel Synthesis Subband Filter block. (You can connect multiple copies of the Two-Channel Synthesis Subband Filter block to create a multilevel dyadic synthesis filter bank.)

You can configure this block to compute the inverse discrete wavelet transform (IDWT) or reconstruct a signal from subbands with smaller bandwidths and slower sample rates. When the block computes the inverse discrete wavelet transform (IDWT) of the input, the output is a frame-based signal with the same dimensions as the input. Each column of the output is the IDWT of the corresponding input column. When reconstructing a signal, the block uses a series of highpass and lowpass FIR filters to reconstruct the signal from the input subbands, as illustrated in the figure "n-Level Asymmetric Dyadic Synthesis Filter Bank". The reconstructed signal has a wider bandwidth and faster sample rate than the input subbands.

You can specify the filter bank's highpass and lowpass filters by providing vectors of filter coefficients. You can do so directly on the block mask, or, if you have a Wavelet Toolbox license, you can specify wavelet-based filters by selecting a wavelet from the **Filter** parameter. You must set the filter bank structure to asymmetric or symmetric, and specify the number of levels in the filter bank.

When you set the **Input** parameter to `Multiple ports`, you must provide each subband to the block through a different input port as a vector or matrix. You should input the highest frequency band through the topmost port. When you set the **Input** parameter to `Single port`, the block input must be a vector or matrix of concatenated subbands.

Note To use a dyadic synthesis filter bank to perfectly reconstruct the output of a dyadic analysis filter bank, the number of levels and tree structures of both filter banks *must* be the same. In addition, the filters in the synthesis filter bank *must* be designed to perfectly reconstruct the outputs of the analysis filter bank. Otherwise, the reconstruction is not perfect.

This block automatically computes wavelet-based perfect reconstruction filters when the wavelet selection in the **Filter** parameter of this block is the *same* as the **Filter** parameter setting of the corresponding Dyadic Analysis Filter Bank block. The use of wavelets requires a Wavelet Toolbox license. To learn how to design your own perfect reconstruction filters, see "References" on page 2-469.

Input Requirements

The inputs to this block are usually the outputs of a Dyadic Analysis Filter Bank block. Since the Dyadic Analysis Filter Bank block can output from either a single port or multiple ports, the Dyadic Synthesis Filter Bank block accepts inputs to either a single port or multiple ports.

The **Input** parameter sets whether the block accepts inputs from a single port or multiple ports, and thus determines the input requirements, as summarized in the following lists and figure.

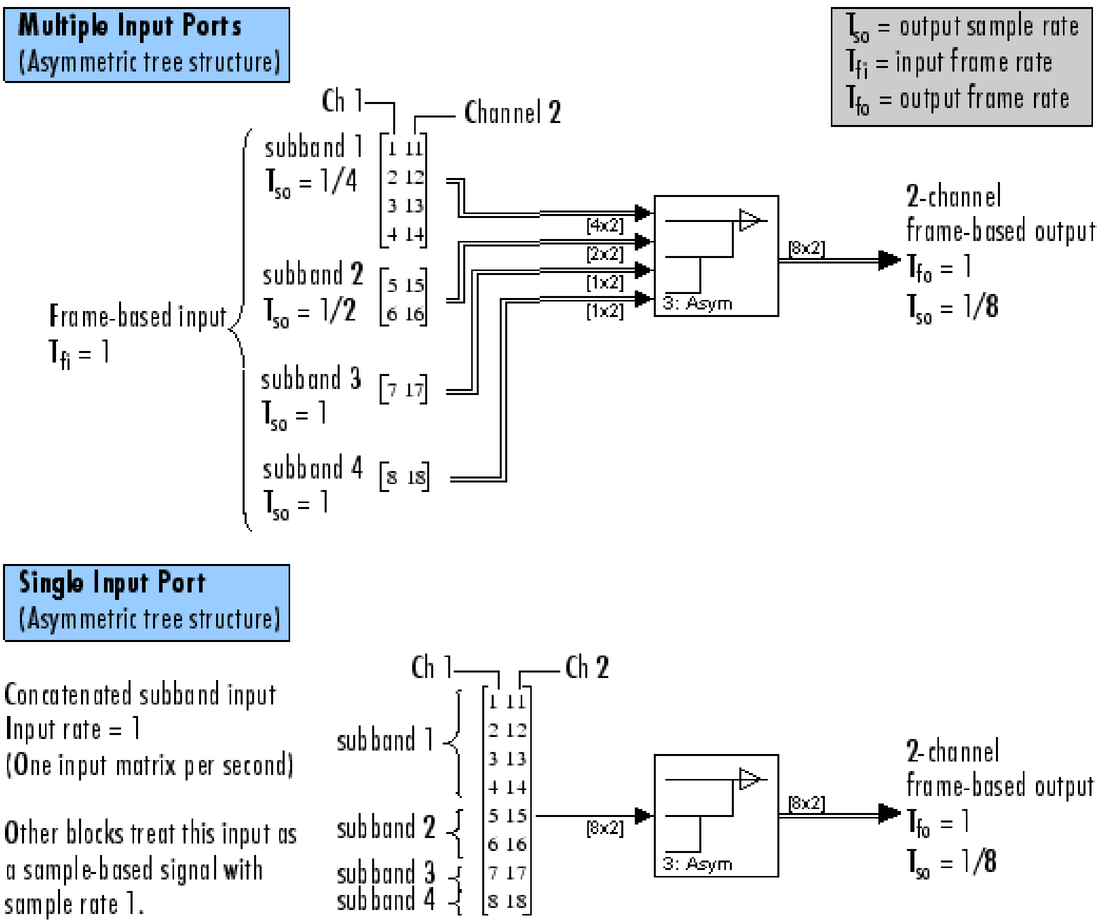
Note Any output of a Dyadic Analysis Filter Bank block whose parameter settings match the corresponding settings of this block is a valid input to this block. For example, the setting of the Dyadic Analysis Filter Bank block parameter, **Output**, must be the same as this block's **Input** parameter (`Single port` or `Multiple ports`).

Valid Inputs for Input Set to Single Port

- Inputs must be vectors or matrices of concatenated subbands. The block always interprets the inputs as sample based.
- Each input column contains the subbands for an independent signal.
- Upper input rows contain the high-frequency subbands, and the lower rows contain the low-frequency subbands.

Valid Inputs for Input Set to Multiple Ports

- Each subband must be provided as a vector or matrix to separate block input ports. The block always interprets the inputs as frame based.
- The columns of each input contains a subband for an independent signal.
- The input to the topmost input port is the subband containing the highest frequencies, and the input to the bottommost port is the subband containing the lowest frequencies.



Valid Inputs to a 3-Level Asymmetric Dyadic Synthesis Filter Bank

For general information about the filter banks, see “Dyadic Synthesis Filter Banks”.

Output Characteristics

The following table summarizes the output characteristics for both frame-based inputs, and concatenated subband inputs. For an illustration of why the output characteristics exist, see the figure Valid Inputs to a 3-Level Asymmetric Dyadic Synthesis Filter Bank on page 2-462.

<table>
<tr><th></th><th>Frame-Based Inputs
(Input = Multiple ports)</th><th>Concatenated Subband Inputs
(Input = Single port)</th></tr>
<tr><td>Output Frame Status</td><td colspan="2">Outputs are always frame based regardless of the input frame status. Each output column is an independent channel, reconstructed from the corresponding channel in the inputs.</td></tr>
<tr><td>Output Frame Rate</td><td>Same as the input frame rate.</td><td>Same as the input rate (the rate of the concatenated subband inputs).</td></tr>
<tr><td>Output Frame Dimensions</td><td>• The output has the same number of columns as the inputs.
• The number of output rows depends on the tree structure of the filter bank:
- Asymmetric — The number of output rows is twice the number of rows in the input to the topmost input port.
- Symmetric — The number of output rows is the product of the number of input ports and the number of rows in an input to any input port.</td><td>The output has the same number of rows and columns as the input.</td></tr>
</table>

For general information about the filter banks, see “Dyadic Synthesis Filter Banks”.

Filter Bank Filters

You must specify the highpass and lowpass filters in the filter bank by setting the **Filter** parameter to one of the following options:

- `User defined` — Allows you to explicitly specify the filters with two vectors of filter coefficients in the **Lowpass FIR filter coefficients** and **Highpass FIR filter coefficients** parameters. The block uses the same lowpass and highpass filters throughout the filter bank. The two filters should be halfband filters, where each filter passes the frequency band that the other filter stops. To use this block to perfectly reconstruct a signal decomposed by a Dyadic Analysis Filter Bank block, the filters in this block *must* be designed to perfectly reconstruct the outputs of the analysis filter bank. To learn how to design your own perfect reconstruction filters, see "References" on page 2-469.

- Wavelet such as `Biorthogonal` or `Daubechies` — The block uses the specified wavelet to construct the lowpass and highpass filters using the Wavelet Toolbox function `wfilters`. Depending on the wavelet, the block might enable either the **Wavelet order** or **Filter order [synthesis / analysis]** parameter. (The latter parameter allows you to specify different wavelet orders for the analysis and synthesis filter stages.) To use this block to reconstruct a signal decomposed by a Dyadic Analysis Filter Bank block, you must set both blocks to use the same wavelets with the same order. You must have a Wavelet Toolbox license to use wavelets.

Specifying Filters with the Filter Parameter and Related Parameters

Filter	Sample Setting for Related Filter Specification Parameters	Corresponding Wavelet Function Syntax
User-defined	Filters based on Daubechies wavelets with wavelet order `3`: • **Lowpass FIR filter coefficients =** `[0.0352 -0.0854 -0.1350 0.4599 0.8069 0.3327]` • **Highpass FIR filter coefficients =**	None

Specifying Filters with the Filter Parameter and Related Parameters (Continued)

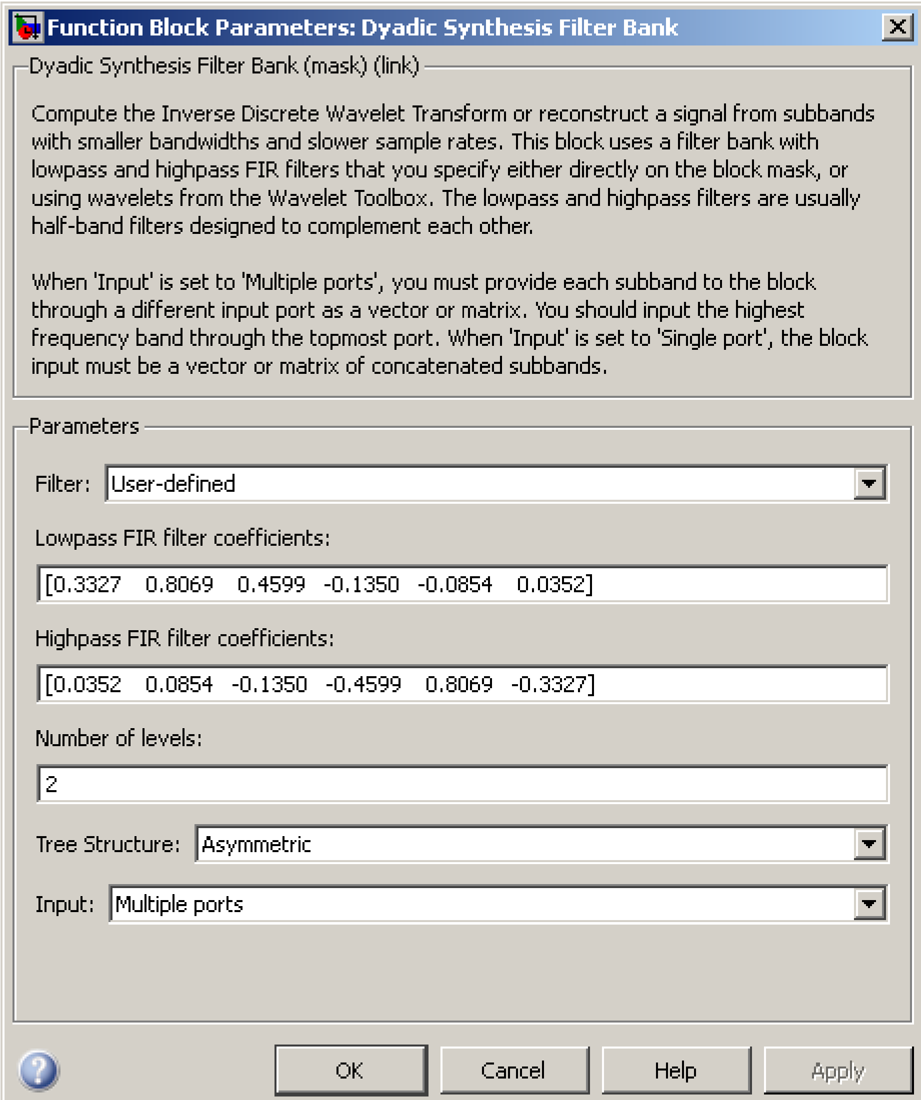
Filter	Sample Setting for Related Filter Specification Parameters	Corresponding Wavelet Function Syntax
	`[-0.3327 0.8069 -0.4599 -0.1350 0.0854 0.0352]`	
Haar	None	`wfilters('haar')`
Daubechies	**Wavelet order** = `4`	`wfilters('db4')`
Symlets	**Wavelet order** = `3`	`wfilters('sym3')`
Coiflets	**Wavelet order** = `1`	`wfilters('coif1')`
Biorthogonal	**Filter order [synthesis / analysis]** = `[3/1]`	`wfilters('bior3.1')`
Reverse Biorthogonal	**Filter order [synthesis / analysis]** = `[3/1]`	`wfilters('rbio3.1')`
Discrete Meyer	None	`wfilters('dmey')`

Examples

See "Examples" on page 2-454 on the Dyadic Analysis Filter Bank block reference page.

Dialog Box

The parameters displayed in the block dialog vary depending on the setting of the **Filter** parameter. Only some of the parameters described below are visible in the dialog box at any one time.


Function Block Parameters: Dyadic Synthesis Filter Bank
Dyadic Synthesis Filter Bank (mask) (link)
Compute the Inverse Discrete Wavelet Transform or reconstruct a signal from subbands with smaller bandwidths and slower sample rates. This block uses a filter bank with lowpass and highpass FIR filters that you specify either directly on the block mask, or using wavelets from the Wavelet Toolbox. The lowpass and highpass filters are usually half-band filters designed to complement each other.
When 'Input' is set to 'Multiple ports', you must provide each subband to the block through a different input port as a vector or matrix. You should input the highest frequency band through the topmost port. When 'Input' is set to 'Single port', the block input must be a vector or matrix of concatenated subbands.
Parameters
Filter: User-defined
Lowpass FIR filter coefficients:
[0.3327 0.8069 0.4599 -0.1350 -0.0854 0.0352]
Highpass FIR filter coefficients:
[0.0352 0.0854 -0.1350 -0.4599 0.8069 -0.3327]
Number of levels:
2
Tree Structure: Asymmetric
Input: Multiple ports
OK
Cancel
Help
Apply

Note To use this block to reconstruct a signal decomposed by a Dyadic Analysis Filter Bank block, all the parameters in this block must be the same as the corresponding parameters in the Dyadic Analysis Filter Bank block (except the **Lowpass FIR filter coefficients** and **Highpass FIR filter coefficients**; see the descriptions of these parameters).

Filter

The type of filter used to determine the high- and low-pass FIR filters in the filter bank:

- Select `User defined` to explicitly specify the filter coefficients in the **Lowpass FIR filter coefficients** and **Highpass FIR filter coefficients** parameters.
- Select a wavelet such as `Biorthogonal` or `Daubechies` to specify a wavelet-based filter. The block uses the Wavelet Toolbox `wfilters` function to construct the filters. Extra parameters such as **Wavelet order** or **Filter order [synthesis / analysis]** might become enabled. For a list of the supported wavelets, see the table Specifying Filters with the Filter Parameter and Related Parameters on page 2-464.

Lowpass FIR filter coefficients

A vector of filter coefficients (descending powers of z) that specifies coefficients used by all the lowpass filters in the filter bank. This parameter is enabled when you set **Filter** to `User defined`. The lowpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the **Highpass FIR filter coefficients** parameter. To perfectly reconstruct a signal decomposed by the Dyadic Analysis Filter Bank, the filters in this block *must* be designed to perfectly reconstruct the outputs of the analysis filter bank. Otherwise, the reconstruction is not perfect. The default values of this parameter specify a perfect reconstruction filter for the default settings of the Dyadic Analysis Filter Bank (based on a Daubechies wavelet with wavelet order `3`).

Highpass FIR filter coefficients

A vector of filter coefficients (descending powers of *z*) that specifies coefficients used by all the highpass filters in the filter bank. This parameter is enabled when you set **Filter** to `User defined`. The highpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the **Lowpass FIR filter coefficients** parameter. To perfectly reconstruct a signal decomposed by the Dyadic Analysis Filter Bank, the filters in this block *must* be designed to perfectly reconstruct the outputs of the analysis filter bank. Otherwise, the reconstruction is not perfect. The default values of this parameter specify a perfect reconstruction filter for the default settings of the Dyadic Analysis Filter Bank (based on a Daubechies wavelet with wavelet order `3`).

Wavelet order

The order of the wavelet selected in the **Filter** parameter. This parameter is enabled only when you set **Filter** to certain types of wavelets, as shown in the table Specifying Filters with the Filter Parameter and Related Parameters on page 2-464.

Filter order [synthesis / analysis]

The order of the wavelet for the synthesis and analysis filter stages. For example, when you set the **Filter** parameter to **Biorthogonal** and set the **Filter order [synthesis / analysis]** parameter to `[2 / 6]`, the block calls the `wfilters` function with input argument `'bior2.6'`. This parameter is enabled only when you set **Filter** to certain types of wavelets, as shown in Specifying Filters with the Filter Parameter and Related Parameters on page 2-464.

Number of levels

The number of filter bank levels. An n-level asymmetric structure has n+1 inputs, and an n-level symmetric structure has 2^n inputs, as shown in “n-Level Asymmetric Dyadic Synthesis Filter Bank” and “n-Level Symmetric Dyadic Synthesis Filter Bank”.

The default setting of this parameter is `2`.

Tree structure

The structure of the filter bank: `Asymmetric`, or `Symmetric`. See the figures “n-Level Asymmetric Dyadic Synthesis Filter Bank” and “n-Level Symmetric Dyadic Synthesis Filter Bank”.

The default setting of this parameter is `Asymmetric` for the Dyadic Synthesis Filter Bank block, and `Symmetric` for the IDWT block.

Input

Set to `Multiple ports` to accept each input subband at a separate port (the topmost port accepts the subband with the highest frequency band). Set to `Single port` to accept one vector or matrix of concatenated subbands at a single port. For more information, see “Input Requirements” on page 2-460.

The default setting of this parameter is `Multiple ports` for the Dyadic Synthesis Filter Bank block, and `Single port` for the IDWT block.

References

Fliege, N. J. *Multirate Digital Signal Processing: Multirate Systems, Filter Banks, Wavelets*. West Sussex, England: John Wiley & Sons, 1994.

Strang, G. and T. Nguyen. *Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1996.

Vaidyanathan, P. P. *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice Hall, 1993.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point
Output	• Double-precision floating point • Single-precision floating point

See Also

Dyadic Analysis Filter Bank	Signal Processing Blockset
IDWT	Signal Processing Blockset
Two-Channel Synthesis Subband Filter	Signal Processing Blockset

See "Multirate Filters" for related information.

Purpose

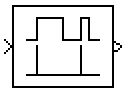
Detect transition from zero to nonzero value

Library

Signal Management / Switches and Counters

`dspswit3`

Description

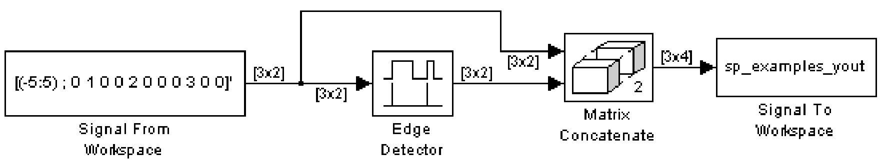


The Edge Detector block generates an impulse (the value 1) in a given output channel when the corresponding channel of the input transitions from zero to a nonzero value. Otherwise, the block generates zeros in each channel.

The output has the same dimension and sample rate as the input. When the input is frame based, the output is frame based; otherwise, the output is sample based. For frame-based input, an edge that is split across two consecutive frames (that is, a zero at the bottom of the first frame, and a nonzero value at the top of the following frame) is counted in the frame that contains the nonzero value.

Examples

In the model below, the Edge Detector block locates the edges (zero to nonzero transitions) in a two-channel frame-based input with frame size 3. The two input channels are horizontally concatenated with the two output channels to create the four-channel workspace variable `sp_examples_yout`.



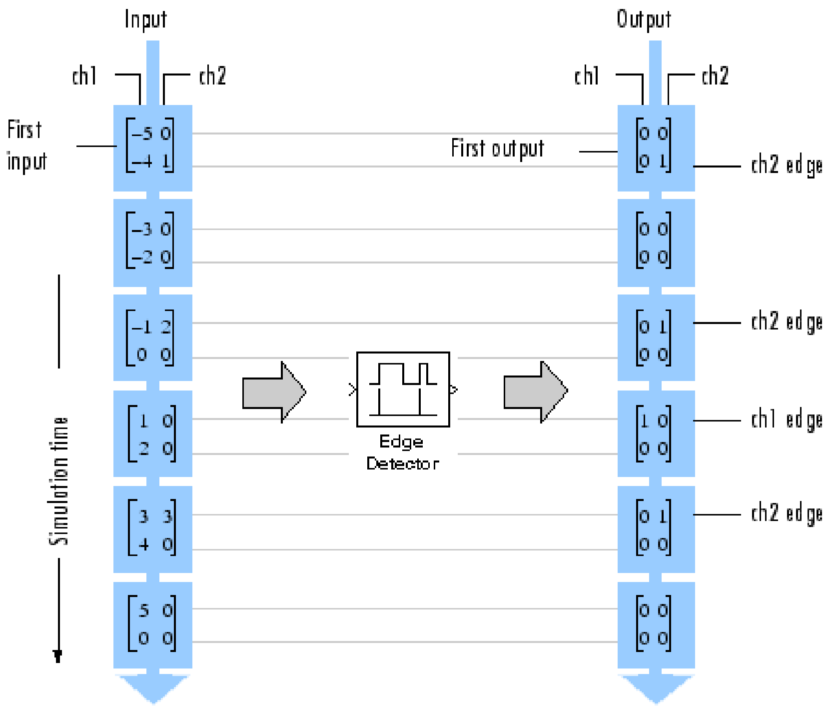
Adjust the block parameters as described below. (Use the default settings for the Signal To Workspace block.)

- Set the Signal From Workspace block parameters as follows:
  - **Signal** = `[(-5:5); 0 1 0 0 2 0 0 0 3 0 0]'`
  - **Sample time** = 1

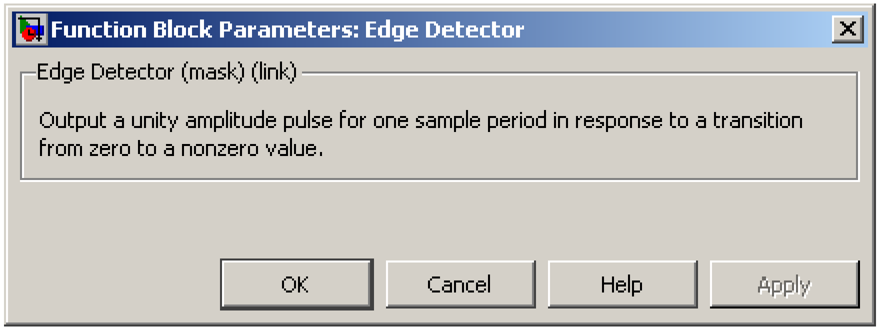
- **Samples per frame** = 3

- Set the Matrix Concatenate block parameters as follows:
  - **Number of inputs** = 2
  - **Mode** = `Multidimensional array`
  - **Concatenate dimension** = 2

As shown below, the block finds edges at sample 7 in channel 1, and at samples 2, 5, and 9 in channel 2.



Dialog Box



Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed and unsigned)
- Boolean — The block might output Boolean values depending on the input data type, and whether Boolean support is enabled or disabled.
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers
- Enumerated

See Also

Counter	Signal Processing Blockset
Event-Count Comparator	Signal Processing Blockset

Purpose

Detect threshold crossing of accumulated nonzero inputs

Library

Signal Management / Switches and Counters

`dspswit3`

Description

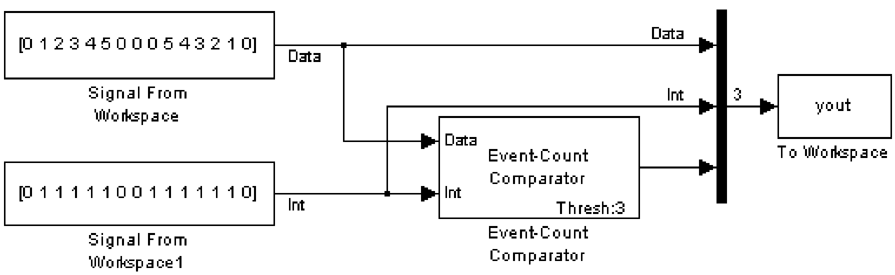
The Event-Count Comparator block records the number of nonzero inputs to the Data port during the period that the block is enabled by a high signal (the value 1) at the Int port. Both inputs must be scalars; the input to the Int port can be sample based or frame based. When the input to the Data port is frame based, the output is frame based; otherwise, the output is sample based.

When the number of accumulated nonzero inputs first equals the **Event threshold** setting, the block waits one additional sample interval, and then sets the output high (1). The block holds the output high until recording is restarted by a low-to-high (0-to-1) transition at the Int port.

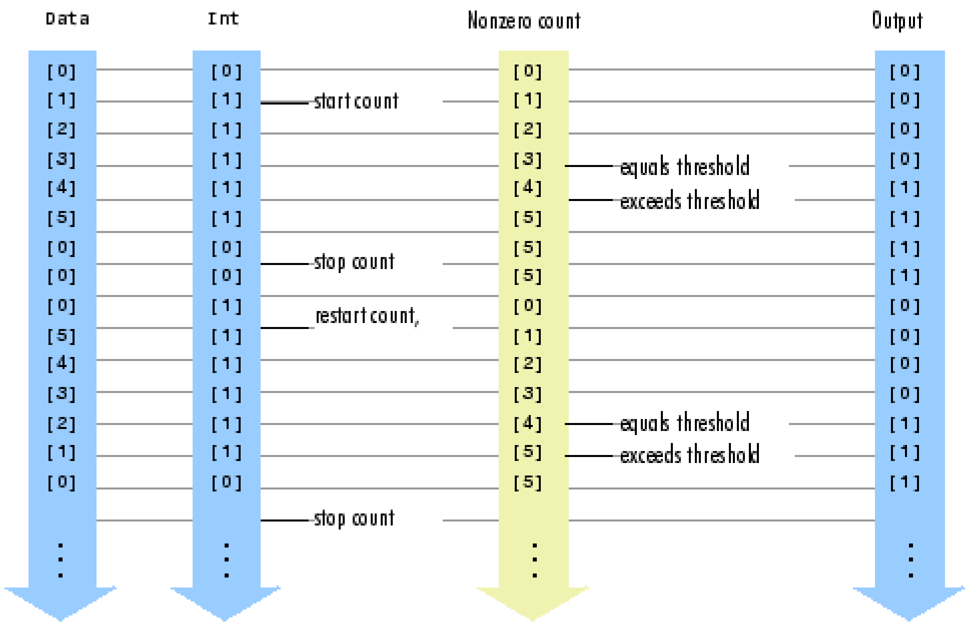
The Event-Count Comparator block accepts real and complex floating-point and fixed-point inputs. However, because the block has discrete state, it does not support constant or continuous sample times. Therefore, at least one input or output port of the Event-Count Comparator block must be connected to a block whose **Sample time** parameter is discrete. The Event-Count Comparator block inherits this non-infinite discrete sample time.

Examples

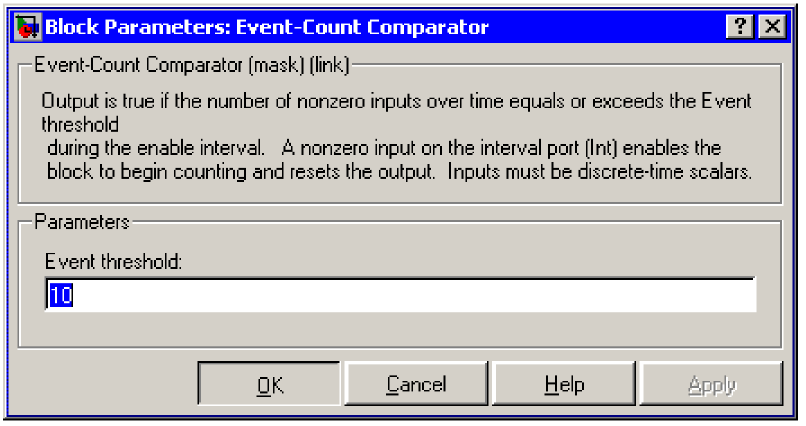
In the model below, the Event-Count Comparator block (**Event threshold** = 3) detects two threshold crossings in the input to the Data port, one at sample 4 and one at sample 12.



All inputs and outputs are multiplexed into the workspace variable `yout`, whose contents are shown in the figure below. The two left columns in the illustration show the inputs to the Data and Int ports, the center column shows the state of the block's internal counter, and the right column shows the block's output.



Dialog Box



Event threshold

Specify the value against which to compare the number of nonzero inputs. Tunable.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed and unsigned)
- Boolean
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers
- Enumerated

See Also

Counter	Signal Processing Blockset
Edge Detector	Signal Processing Blockset

Purpose

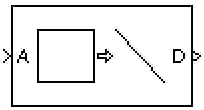
Extract main diagonal of input matrix

Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description

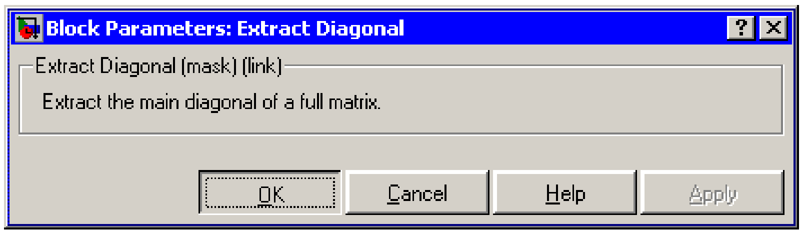


The Extract Diagonal block populates the 1-D output vector with the elements on the main diagonal of the M-by-N input matrix `A`.

```
D = diag(A)      Equivalent MATLAB code
```

The output vector has length `min(M,N)`, and is always sample based.

Dialog Box



Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed and unsigned)
- Boolean — Block outputs are always Boolean.
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers

See Also

Constant Diagonal Matrix	Signal Processing Blockset
Create Diagonal Matrix	Signal Processing Blockset

Extract Triangular Matrix	Signal Processing Blockset
`diag`	MATLAB

Purpose

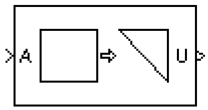
Extract lower or upper triangle from input matrices

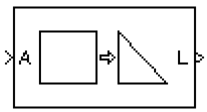
Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description





The Extract Triangular Matrix block creates a triangular matrix output from the upper or lower triangular elements of an *M*-by-*N* input matrix. A length-*M* 1-D vector input is treated as an *M*-by-1 matrix.

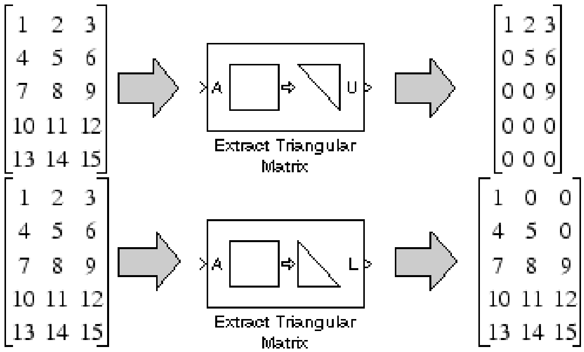
The **Extract** parameter selects between the two components of the input:

- `Upper` — Copies the elements on and above the main diagonal of the input matrix to an output matrix of the same size. The first *row* of the output matrix is therefore identical to the first *row* of the input matrix. The elements below the main diagonal of the output matrix are zero.
- `Lower` — Copies the elements on and below the main diagonal of the input matrix to an output matrix of the same size. The first *column* of the output matrix is therefore identical to the first *column* of the input matrix. The elements above the main diagonal of the output matrix are zero.

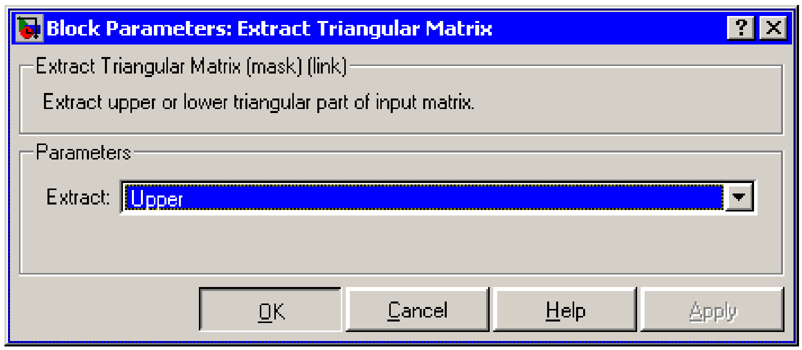
The output has the same frame status as the input.

Examples

The example below shows the extraction of upper and lower triangles from a 5-by-3 input matrix.



Dialog Box



Extract

The component of the matrix to copy to the output, upper triangle or lower triangle.

Supported Data Types

Port	Supported Data Types
A	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
U	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
L	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Autocorrelation LPC	Signal Processing Blockset
Cholesky Factorization	Signal Processing Blockset
Constant Diagonal Matrix	Signal Processing Blockset
Extract Diagonal	Signal Processing Blockset

Forward Substitution	Signal Processing Blockset
LDL Factorization	Signal Processing Blockset
LU Factorization	Signal Processing Blockset
`tril`	MATLAB
`triu`	MATLAB

Purpose

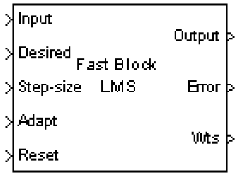
Compute output, error, and weights using LMS adaptive algorithm

Library

Filtering / Adaptive Filters

`dspadpt3`

Description



Fast Block LMS Filter

The Fast Block LMS Filter block implements an adaptive least mean-square (LMS) filter, where the adaptation of the filter weights occurs once for every block of data samples. The block estimates the filter weights, or coefficients, needed to convert the input signal into the desired signal. Connect the signal you want to filter to the Input port. This input signal can be a sample-based scalar or a single-channel frame-based signal. Connect the signal you want to model to the Desired port. The desired signal must have the same data type, frame status, complexity, and dimensions as the input signal. The Output port outputs the filtered input signal, which can be sample or frame based. The Error port outputs the result of subtracting the output signal from the desired signal.

The block calculates the filter weights using the Block LMS Filter equations. For more information, see Block LMS Filter. The Fast Block LMS Filter block implements the convolution operation involved in the calculations of the filtered output, y, and the weight update function in the frequency domain using the FFT algorithm used in the Overlap-Save FFT Filter block. See Overlap-Save FFT Filter for more information.

Use the **Filter length** parameter to specify the length of the filter weights vector.

The **Block size** parameter determines how many samples of the input signal are acquired before the filter weights are updated. The input frame length must be a multiple of the **Block size** parameter.

The **Step-size (mu)** parameter corresponds to μ in the equations. You can either specify a step-size using the input port, Step-size, or enter a value in the Block Parameters: Block LMS Filter dialog box.

Use the **Leakage factor (0 to 1)** parameter to specify the leakage factor, $0 < 1 - \mu\alpha \le 1$, in the leaky LMS algorithm shown below.

$$\mathbf{w}(k) = (1-\mu\alpha)\mathbf{w}(k-1) - f(\mathbf{u}(n), e(n), \mu)$$

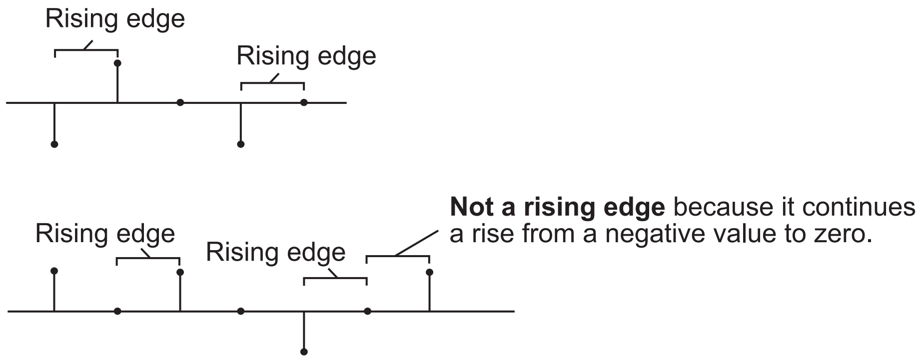
Enter the initial filter weights, $\mathbf{w}(0)$, as a vector or a scalar in the **Initial value of filter weights** text box. When you enter a scalar, the block uses the scalar value to create a vector of filter weights. This vector has length equal to the filter length and all of its values are equal to the scalar value.

When you select the **Adapt port** check box, an Adapt port appears 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 at their current values.

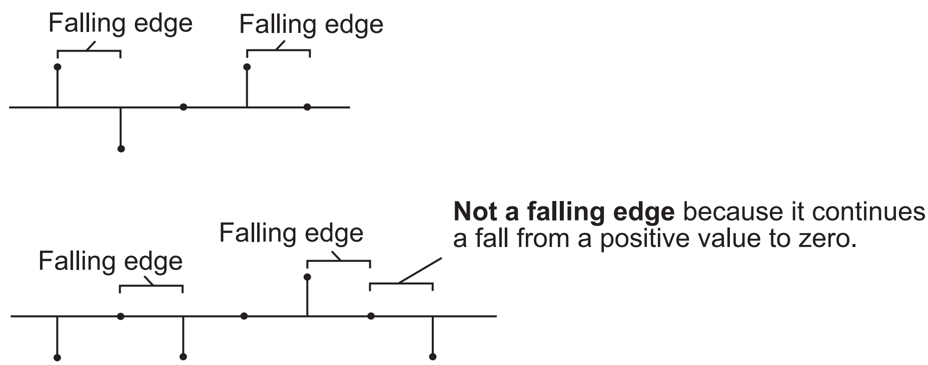
When you want to reset the value of the filter weights to their initial values, use the **Reset input** parameter. The block resets the filter weights whenever a reset event is detected at the Reset port. The reset signal rate must be the same rate as the data signal input.

From the **Reset input** list, select `None` to disable the Reset port. To enable the Reset port, select one of the following from the **Reset input** list:

- `Rising edge` — Triggers a reset operation when the Reset input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)



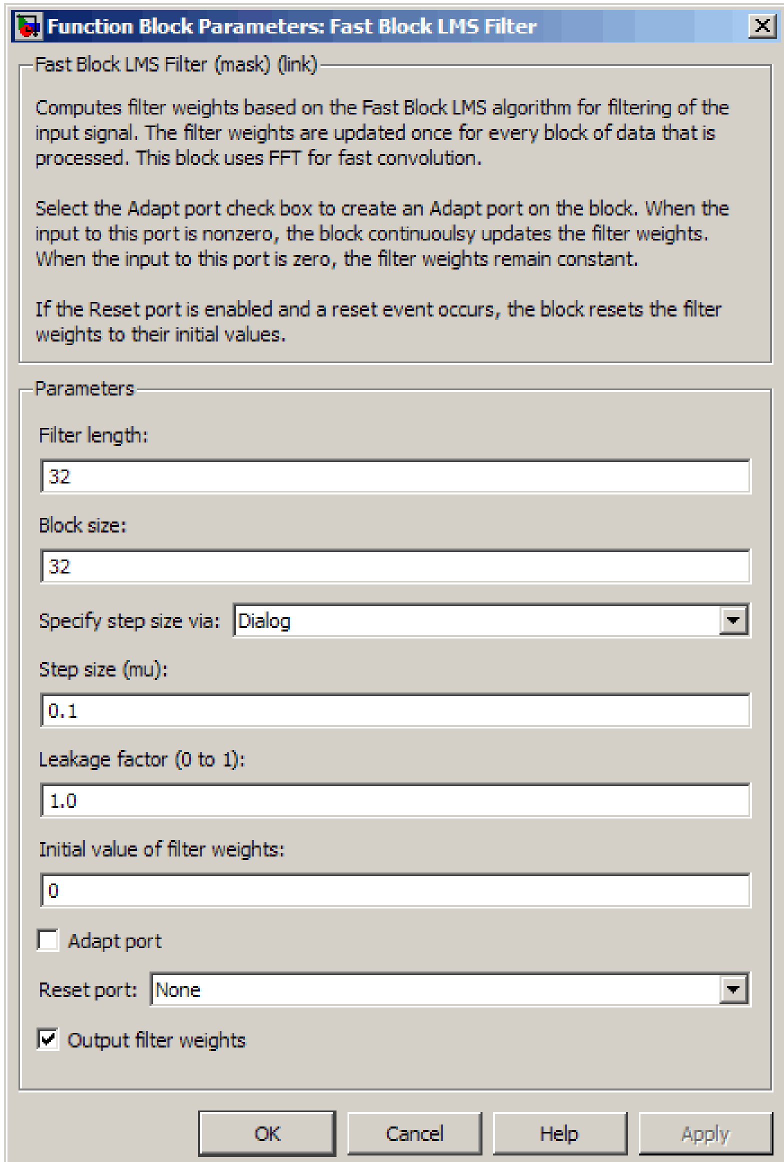
- `Falling edge` — Triggers a reset operation when the Reset input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)



- `Either edge` — Triggers a reset operation when the Reset input is a `Rising edge` or `Falling edge` (as described above)
- `Non-zero sample` — Triggers a reset operation at each sample time that the Reset input is not zero

Select the **Output filter weights** check box to create a Wts port on the block. For each iteration, the block outputs the current updated filter weights from this port.

Dialog Box


Function Block Parameters: Fast Block LMS Filter
Fast Block LMS Filter (mask) (link)
Computes filter weights based on the Fast Block LMS algorithm for filtering of the input signal. The filter weights are updated once for every block of data that is processed. This block uses FFT for fast convolution.
Select the Adapt port check box to create an Adapt port on the block. When the input to this port is nonzero, the block continuoulsy 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.
Parameters
Filter length:
32
Block size:
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

Filter length
Enter the length of the FIR filter weights vector. The sum of the block size and the filter length must be a power of 2.

Block size
Enter the number of samples to acquire before the filter weights are updated. The input frame length must be an integer multiple of the block size. The sum of the block size and the filter length must be a power of 2.

Specify step-size via
Select `Dialog` to enter a value for mu, or select `Input port` to specify mu using the Step-size input port.

Step-size (mu)
Enter the step-size. Tunable.

Leakage factor (0 to 1)
Enter the leakage factor, $0 < 1 - \mu\alpha \leq 1$. Tunable.

Initial value of filter weights
Specify the initial values of the FIR filter weights.

Adapt port
Select this check box to enable the Adapt input port.

Reset input
Select this check box to enable the Reset input port.

Output filter weights
Select this check box to export the filter weights from the Wts port.

References

Hayes, M.H. *Statistical Digital Signal Processing and Modeling.* New York: John Wiley & Sons, 1996.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point
Desired	• Must be the same as Input for floating-point signals • Must be any fixed-point data type when Input is fixed point
Step-size	• Must be the same as Input for floating-point signals • Must be any fixed-point data type when Input is fixed point
Adapt	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Reset	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Must be the same as Input for floating-point signals • Must be the same as Desired for fixed-point signals

Port	Supported Data Types
Error	• Must be the same as Input for floating-point signals • Must be the same as Desired for fixed-point signals
Wts	• Must be the same as Input for floating-point signals • Obeys the **Weights** parameter for fixed-point signals

Purpose

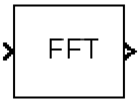
Compute fast Fourier transform (FFT) of input

Library

Transforms

`dspxfrm3`

Description



The FFT block computes the fast Fourier transform (FFT) of each row of a sample-based 1-by-P input vector, u, or across the first dimension (P) of an N-D input array, u. When you select the **Inherit FFT length from input dimensions** check box, P must be an integer power of two, and the **FFT length**, M, is set equal to P. If you do not select the check box, P can be any length, and the value of the **FFT length** parameter must be a positive integer power of two. For user-specified FFT lengths, when M does not equal P, zero padding or modulo-M data wrapping occurs before the FFT operation, as per Orfanidis [1]:

```
y = fft(u,M)  % P   M

y(:,l) = fft(datawrap(u(:,l),M))  % P > M; l = 1,...,N
```

When the input length, P, is greater than the FFT length, M, you may see magnitude increases in your FFT output. These magnitude increases occur because the FFT block uses modulo-M data wrapping to preserve all available input samples.

To avoid such magnitude increases, you can truncate the length of your input sample, P, to the FFT length, M. To do so, place a Pad block before the FFT block in your model.

The kth entry of the lth output channel, $y(k, l)$, equals the kth point of the M-point discrete Fourier transform (DFT) of the lth input channel:

$$y(k,l) = \sum_{p=1}^{P} u(p,l)e^{-j2\pi(p-1)(k-1)/M} \qquad k = 1,\dots,M$$

This block supports real and complex floating-point and fixed-point inputs.

Input and Output Characteristics

The following table describes valid inputs to the FFT block, their corresponding outputs, and the dimension along which the block computes the DFT.

- Valid inputs to the FFT block. Inputs can be real- or complex-valued, and must be in linear order.
- The dimension along which the block computes the DFT.
- The corresponding block output characteristics. The output port rate must equal the input port rate.

Valid Block Inputs	Dimension Along Which Block Computes DFT	Corresponding Block Output Characteristics
N-D array	First dimension	• Sample based. • Complex valued. • N-D array with the size of the first dimension equal to the FFT length, M, and all other dimensions the same size as the input. • Each output column contains the M-point DFT of the corresponding input channel in linear or bit-reversed order.
Sample-based 1-by-P row vector	Row	• Sample based. • Complex valued. • 1-by-M row vector.

Valid Block Inputs	Dimension Along Which Block Computes DFT	Corresponding Block Output Characteristics
		• Each output row contains the M-point DFT of the corresponding input channel in linear or bit-reversed order.
Unoriented length-P 1-D vector	Vector	Unoriented, length-M, complex-valued 1-D output vector containing M-point DFT of input in linear or bit-reversed order.

Selecting the Twiddle Factor Computation Method

The **Twiddle factor computation** parameter determines how the block computes the necessary sine and cosine terms to calculate the term $e^{-j2\pi(p-1)(k-1)/M}$, shown in the first equation of this block reference page.

The block only supports `Table lookup` mode for fixed-point signals.

The **Twiddle factor computation** parameter has two settings, each with its advantages and disadvantages, as described in the following table.

Twiddle Factor Computation Parameter Setting	Sine and Cosine Computation Method	Effect on Block Performance
`Table lookup`	The block computes and stores the trigonometric values before the simulation starts and retrieves them during the simulation. When you generate code from the block, the processor running the generated code stores the trigonometric values computed by the block, and retrieves the values during code execution.	The block usually runs much more quickly, but requires extra memory for storing the precomputed trigonometric values. You can optimize the table for memory consumption or speed, as described in "Optimizing the Table of Trigonometric Values" on page 2-495.
`Trigonometric fcn`	The block computes sine and cosine values during the simulation. When you generate code from the block, the processor running the generated code computes the sine and cosine values while the code runs.	The block usually runs more slowly, but does not need extra data memory. For code generation, the block requires a support library to emulate the trigonometric functions, increasing the size of the generated code.

Optimizing the Table of Trigonometric Values

When you set the **Twiddle factor computation** parameter to `Table lookup`, you also need to set the **Optimize table for** parameter.

This parameter optimizes the table of trigonometric values for speed or memory by varying the number of table entries as summarized in the following table.

Optimize Table for Parameter Setting	Number of Table Entries for N-Point FFT	Memory Required for Single-Precision, 512-Point FFT
`Speed`	$3N/4$ — floating point N — fixed point	$\left(\frac{3\times 512}{4}\,\text{table entries}\right)\times\left(4\,\frac{\text{bytes}}{\text{table entry}}\right)$ $= 1536\,\text{bytes}$
`Memory`	$N/4$ — floating point Not supported for fixed point	$\left(\frac{512}{4}\,\text{table entries}\right)\times\left(4\,\frac{\text{bytes}}{\text{table entry}}\right)$ $= 512\,\text{bytes}$

Ordering Output Column Entries

You can set the **Output in bit-reversed order** parameter to specify the ordering of the column elements of the block output. If you select the **Output in bit-reversed order** check box, the output appears in bit-reversed order. If you clear the **Output in bit-reversed order** check box, the output appears in linear order.

Note The FFT block calculates its output in bit-reversed order. Linearly ordering the FFT block output requires an extra bit-reversal operation. Thus, in many situations, you can increase the speed of the FFT block by selecting the **Output in bit-reversed order** check box.

For more information ordering of the output, see “Linear and Bit-Reversed Output Order”.

Algorithms Used for FFT Computation

Depending on whether the block’s input is real- or complex-valued and whether you want the output in linear or bit-reversed order, the block uses one or more of the following algorithms as summarized in the following table:

- Bit-reversal operation
- Double-signal algorithm
- Half-length algorithm
- Radix-2 decimation-in-time (DIT) algorithm
- Radix-2 decimation-in-frequency (DIF) algorithm

Complexity of Input	Output Ordering	Algorithms Used for FFT Computation
Complex	Linear	Bit-reversal operation and radix-2 DIT
Complex	Bit-reversed	Radix-2 DIF
Real	Linear	Bit-reversal operation and radix-2 DIT in conjunction with the half-length and double-signal algorithms
Real	Bit-reversed	Radix-2 DIF in conjunction with the half-length and double-signal algorithms

The efficiency of the FFT algorithm can be enhanced for real input signals by forming complex-valued sequences from the real-valued sequences prior to the computation of the DFT. When there are $2N+1$ real input channels, the FFT block forms these complex-valued sequences by applying the double-signal algorithm to the first $2N$ input channels, and the half-length algorithm to the last odd-numbered channel.

For real input signals with fixed-point data types, it is possible to see different numerical results in the output of the last odd numbered channel, even when all input channels are identical. This numerical difference results from differences in the double-signal algorithm and the half-length algorithm.

You can eliminate this numerical difference in two ways:

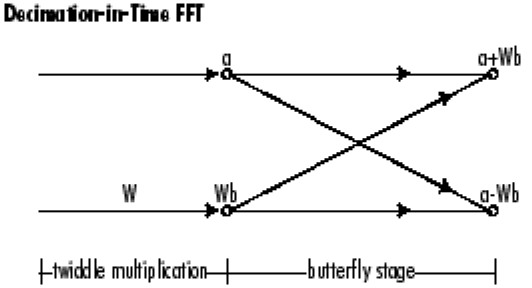
- Using full precision arithmetic for fixed-point input signals
- Changing the input data type to floating point

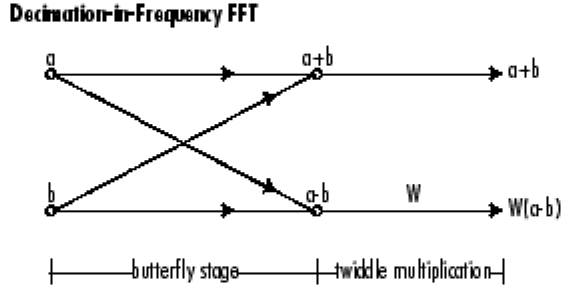
For more information on the double-signal and half-length algorithms, see Proakis [2]. "Efficient Computation of the DFT of Two Real Sequences" on page 475 describes the double signal algorithm. "Efficient Computation of the DFT of a 2N-Point Real Sequence" on page 476 describes the half-length algorithm.

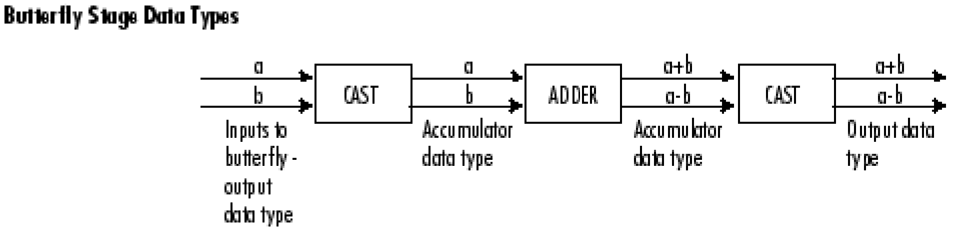
Fixed-Point Data Types

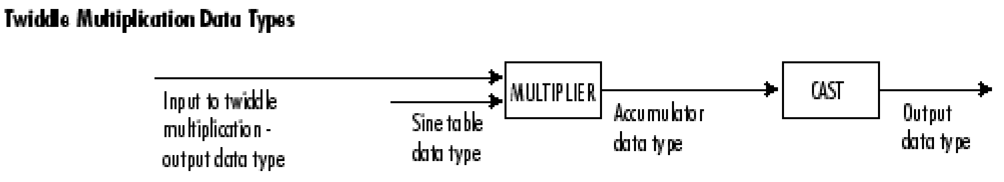
The following diagrams show the data types used within the FFT block for fixed-point signals. You can set the sine table, accumulator, product output, and output data types displayed in the diagrams in the FFT block dialog as discussed in "Dialog Box" on page 2-500.

The block first casts inputs to the FFT block to the output data type and stores them in the output buffer. Each butterfly stage then processes signals in the accumulator data type, and the block then casts the final output of the butterfly back into the output data type. The block multiplies in a twiddle factor before each butterfly stage in a decimation-in-time FFT, and after each butterfly stage in a decimation-in-frequency FFT.









The output of the multiplier appears in the accumulator data type because both of the inputs to the multiplier are complex. For details on the complex multiplication performed, see “Multiplication Data Types”.

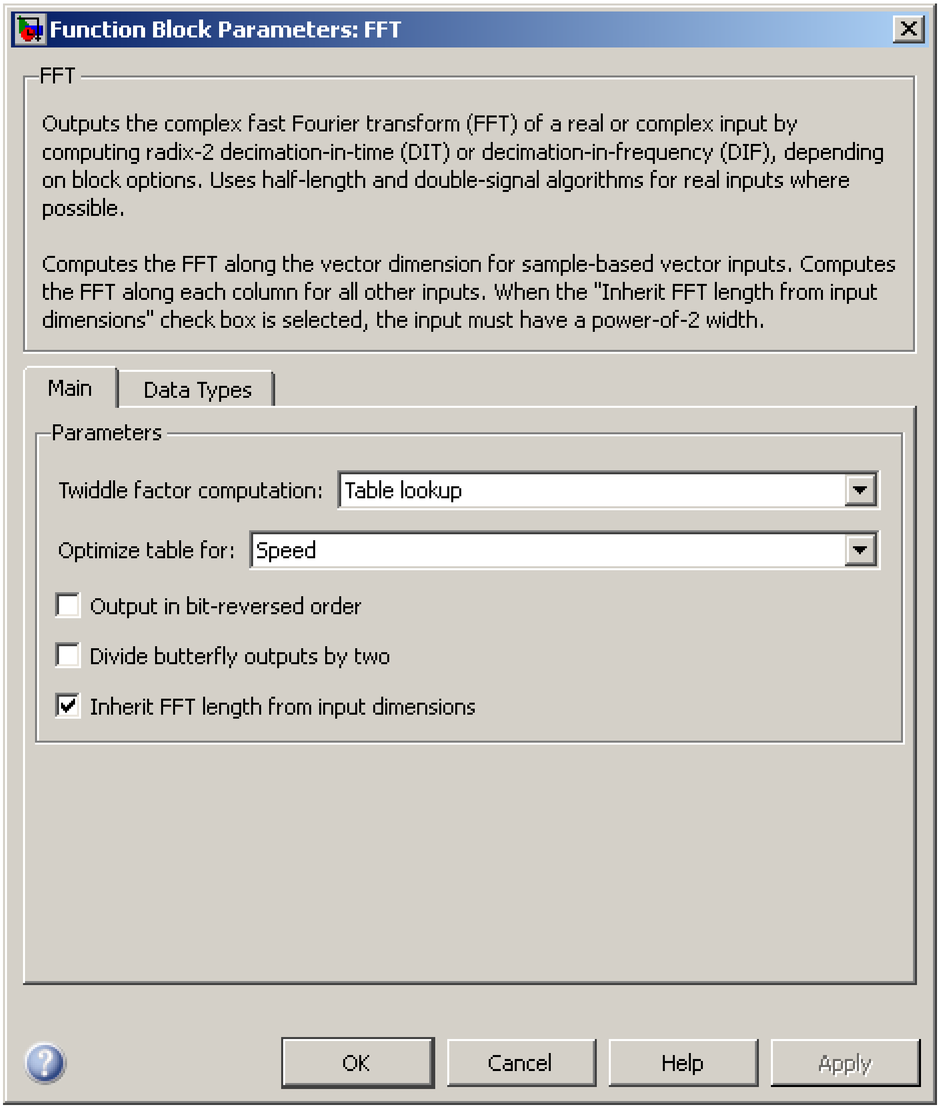
Note When the block input is fixed point, all internal data types are signed fixed point.

Examples

See the section on “Transforming Time-Domain Data into the Frequency Domain” in the *Signal Processing Blockset User’s Guide*.

Dialog Box

The **Main** pane of the FFT block dialog appears as follows.



Twiddle factor computation

Specify the computation method of the term $e^{-j2\pi(p-1)(k-1)/M}$, shown in the first equation of this block reference page.

In `Table lookup` mode, the block computes and stores the sine and cosine values before the simulation starts.

In `Trigonometric fcn` mode, the block computes the sine and cosine values during the simulation. See "Selecting the Twiddle Factor Computation Method" on page 2-494 for more information.

This parameter must be set to `Table lookup` for fixed-point signals.

Optimize table for

Select the optimization of the table of sine and cosine values for `Speed` or `Memory`. This parameter becomes available only when you set the **Twiddle factor computation** parameter to `Table lookup`. See "Selecting the Twiddle Factor Computation Method" on page 2-494 for more information.

This parameter must be set to `Speed` for fixed-point signals.

Output in bit-reversed order

Designate the order of the output channel elements relative to the ordering of the input elements. When you select this check box, the output channel elements appear in bit-reversed order relative to the input ordering. Otherwise, the output column elements appear in linear order relative to the input ordering.

Linearly ordering the output requires extra data sorting manipulation, so in some situations it might be better to output in bit-reversed order.

Divide butterfly outputs by two

When you select this parameter, the output of each butterfly of the FFT is divided by two. When you do not select this parameter, the block does not scale the output.

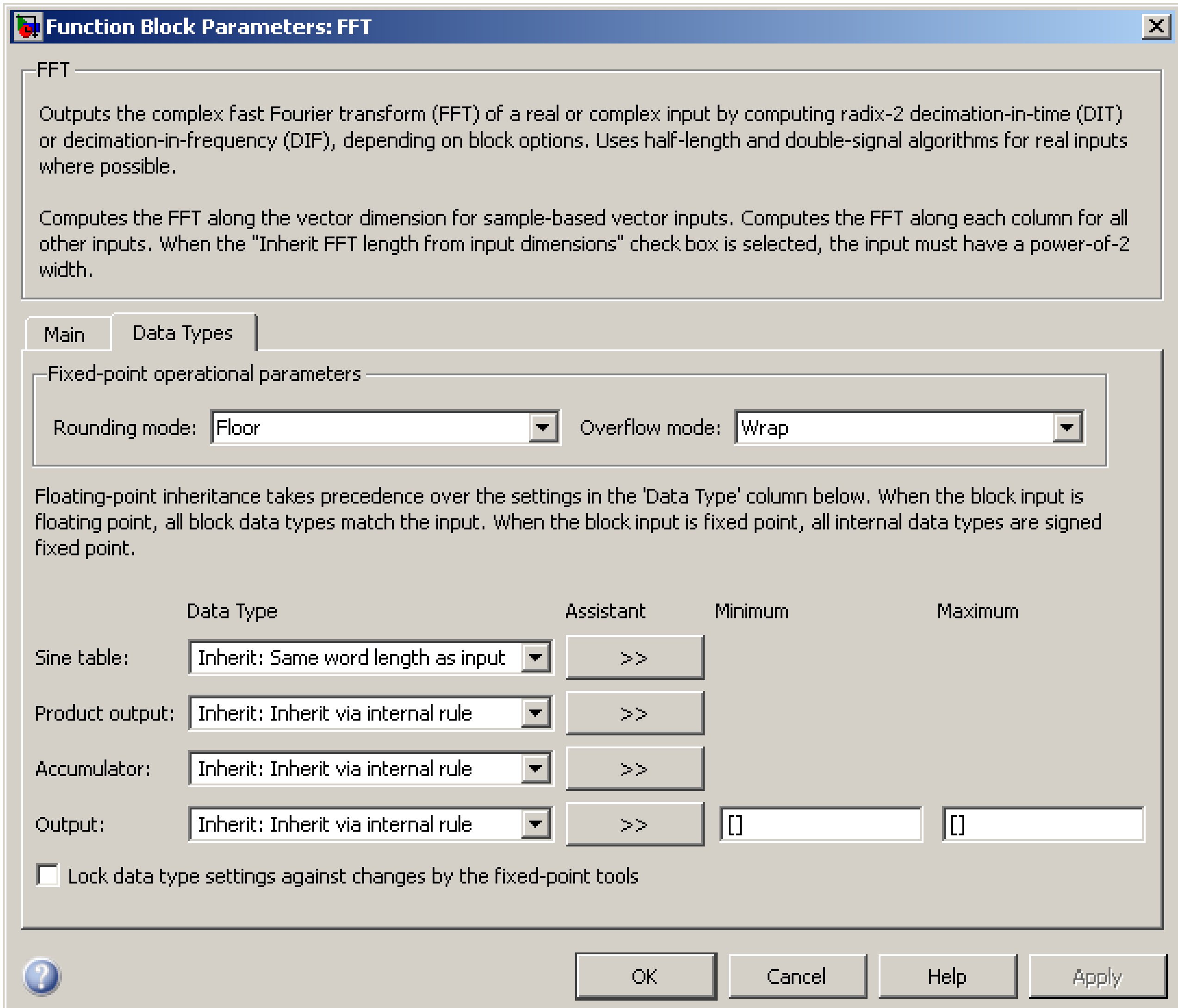
Inherit FFT length from input dimensions

Select to inherit the FFT length from the input dimensions. When you select this parameter, the input length *P* must be a power of two. When you do not select this parameter, the **FFT length** parameter becomes available.

FFT length

Specify a power-of-two FFT length. This parameter becomes available only when you do not select the **Inherit FFT length from input dimensions** parameter.

The **Data Types** pane of the FFT block dialog appears as follows.


Function Block Parameters: FFT
FFT
Outputs the complex fast Fourier transform (FFT) of a real or complex input by computing radix-2 decimation-in-time (DIT) or decimation-in-frequency (DIF), depending on block options. Uses half-length and double-signal algorithms for real inputs where possible.
Computes the FFT along the vector dimension for sample-based vector inputs. Computes the FFT along each column for all other inputs. When the "Inherit FFT length from input dimensions" check box is selected, the input must have a power-of-2 width.
Main
Data Types
Fixed-point operational parameters
Rounding mode:
Floor
Overflow mode:
Wrap
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input. When the block input is fixed point, all internal data types are signed fixed point.
Data Type
Assistant
Minimum
Maximum
Sine table:
Inherit: Same word length as input
>>
Product output:
Inherit: Inherit via internal rule
>>
Accumulator:
Inherit: Inherit via internal rule
>>
Output:
Inherit: Inherit via internal rule
>>
[]
[]
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Rounding mode

Select the rounding mode for fixed-point operations. The sine table values do not obey this parameter; instead, they always round to `Nearest`.

Overflow mode

Select the overflow mode for fixed-point operations. The sine table values do not obey this parameter; instead, they are always saturated.

Sine table data type

Choose how you specify the word length of the values of the sine table. The fraction length of the sine table values always equals the word length minus one. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same word length as input`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16)`

The sine table values do not obey the **Rounding mode** and **Overflow mode** parameters; instead, they are always saturated and rounded to `Nearest`.

Product output data type

Specify the product output data type. See "Fixed-Point Data Types" on page 2-497 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Accumulator data type

Specify the accumulator data type. See "Fixed-Point Data Types" on page 2-497 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Output data type

Specify the output data type. See "Fixed-Point Data Types" on page 2-497 for illustrations depicting the use of the output data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`.

  When you select `Inherit:  Inherit via internal rule`, the block calculates the output word length and fraction length automatically. The internal rule first calculates an ideal output word length and fraction length using the following equations:

$$WL_{ideal\ output} = WL_{input} + floor(\log_2(FFT\ length - 1)) + 1$$

$$FL_{ideal\ output} = FL_{input}$$

Using these ideal results, the internal rule then selects word lengths and fraction lengths that are appropriate for your hardware. For more information, see "Inherit via Internal Rule".

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

References

[1] Orfanidis, S. J. *Introduction to Signal Processing*. Upper Saddle River, NJ: Prentice Hall, 1996, p. 497.

[2] Proakis, John G. and Dimitris G. Manolakis. *Digital Signal Processing*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 1996.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

DCT	Signal Processing Blockset
IFFT	Signal Processing Blockset
Pad	Signal Processing Blockset
`fft`	MATLAB
`ifft`	MATLAB
`bitrevorder`	Signal Processing Toolbox

Filter Realization Wizard

Purpose

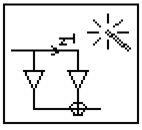
Construct filter realizations using Digital Filter block or Sum, Gain, and Delay blocks

Library

Filtering / Filter Implementations

`dsparch4`

Description



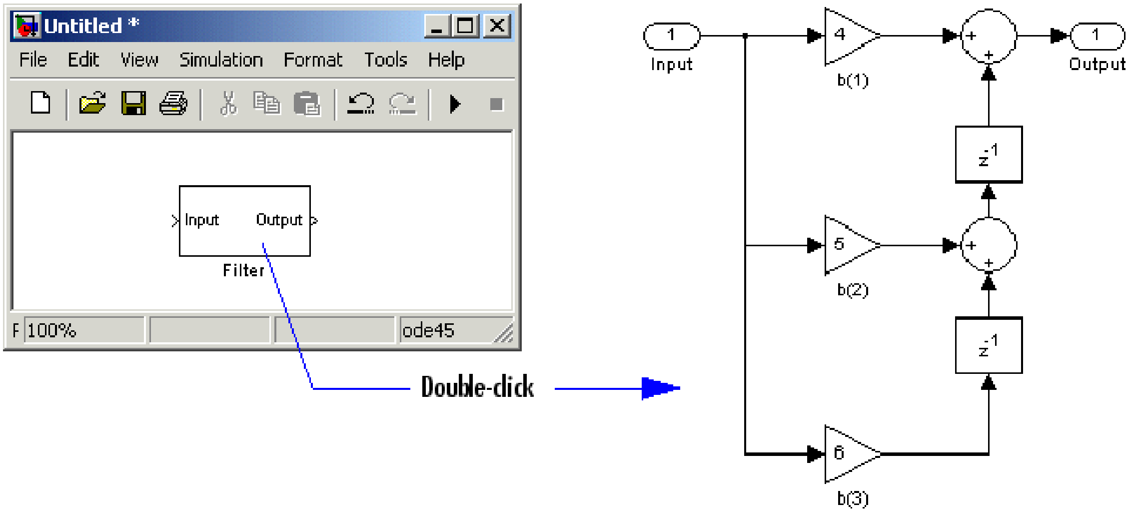
Note Use this block to implement fixed-point or floating-point digital filters using Sum, Gain, and Delay blocks or the Digital Filter block. You can either design a filter by using the block's filter design and analysis parameters, or import the coefficients of a filter you have designed elsewhere.

The following blocks also implement digital filters, but serve slightly different purposes:

- Digital Filter — Use to implement floating-point or fixed-point filters that you have already designed
- Digital Filter Design — Use to design, analyze, and then implement floating-point filters.

The Filter Realization Wizard is a tool for automatically implementing a digital filter. You must specify a filter, its structure, and the data types for the filter's inputs, outputs, and computations. The filter can support double-precision, single-precision, or fixed-point data types.

The Filter Realization Wizard can implement a digital filter in one of two ways. It can use a Digital Filter block, or it can create a subsystem block that implements the specified filter using Sum, Gain, and Delay blocks. If the Filter Realization Wizard creates a Digital Filter block, double-click the block to open the Block Parameters: Filter dialog box. If it creates a subsystem, double-click the subsystem block to see the filter implementation as shown in the figure below.



The subsystem block applies the specified filter to any sample-based input signal, or any frame-based row vector signal, and outputs the result. For more information about filter implementation, see "Specifying the Filter Implementation" on page 2-512.

The parameters of the Filter Realization Wizard are a part of a larger GUI, the Filter Design and Analysis Tool (`fdatool`), from the Signal Processing Toolbox product. You can use the tools in FDATool to design and analyze your filter, and then use the Filter Realization Wizard parameters to implement the filter in your models.

Sections of This Reference Page

- "References" on page 2-517
- "Supported Data Types" on page 2-517
- "See Also" on page 2-517

Valid Inputs and Corresponding Outputs

When the Filter Realization Wizard implements the specified filter by creating a new subsystem block, the block applies the specified filter to an input signal and outputs the result.

Valid Inputs

The subsystem block accepts inputs that are

- Sample-based vectors and matrices
- Frame-based row vectors (nonrecursive structures only)

Corresponding Outputs

The output of the subsystem block has the same dimensions and frame status as the input.

What Is Considered an Independent Channel

The subsystem block treats each *element* of a vector or matrix as an independent channel.

Specifying the Filter and Its Data Type Support

To specify a purely double-precision filter, you can either design a filter using the **Design Filter** panel, or import a filter using the **Import Filter** panel. (You can import `dfilt` filter objects as well as vectors of filter coefficients designed using Signal Processing Toolbox functions and Filter Design Toolbox functions.)

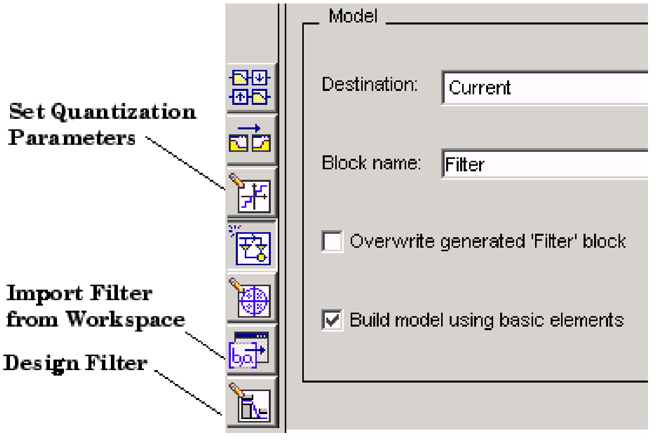
You can also specify a fixed-point filter or a single-precision filter. You can specify such filters by using the **Set Quantization Parameters** panel, which requires the Filter Design Toolbox product.

Note *Running* a model containing implementations of fixed-point filters requires the Simulink Fixed Point product, but you can still edit models containing such filter implementations without it. See the Simulink Fixed Point documentation for more information.

See the following topics to learn how to use the panels to specify your filter:

- For more information on the **Design Filter** panel, see "FDATool: A Filter Design and Analysis GUI" in the Signal Processing Toolbox documentation.
- For more information on the **Import Filter** panel, see "Importing a Filter Design" in the Signal Processing Toolbox documentation.
- For more information on the **Set Quantization Parameters** panel, see "Switching FDATool to Quantization Mode" in the Filter Design Toolbox documentation.

To open a panel, click the appropriate button in the lower-left corner of FDATool.



Supported Filter Structures

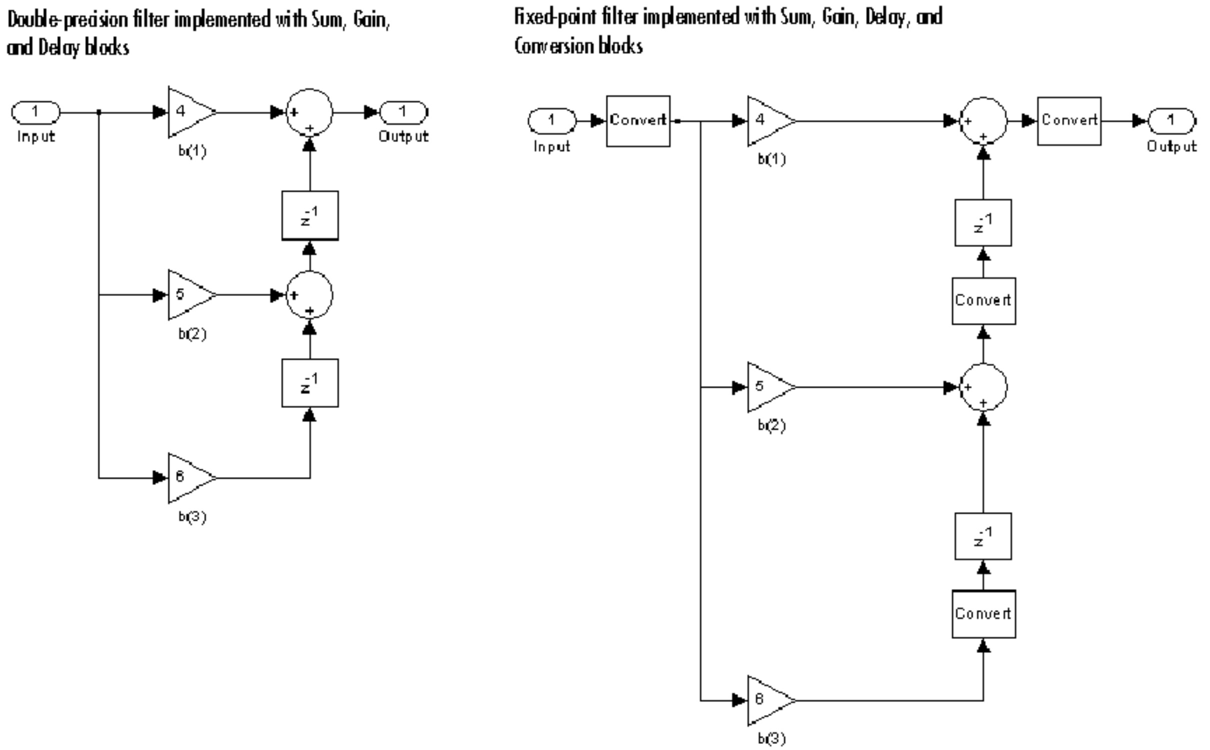
The Filter Realization Wizard supports the following structures:

- Direct form I
- Direct form I, second-order sections
- Direct form I transposed
- Direct form I transposed, second-order sections
- Direct form II
- Direct form II, second-order sections
- Direct form II transposed
- Direct form II transposed, second-order sections
- Direct form FIR
- Direct form FIR transposed
- Direct form symmetric FIR
- Direct form antisymmetric FIR
- Lattice all-pass
- Lattice AR
- Lattice ARMA
- Lattice MA for maximum phase
- Lattice MA for minimum phase
- Cascade
- Parallel

Specifying the Filter Implementation

You can determine how the Filter Realization Wizard models the specified filter using the **Build model using basic elements** check box. When you select this check box, the Filter Realization Wizard creates a subsystem block that implements your filter using Sum, Gain, and Delay blocks. When you clear this check box, the Filter Realization Wizard uses a Digital Filter block to implement your filter. The **Build model using basic elements** check box is only available when your filter can be implemented using a Digital Filter block.

If you have Signal Processing Blockset software, Signal Processing Toolbox software, and Filter Design Toolbox software installed on your system, the Filter Realization Wizard can generate a subsystem that represents either a double-precision or fixed-point filter. You must install the Simulink Fixed Point product to simulate a fixed-point filter. You can still edit the blocks used to implement the filter without installing the Simulink Fixed Point product.



Implementations of Double-Precision and Fixed-Point Filters

Corresponding Method for dfilt

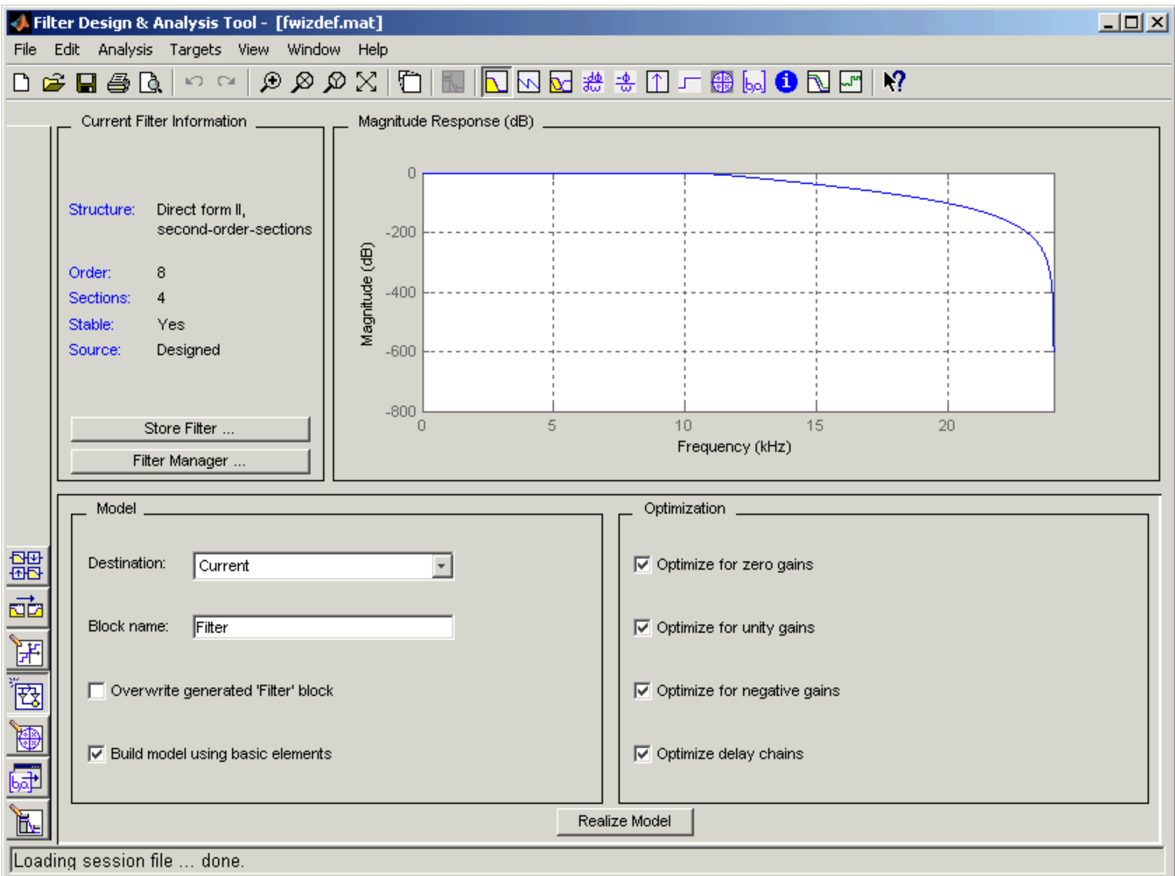
The Signal Processing Toolbox `dfilt` (digital filter) object in has a method, `realizemdl`, that allows you to access the capabilities of the Filter Realization Wizard from the command line.

For more information about the `realizemdl` method, see the following:

- The topic on “Methods” in the `dfilt` reference page in the Signal Processing Toolbox documentation
- The `realizemdl` reference page in the Filter Design Toolbox documentation

Dialog Box

Note The following parameters for the Filter Realization Wizard are in the **Realize Model** pane of the Filter Design and Analysis Tool (FDATool) GUI. To open different panels of FDATool, click the different buttons at the lower-left corner. For more information about relevant panels, see "Specifying the Filter and Its Data Type Support" on page 2-510.



Destination
Specify where the new filter block should be created. This can be in a new model or in the current (most recently selected) model.

Block Name
Enter the name of the new filter block.

Overwrite generated block "Filter" block
When selected, the block overwrites any filter block in the current model with the name specified in the **Block Name** parameter. This parameter is enabled when the **Destination** parameter is set to `Current`.

Build model using basic elements
Select this check box to implement your filter using Sum, Gain, and Delay blocks. Clear this check box to implement your filter using the Digital Filter block. This parameter is only available when your filter can be modeled using the Digital Filter block.

Note that when your filter is implemented using Sum, Gain, and Delay blocks, inputs to the filter must be sample based.

Optimize for zero gains
When selected, the block removes zero-gain paths from the filter structure. For an example, see "Optimizing the Filter Structure".

Optimize for unity gains
When selected, the block substitutes gains equal to 1 with a wire (short circuit). For an example, see "Optimizing the Filter Structure".

Optimize for negative gains
When selected, the block substitutes gains equal to -1 with a wire (short circuit), and changes the corresponding sums to subtractions. For an example, see "Optimizing the Filter Structure".

Optimize delay chains

When selected, the block substitutes any delay chains made up of *n* unit delays with a single delay by *n*. For an example, see "Optimizing the Filter Structure".

Realize Model

Click to create a subsystem block that implements the specified filter using Sum, Gain, and Delay blocks. To see the filter implementation, double-click the subsystem block. The subsystem block applies the specified filter to any sample-based input signal or frame-based row vector signal, and outputs the result.

Note For more information about relevant parameters in other panels of FDATool, see "Specifying the Filter and Its Data Type Support" on page 2-510.

References

Oppenheim, A. V. and R. W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

Proakis, J. and D. Manolakis. *Digital Signal Processing*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

Supported Data Types

- Double-precision floating point
- Single-precision floating point — Supported only when you install the following products: Filter Design Toolbox and Simulink Fixed Point
- Fixed point (signed and unsigned) — Supported only when you install the following products: Filter Design Toolbox, "Simulink Fixed Point", and Fixed-Point Toolbox

See Also

Digital Filter	Signal Processing Blockset
Digital Filter Design	Signal Processing Blockset
`filter`	Filter Design Toolbox

`realizemdl`	Filter Design Toolbox
`dfilt`	Signal Processing Toolbox
`filter`	Signal Processing Toolbox

- “Filters” — Examples of when and how to use Signal Processing Blockset filtering blocks
- “Choosing Between Filter Design Blocks”

Purpose

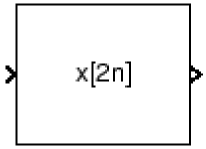
Filter and downsample input signals

Library

Filtering / Multirate Filters

`dspmlti4`

Description



The FIR Decimation block resamples the discrete-time input at a rate *K* times slower than the input sample rate, where the integer *K* is specified by the **Decimation factor** parameter. This process consists of two steps:

- The block filters the input data using a direct-form FIR filter.
- The block downsamples the filtered data to a lower rate by discarding *K*–1 consecutive samples following every sample retained.

The FIR Decimation block implements the above FIR filtering and downsampling steps together using a polyphase filter structure, which is more efficient than straightforward filter-then-decimate algorithms. See Fliege [1] for more information.

The **FIR filter coefficients** parameter specifies the numerator coefficients of the FIR filter transfer function *H*(*z*).

$$H(z) = B(z) = b_1 + b_2 z^{-1} + \ldots + b_m z^{-(m-1)}$$

The length-*m* coefficient vector, `[b(1) b(2) ...  b(m)]`, can be generated by one of the filter design functions in Signal Processing Toolbox software, such as the `fir1` function used in Example 1 below. The filter should be lowpass with normalized cutoff frequency no greater than 1/*K*. All filter states are internally initialized to zero.

The FIR Decimation block supports real and complex floating-point and fixed-point inputs, except for complex unsigned fixed-point inputs. This block supports triggered subsystems when you select `Maintain input frame rate` for the **Framing** parameter.

Sample-Based Operation

An M-by-N sample-based matrix input is treated as $M*N$ independent channels, and the block decimates each channel over time. The output sample period is K times longer than the input sample period ($T_{so} = KT_{si}$), and the input and output sizes are identical.

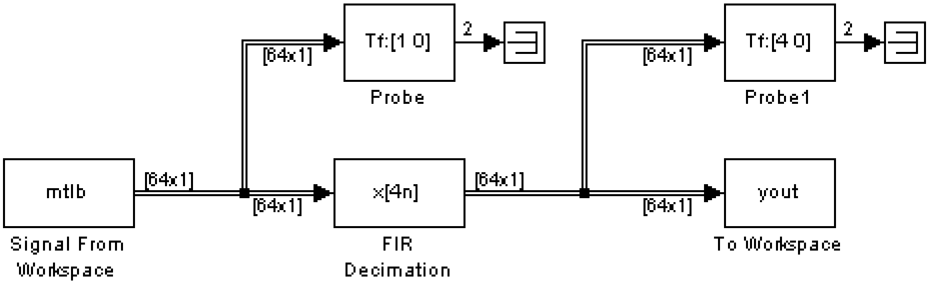
Frame-Based Operation

An M_i-by-N frame-based matrix input is treated as N independent channels, and the block decimates each channel over time. The **Framing** parameter determines how the block adjusts the rate at the output to accommodate the reduced number of samples. There are two available options:

- `Maintain input frame size`

  The block generates the output at the decimated rate by using a proportionally longer frame *period* at the output port than at the input port. For decimation by a factor of K, the output frame period is K times longer than the input frame period ($T_{fo} = KT_{fi}$), but the input and output frame sizes are equal.

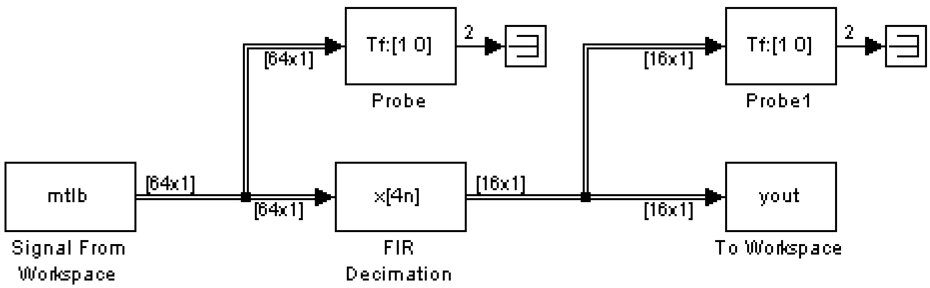
  The following model shows a single-channel input with a frame period of 1 second (**Sample time** = `1/64` and **Samples per frame** = `64` in the Signal From Workspace block) being decimated by a factor of 4 to a frame period of 4 seconds. The input and output frame sizes are identical.



- `Maintain input frame rate`

The block generates the output at the decimated rate by using a proportionally smaller frame size than the input. For decimation by a factor of K, the output frame size is K times smaller than the input frame size ($M_o = M_i/K$), but the input and output frame rates are equal. The input frame size, M_i, must be a multiple of the decimation factor, K.

The following model shows a single-channel input of frame size 64 being decimated by a factor of 4 to a frame size of 16. The block's input and output frame rates are identical.



Latency

The FIR Decimation block has *zero-tasking latency* for all single-tasking and multitasking cases, except one:

Sample-Based Latency	Frame-Based Latency — Maintain input frame rate	Frame-Based Latency — Maintain input frame size
None	None	One frame (M_i samples)

Zero-tasking latency means that the block propagates the first filtered input sample (received at t=0) as the first output sample, followed by filtered input samples K+1, 2K+1, and so on.

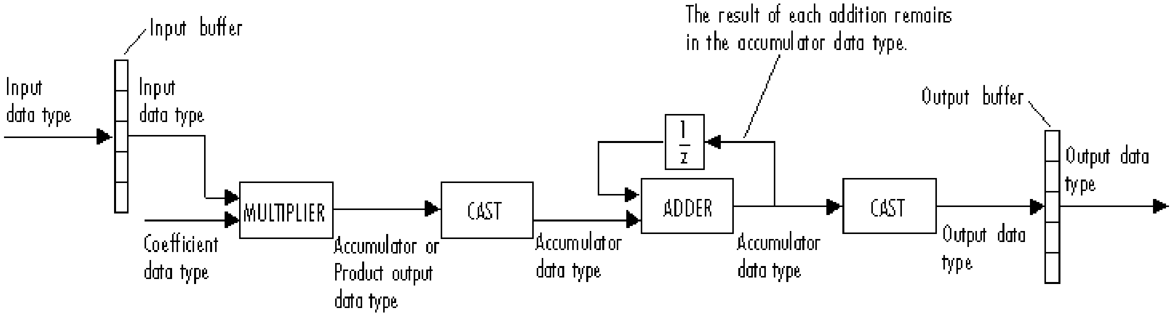
In cases of *one-frame latency*, the value of the first M_i output rows is defined by the **Output buffer initial conditions** parameter, where M_i

is the input frame size. The default value of the **Output buffer initial conditions** parameter is 0, but you can enter a matrix containing one value for each channel, or a scalar to be applied to all signal channels. The first filtered input sample (first filtered row of the input matrix) appears in the output as sample M_i+1, followed by filtered input samples K+1, 2K+1, and so on.

Note For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

Fixed-Point Data Types

The following diagram shows the data types used within the FIR Decimation block for fixed-point signals.



You can set the coefficient, product output, accumulator, and output data types in the block dialog as discussed in "Dialog Box" on page 2-525. This diagram shows that data is stored in the input buffer in the same data type and scaling as the input. Filtered data is stored in the output buffer in the output data type and scaling that you set in the block dialog. Any initial conditions are also stored in the output buffer in the output data type and scaling you set in the block dialog.

The output of the multiplier is in the product output data type when at least one of the inputs to the multiplier is real. When both of the

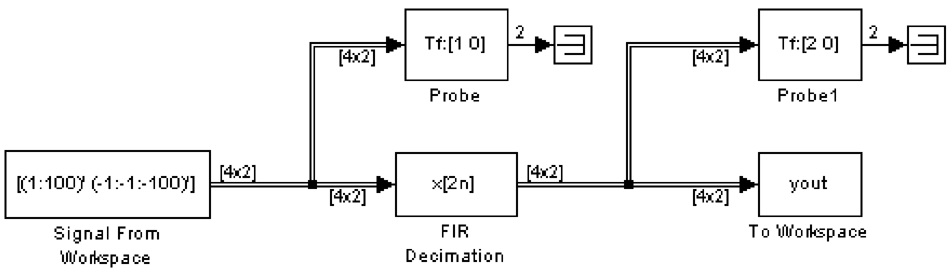
inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see “Multiplication Data Types”.

Note When the block input is fixed point, all internal data types are signed fixed point.

Examples

Example 1

Construct the following frame-based model.



Adjust the block parameters as follows:

- Configure the Signal From Workspace block to generate a two-channel signal with frame size of 4 and sample period of 0.25. This represents an output frame period of 1 (0.25*4). The first channel should contain the positive ramp signal 1, 2, ..., 100, and the second channel should contain the negative ramp signal -1, -2, ..., -100.
  - **Signal** = `[(1:100)' (-1:-1:-100)']`
  - **Sample time** = `0.25`
  - **Samples per frame** = `4`
- Configure the FIR Decimation block to decimate the two-channel input by decreasing the output frame rate by a factor of 2 relative to

the input frame rate. Use a third-order filter with normalized cutoff frequency, f_{n0}, of 0.25. (Note that f_{n0} satisfies $f_{n0} \leq 1/K$.)

- **FIR filter coefficients** = `fir1(3,0.25)`
- **Downsample factor** = `2`
- **Framing** = `Maintain input frame size`

The filter coefficient vector generated by `fir1(3,0.25)` is

```
[0.0386 0.4614 0.4614 0.0386]
```

or, equivalently,

$$H(z) = B(z) = 0.0386 + 0.04614z^{-1} + 0.04614z^{-2} + 0.0386z^{-3}$$

- Configure the Probe blocks by clearing the **Probe width**, **Probe complex signal**, and **Probe signal dimensions** check boxes (if desired).

This model is multirate because there are at least two distinct sample rates, as shown by the two Probe blocks. To run this model in the Simulink multitasking mode, make the following settings in the **Solver** pane of the Configuration Parameters dialog box:

- From the **Type** list, select `Fixed-step`.
- From the **Solver** list, select `Discrete (no continuous states)`.
- From the **Tasking mode for periodic sample times** list, select `MultiTasking`.
- Set the **Stop time** to `30`.

Run the model and look at the output, `yout`. The first few samples of each channel are shown below.

```
yout =
         0         0
         0         0
```

```
         0          0
         0          0
    0.0386    -0.0386
    1.5000    -1.5000
    3.5000    -3.5000
    5.5000    -5.5000
    7.5000    -7.5000
    9.5000    -9.5000
   11.5000   -11.5000
```

Since this is a frame-based multirate model, the first four (M_i) output rows are zero. The first filtered input matrix row appears in the output as sample 5 (that is, sample M_i+1).

Example 2

The `doc_polyphasedec` model illustrates the underlying polyphase implementations of the FIR Decimation block. Run the model and view the results on the scope. The output of the FIR Decimation block is the same as the output of the Polyphase Decimation Filter block.

Example 3

The `doc_mrf_nlp` model illustrates the use of the FIR Decimation block in a number of multistage multirate filters.

Dialog Box

Coefficient Source

The FIR Decimation block can operate in two different modes. Select the mode in the **Coefficient source** group box. If you select

- **Dialog parameters**, you enter information about the filter such as structure and coefficients in the block mask.
- **Multirate filter object (MFILT)**, you specify the filter using a Filter Design Toolbox `mfilt` object.

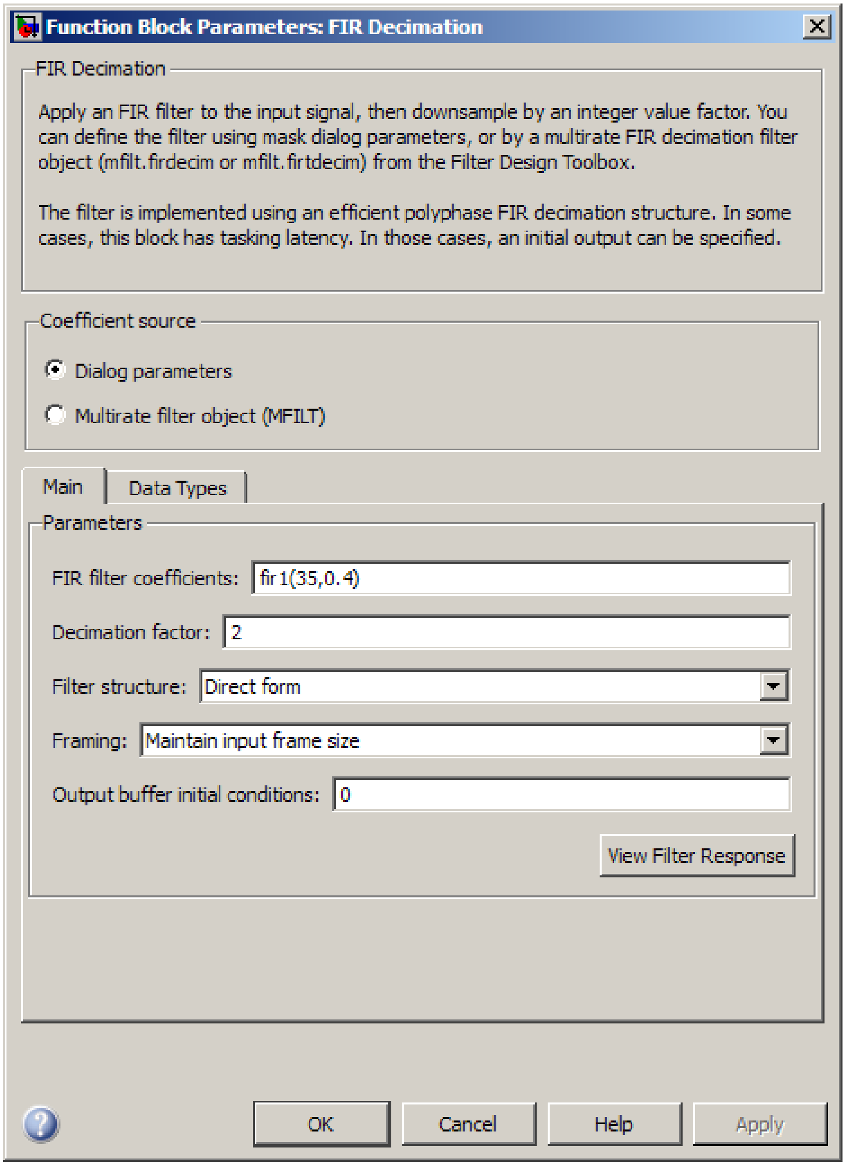
Different items appear on the FIR Decimation block dialog depending on whether you select **Dialog parameters** or **Multirate filter object**

(MFILT) in the **Coefficient source** group box. See the following sections for details:

- "Specify Filter Characteristics in Dialog" on page 2-526
- "Specify Multirate Filter Object" on page 2-534

Specify Filter Characteristics in Dialog

The **Main** pane of the FIR Decimation block dialog appears as follows when **Dialog parameters** is selected in the **Coefficient source** group box.


Function Block Parameters: FIR Decimation
FIR Decimation
Apply an FIR filter to the input signal, then downsample by an integer value factor. You can define the filter using mask dialog parameters, or by a multirate FIR decimation filter object (mfilt.firdecim or mfilt.firtdecim) from the Filter Design Toolbox.
The filter is implemented using an efficient polyphase FIR decimation structure. In some cases, this block has tasking latency. In those cases, an initial output can be specified.
Coefficient source
Dialog parameters
Multirate filter object (MFILT)
Main
Data Types
Parameters
FIR filter coefficients:
fir1(35,0.4)
Decimation factor:
2
Filter structure:
Direct form
Framing:
Maintain input frame size
Output buffer initial conditions:
0
View Filter Response
OK
Cancel
Help
Apply

FIR filter coefficients
Specify the lowpass FIR filter coefficients, in descending powers of *z*.

Decimation factor
Specify the integer factor, *K*, by which to decrease the sample rate of the input sequence.

Filter Structure
Choose whether to implement a `Direct form` or `Direct form transposed` filter.

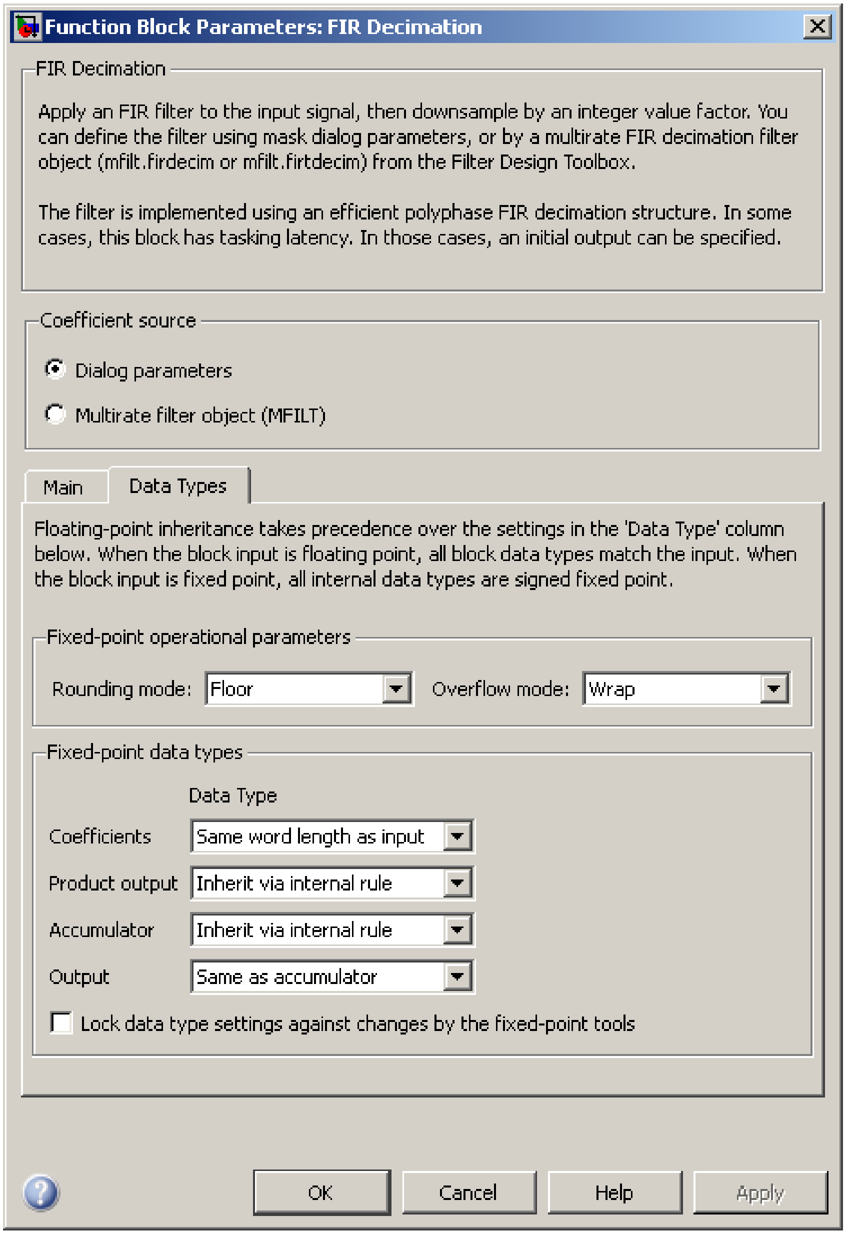
Framing
For frame-based operation, specify the method by which to implement the decimation; reduce the output frame rate, or reduce the output frame size. This parameter cannot be set to `Maintain input frame rate` for sample-based signals.

Output buffer initial conditions
For the case of *one-frame latency*, this parameter specifies the output at the output port until the first filtered input sample is available. The default value of this parameter is `0`, but you can enter a matrix containing one value for each channel, or a scalar to be applied to all signal channels.

View filter response
This button opens the Filter Visualization Tool (`fvtool`) from the Signal Processing Toolbox product and displays the filter response of the filter defined in the block. For more information on FVTool, see the Signal Processing Toolbox documentation.

Note This button is only available when the Filter Design Toolbox product is installed. If you specify a filter in the **Multirate filter variable** parameter, you must apply the filter by clicking the **Apply** button before using the **View filter response** button.

The **Data Types** pane of the FIR Decimation block dialog appears as follows when **Dialog parameters** is specified in the **Coefficient source** group box.


Function Block Parameters: FIR Decimation
FIR Decimation
Apply an FIR filter to the input signal, then downsample by an integer value factor. You can define the filter using mask dialog parameters, or by a multirate FIR decimation filter object (mfilt.firdecim or mfilt.firtdecim) from the Filter Design Toolbox.
The filter is implemented using an efficient polyphase FIR decimation structure. In some cases, this block has tasking latency. In those cases, an initial output can be specified.
Coefficient source
Dialog parameters
Multirate filter object (MFILT)
Main
Data Types
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input. When the block input is fixed point, all internal data types are signed fixed point.
Fixed-point operational parameters
Rounding mode:
Floor
Overflow mode:
Wrap
Fixed-point data types
Data Type
Coefficients
Same word length as input
Product output
Inherit via internal rule
Accumulator
Inherit via internal rule
Output
Same as accumulator
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Rounding mode
Select the rounding mode for fixed-point operations. The filter coefficients do not obey this parameter; they always round to `Nearest`.

Overflow mode
Select the overflow mode for fixed-point operations. The filter coefficients do not obey this parameter; they are always saturated.

Coefficients
Choose how you specify the word length and the fraction length of the filter coefficients:

- When you select `Same word length as input`, the word length of the filter coefficients match that of the input to the block. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.
- When you select `Specify word length`, you can enter the word length of the coefficients, in bits. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the coefficients, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the coefficients. This block requires power-of-two slope and a bias of zero.

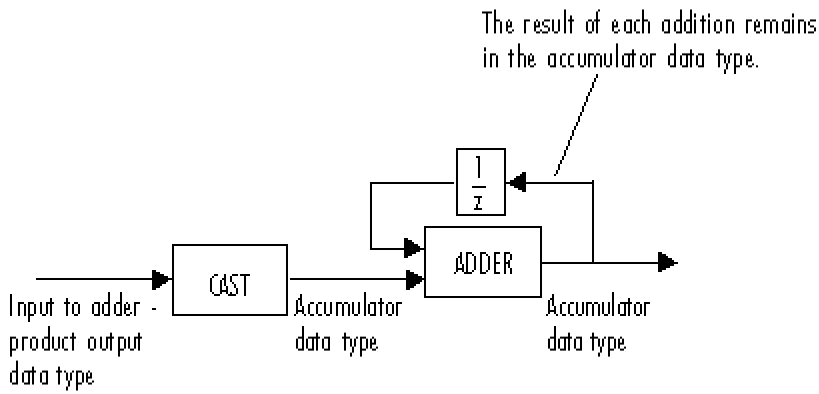
The filter coefficients do not obey the **Rounding mode** and the **Overflow mode** parameters; they are always saturated and rounded to `Nearest`.

Product output
Use this parameter to specify how you would like to designate the product output word and fraction lengths. See "Fixed-Point

Data Types" on page 2-522 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block:

- When you select `Inherit via internal rule`, the product output word length and fraction length are calculated automatically. For information about how the product output word and fraction lengths are calculated when an internal rule is used, see "Inherit via Internal Rule".
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator



As depicted in this graphic, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how you would like to designate this accumulator word and fraction lengths.

You also use this parameter to specify the accumulator word and fraction lengths resulting from a complex-complex multiplication in the block. See “Multiplication Data Types” for more information.

- When you select `Inherit via internal rule`, the accumulator word length and fraction length are calculated automatically. For information about how the accumulator word and fraction lengths are calculated when an internal rule is used, see “Inherit via Internal Rule”.
- When you select `Same as product output`, these characteristics match those of the product output.
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Choose how you specify the output word length and fraction length:

- When you select `Same as accumulator`, these characteristics match those of the accumulator.

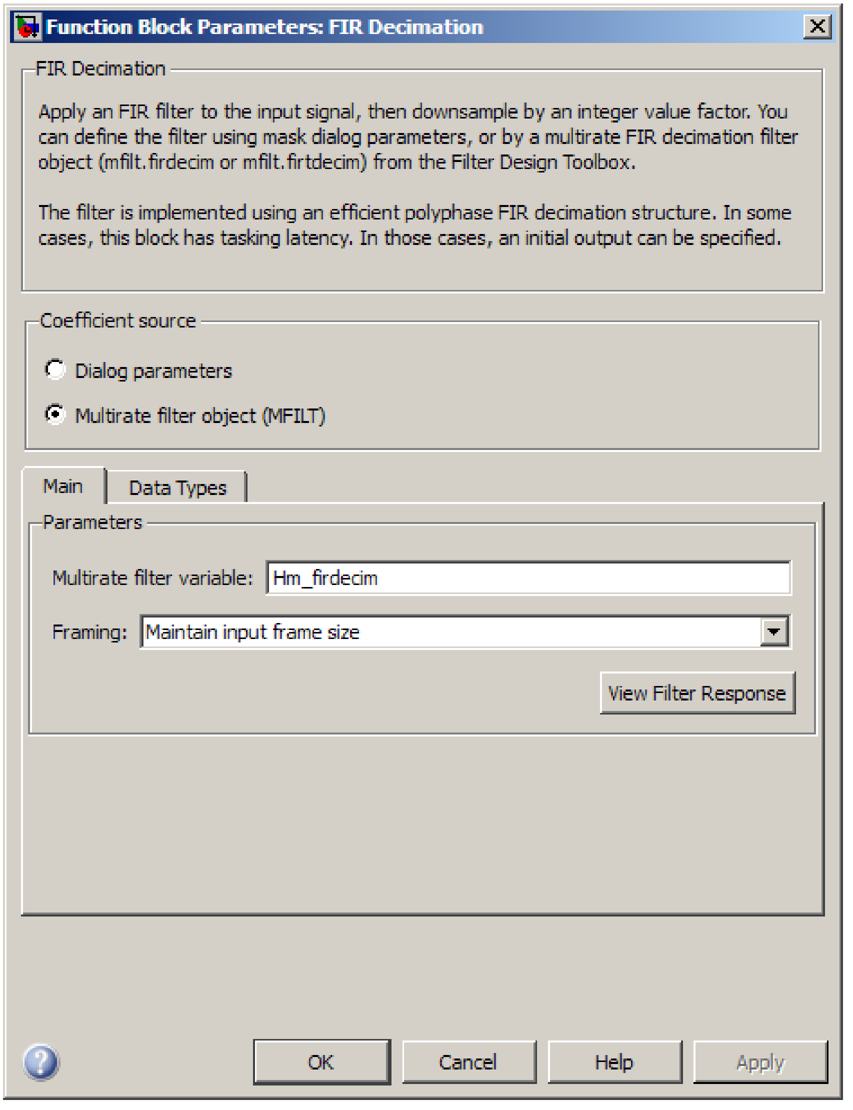
  A special case occurs when `Inherit via internal rule` is specified for **Accumulator**, and block inputs and coefficients are complex. In that case, the output word length is one less than the accumulator word length.
- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Specify Multirate Filter Object

The **Main** pane of the FIR Decimation block dialog appears as follows when **Multirate filter object (MFILT)** is specified in the **Coefficient source** group box.


Function Block Parameters: FIR Decimation
FIR Decimation
Apply an FIR filter to the input signal, then downsample by an integer value factor. You can define the filter using mask dialog parameters, or by a multirate FIR decimation filter object (mfilt.firdecim or mfilt.firtdecim) from the Filter Design Toolbox.
The filter is implemented using an efficient polyphase FIR decimation structure. In some cases, this block has tasking latency. In those cases, an initial output can be specified.
Coefficient source
Dialog parameters
Multirate filter object (MFILT)
Main
Data Types
Parameters
Multirate filter variable:
Hm_firdecim
Framing:
Maintain input frame size
View Filter Response
OK
Cancel
Help
Apply

Multirate filter variable

Specify the multirate filter object (`mfilt`) that you would like the block to implement. You can do this in one of three ways:

- You can fully specify the `mfilt` object in the block mask.
- You can enter the variable name of a `mfilt` object that is defined in any workspace.
- You can enter a variable name for a `mfilt` object that is not yet defined, as shown in the default value.

For more information on creating `mfilt` objects, see the `mfilt` function reference page in the Filter Design Toolbox documentation.

Framing

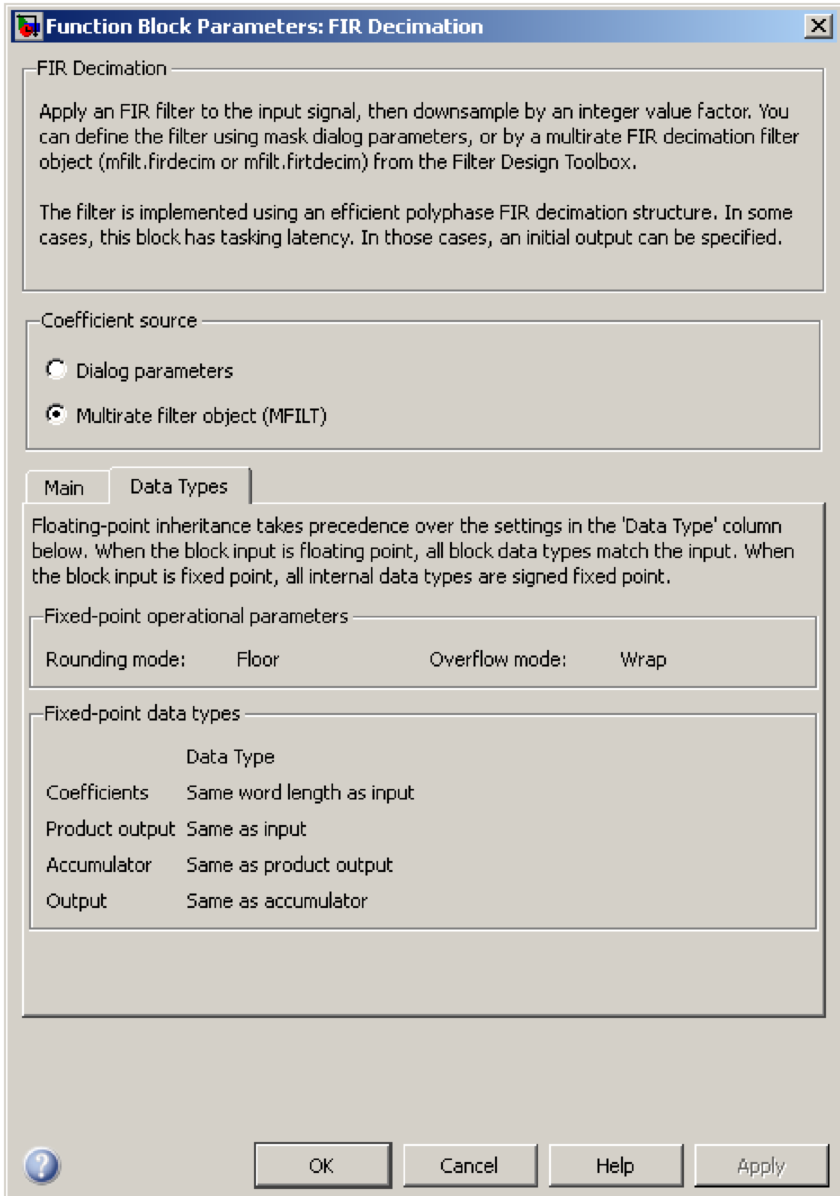
For frame-based operation, specify the method by which to implement the decimation; reduce the output frame rate, or reduce the output frame size. This parameter cannot be set to `Maintain input frame rate` for sample-based signals.

View filter response

This button opens the Filter Visualization Tool (`fvtool`) from the Signal Processing Toolbox product and displays the filter response of the `mfilt` object specified in the **Multirate filter variable** parameter. For more information on FVTool, see the Signal Processing Toolbox documentation.

Note This button is only available when the Filter Design Toolbox product is installed. If you specify a filter in the **Multirate filter variable** parameter, you must apply the filter by clicking the **Apply** button before using the **View filter response** button.

The **Data Types** pane of the FIR Decimation block dialog appears as follows when **Multirate filter object (MFILT)** is specified in the **Coefficient source** group box.


Function Block Parameters: FIR Decimation
FIR Decimation
Apply an FIR filter to the input signal, then downsample by an integer value factor. You can define the filter using mask dialog parameters, or by a multirate FIR decimation filter object (mfilt.firdecim or mfilt.firtdecim) from the Filter Design Toolbox.
The filter is implemented using an efficient polyphase FIR decimation structure. In some cases, this block has tasking latency. In those cases, an initial output can be specified.
Coefficient source
Dialog parameters
Multirate filter object (MFILT)
Main
Data Types
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input. When the block input is fixed point, all internal data types are signed fixed point.
Fixed-point operational parameters
Rounding mode: Floor
Overflow mode: Wrap
Fixed-point data types
Data Type
Coefficients Same word length as input
Product output Same as input
Accumulator Same as product output
Output Same as accumulator
OK
Cancel
Help
Apply

The fixed-point settings of the filter object specified on the **Main** pane are displayed on the **Data Types** pane. You cannot change these settings directly on the block mask. To change the fixed-point settings you must edit the filter object directly.

For more information on multirate filter objects, see the `mfilt` function reference page in the Filter Design Toolbox documentation.

References

[1] Fliege, N. J. *Multirate Digital Signal Processing: Multirate Systems, Filter Banks, Wavelets*. West Sussex, England: John Wiley & Sons, 1994.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Downsample	Signal Processing Blockset
FIR Interpolation	Signal Processing Blockset
FIR Rate Conversion	Signal Processing Blockset
`decimate`	Signal Processing Toolbox

`fir1`	Signal Processing Toolbox
`fir2`	Signal Processing Toolbox
`firls`	Signal Processing Toolbox

Purpose

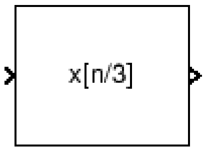
Upsample and filter input signals

Library

Filtering / Multirate Filters

`dspmlti4`

Description



The FIR Interpolation block resamples the discrete-time input at a rate L times faster than the input sample rate, where the integer L is specified by the **Interpolation factor** parameter. This process consists of two steps:

- The block upsamples the input to a higher rate by inserting L-1 zeros between samples.
- The block filters the upsampled data with a direct-form FIR filter.

The FIR Interpolation block implements the above upsampling and FIR filtering steps together using a polyphase filter structure, which is more efficient than straightforward upsample-then-filter algorithms. See N.J. Fliege, *Multirate Digital Signal Processing: Multirate Systems, Filter Banks, Wavelets* for more information.

The **FIR filter coefficients** parameter specifies the numerator coefficients of the FIR filter transfer function $H(z)$.

$$H(z) = B(z) = b_1 + b_2 z^{-1} + \ldots + b_m z^{-(m-1)}$$

The coefficient vector, `[b(1) b(2) ...  b(m)]`, can be generated by one of the Signal Processing Toolbox filter design functions (such as `fir1`), and should have a length greater than the interpolation factor (m>L). The filter should be lowpass with normalized cutoff frequency no greater than 1/L. All filter states are internally initialized to zero.

The FIR Interpolation block supports real and complex floating-point and fixed-point inputs except for complex unsigned fixed-point inputs. This block supports triggered subsystems when you select `Maintain input frame rate` for the **Framing** parameter.

Sample-Based Operation

An M-by-N sample-based matrix input is treated as $M*N$ independent channels, and the block interpolates each channel over time. The output sample period is L times shorter than the input sample period (T_{so} = T_{si}/L), and the input and output sizes are identical.

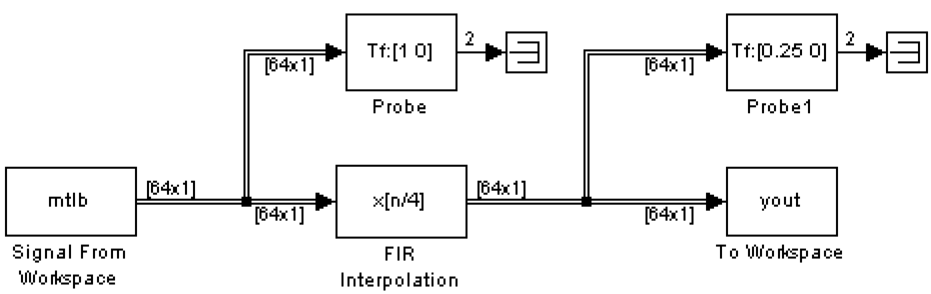
Frame-Based Operation

An M_i-by-N frame-based matrix input is treated as N independent channels, and the block interpolates each channel over time. The **Framing** parameter determines how the block adjusts the rate at the output to accommodate the added samples. There are two available options:

- `Maintain input frame size`

  The block generates the output at the interpolated rate by using a proportionally shorter frame *period* at the output port than at the input port. For interpolation by a factor of L, the output frame period is L times shorter than the input frame period (T_{fo} = T_{fi}/L), but the input and output frame sizes are equal.

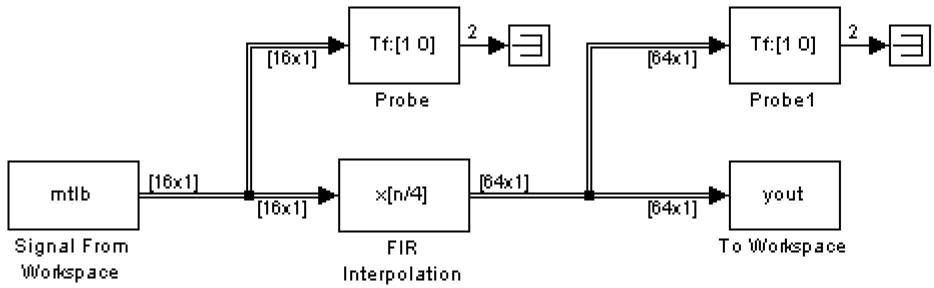
  The example below shows a single-channel input with a frame period of 1 second (**Sample time** = `1/64` and **Samples per frame** = `64` in the Signal From Workspace block) being interpolated by a factor of 4 to a frame period of 0.25 second. The input and output frame sizes are identical.



- `Maintain input frame rate`

The block generates the output at the interpolated rate by using a proportionally larger frame *size* than the input. For interpolation by a factor of L, the output frame size is L times larger than the input frame size ($M_o = M_i{*}L$), but the input and output frame rates are equal.

The example below shows a single-channel input of frame size 16 being interpolated by a factor of 4 to a frame size of 64. The block's input and output frame rates are identical.



Zero Latency

The FIR Interpolation block has *zero-tasking latency* for all single-rate operations. The block is single rate for the particular combinations of sampling mode and parameter settings shown in the table below.

Sampling Mode	Parameter Settings
Sample based	**Interpolation factor** parameter, L, is `1`.
Frame based	**Interpolation factor** parameter, L, is `1`, *or* **Framing** parameter is `Maintain input frame rate`.

Note that in sample-based mode, single-rate operation occurs only in the trivial case of factor-of-1 interpolation.

The block also has zero latency for sample-based multirate operations in the Simulink single-tasking mode. Zero-tasking latency means that the block propagates the first filtered input (received at t=0) as the first input sample, followed by L-1 interpolated values, the second filtered input sample, and so on.

Nonzero Latency

The FIR Interpolation block is multirate for all settings other than those in the previous table. The amount of latency for multirate operation depends on the Simulink tasking mode and the block's sampling mode, as shown in the following table.

Multirate...	Sample-Based Latency	Frame-Based Latency
Single-tasking	None	None
Multitasking	L samples	L frames (M_i samples per frame)

When the block exhibits latency, the default initial condition is zero. Alternatively, you can use the **Output buffer initial conditions** parameter to specify a matrix of initial conditions containing one value for each channel, or a scalar to be applied as the initial condition for all signal channels. The **Output buffer initial conditions** parameter is divided by the **Interpolation factor** and output at the output port until the first filtered input sample is available.

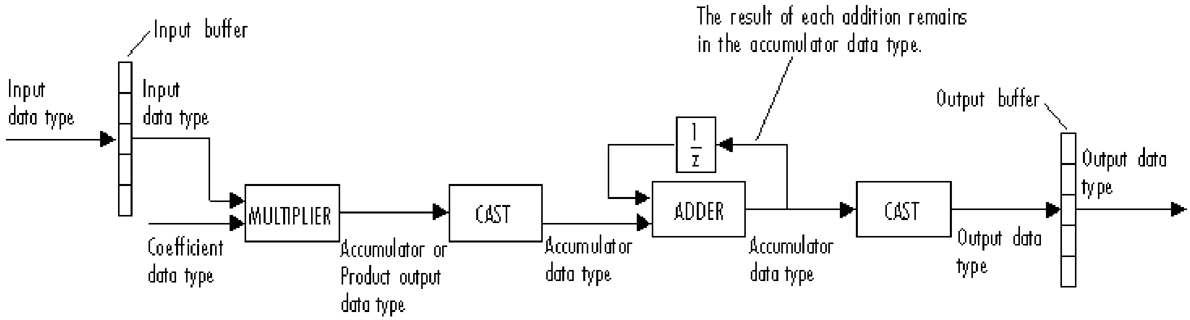
In sample-based cases, the scaled initial conditions appear at the start of each channel, followed immediately by the first filtered input sample, L-1 interpolated values, and so on.

In frame-based cases, with the default initial condition, the first M_iL output rows contain zeros, where M_i is the input frame size. The first filtered input sample (first filtered row of the input matrix) appears in the output as sample M_iL+1, followed by L-1 interpolated values, the second filtered input sample, and so on. See the following example for an illustration of this case.

Note For more information on latency and the Simulink tasking modes, see “Excess Algorithmic Delay (Tasking Latency)” and “Scheduling Considerations” in the *Real-Time Workshop User’s Guide*.

Fixed-Point Data Types

The following diagram shows the data types used within the FIR Interpolation block for fixed-point signals.



You can set the coefficient, product output, accumulator, and output data types in the block dialog as discussed in “Dialog Box” on page 2-548. The diagram shows that input data is stored in the input buffer in the same data type and scaling as the input. Filtered data is stored in the output buffer in the output data type and scaling that you set in the block dialog. Any initial conditions are also stored in the output buffer in the output data type and scaling you set in the block dialog.

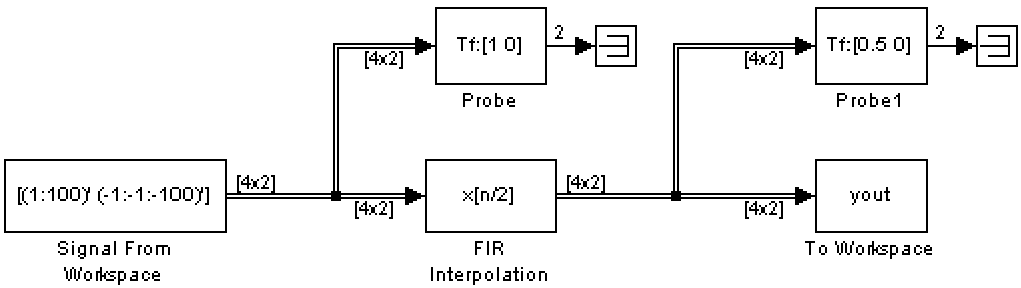
The output of the multiplier is in the product output data type when at least one of the inputs to the multiplier is real. When both of the inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see “Multiplication Data Types”.

Note When the block input is fixed point, all internal data types are signed fixed point.

Examples

Example 1

Construct the frame-based model shown below.



Adjust the block parameters as follows:

- Configure the Signal From Workspace block to generate a two-channel signal with frame size of 4 and sample period of 0.25. This represents an output frame period of 1 (0.25*4). The first channel should contain the positive ramp signal 1, 2, ..., 100, and the second channel should contain the negative ramp signal -1, -2, ..., -100.
  - **Signal** = `[(1:100)' (-1:-1:-100)']`
  - **Sample time** = `0.25`
  - **Samples per frame** = `4`
- Configure the FIR Interpolation block to interpolate the two-channel input by increasing the output frame rate by a factor of 2 relative to the input frame rate. Use a third-order filter (m=3) with normalized cutoff frequency, f_{n0}, of 0.25. (Note that f_{n0} and m satisfy $f_{n0} \leq 1/L$ and $m > L$.)

- **FIR filter coefficients** = `fir1(3,0.25)`
- **Interpolation factor** = `2`
- **Framing** = `Maintain input frame size`

The filter coefficient vector generated by `fir1(3,0.25)` is

```
[0.0386 0.4614 0.4614 0.0386]
```

or, equivalently,

$$H(z) = B(z) = 0.0386 + 0.04614z^{-1} + 0.04614z^{-2} + 0.0386z^{-3}$$

- Configure the Probe blocks by clearing the **Probe width**, **Probe complex signal**, and **Probe signal dimensions** check boxes (if desired).

This model is multirate because there are at least two distinct sample rates, as shown by the two Probe blocks. To run this model in the Simulink multitasking mode, open the Configuration Parameters dialog box. In the **Select** pane, click **Solver**. From the **Type** list, select `Fixed-step`, and from the **Solver** list, select `Discrete (no continuous states)`. From the **Tasking mode for periodic sample times** list, select `MultiTasking`. Also set the **Stop time** to `30`.

Run the model and look at the output, `yout`. The first few samples of each channel are shown below.

```
dsp_examples_yout =
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
         0         0
    0.0386   -0.0386
```

```
0.4614   -0.4614
0.5386   -0.5386
0.9614   -0.9614
1.0386   -1.0386
```

Since we ran this frame-based multirate model in multitasking mode, the first eight (M_iL) output rows are zero. The first filtered input matrix row appears in the output as sample 9 (that is, sample M_iL+1). Every other row is an interpolated value.

Example 2

The `doc_polyphaseinterp` model illustrates the underlying polyphase implementations of the FIR Interpolation block. Run the model and view the results on the scope. The output of the FIR Interpolation block is the same as the output of the Polyphase Interpolation Filter block.

Example 3

The `doc_mrf_nlp` model illustrates the use of the FIR Interpolation block in a number of multistage multirate filters.

Dialog Box

Coefficient Source

The FIR Interpolation block can operate in two different modes. Select the mode in the **Coefficient source** group box. If you select

- **Dialog parameters**, you enter information about the filter such as structure and coefficients in the block mask.
- **Multirate filter object (MFILT)**, you specify the filter using a Filter Design Toolbox `mfilt` object.

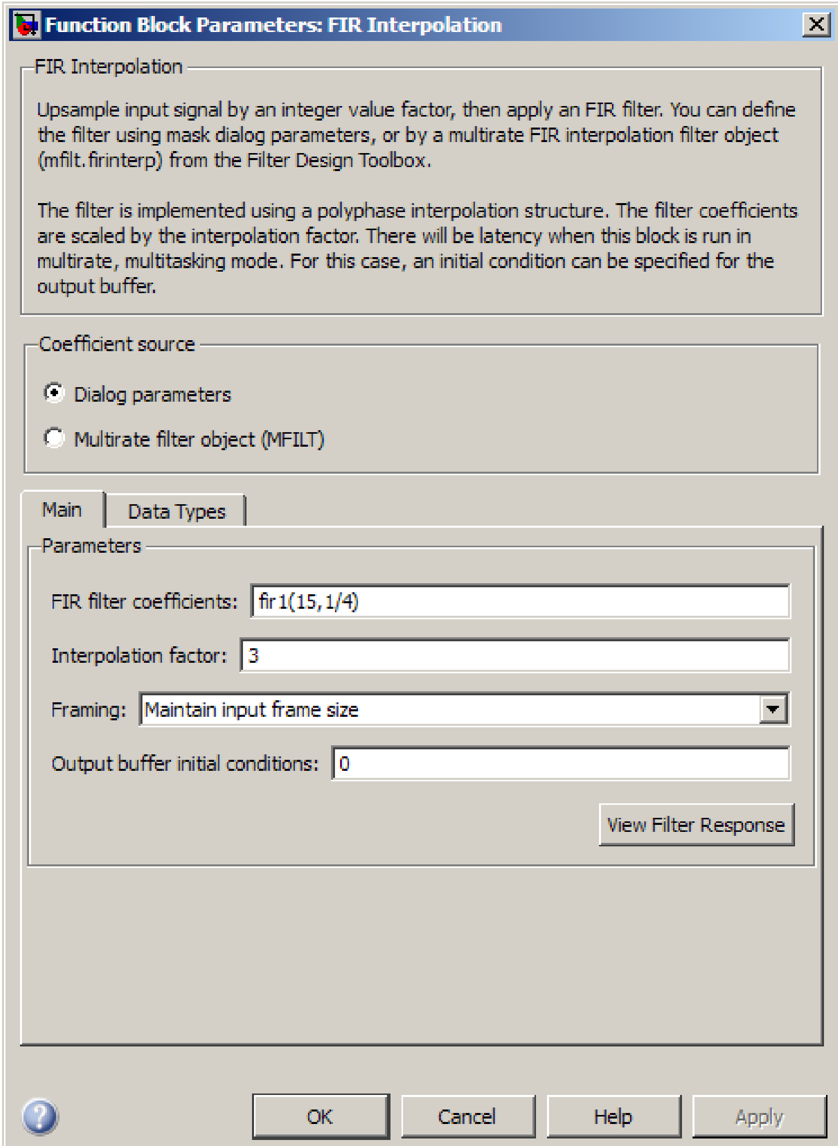
Different items appear on the FIR Interpolation block dialog depending on whether you select **Dialog parameters** or **Multirate filter object (MFILT)** in the **Coefficient source** group box. See the following sections for details:

- "Specify Filter Characteristics in Dialog" on page 2-549

- "Specify Multirate Filter Object" on page 2-557

Specify Filter Characteristics in Dialog

The **Main** pane of the FIR Interpolation block dialog appears as follows when **Dialog parameters** is selected in the **Coefficient source** group box.


Function Block Parameters: FIR Interpolation
FIR Interpolation
Upsample input signal by an integer value factor, then apply an FIR filter. You can define the filter using mask dialog parameters, or by a multirate FIR interpolation filter object (mfilt.firinterp) from the Filter Design Toolbox.
The filter is implemented using a polyphase interpolation structure. The filter coefficients are scaled by the interpolation factor. There will be latency when this block is run in multirate, multitasking mode. For this case, an initial condition can be specified for the output buffer.
Coefficient source
Dialog parameters
Multirate filter object (MFILT)
Main
Data Types
Parameters
FIR filter coefficients:
fir1(15,1/4)
Interpolation factor:
3
Framing:
Maintain input frame size
Output buffer initial conditions:
0
View Filter Response
OK
Cancel
Help
Apply

FIR filter coefficients

Specify the FIR filter coefficients, in descending powers of z.

Interpolation factor

Specify the integer factor, L, by which to increase the sample rate of the input sequence.

Framing

For frame-based operation, specify the method by which to implement the interpolation: increase the output frame rate, or increase the output frame size. This parameter cannot be set to `Maintain input frame rate` for sample-based signals.

Output buffer initial conditions

In cases of nonzero latency, the block divides this parameter by the **Interpolation factor** and outputs the results at the output port until the first filtered input sample is available. The default initial condition value is `0`, but you can enter a matrix containing one value for each channel, or a scalar to be applied to all signal channels.

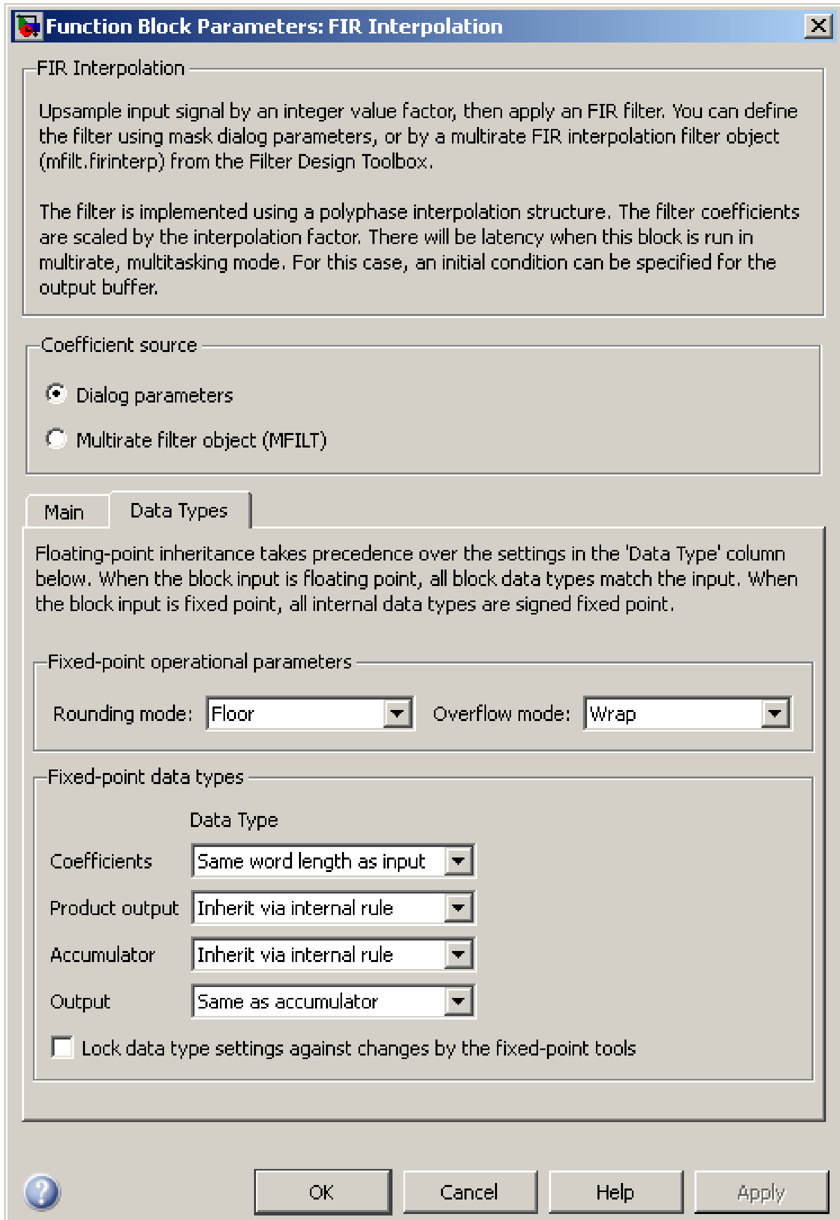
Output buffer initial conditions are stored in the output data type and scaling.

View filter response

This button opens the Filter Visualization Tool (`fvtool`) from the Signal Processing Toolbox product and displays the filter response of the filter defined in the block. For more information on FVTool, see the Signal Processing Toolbox documentation.

Note This button is only available when the Filter Design Toolbox product is installed. If you specify a filter in the **Multirate filter variable** parameter, you must apply the filter by clicking the **Apply** button before using the **View filter response** button.

The **Data Types** pane of the FIR Interpolation block dialog appears as follows when **Dialog parameters** is specified in the **Coefficient source** group box.


Function Block Parameters: FIR Interpolation
FIR Interpolation
Upsample input signal by an integer value factor, then apply an FIR filter. You can define the filter using mask dialog parameters, or by a multirate FIR interpolation filter object (mfilt.firinterp) from the Filter Design Toolbox.
The filter is implemented using a polyphase interpolation structure. The filter coefficients are scaled by the interpolation factor. There will be latency when this block is run in multirate, multitasking mode. For this case, an initial condition can be specified for the output buffer.
Coefficient source
Dialog parameters
Multirate filter object (MFILT)
Main
Data Types
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input. When the block input is fixed point, all internal data types are signed fixed point.
Fixed-point operational parameters
Rounding mode:
Floor
Overflow mode:
Wrap
Fixed-point data types
Data Type
Coefficients
Same word length as input
Product output
Inherit via internal rule
Accumulator
Inherit via internal rule
Output
Same as accumulator
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Rounding mode

Select the rounding mode for fixed-point operations. The filter coefficients do not obey this parameter; they always round to `Nearest`.

Overflow mode

Select the overflow mode for fixed-point operations. The filter coefficients do not obey this parameter; they are always saturated.

Coefficients

Choose how you specify the word length and fraction length of the filter coefficients:

- When you select `Same word length as input`, the word length of the filter coefficients match that of the input to the block. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.
- When you select `Specify word length`, you can enter the word length of the coefficients, in bits. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the coefficients, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the coefficients. This block requires power-of-two slope and a bias of zero.

The filter coefficients do not obey the **Rounding mode** and the **Overflow mode** parameters; they are always saturated and rounded to `Nearest`.

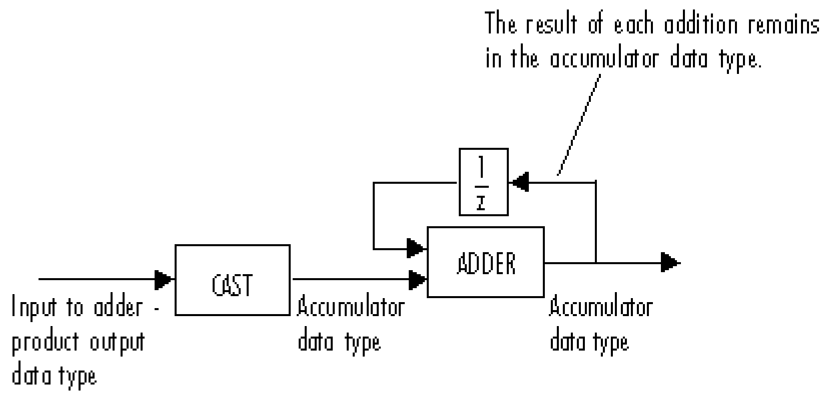
Product output

Use this parameter to specify how you would like to designate the product output word and fraction lengths. See "Fixed-Point

Data Types" on page 2-545 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block:

- When you select `Inherit via internal rule`, the product output word length and fraction length are calculated automatically. For information about how the product output word and fraction lengths are calculated when an internal rule is used, see "Inherit via Internal Rule".
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator



As depicted above, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how you would like to designate this accumulator word and fraction lengths.

You also use this parameter to specify the accumulator word and fraction lengths resulting from a complex-complex multiplication in the block. See "Multiplication Data Types" for more information:

- When you select `Inherit via internal rule`, the accumulator word length and fraction length are calculated automatically. For information about how the accumulator word and fraction lengths are calculated when an internal rule is used, see "Inherit via Internal Rule".
- When you select `Same as product output`, these characteristics match those of the product output.
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Choose how you specify the output word length and fraction length:

- When you select `Same as accumulator`, these characteristics match those of the accumulator.

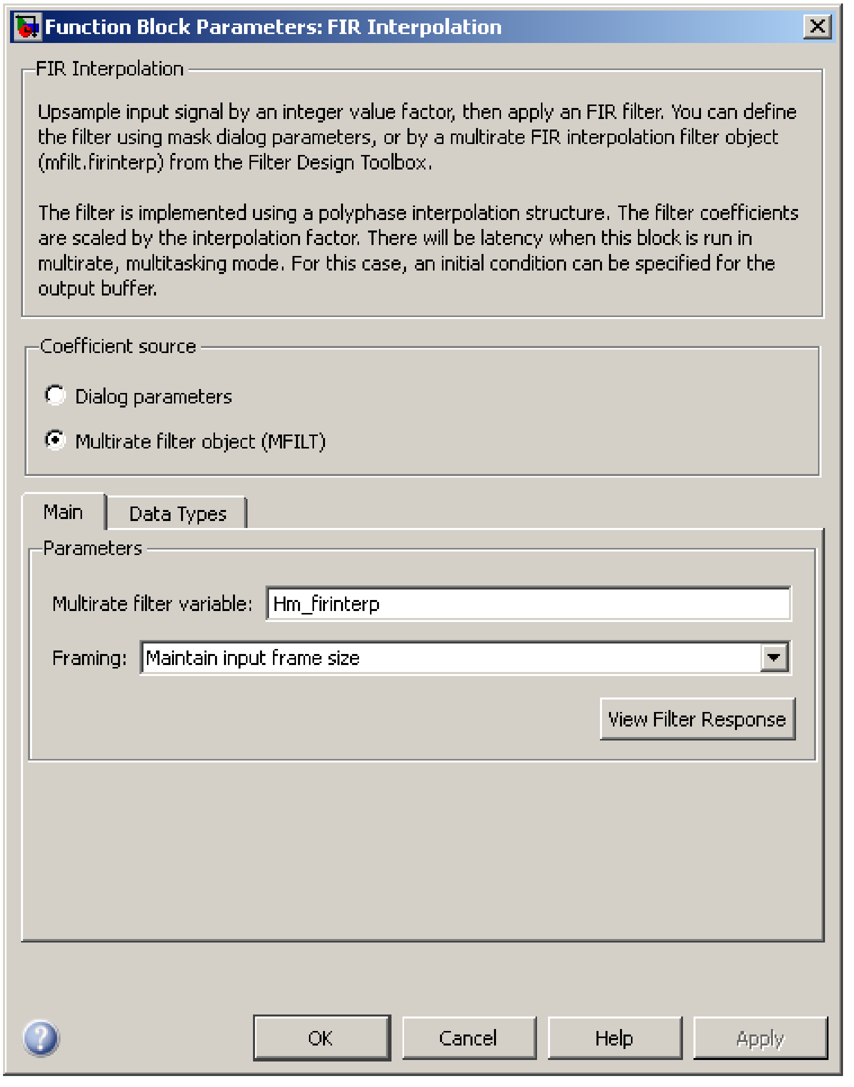
  A special case occurs when `Inherit via internal rule` is specified for **Accumulator**, and block inputs and coefficients are complex. In that case, the output word length be one less than the accumulator word length.
- When you select `Same as product output`, these characteristics match those of the product output.

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Specify Multirate Filter Object

The **Main** pane of the FIR Interpolation block dialog appears as follows when **Multirate filter object (MFILT)** is specified in the **Coefficient source** group box.


Function Block Parameters: FIR Interpolation
FIR Interpolation
Upsample input signal by an integer value factor, then apply an FIR filter. You can define the filter using mask dialog parameters, or by a multirate FIR interpolation filter object (mfilt.firinterp) from the Filter Design Toolbox.
The filter is implemented using a polyphase interpolation structure. The filter coefficients are scaled by the interpolation factor. There will be latency when this block is run in multirate, multitasking mode. For this case, an initial condition can be specified for the output buffer.
Coefficient source
Dialog parameters
Multirate filter object (MFILT)
Main
Data Types
Parameters
Multirate filter variable:
Hm_firinterp
Framing:
Maintain input frame size
View Filter Response
OK
Cancel
Help
Apply

Multirate filter variable

Specify the multirate filter object (`mfilt`) that you would like the block to implement. You can do this in one of three ways:

- You can fully specify the `mfilt` object in the block mask.
- You can enter the variable name of a `mfilt` object that is defined in any workspace.
- You can enter a variable name for a `mfilt` object that is not yet defined, as shown in the default value.

For more information on creating `mfilt` objects, see the `mfilt` function reference page in the Filter Design Toolbox documentation.

Framing

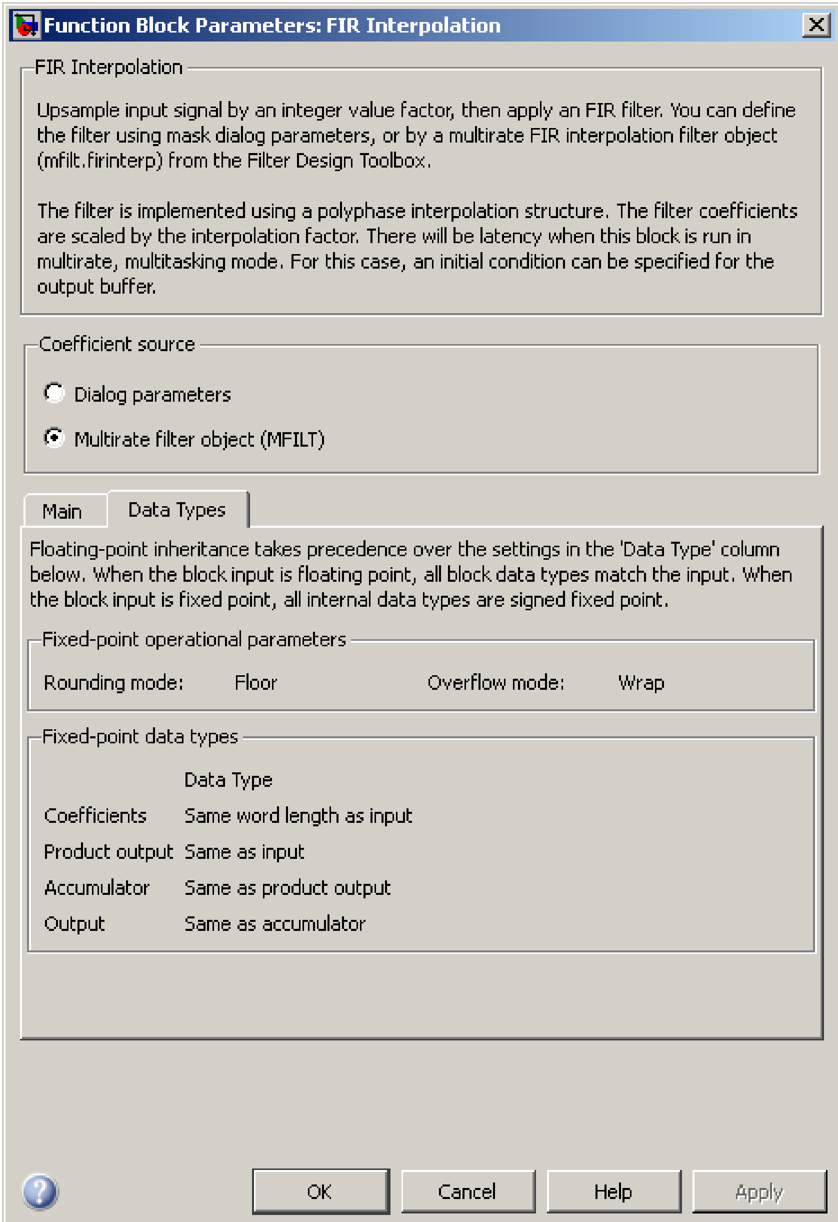
For frame-based operation, specify the method by which to implement the interpolation: increase the output frame rate, or increase the output frame size. This parameter cannot be set to `Maintain input frame rate` for sample-based signals.

View filter response

This button opens the Filter Visualization Tool (`fvtool`) from the Signal Processing Toolbox product and displays the filter response of the `mfilt` object specified in the **Multirate filter variable** parameter. For more information on FVTool, see the Signal Processing Toolbox documentation.

Note This button is only available when the Filter Design Toolbox product is installed. If you specify a filter in the **Multirate filter variable** parameter, you must apply the filter by clicking the **Apply** button before using the **View filter response** button.

The **Data Types** pane of the FIR Interpolation block dialog appears as follows when **Multirate filter object (MFILT)** is specified in the **Coefficient source** group box.


Function Block Parameters: FIR Interpolation
FIR Interpolation
Upsample input signal by an integer value factor, then apply an FIR filter. You can define the filter using mask dialog parameters, or by a multirate FIR interpolation filter object (mfilt.firinterp) from the Filter Design Toolbox.
The filter is implemented using a polyphase interpolation structure. The filter coefficients are scaled by the interpolation factor. There will be latency when this block is run in multirate, multitasking mode. For this case, an initial condition can be specified for the output buffer.
Coefficient source
Dialog parameters
Multirate filter object (MFILT)
Main
Data Types
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input. When the block input is fixed point, all internal data types are signed fixed point.
Fixed-point operational parameters
Rounding mode:
Floor
Overflow mode:
Wrap
Fixed-point data types
Data Type
Coefficients
Same word length as input
Product output
Same as input
Accumulator
Same as product output
Output
Same as accumulator
OK
Cancel
Help
Apply

The fixed-point settings of the filter object specified on the **Main** pane are displayed on the **Data Types** pane. You cannot change these settings directly on the block mask. To change the fixed-point settings you must edit the filter object directly.

For more information on multirate filter objects, see the `mfilt` function reference page in the Filter Design Toolbox documentation.

References

Fliege, N. J. *Multirate Digital Signal Processing: Multirate Systems, Filter Banks, Wavelets*. West Sussex, England: John Wiley & Sons, 1994.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

FIR Decimation	Signal Processing Blockset
FIR Rate Conversion	Signal Processing Blockset
Upsample	Signal Processing Blockset
`fir1`	Signal Processing Toolbox

`fir2`	Signal Processing Toolbox
`firls`	Signal Processing Toolbox
`interp`	Signal Processing Toolbox

Purpose

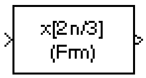
Upsample, filter, and downsample input signals

Library

Filtering / Multirate Filters

`dspmlti4`

Description



The FIR Rate Conversion block resamples the discrete-time input to a period K/L times the input sample period, where the integer K is specified by the **Decimation factor** parameter and the integer L is specified by the **Interpolation factor** parameter. The resampling process consists of the following steps:

1. The block upsamples the input to a higher rate by inserting L-1 zeros between input samples.

2. The upsampled data is passed through a direct-form II transpose FIR filter.

3. The block downsamples the filtered data to a lower rate by discarding K-1 consecutive samples following each sample retained.

K and L must be *relatively prime* integers; that is, the ratio K/L cannot be reducible to a ratio of smaller integers. The FIR Rate Conversion block implements the above three steps together using a polyphase filter structure, which is more efficient than straightforward upsample-filter-decimate algorithms. See Orfanidis [1] for more information.

The **FIR filter coefficients** parameter specifies the numerator coefficients of the FIR filter transfer function $H(z)$.

$$H(z) = B(z) = b_1 + b_2 z^{-1} + \ldots + b_m z^{-(m-1)}$$

The coefficient vector, `[b(1) b(2) ...  b(m)]`, can be generated by one of the Signal Processing Toolbox filter design functions (such as `fir1`), and should have a length greater than the interpolation factor ($m>L$). The filter should be lowpass with normalized cutoff frequency no greater than min($1/L$,$1/K$). All filter states are internally initialized to zero.

The FIR Rate Conversion block supports real and complex floating-point and fixed-point inputs except for complex unsigned fixed-point inputs.

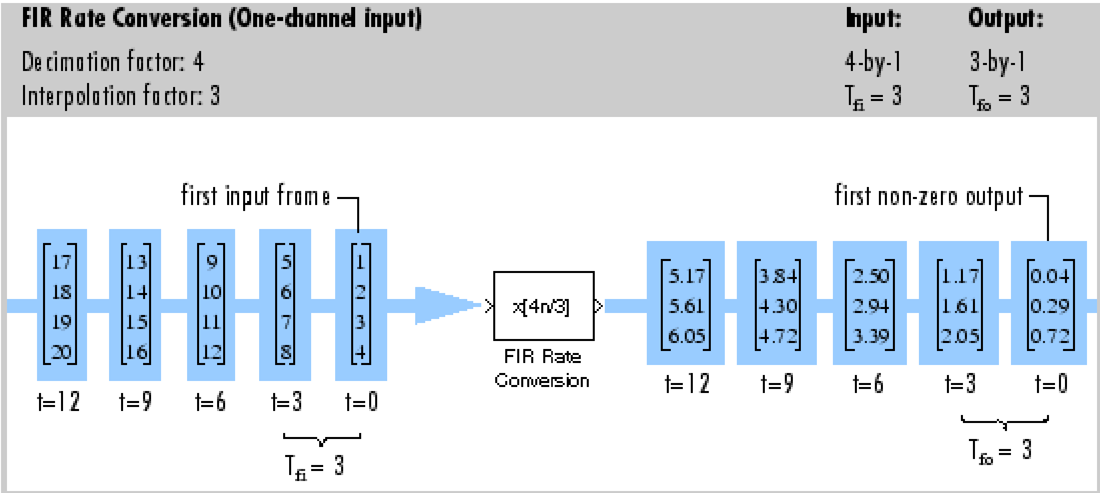
Frame-Based Operation

This block accepts *only* frame-based inputs. An M_i-by-N frame-based matrix input is treated as N independent channels, and the block resamples each channel independently over time.

The **Interpolation factor**, L, and **Decimation factor**, K, must satisfy the relation

$$\frac{K}{L} = \frac{M_i}{M_o}$$

for an *integer* output frame size M_o. The simplest way to satisfy this requirement is to let the **Decimation factor** equal the input frame size, M_i. The output frame size, M_o, is then equal to the **Interpolation factor**. This change in the frame size, from M_i to M_o, produces the desired rate conversion while leaving the output frame period the same as the input ($T_{fo} = T_{fi}$).

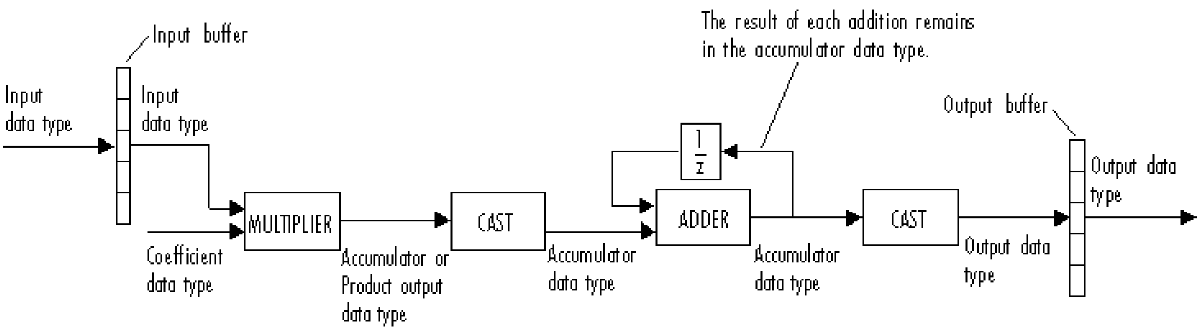


Latency

The FIR Rate Conversion block has no tasking latency. The block propagates the first filtered input (received at t=0) as the first output sample.

Fixed-Point Data Types

The following diagram shows the data types used within the FIR Rate Conversion block for fixed-point signals.



You can set the coefficient, product output, accumulator, and output data types in the block dialog as discussed in "Dialog Box" on page 2-567. The diagram shows that input data is stored in the input buffer in the same data type and scaling as the input. Filtered data is stored in the output buffer in the output data type and scaling that you set in the block dialog. Any initial conditions are also stored in the output buffer in the output data type and scaling you set in the block dialog.

The output of the multiplier is in the product output data type when at least one of the inputs to the multiplier is real. When both of the inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see "Multiplication Data Types".

Note When the block input is fixed point, all internal data types are signed fixed point.

Examples

The `doc_audio_src` model provides a simple illustration of one way to convert a speech signal from one sample rate to another. In this model, the data is first sampled at 22,050 Hz and then resampled at 8000 Hz. If you listen to the output, you can hear that the high frequency content has been removed from the signal, although the speech sounds basically the same.

Diagnostics

An error is generated when the relation between K and L shown above is not satisfied.

```
(Input port width)/(Output port width) must equal the
(Decimation factor)/(Interpolation factor).
```

A warning is generated when L and K are not relatively prime; that is, when the ratio L/K can be reduced to a ratio of smaller integers.

```
Warning: Integer conversion factors are not relatively prime
in block 'modelname/FIR Rate Conversion
```

```
(Frame)'. Converting ratio L/M to l/m.
```

The block scales the ratio to be relatively prime and continues the simulation.

Dialog Box

Coefficient Source

The FIR Rate Conversion block can operate in two different modes. Select the mode in the **Coefficient source** group box. If you select

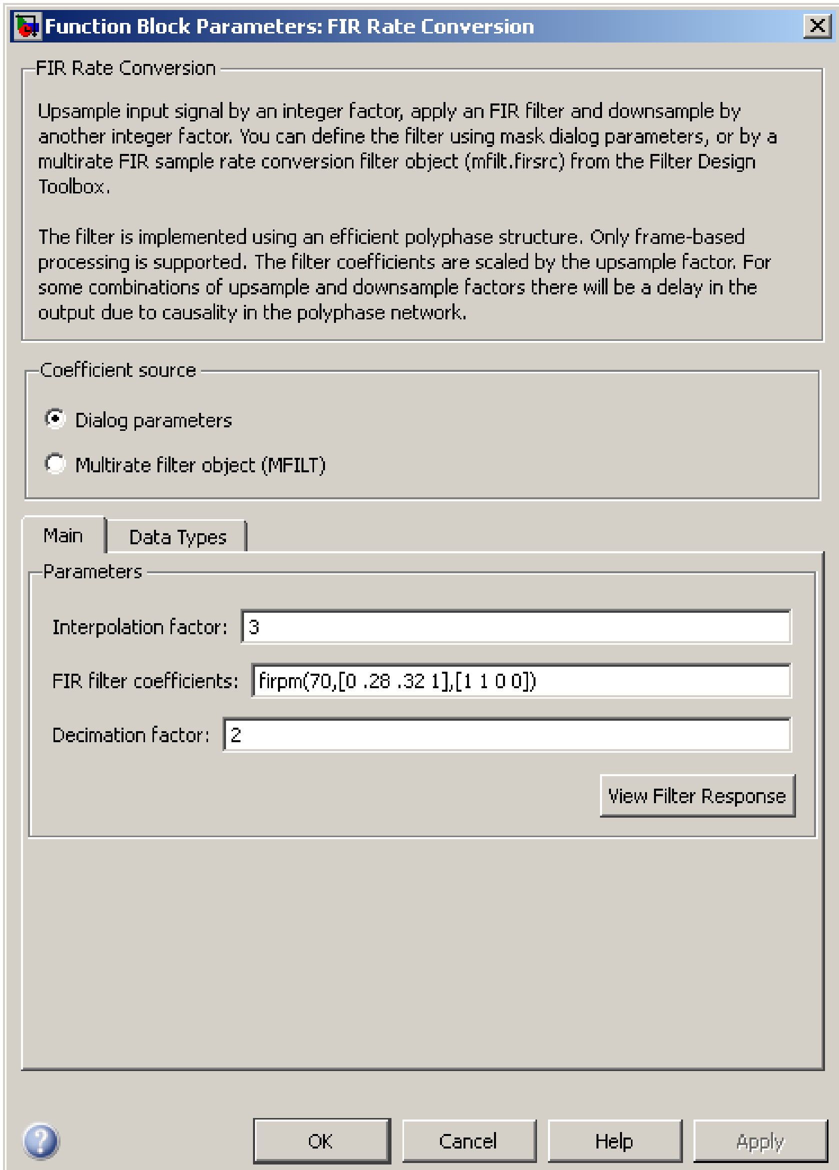
- **Dialog parameters**, you enter information about the filter such as structure and coefficients in the block mask.
- **Multirate filter object (MFILT)**, you specify the filter using a Filter Design Toolbox `mfilt` object.

Different items appear on the FIR Rate Conversion block dialog depending on whether you select **Dialog parameters** or **Multirate filter object (MFILT)** in the **Coefficient source** group box. See the following sections for details:

- "Specify Filter Characteristics in Dialog" on page 2-567
- "Specify Multirate Filter Object" on page 2-574

Specify Filter Characteristics in Dialog

The **Main** pane of the FIR Rate Conversion block dialog appears as follows when **Dialog parameters** is selected in the **Coefficient source** group box.


Function Block Parameters: FIR Rate Conversion
FIR Rate Conversion
Upsample input signal by an integer factor, apply an FIR filter and downsample by another integer factor. You can define the filter using mask dialog parameters, or by a multirate FIR sample rate conversion filter object (mfilt.firsrc) from the Filter Design Toolbox.
The filter is implemented using an efficient polyphase structure. Only frame-based processing is supported. The filter coefficients are scaled by the upsample factor. For some combinations of upsample and downsample factors there will be a delay in the output due to causality in the polyphase network.
Coefficient source
Dialog parameters
Multirate filter object (MFILT)
Main
Data Types
Parameters
Interpolation factor:
3
FIR filter coefficients:
firpm(70,[0 .28 .32 1],[1 1 0 0])
Decimation factor:
2
View Filter Response
OK
Cancel
Help
Apply

Interpolation factor
Specify the integer factor, *L*, by which to upsample the signal before filtering.

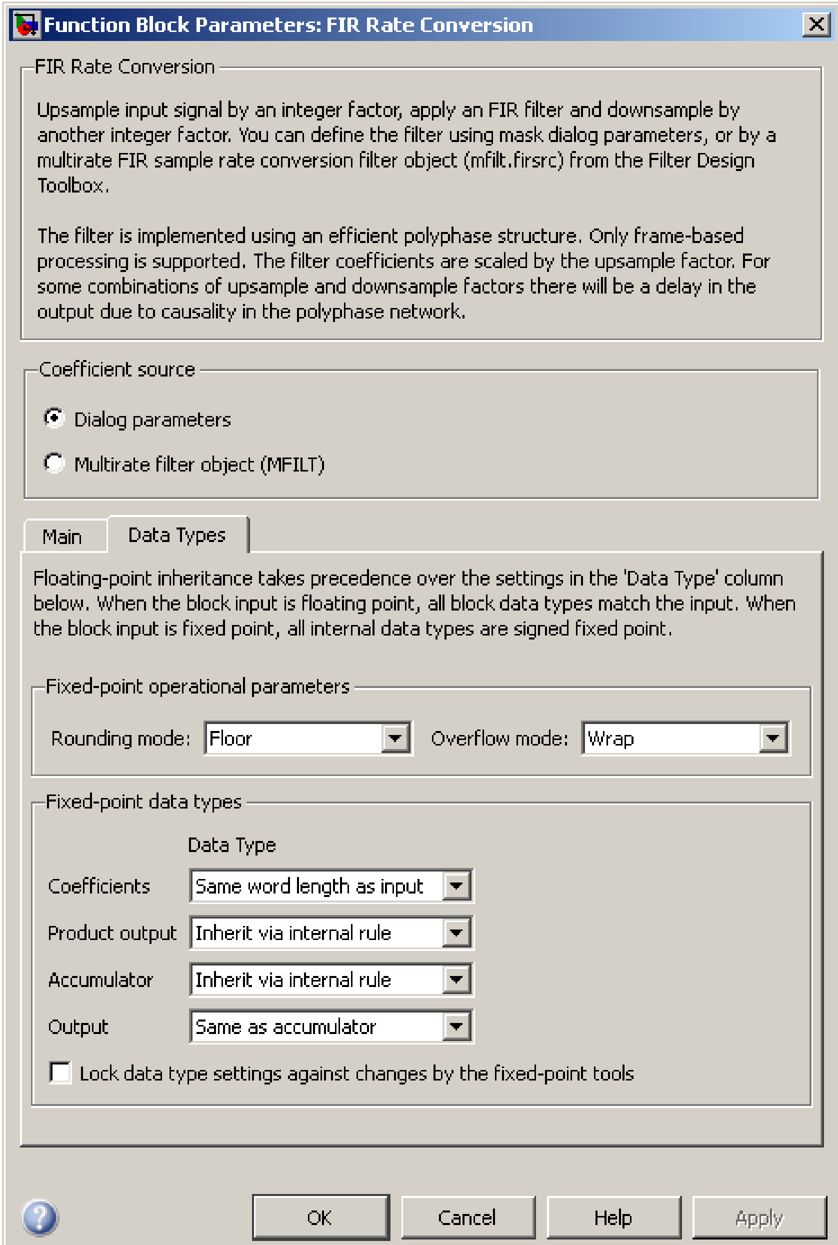
FIR filter coefficients
Specify the FIR filter coefficients in descending powers of *z*.

Decimation factor
Specify the integer factor, *K*, by which to downsample the signal after filtering.

View filter response
This button opens the Filter Visualization Tool (`fvtool`) from the Signal Processing Toolbox product and displays the filter response of the filter defined in the block. For more information on FVTool, see the Signal Processing Toolbox documentation.

Note This button is only available when the Filter Design Toolbox product is installed. If you specify a filter in the **Multirate filter variable** parameter, you must apply the filter by clicking the **Apply** button before using the **View filter response** button.

The **Data Types** pane of the FIR Rate Conversion block dialog appears as follows when **Dialog parameters** is specified in the **Coefficient source** group box.



Function Block Parameters: FIR Rate Conversion
FIR Rate Conversion
Upsample input signal by an integer factor, apply an FIR filter and downsample by another integer factor. You can define the filter using mask dialog parameters, or by a multirate FIR sample rate conversion filter object (mfilt.firsrc) from the Filter Design Toolbox.
The filter is implemented using an efficient polyphase structure. Only frame-based processing is supported. The filter coefficients are scaled by the upsample factor. For some combinations of upsample and downsample factors there will be a delay in the output due to causality in the polyphase network.
Coefficient source
Dialog parameters
Multirate filter object (MFILT)
Main
Data Types
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input. When the block input is fixed point, all internal data types are signed fixed point.
Fixed-point operational parameters
Rounding mode:
Floor
Overflow mode:
Wrap
Fixed-point data types
Data Type
Coefficients
Same word length as input
Product output
Inherit via internal rule
Accumulator
Inherit via internal rule
Output
Same as accumulator
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Rounding mode
Select the rounding mode for fixed-point operations. The filter coefficients do not obey this parameter; they always round to `Nearest`.

Overflow mode
Select the overflow mode for fixed-point operations. The filter coefficients do not obey this parameter; they are always saturated.

Coefficients
Choose how you specify the word length and fraction length of the filter coefficients.

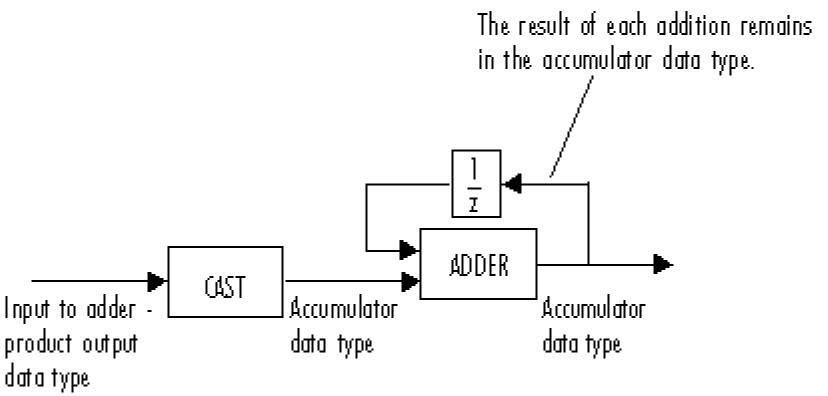
- When you select `Same word length as input`, the word length of the filter coefficients match that of the input to the block. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.
- When you select `Specify word length`, you can enter the word length of the coefficients, in bits. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the coefficients, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the coefficients. This block requires power-of-two slope and a bias of zero.
- The coefficients do not obey the **Rounding mode** and the **Overflow mode** parameters; they are always saturated and rounded to `Nearest`.

Product output
Use this parameter to specify how you would like to designate the product output word and fraction lengths. See “Fixed-Point Data Types” on page 2-565 and “Multiplication Data Types” for

illustrations depicting the use of the product output data type in this block.

- When you select `Inherit via internal rule`, the product output word length and fraction length are calculated automatically. For information about how the product output word and fraction lengths are calculated when an internal rule is used, see "Inherit via Internal Rule".
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator



As depicted above, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how you would like to designate this accumulator word and fraction lengths.

You also use this parameter to specify the accumulator word and fraction lengths resulting from a complex-complex multiplication in the block. See “Multiplication Data Types” for more information.

- When you select `Inherit via internal rule`, the accumulator word length and fraction length are calculated automatically. For information about how the accumulator word and fraction lengths are calculated when an internal rule is used, see “Inherit via Internal Rule”.
- When you select `Same as product output`, these characteristics match those of the product output.
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Choose how you specify the output word length and fraction length:

- When you select `Same as accumulator`, these characteristics match those of the accumulator.

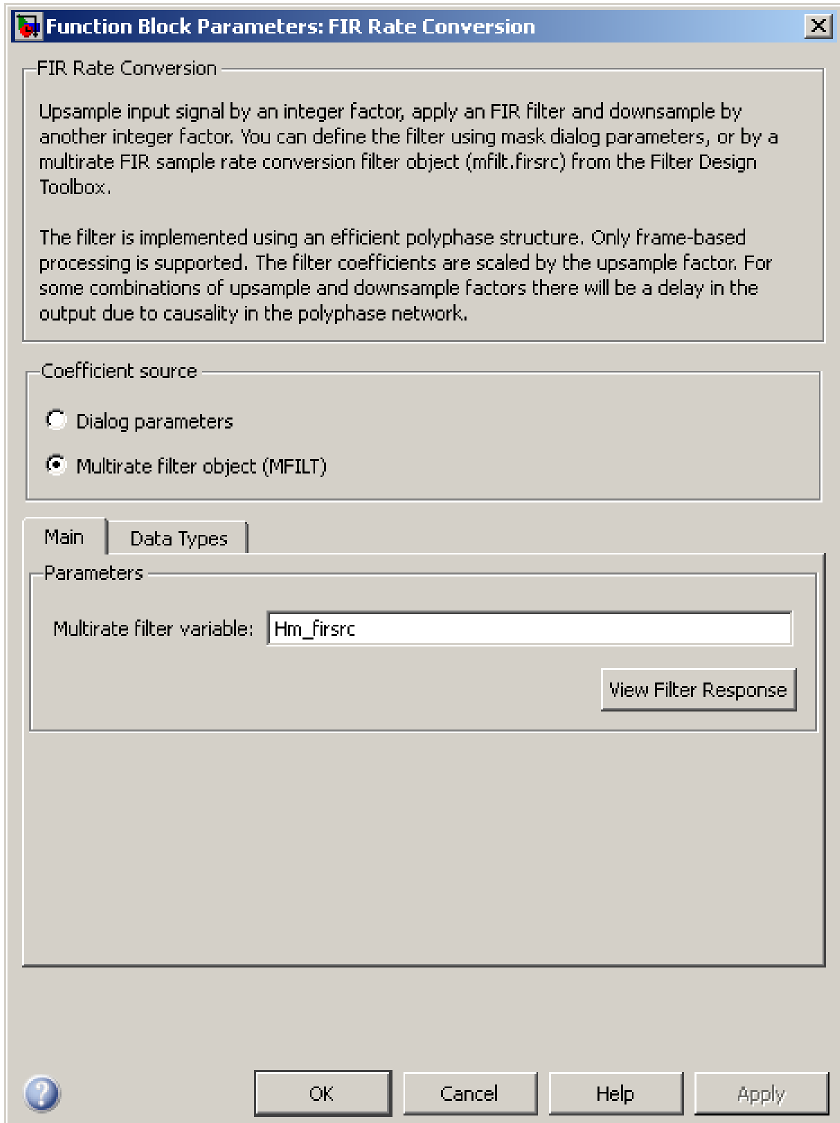
  A special case occurs when `Inherit via internal rule` is specified for **Accumulator**, and block inputs and coefficients are complex. In that case, the output word length is one less than the accumulator word length.
- When you select `Same as product output`, these characteristics match those of the product output.
- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Specify Multirate Filter Object

The **Main** pane of the FIR Rate Conversion block dialog appears as follows when **Multirate filter object (MFILT)** is specified in the **Coefficient source** group box.


Function Block Parameters: FIR Rate Conversion
FIR Rate Conversion
Upsample input signal by an integer factor, apply an FIR filter and downsample by another integer factor. You can define the filter using mask dialog parameters, or by a multirate FIR sample rate conversion filter object (mfilt.firsrc) from the Filter Design Toolbox.
The filter is implemented using an efficient polyphase structure. Only frame-based processing is supported. The filter coefficients are scaled by the upsample factor. For some combinations of upsample and downsample factors there will be a delay in the output due to causality in the polyphase network.
Coefficient source
Dialog parameters
Multirate filter object (MFILT)
Main
Data Types
Parameters
Multirate filter variable:
Hm_firsrc
View Filter Response
OK
Cancel
Help
Apply

Multirate filter variable

Specify the multirate filter object (`mfilt`) that you would like the block to implement. You can do this in one of three ways:

- You can fully specify the `mfilt` object in the block mask.
- You can enter the variable name of a `mfilt` object that is defined in any workspace.
- You can enter a variable name for a `mfilt` object that is not yet defined, as shown in the default value.

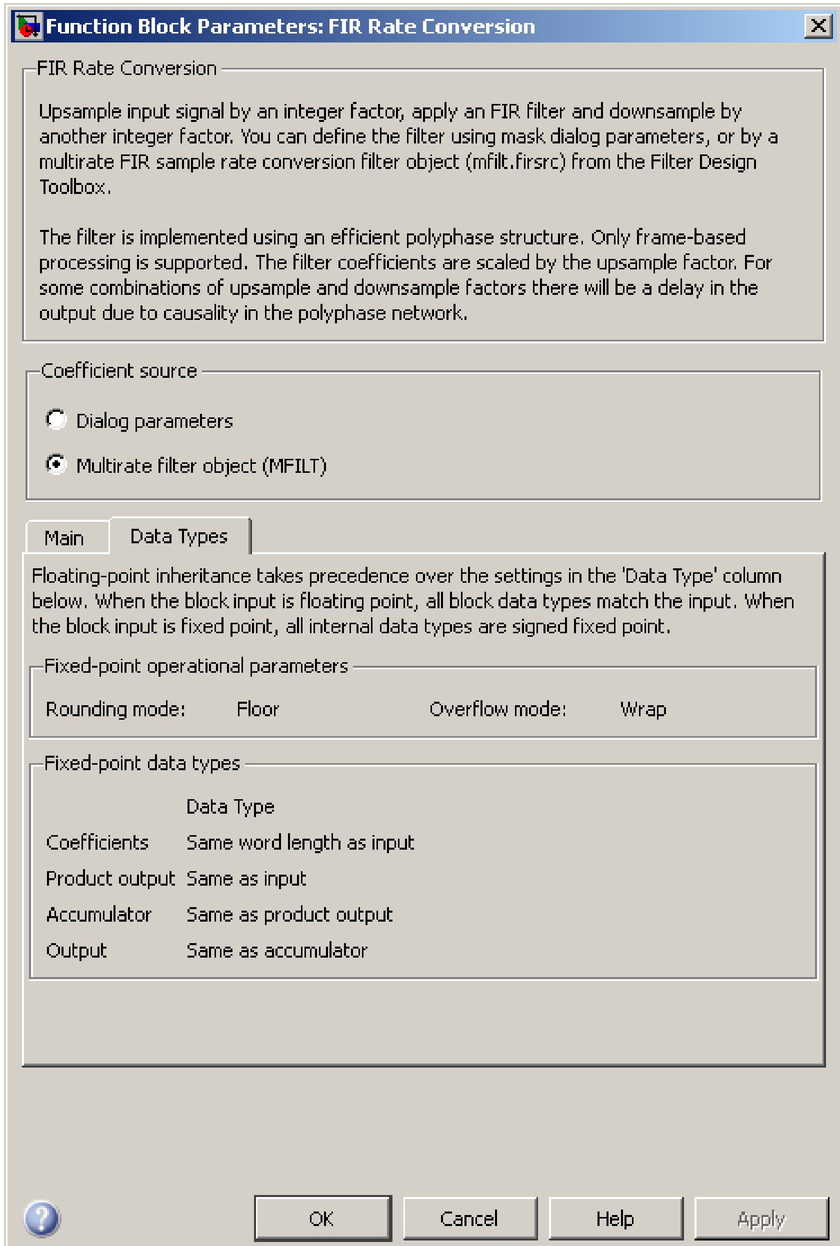
For more information on creating `mfilt` objects, see the `mfilt` function reference page in the Filter Design Toolbox documentation.

View filter response

This button opens the Filter Visualization Tool (`fvtool`) from the Signal Processing Toolbox product and displays the filter response of the `mfilt` object specified in the **Multirate filter variable** parameter. For more information on FVTool, see the Signal Processing Toolbox documentation.

Note This button is only available when the Filter Design Toolbox product is installed. If you specify a filter in the **Multirate filter variable** parameter, you must apply the filter by clicking the **Apply** button before using the **View filter response** button.

The **Data Types** pane of the FIR Rate Conversion block dialog appears as follows when **Multirate filter object (MFILT)** is specified in the **Coefficient source** group box.


Function Block Parameters: FIR Rate Conversion
FIR Rate Conversion
Upsample input signal by an integer factor, apply an FIR filter and downsample by another integer factor. You can define the filter using mask dialog parameters, or by a multirate FIR sample rate conversion filter object (mfilt.firsrc) from the Filter Design Toolbox.
The filter is implemented using an efficient polyphase structure. Only frame-based processing is supported. The filter coefficients are scaled by the upsample factor. For some combinations of upsample and downsample factors there will be a delay in the output due to causality in the polyphase network.
Coefficient source
Dialog parameters
Multirate filter object (MFILT)
Main
Data Types
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input. When the block input is fixed point, all internal data types are signed fixed point.
Fixed-point operational parameters
Rounding mode: Floor
Overflow mode: Wrap
Fixed-point data types
Data Type
Coefficients Same word length as input
Product output Same as input
Accumulator Same as product output
Output Same as accumulator
OK
Cancel
Help
Apply

The fixed-point settings of the filter object specified on the **Main** pane are displayed on the **Data Types** pane. You cannot change these settings directly on the block mask. To change the fixed-point settings you must edit the filter object directly.

For more information on multirate filter objects, see the `mfilt` function reference page in the Filter Design Toolbox documentation.

References

[1] Orfanidis, S. J. *Introduction to Signal Processing*. Prentice Hall, 1996.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Downsample	Signal Processing Blockset
FIR Decimation	Signal Processing Blockset
FIR Interpolation	Signal Processing Blockset
Upsample	Signal Processing Blockset
`fir1`	Signal Processing Toolbox

`fir2`	Signal Processing Toolbox
`firls`	Signal Processing Toolbox
`upfirdn`	Signal Processing Toolbox

See the following sections for related information:

- "Converting Sample and Frame Rates"
- "Multirate Filters"

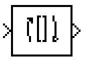
Flip

Purpose Flip input vertically or horizontally

Library Signal Management / Indexing

`dspindex`

Description



The Flip block vertically or horizontally reverses the M-by-N input matrix, u. The output always has the same dimension and frame status as the input.

When you select `Columns` from the **Flip along** menu, the block *vertically* flips the input so that the first row of the input is the last row of the output.

```
y = flipud(u)      % Equivalent MATLAB code
```

For convenience, length-M 1-D vector inputs are treated as M-by-1 column vectors for vertical flipping.

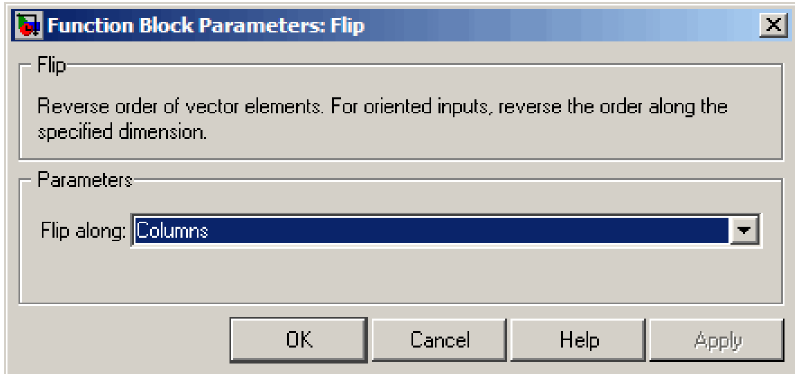
When you select `Rows` from the **Flip along** menu, the block *horizontally* flips the input so that the first column of the input is the last column of the output.

```
y = fliplr(u)      % Equivalent MATLAB code
```

For convenience, length-N 1-D vector inputs are treated as 1-by-N row vectors for horizontal flipping. The output always has the same dimension and frame status as the input.

This block supports Simulink virtual buses.

Dialog Box



Flip along

The dimension along which to flip the input. `Columns` specifies vertical flipping, while `Rows` specifies horizontal flipping.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated

Port	Supported Data Types
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated

See Also

Selector	Simulink
Transpose	Signal Processing Blockset
Variable Selector	Signal Processing Blockset
`flipud`	MATLAB
`fliplr`	MATLAB

Purpose

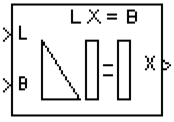
Solve LX=B for X when L is lower triangular matrix

Library

Math Functions / Matrices and Linear Algebra / Linear System Solvers

`dspsolvers`

Description



The Forward Substitution block solves the linear system LX=B by simple forward substitution of variables, where:

- L is the lower triangular M-by-M matrix input to the L port.
- B is the M-by-N matrix input to the B port.

The sample-based output is the M-by-N matrix X that is the solution of the equations. The block does not check the rank of the inputs.

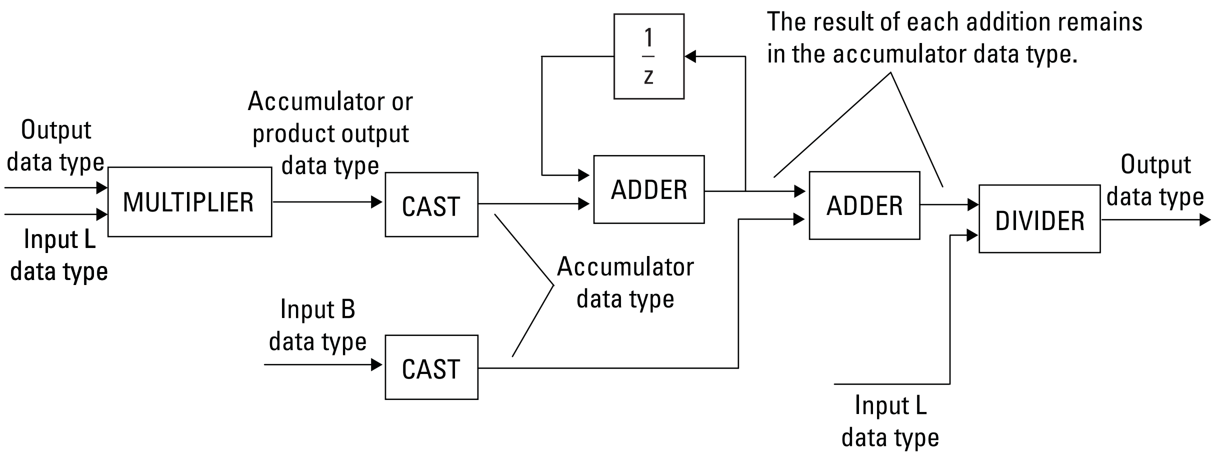
The block only uses the elements in the *lower triangle* of input L and ignores the upper elements. When you select **Input L is unit-lower triangular**, the block assumes the elements on the diagonal of L are 1s. This is useful when matrix L is the result of another operation, such as an LDL decomposition, that uses the diagonal elements to represent the D matrix.

The block treats a length-M vector input at port B as an M-by-1 matrix.

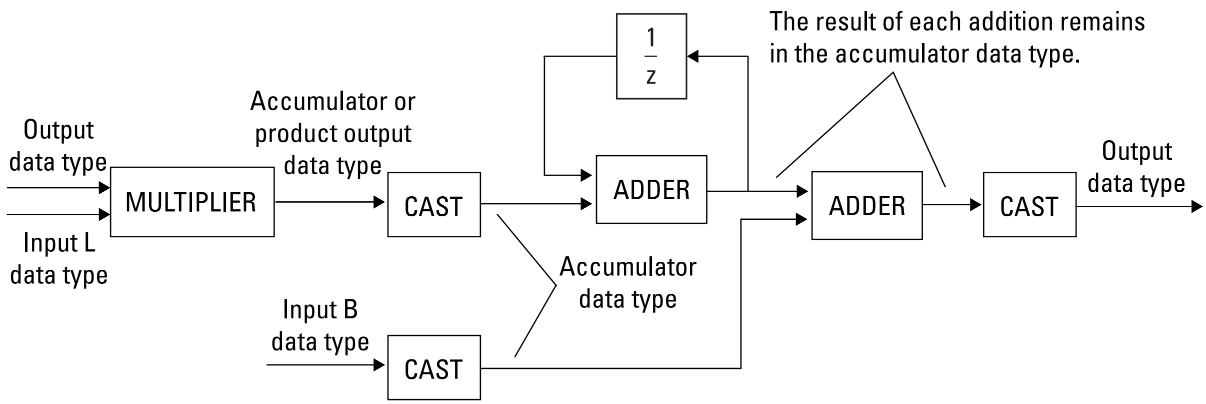
Fixed-Point Data Types

The following diagram shows the data types used within the Forward Substitution block for fixed-point signals.

When input L is not unit-lower triangular:



When input L is unit-lower triangular:

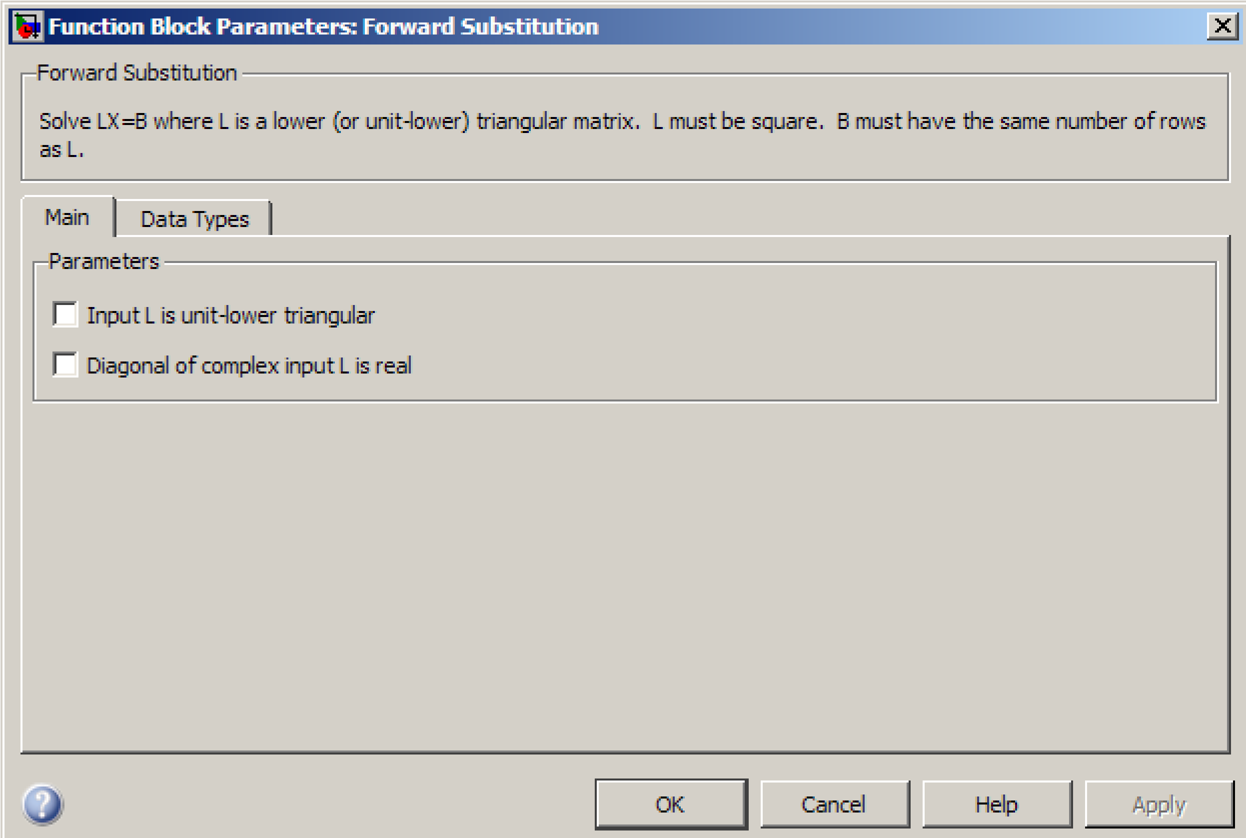


You can set the product output, accumulator, and output data types in the block dialog box, as discussed in the following section.

The output of the multiplier is in the product output data type when the input is real. When the input is complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see “Multiplication Data Types”.

Dialog Box

The **Main** pane of the Forward Substitution block dialog box appears as follows.



Input L is unit-lower triangular

Select this check box only when all elements on the diagonal of L have a value of 1. When you do so, the block optimizes its behavior by skipping an unnecessary divide operation.

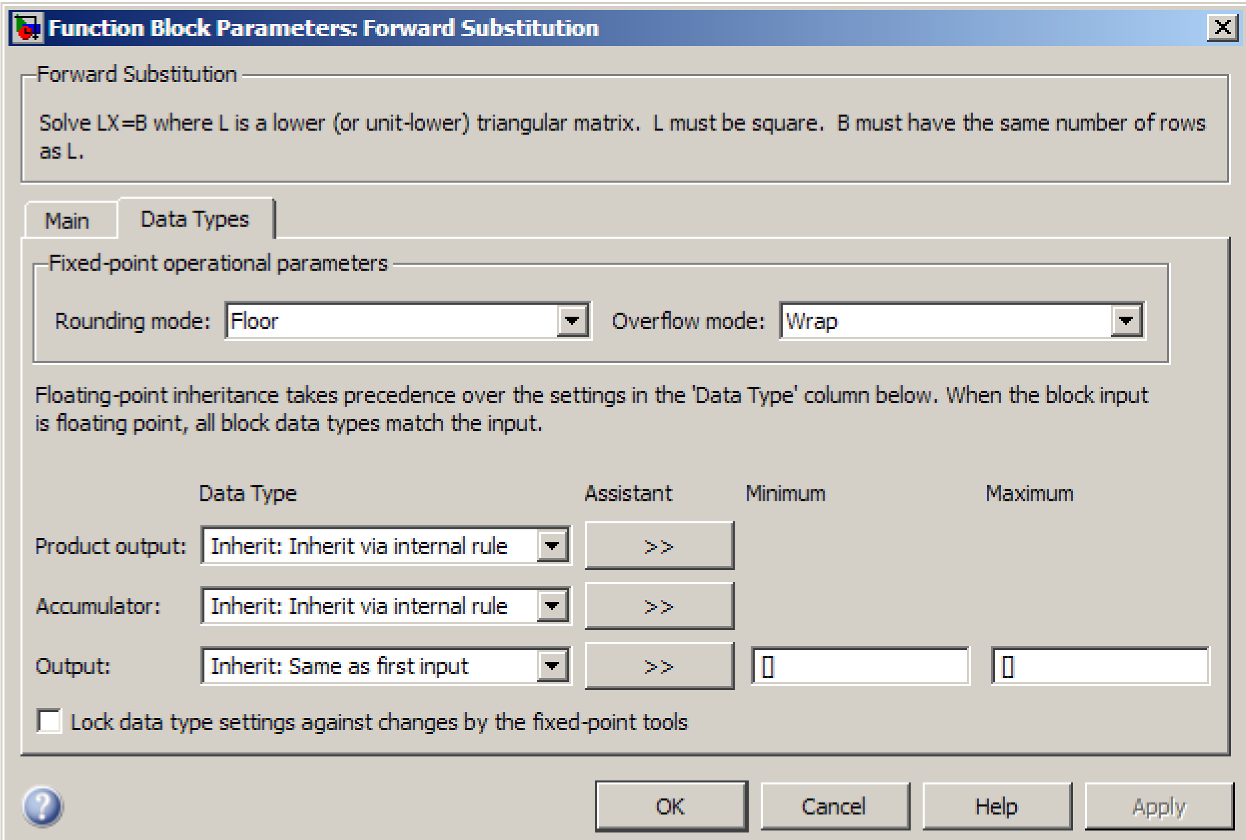
Do not select this check box if there are any elements on the diagonal of L that do not have a value of 1. When you clear the **Input L is unit-lower triangular** check box, the block always performs the necessary divide operation.

Diagonal of complex input L is real

Select to optimize simulation speed when the diagonal elements of complex input *L* are real. This parameter is only visible when **Input L is unit-upper triangular** is not selected.

Note When L is a complex fixed-point signal, you must select either **Input L is unit-lower triangular** or **Diagonal of complex input L is real**. In such a case, the block ignores any imaginary part of the diagonal of L.

The **Data Types** pane of the Forward Substitution block dialog box appears as follows.



Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Rounding mode
: Select the rounding mode for fixed-point operations.

Overflow mode
: Select the overflow mode for fixed-point operations.

Product output data type
: Specify the product output data type. See “Fixed-Point Data Types” on page 2-583 and “Multiplication Data Types” for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Accumulator data type
: Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-583 for illustrations depicting the use of the accumulator data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Output data type

Specify the output data type. See “Fixed-Point Data Types” on page 2-583 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as first input`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User’s Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see “Checking Signal Ranges”)
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see “Checking Signal Ranges”)
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
L	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
B	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
X	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

Backward Substitution	Signal Processing Blockset
Cholesky Solver	Signal Processing Blockset
LDL Solver	Signal Processing Blockset
Levinson-Durbin	Signal Processing Blockset
LU Solver	Signal Processing Blockset
QR Solver	Signal Processing Blockset

See “Linear System Solvers” for related information.

Purpose

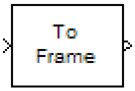
Specify sampling mode of output signal

Library

Signal Management / Signal Attributes

`dspsigattribs`

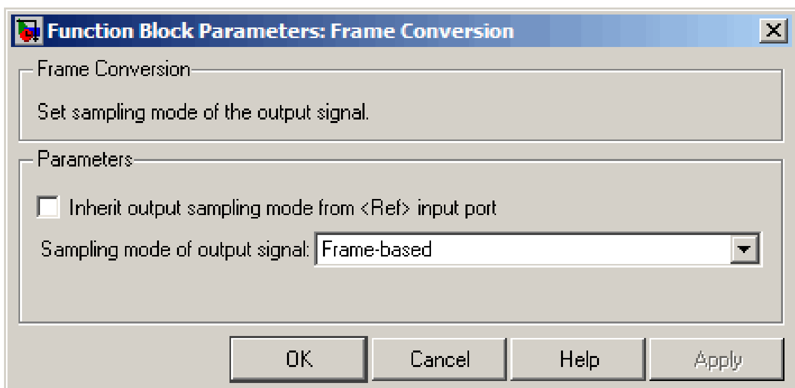
Description



The Frame Conversion block passes the input through to the output and sets the output sampling mode to the value of the **Sampling mode of output signal** parameter, which can be either `Frame-based` or `Sample-based`. The output sampling mode can also be inherited from the signal at the Ref (reference) input port, which you make visible by selecting the **Inherit output sampling mode from <Ref> input port** check box.

The Frame Conversion block does not make any changes to the input signal other than the sampling mode. In particular, the block does not rebuffer or resize 2-D inputs. Because 1-D vectors cannot be frame based, when the input is a length-M 1-D vector and the block is in `Frame-based` mode, the output is a frame-based M-by-1 matrix — that is, a single channel.

Dialog Box



Inherit output sampling mode from <Ref> input port
Select to enable the Ref port from which the block inherits the output sampling mode.

Sampling mode of output signal
Specify the sampling mode of the output signal, `Frame-based` or `Sample-based`.

Supported Data Types

Port	Supported Data Types
In	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated
Ref	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated

See Also

Buffer	Signal Processing Blockset
Check Signal Attributes	Signal Processing Blockset
Convert 1-D to 2-D	Signal Processing Blockset
Convert 2-D to 1-D	Signal Processing Blockset
Inherit Complexity	Signal Processing Blockset
Unbuffer	Signal Processing Blockset
Probe	Simulink
Reshape	Simulink
Signal Specification	Simulink

Frame Status Conversion (Obsolete)

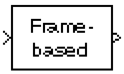
Purpose

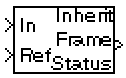
Specify frame status of output as sample based or frame based

Library

`dspobslib`

Description





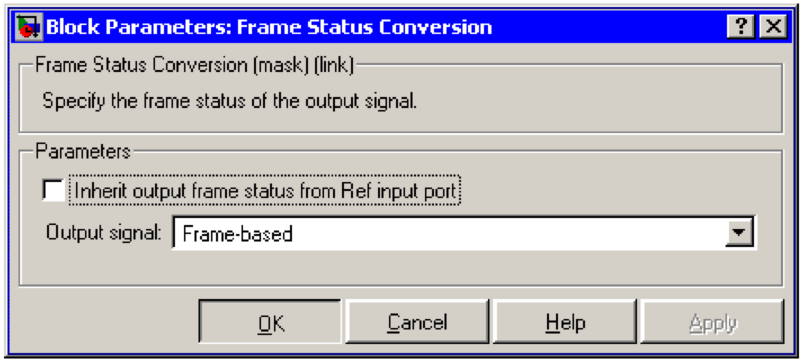
Note The Frame Status Conversion block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the Frame Conversion block.

The Frame Status Conversion block passes the input through to the output, and sets the output frame status to the **Output signal** parameter, which can be either `Frame-based` or `Sample-based`. The output frame status can also be inherited from the signal at the `Ref` (reference) input port, which is made visible by selecting the **Inherit output frame status from Ref input port** check box.

When the **Output signal** parameter setting or the inherited signal's frame status differs from the input frame status, the block changes the input frame status accordingly, but does not otherwise alter the signal. In particular, the block does not rebuffer or resize 2-D inputs. Because 1-D vectors cannot be frame based, when the input is a length-M 1-D vector, and the **Output signal** parameter is set to `Frame-based`, the output is a frame-based M-by-1 matrix (that is, a single channel).

When the **Output signal** parameter or the inherited signal's frame status matches the input frame status, the block passes the input through to the output unaltered.

Dialog Box



Inherit output frame status from Ref input port
When selected, enables the `Ref` input port from which the block inherits the output frame status.

Output signal
The output frame status, `Frame-based` or `Sample-based`.

Supported Data Types

Port	Supported Data Types
In	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Ref	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • Boolean

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Check Signal Attributes	Signal Processing Blockset
Convert 1-D to 2-D	Signal Processing Blockset
Convert 2-D to 1-D	Signal Processing Blockset
Inherit Complexity	Signal Processing Blockset

Purpose

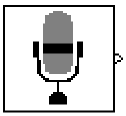
Read audio data from computer's audio device

Library

Signal Processing Sources

`dspsrcs4`

Description



The From Audio Device block reads audio data from an audio device in real time. This block has the following limitations:

- Not supported for use with the Simulink Model block.
- Not currently supported on Solaris™ platforms.

Use the **Device** parameter to specify the device from which to acquire audio. This parameter is automatically populated based on the audio devices installed on your system. If you plug or unplug an audio device from your system, type `clear mex` at the MATLAB command prompt to update this list.

Use the **Number of channels** parameter to specify the number of audio channels in the signal. For example:

- Enter `2` if the audio source is two channels (stereo).
- Enter `1` if the audio source is single channel (mono).
- Enter `6` if you are working with a 5.1 speaker system.

The block's output is an M-by-N matrix, where M is the number of consecutive samples and N is the number of audio channels.

Use the **Sample rate (Hz)** parameter to specify the number of samples per second in the signal. If the audio data is processed in uncompressed pulse code modulation (PCM) format, it should typically be sampled at one of the standard audio device rates: 8000, 11025, 22050, 44100, or 48000 Hz.

Use the **Device data type** parameter to specify the data type of the audio data that the device is placing in the buffer. You can choose:

- `8-bit integer`
- `16-bit integer`
- `24-bit integer`
- `32-bit float`
- `Determine from output data type`

If you choose `Determine from output data type`, the following table summarizes the block's behavior.

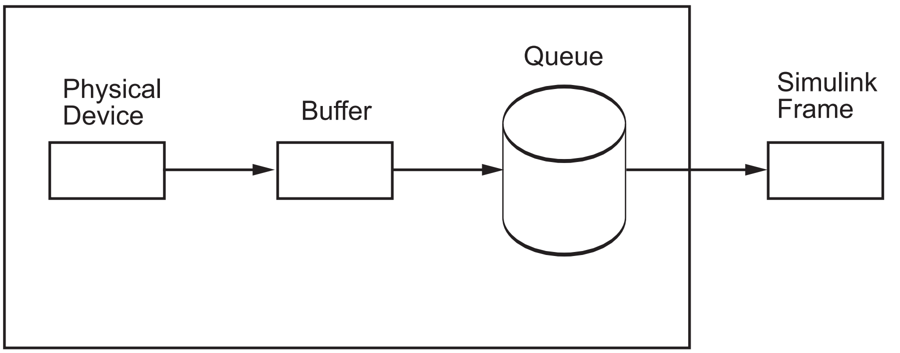
Output Data Type	Device Data Type
Double-precision floating point or single-precision floating point	32-bit floating point
32-bit integer	24-bit integer
16-bit integer	16-bit integer
8-bit integer	8-bit integer

If you choose `Determine from output data type` and the device does not support a data type, the block uses the next lowest precision data type supported by the device.

Use the **Frame size (samples)** parameter to specify the number of samples in the block's output. Use the **Output data type** parameter to specify the data type of audio data output by the block.

Buffering

The From Audio Device block buffers the data from the audio device using the process illustrated by the following figure.



From Audio Device Block

1. At the start of the simulation, the audio device begins writing the input data to a buffer. This data has the data type specified by the **Device data type** parameter.

2. When the buffer is full, the From Audio Device block writes the contents of the buffer to the queue. Specify the size of this queue using the **Queue duration (seconds)** parameter.

3. As the audio device appends audio data to the bottom of the queue, the From Audio Device block pulls data from the top of the queue to fill the Simulink frame. This data has the data type specified by the **Output data type** parameter.

Select the **Automatically determine buffer size** check box to allow the block to calculate a conservative buffer size using the following equation:

$$size = 2^{\left\lfloor \log_2 \frac{sr}{10} \right\rfloor}$$

In this equation, *size* is the buffer size, and *sr* is the sample rate. If you clear this check box, the **Buffer size (samples)** parameter appears on the block. Use this parameter to specify the buffer size in samples.

When the simulation throughput rate is lower than the hardware throughput rate, the queue, which is initially empty, fills up. If the queue is full, the block drops the incoming data from the audio device. When the simulation throughput rate is higher than the hardware throughput rate, the From Audio Device block waits for new samples to become available.

Troubleshooting

When Simulink cannot keep up with an audio device that is operating in real time, the queue fills up and the block begins to lose audio data. To receive a warning, which will indicate the number of samples lost, type the following command on the MATLAB command line:

```
warning('on', ...
    'spblks:block:FromAudioDevice:fromAudioDeviceDroppedSamples');
```

Here are several ways to deal with this situation:

- *Increase the queue duration.*

  The **Queue duration (seconds)** parameter specifies the duration of the signal, in seconds, that can be buffered during the simulation. This is the maximum length of time that the block's data demand can lag behind the hardware's data supply.

- *Increase the buffer size.*

  The size of the buffer processed in each interrupt from the audio device affects the performance of your model. If the buffer is too small, a large portion of hardware resources are used to write data to the queue. If the buffer is too big, Simulink must wait for the device to fill the buffer before it moves the data to the queue, which introduces latency.

- *Increase the simulation throughput rate.*

  Two useful methods for improving simulation throughput rates are increasing the signal frame size and compiling the simulation into native code:

   - Increase frame sizes and convert sample-based signals to frame-based signals throughout the model to reduce the amount of block-to-block communication overhead. This can increase throughput rates in many cases. However, larger frame sizes generally result in greater model latency due to initial buffering operations.
   - Generate executable code with Real-Time Workshop code generation software. Native code runs much faster than Simulink and should provide rates adequate for real-time audio processing.

Other ways to improve throughput rates include simplifying the model and running the simulation on a faster PC processor. For other ideas on improving simulation performance, see "Delay and Latency" and "Improving Simulation Performance and Accuracy" in the Simulink documentation.

Audio Hardware API

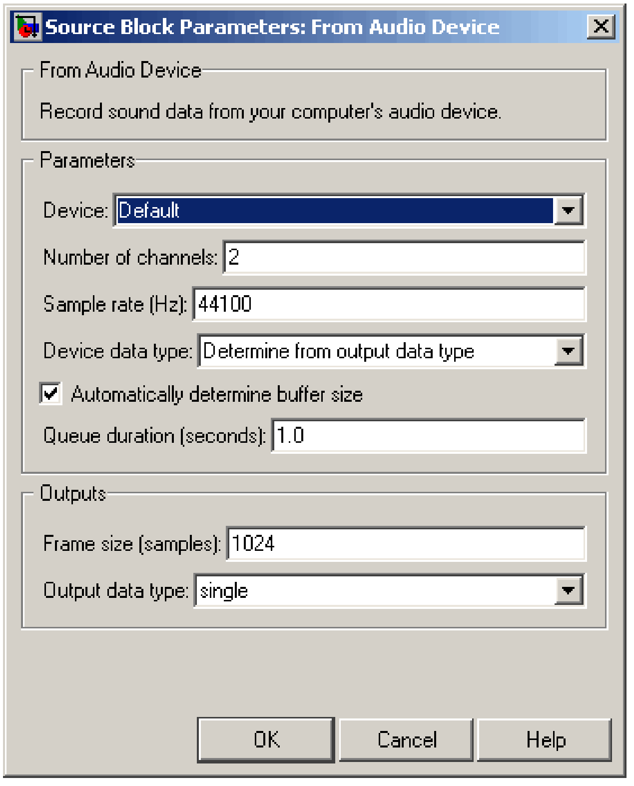
The To Audio Device and From Audio Device blocks use the open-source PortAudio library in order to communicate with the audio hardware on a given computer. The PortAudio library supports a range of API's designed to communicate with the audio hardware on a given platform. The following API choices were made when building the PortAudio library for the Signal Processing Blockset product:

- Windows: DirectSound, WDM—KS
- Linux: OSS, ALSA
- Mac: CoreAudio

To select or change the Audio Hardware API, select **Preferences** from the MATLAB **File** menu. Then select Signal Processing Blockset from the tree menu.

If you are interested in using a different audio API, please search for PortAudio on the Matlab Central website.

Dialog Box



Device
Specify the device from which to acquire audio data.

Number of channels
Specify the number of audio channels.

Sample rate (Hz)
Specify the number of samples per second in the signal.

Device data type
Specify the data type used by the device to acquire audio data.

Automatically determine buffer size
Select this check box to enable the block to use a conservative buffer size.

Buffer size (samples)
Specify the size of the buffer that the block uses to communicate with the audio device. This parameter is visible when the **Automatically determine buffer size** check box is cleared.

Queue duration (seconds)
Specify the size of the queue in seconds.

Frame size (samples)
Specify the number of samples in the block's output signal.

Output data type
Select the data type of the block's output.

Supported Data Types

Port	Supported Data Types
Output	• Double-precision floating point • Single-precision floating point • 32-bit signed integers • 16-bit signed integers • 8-bit unsigned integers

See Also

From Multimedia File	Signal Processing Blockset
To Audio Device	Signal Processing Blockset
`audiorecorder`	MATLAB

From Multimedia File

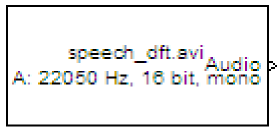
Purpose

Read multimedia file

Library

Signal Processing Sources

`dspsrcs4`



From Multimedia File1

Description

The From Multimedia File block reads audio frames, video frames, or both from a multimedia file. The block imports data from the file into a Simulink model.

Reading video files requires the Video and Image Processing Blockset™ product.

Note This block supports code generation for the host computer that has file I/O available. You cannot use this block with Real-Time Windows Target™ software because that product does not support file I/O.

Supported Platforms and File Types

With the necessary Windows® DirectShow codecs installed on your system, the From Multimedia File Block supports many video and audio file formats.

This block performs best on platforms with Version 9.0 or later of DirectX® software. For non-Windows platforms, this block supports only uncompressed AVI files. It does not support OpenDML extensions to the AVI standard on AVI files larger than 4 GB.

Windows Platforms Supported File Formats

Multimedia Types	File Name Extensions
Video files	`.qt`, `.mov`, `.avi`, `.asf`, `.asx`, `.wmv`, `.mpg`, `.mpeg`, `.mp2`, `.mp4`
Audio files	`.wav`, `.wma`, `.aif`, `.aifc`, `.aiff`, `.mp3`, `.au`, `.snd`

Windows XP x64 platform ships with a limited set of 64-bit video and audio codecs. If the From Multimedia File block cannot a compressed multimedia file, try one of the two alternatives:

- Run the 32-bit version of MATLAB on your Windows XP x64 platform. Windows XP x64 ships with many 32-bit codecs.
- Save the multimedia file to a file format supported by the From Multimedia File block.

If you use Windows, use Windows Media player Version 11 or later with this block for best results.

Non-Windows Platform Supported File Formats

Multimedia Types	File Name Extensions
Video files	`.avi`

Ports

The output ports of the From Multimedia File block change according to the content of the multimedia file. If the file contains only video frames, the **Image** or intensity **I** port appears on the block. If the file contains only audio samples, the **Audio** port appears on the block. If the file contains both audio and video, you can select the data to emit. The following table describes available ports.

Port	Description
Image	M-by-N-by-P color video signal where P is the number of color planes.
I	M-by-N matrix of intensity values.
R, G, B	Matrix that represents one plane of the RGB video stream. Outputs from the R, G, or B ports must have same dimensions.
Audio	Vector of audio data.
Y, Cb, Cr	Matrix that represents one frame of the YCbCr video stream. The Y, Cb, Cr ports produce the following outputs: Y: $M\ x\ N$ Cb: $M\ x \frac{N}{2}$ Cr: $M\ x \frac{N}{2}$

Sample Rates

The sample rate that the block uses depends on the audio and video sample rate. While the FMMF block operates at a single rate in Simulink, the underlying audio and video streams can produce different rates. In some cases, when the block outputs both audio and video, makes a small adjustment to the video rate.

Sample Time Calculations Used for Video and Audio Files

For Video Files Only:

Sample time = $\frac{1}{FPS}$, where FPS is the Frames per Second.

For Audio Files Only:

Sample time = $\frac{1024}{SampleRate}$, where 1024 is the size of the audio frame, set by the block.

For Video and Audio Files:

$$\text{Sample time} = \frac{ceil(AudioSampleRate/FPS)}{AudioSampleRate}.$$

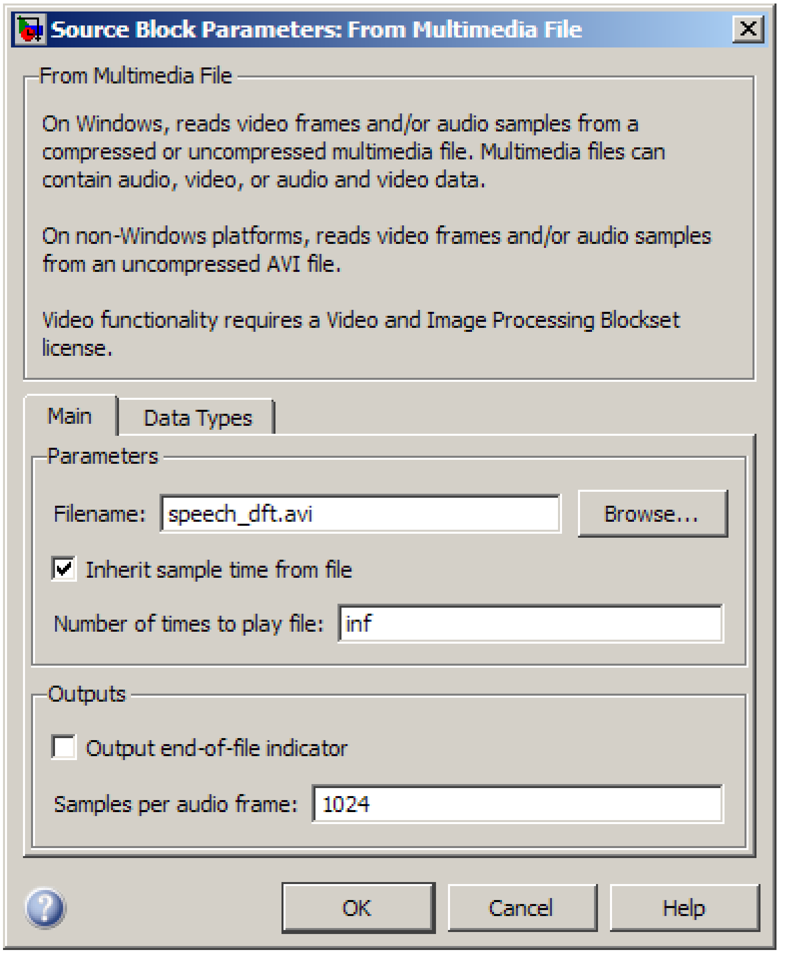
When audio sample time calculation, $\frac{AudioSampleRate}{FPS}$ is noninteger, the equation cannot reduce to $\frac{1}{FPS}$.

In this case, to prevent synchronization problems, the block drops the corresponding video frame when the audio stream leads the video stream by more than $\frac{1}{FPS}$.

In summary, the block outputs one video frame at each Simulink time step. To calculate the number of audio samples to output at each time step, the block divides the audio sample rate by the video frame rate (fps). If the audio sample rate does not divide evenly by the number of video frames per second, the block rounds the number of audio samples up to the nearest whole number. If necessary, the block periodically drops a video frame to maintain synchronization for large files.

Dialog Box

The **Main** pane of the From Multimedia File block dialog appears as follows.



File name

Specify the name of the multimedia file from which to read. The block determines the type of file (audio and video, audio only, or video only) and provides the associated parameters.

If the location of the file does not appear on your MATLAB path, use the **Browse** button to specify the full path. Otherwise, if the location of this file appears on your MATLAB path, enter only the file name. On Windows platforms, this parameter supports URLs that point to MMS (Microsoft Media Server) streams.

Inherit sample time from file

Select the **Inherit sample time from file** check box if you want the block sample time to be the same as the multimedia file. If you clear this check box, enter the block sample time in the **Desired sample time** parameter field. The file that the From Multimedia File block references, determines the block default sample time. You can also set the sample time for this block manually. If you do not know the intended sample rate of the video, let the block inherit the sample rate from the multimedia file.

Desired sample time

Specify the block sample time. This parameter becomes available if you clear the **Inherit sample time from file** check box.

Number of times to play file

Enter a positive integer or `inf` to represent the number of times to play the file.

Output end-of-file indicator

Use this check box to determine whether the output is the last video frame or audio sample in the multimedia file. When you select this check box, a Boolean output port labeled EOF appears on the block. The output from the EOF port defaults to 1 when the last video frame or audio sample is output from the block. Otherwise, the output from the EOF port defaults to 0.

Multimedia outputs

Specify `Video and audio`, `Video only`, or `Audio only` output file type. This parameter becomes available only when a video signal has both audio and video.

Samples per audio frame

Specify number of samples per audio frame. This parameter becomes available only for audio-only files.

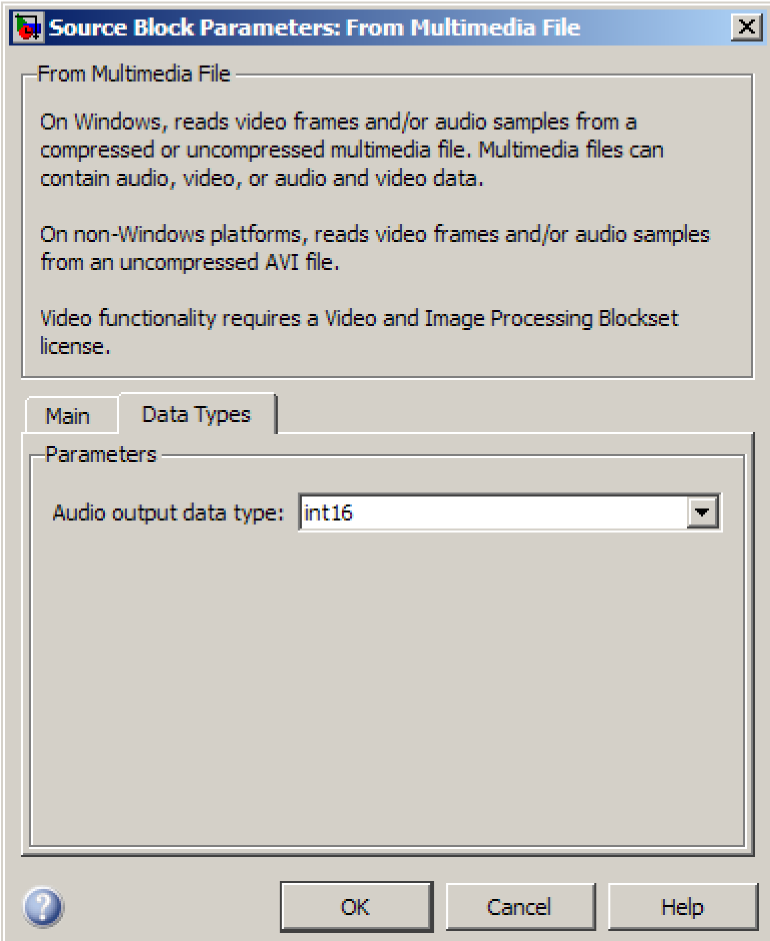
Output color format

Specify whether you want the block to output `RGB`, `Intensity`, or `YCbCr 4:2:2` video frames. This parameter becomes available only for a signal that contains video. If you select `RGB`, use the **Image signal** parameter to specify how to output a color signal.

Image signal

Specify how to output a color video signal. If you select `One multidimensional signal`, the block outputs an *M*-by-*N*-by-*P* color video signal, where *P* is the number of color planes, at one port. If you select `Separate color signals`, additional ports appear on the block. Each port outputs one *M*-by-*N* plane of an RGB video stream. This parameter becomes available only if you set the **Image color space** parameter to `RGB` and the signal contains video.

The **Data Types** pane of the To Multimedia File block dialog box appears as follows.



Audio output data type

Set the data type of the audio samples output at the Audio port. This parameter becomes available only if the multimedia file contains audio. You can choose `double`, `single`, `int16`, or `uint8` types.

Video output data type

Set the data type of the video frames output at the **R**, **G**, **B**, or **Image** ports. This parameter becomes available only if the multimedia file contains video. You can choose `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, or `Inherit from file` types.

Supported Data Types

For sink blocks to display video data properly, double- and single-precision floating-point pixel values must be between `0` and `1`. For other data types, the pixel values must be between the minimum and maximum values supported by their data type.

Port	Supported Data Types	Supports Complex Values?
Image	• Double-precision floating point • Single-precision floating point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers	No
R, G, B	Same as the Image port	No
Audio	• Double-precision floating point • Single-precision floating point • 16-bit signed integers • 8-bit unsigned integers	No
Y, Cb,Cr	Same as the Image port	No

See Also

To Multimedia File	Signal Processing Blockset
Image From Workspace	Video and Image Processing Blockset
To Video Display	Video and Image Processing Blockset
Video From Workspace	Video and Image Processing Blockset
Video Viewer	Video and Image Processing Blockset
“Working with Sample Times”	Simulink

From Wave Device (Obsolete)

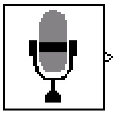
Purpose

Read audio data from standard audio device in real-time (32-bit Windows operating systems only)

Library

`dspwin32`

Description



Note The From Wave Device block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the From Audio Device block.

The From Wave Device block reads audio data from a standard Windows audio device in real-time. It is compatible with most popular Windows hardware, including Sound Blaster cards. (Models that contain both this block and the To Wave Device block require a *duplex-capable* sound card.)

The **Use default audio device** parameter allows the block to detect and use the system's default audio hardware. This option should be selected on systems that have a single sound device installed, or when the default sound device on a multiple-device system is the desired source. In cases when the default sound device is not the desired input source, clear **Use default audio device**, and select the desired device in the **Audio device menu** parameter.

When the audio source contains two channels (stereo), the **Stereo** check box should be selected. When the audio source contains a single channel (mono), the **Stereo** check box should be cleared. For stereo input, the block's output is an M-by-2 matrix containing one frame (M consecutive samples) of audio data from each of the two channels. For mono input, the block's output is an M-by-1 matrix containing one frame (M consecutive samples) of audio data from the mono input. The frame size, M, is specified by the **Samples per frame** parameter. For M=1, the output is sample based; otherwise, the output is frame based.

The audio data is processed in uncompressed pulse code modulation (PCM) format, and should typically be sampled at one of the standard Windows audio device rates: 8000, 11025, 22050, or 44100 Hz. You can select one of these rates from the **Sample rate** parameter. To specify a different rate, select the **User-defined** option and enter a value in the **User-defined sample rate** parameter.

The **Sample Width (bits)** parameter specifies the number of bits used to represent the signal samples read by the audio device. The following settings are available:

- `8` — allocates 8 bits to each sample, allowing a resolution of 256 levels
- `16` — allocates 16 bits to each sample, allowing a resolution of 65536 levels
- `24` — allocates 24 bits to each sample, allowing a resolution of 16777216 levels (only for use with 24-bit audio devices)

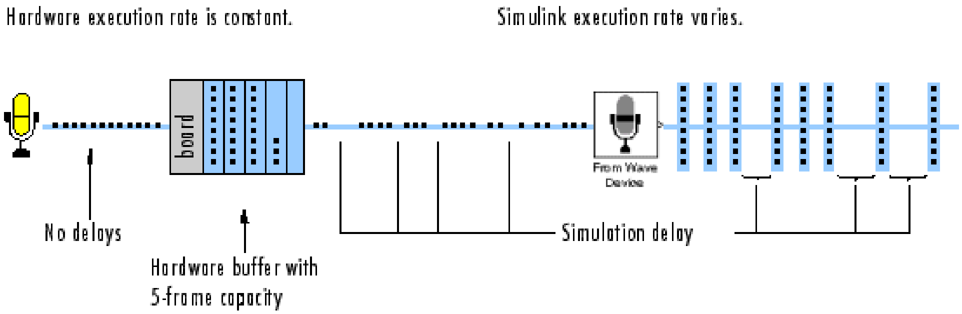
Higher sample width settings require more memory but yield better fidelity. The output from the block is independent of the **Sample width (bits)** setting. The output data type is determined by the **Data type** parameter setting.

Buffering

Since the audio device accepts real-time audio input, Simulink software must read a continuous stream of data from the device throughout the simulation. Delays in reading data from the audio hardware can result in hardware errors or distortion of the signal. This means that the From Wave Device block must read data from the audio hardware as quickly as the hardware itself acquires the signal. However, the block often *cannot* match the throughput rate of the audio hardware, especially when the simulation is running from within Simulink rather than as generated code. (Simulink operations are generally slower than comparable hardware operations, and execution speed routinely varies during the simulation as the host operating system services other processes.) The block must therefore rely on a buffering strategy to ensure that signal data can be read on schedule without losing samples.

At the start of the simulation, the audio device begins writing the input data to a (hardware) buffer with a capacity of T_b seconds. The From Wave Device block immediately begins pulling the earliest samples off the buffer (first in, first out) and collecting them in length-M frames for output. As the audio device continues to append inputs to the bottom of the buffer, the From Wave Device block continues to pull inputs off the top of the buffer at the best possible rate.

The following figure shows an audio signal being acquired and output with a frame size of 8 samples. The buffer of the sound board is approaching its five-frame capacity at the instant shown, which means that the hardware is adding samples to the buffer more rapidly than the block is pulling them off. (If the signal sample rate was 8 kHz, this small buffer could hold approximately 0.005 second of data.



When the simulation throughput rate is higher than the hardware throughput rate, the buffer remains empty throughout the simulation. If necessary, the From Wave Device block simply waits for new samples to become available on the buffer (the block does not interpolate between samples). More typically, the simulation throughput rate is lower than the hardware throughput rate, and the buffer tends to fill over the duration of the simulation.

Troubleshooting

When the buffer size is too small in relation to the simulation throughput rate, the buffer might fill before the entire length of signal is processed. This usually results in a device error or undesired device output. When this problem occurs, you can choose to either increase the buffer size or the simulation throughput rate:

- *Increase the buffer size*

  The **Queue duration** parameter specifies the duration of signal, T_b (in real-time seconds), that can be buffered in hardware during the simulation. Equivalently, this is the maximum length of time that the block's data acquisition can lag the hardware's data acquisition. The number of frames buffered is approximately

  $$\frac{T_b F_s}{M}$$

  where F_s is the sample rate of the signal and M is the number of samples per frame. The required buffer size for a given signal depends on the signal length, the frame size, and the speed of the simulation. Note that increasing the buffer size might increase model latency.

- *Increase the simulation throughput rate*

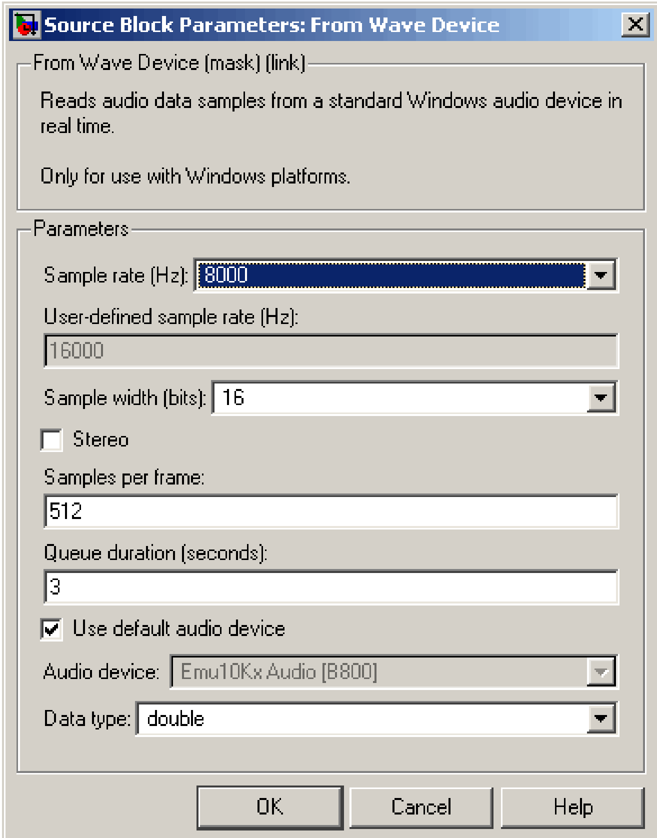
  Two useful methods for improving simulation throughput rates are increasing the signal frame size and compiling the simulation into native code:

  - Increase frame sizes (and convert sample-based signals to frame-based signals) throughout the model to reduce the amount of block-to-block communication overhead. This can drastically increase throughput rates in many cases. However, larger frame sizes generally result in greater model latency due to initial buffering operations.

  - Generate executable code with Real Time Workshop. Native code runs much faster than Simulink, and should provide rates adequate for real-time audio processing.

More general ways to improve throughput rates include simplifying the model, and running the simulation on a faster PC processor. See "Delay and Latency" and "Improving Simulation Performance and Accuracy" in the Simulink documentation for other ideas on improving simulation performance.

Dialog Box



Sample rate (Hz)
: The sample rate of the audio data to be acquired. Select one of the standard Windows rates or the `User-defined` option.

User-defined sample rate (Hz)
: The (nonstandard) sample rate of the audio data to be acquired.

Sample width (bits)
: The number of bits used to represent each signal sample.

Stereo
Specifies stereo (two-channel) inputs when selected, mono (one-channel) inputs when cleared. Stereo output is M-by-2; mono output is M-by-1.

Samples per frame
The number of audio samples in each successive output frame, M. When the value of this parameter is 1, the block outputs a sample-based signal.

Queue duration (seconds)
The length of signal (in seconds) to buffer to the hardware at the start of the simulation.

Use default audio device
Reads audio input from the system's default audio device when selected. Clear to enable the **Audio device ID** parameter and select a device.

Audio device
The name of the audio device from which to read the audio output (lists the names of the installed audio device drivers). Select **Use default audio device** when the system has only a single audio card installed.

Data type
The data type of the output: double-precision, single-precision, signed 16-bit integer, or unsigned 8-bit integer.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- 16-bit signed integer
- 8-bit unsigned integer

See Also

From Wave File (Obsolete)	Signal Processing Blockset
To Wave Device (Obsolete)	Signal Processing Blockset
`audiorecorder`	MATLAB

From Wave File (Obsolete)

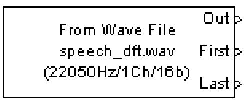
Purpose

Read audio data from Microsoft Wave (`.wav`) file

Library

`dspwin32`

Description



From Wave File

Note The From Wave File block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the From Multimedia File block.

The From Wave File block streams audio data from a Microsoft® Wave (`.wav`) file and generates a signal with one of the data types and amplitude ranges in the following table.

Note AVI files are the only supported file type for non-Windows platforms.

Output Data Type	Output Amplitude Range
`double`	±1
`single`	±1
`int16`	-32768 to 32767 (-2^{15} to 2^{15} - 1)
`uint8`	0 to 255

The audio data must be in uncompressed pulse code modulation (PCM) format.

```
y = wavread('filename')       % Equivalent MATLAB code
```

The block supports 8-, 16-, 24-, and 32-bit Microsoft Wave (`.wav`) files.

The **File name** parameter can specify an absolute or relative path to the file. When the file is on the MATLAB path or in the current folder (the folder returned by typing `pwd` at the MATLAB command line), you need only specify the file's name. You do not need to specify the `.wav` extension.

For an audio file containing C channels, the block's output is an M-by-C matrix containing one frame (M consecutive samples) of audio data from each channel. The frame size, M, is specified by the **Samples per output frame** parameter. For M=1, the output is sample based; otherwise, the output is frame based.

The output frame period, T_{fo}, is

$$T_{fo} = \frac{M}{F_s}$$

where F_s is the data sample rate in Hz.

To reduce the required number of file accesses, the block acquires L consecutive samples from the file during each access, where L is specified by the **Minimum number of samples for each read from file** parameter ($L \geq M$). For $L < M$, the block instead acquires M consecutive samples during each access. Larger values of L result in fewer file accesses, which reduces run-time overhead.

Use the **Data type** parameter to specify the data type of the block's output. Your choices are `double`, `single`, `uint8`, or `int16`.

Select the **Loop** check box if you want to play the file more than once. Then, enter the number of times to play the file. The number you enter must be a positive integer or `inf`.

Use the **Number of times to play file** parameter to enter the number of times to play the file. The number you enter must be a positive integer or `inf`, to play the file until you stop the simulation.

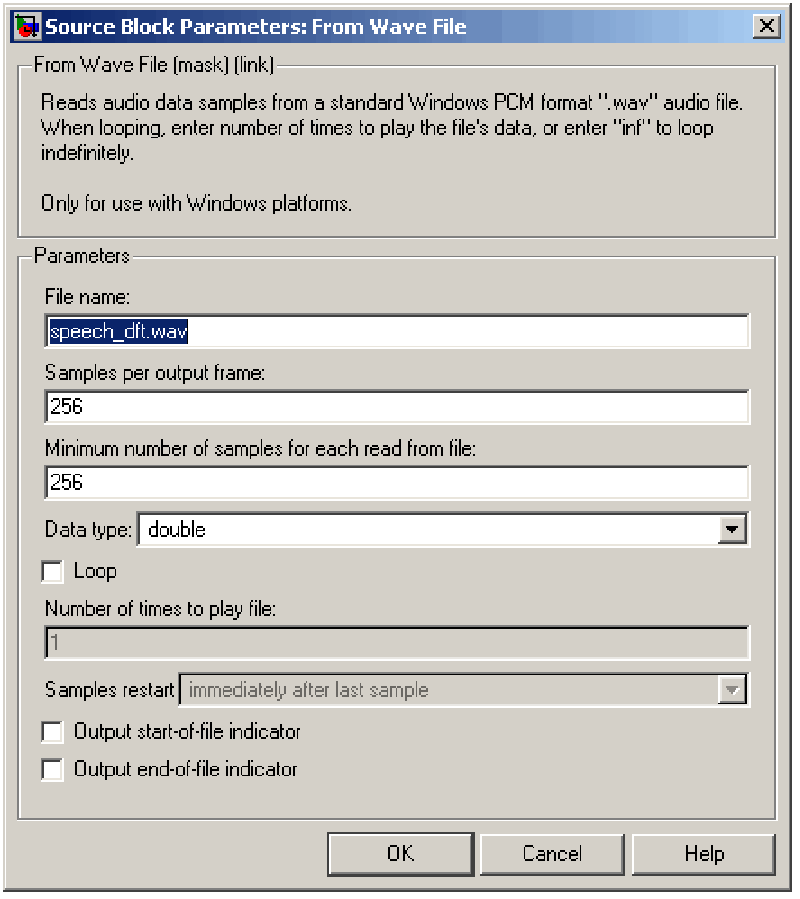
The **Samples restart** parameter determines whether the samples from the audio file repeat immediately or repeat at the beginning of the next frame output from the output port. When you select `immediately after last sample`, the samples repeat immediately. When you select `at beginning of next frame`, the frame containing the last sample value from the audio file is zero padded until the frame is filled. The block then places the first sample of the audio file in the first position of the next output frame.

Use the **Output start-of-file indicator** parameter to determine when the first audio sample in the file is output from the block. When you select this check box, a Boolean output port labeled SOF appears on the block. The output from the SOF port is 1 when the first audio sample in the file is output from the block. Otherwise, the output from the SOF port is 0.

Use the **Output end-of-file indicator** parameter to determine when the last audio sample in the file is output from the block. When you select this check box, a Boolean output port labeled EOF appears on the block. The output from the EOF port is 1 when the last audio sample in the file is output from the block. Otherwise, the output from the EOF port is 0.

The block icon shows the name, sample rate (in Hz), number of channels (1 or 2), and sample width (in bits) of the data in the specified audio file. All sample rates are supported; the sample width must be either 8, 16, 24, or 32 bits.

Dialog Box



File name

Enter the path and name of the file to read. Paths can be relative or absolute.

Samples per output frame

Enter the number of samples in each output frame, *M*. When the value of this parameter is 1, the block outputs a sample-based signal.

Minimum number of samples for each read from file
Enter the number of consecutive samples to acquire from the file with each file access, *L*.

Data type
Select the output data type: `double`, `single`, `uint8`, or `int16`. The data type setting determines the output's amplitude range, as shown in the preceding table.

Loop
Select this check box if you want to play the file more than once.

Number of times to play file
Enter the number of times you want to play the file.

Samples restart
Select `immediately after last sample` to repeat the audio file immediately. Select `at beginning of next frame` to place the first sample of the audio file in the first position of the next output frame.

Output start-of-file indicator
Use this check box to determine whether the output contains the first audio sample in the file.

Output end-of-file indicator
Use this check box to determine whether the output contains the last audio sample in the file.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- 16-bit signed integer
- 8-bit unsigned integer

See Also

From Audio Device	Signal Processing Blockset
Signal From Workspace	Signal Processing Blockset

To Multimedia File	Signal Processing Blockset
`wavread`	MATLAB

Purpose

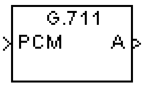
Quantize narrowband speech input signals

Library

Quantizers

`dspquant2`

Description



G.711 Codec

The G711 Codec block is a logarithmic scalar quantizer designed for narrowband speech. Narrowband speech is defined as a voice signal with an analog bandwidth of 4 kHz and a Nyquist sampling frequency of 8 kHz. The block quantizes a narrowband speech input signal so that it can be transmitted using only 8-bits. The G711 Codec block has three modes of operation: encoding, decoding, and conversion. You can choose the block's mode of operation by setting the **Mode** parameter.

If, for the **Mode** parameter, you choose `Encode PCM to A-law`, the block assumes that the linear PCM input signal has a dynamic range of 13 bits. Because the block always operates in saturation mode, it assigns any input value above $2^{12}-1$ to $2^{12}-1$ and any input value below -2^{12} to -2^{12}. The block implements an A-law quantizer on the input signal and outputs A-law index values. When you choose `Encode PCM to mu-law`, the block assumes that the linear PCM input signal has a dynamic range of 14 bits. Because the block always operates in saturation mode, it assigns any input value above $2^{13}-1$ to $2^{13}-1$ and any input value below -2^{13} to -2^{13}. The block implements a mu-law quantizer on the input signal and outputs mu-law index values.

If, for the **Mode** parameter, you choose `Decode A-law to PCM`, the block decodes the input A-law index values into quantized output values using an A-law lookup table. When you choose `Decode mu-law to PCM`, the block decodes the input mu-law index values into quantized output values using a mu-law lookup table.

If, for the **Mode** parameter, you choose `Convert A-law to mu-law`, the block converts the input A-law index values to mu-law index values. When you choose `Convert mu-law to A-law`, the block converts the input mu-law index values to A-law index values.

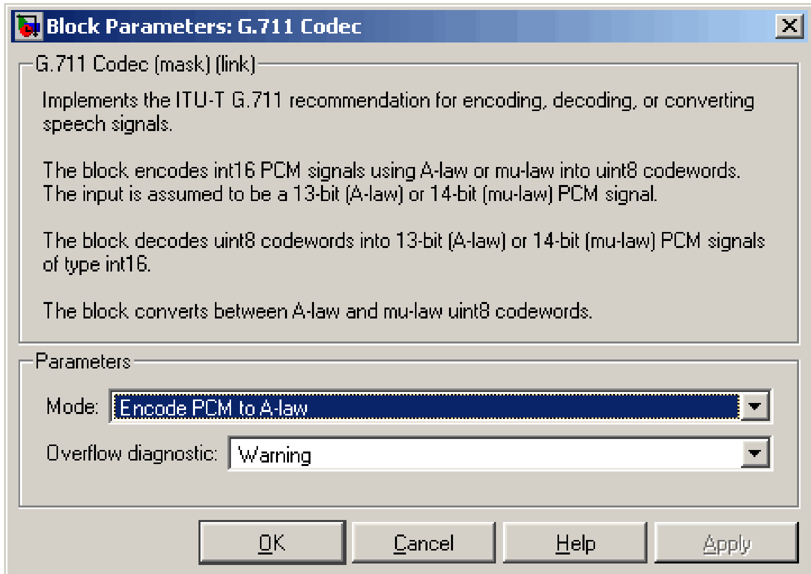
Note Set the **Mode** parameter to `Convert A-law to mu-law` or `Convert mu-law to A-law` only when the input to the block is A-law or mu-law index values.

If, for the **Mode** parameter, you choose `Encode PCM to A-law` or `Encode PCM to mu-law`, the **Overflow diagnostic** parameter appears on the block parameters dialog box. Use this parameter to determine the behavior of the block when overflow occurs. The following options are available:

- `Ignore` — Proceed with the computation and do not issue a warning message.
- `Warning` — Display a warning message in the MATLAB Command Window, and continue the simulation.
- `Error` — Display an error dialog box and terminate the simulation.

Note Like all diagnostic parameters on the Configuration Parameters dialog box, **Overflow diagnostic** parameter is set to `Ignore` in the code generated for this block by Real-Time Workshop code generation software.

Dialog Box



Mode

- When you choose `Encode PCM to A-law`, the block implements an A-law encoder.
- When you choose `Encode PCM to mu-law`, the block implements a mu-law encoder.
- When you choose `Decode A-law to PCM`, the block decodes the input index values into quantized output values using an A-law lookup table.
- When you choose `Decode mu-law to PCM`, the block decodes the input index values into quantized output values using a mu-law lookup table.
- When you choose `Convert A-law to mu-law`, the block converts the input A-law index values to mu-law index values.
- When you choose `Convert mu-law to A-law`, the block converts the input mu-law index values to A-law index values.

Overflow diagnostic

Use this parameter to determine the behavior of the block when overflow occurs.

- Select `Ignore` to proceed with the computation without a warning message.
- Select `Warning` to display a warning message in the MATLAB Command Window and continue the simulation.
- Select `Error` to display an error dialog box and terminate the simulation.

This parameter is only visible if, for the **Mode** parameter, you select `Encode PCM to A-law` or `Encode PCM to mu-law`.

References

ITU-T Recommendation G.711, "Pulse Code Modulation (PCM) of Voice Frequencies," *General Aspects of Digital Transmission Systems*; *Terminal Equipments*, International Telecommunication Union (ITU), 1993.

Supported Data Types

Port	Supported Data Types
PCM	• 16-bit signed integers
A	• 8-bit unsigned integers
mu	• 8-bit unsigned integers

See Also

Quantizer	Simulink
Scalar Quantizer Decoder	Signal Processing Blockset
Scalar Quantizer Design	Signal Processing Blockset
Uniform Decoder	Signal Processing Blockset
Uniform Encoder	Signal Processing Blockset
Vector Quantizer Decoder	Signal Processing Blockset

Vector Quantizer Design	Signal Processing Blockset
Vector Quantizer Encoder	Signal Processing Blockset

Halfband Filter

Purpose

Design halfband filter

Library

Filtering / Filter Designs

`dspfdesign`

Description

This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. You must have a Filter Design Toolbox license to design filters with this block. However, you can run models containing this block without a license. This allows you to run a model sent to you by a colleague who has designed a filter using this block, even if you do not have the Filter Design Toolbox product.

Dialog Box

See “Halfband Filter Design Dialog Box — Main Pane” in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Purpose

Design highpass filter

Library

Filtering / Filter Designs

`dspfdesign`

Description

This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. Without a Filter Design Toolbox license, you can run models that contain this block, and can edit some, but not all, block parameters. To enable the full filter design functionality of this block, you must have a Filter Design Toolbox license.

Dialog Box

See "Highpass Filter Design Dialog Box — Main Pane" in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Purpose

Design Hilbert filter

Library

Filtering / Filter Designs

`dspfdesign`

Description

This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. Without a Filter Design Toolbox license, you can run models that contain this block, and can edit some, but not all, block parameters. To enable the full filter design functionality of this block, you must have a Filter Design Toolbox license.

Dialog Box

See "Hilbert Filter Design Dialog Box — Main Pane" in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Histogram

Purpose Generate histogram of input or sequence of inputs

Library Statistics

`dspstat3`

Description The Histogram block computes the frequency distribution of the elements in a vector input, of the elements in each channel of a frame-based matrix input, or of the elements in a sample based N-D array. The **Running histogram** parameter selects between basic operation and running operation, described below. The Histogram block accepts real and complex fixed-point and floating-point inputs.

The block distributes the elements of the input into the number of discrete bins specified by the **Number of bins** parameter, `n`.

```
y = hist(u,n)     % Equivalent MATLAB code
```

Complex fixed-point inputs are distributed according to their magnitude squared values; complex floating-point inputs are distributed by their normalized values.

The histogram value for a given bin represents the frequency of occurrence of the input values bracketed by that bin. You specify the upper boundary of the highest-valued bin in the **Upper limit of histogram** parameter, B_M, and the lower boundary of the lowest-valued bin in the **Lower limit of histogram** parameter, B_m. The bins have equal width of

$$\Delta = \frac{B_M - B_m}{n}$$

and centers located at

$$B_m + \left(k + \frac{1}{2}\right)\Delta \qquad k = 0,1,2,\ldots,n-1$$

Input values that fall on the border between two bins are placed into the lower valued bin; that is, each bin includes its upper boundary. For example, a bin of width 4 centered on the value 5 contains the input value 7, but not the input value 3. Input values greater than the **Upper limit of histogram** parameter or less than **Lower limit of histogram** parameter are placed into the highest valued or lowest valued bin, respectively.

The values you enter for the **Upper limit of histogram** and **Lower limit of histogram** parameters must be real-valued scalars. `NaN` and `inf` are not valid values for the **Upper limit of histogram** and **Lower limit of histogram** parameters.

Basic Operation

When the **Running histogram** check box is not selected, the Histogram block computes the frequency distribution of the current input.

For frame-based M-by-N inputs, (including 1-by-N row vectors and M-by-1 column vectors), the Histogram block computes a histogram for each channel of the M-by-N matrix independently. The block outputs an n-by-N matrix, where n is the **Number of bins** specified in the Histogram block. The jth column of the output matrix contains the histogram for the data in the jth column of the M-by-N input matrix.

For all sample-based N-D input arrays, including length-M 1-D vectors and 1-by-N row vectors, the Histogram block computes the frequency distribution of the input data. The block outputs an n-by-1 vector, where n is the **Number of bins** specified in the Histogram block.

Running Operation

When you select the **Running histogram** check box, the Histogram block computes the frequency distribution of both the past and present data for successive inputs. The block resets the histogram (by emptying all of the bins) when it detects a reset event at the optional Rst port. See “Resetting the Running Histogram” on page 2-642 for more information on how to trigger a reset.

For frame-based M-by-N inputs (including 1-by-N row vectors and M-by-1 column vectors), the Histogram block computes a running histogram for each channel of the M-by-N matrix. The block outputs an n-by-N matrix, where n is the **Number of bins** specified in the Histogram block. The jth column of the output matrix contains the running histogram for the jth column of the M-by-N input matrix.

For all sample-based N-D input arrays, including length-M 1-D vectors, the Histogram block computes a running histogram for the data in the first dimension of the input. The block outputs an n-by-1 vector, where n is the **Number of bins** specified in the Histogram block.

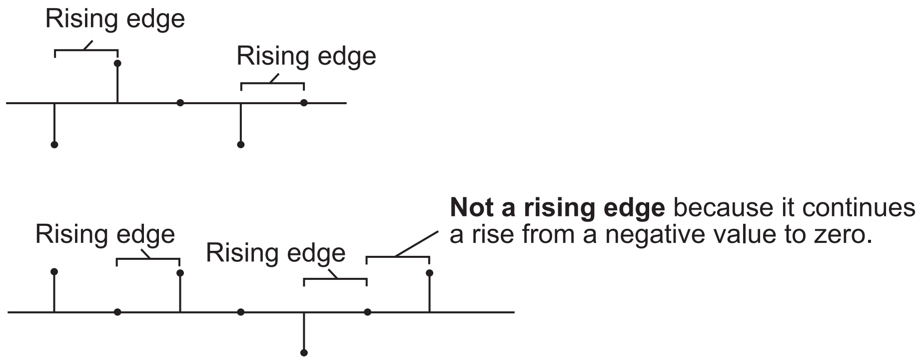
Note When the histogram block is used in running mode and the input data type is non-floating point, the output of the histogram is stored as a `uint32` data type. The largest number that can be represented by this data type is $2^{32}-1$. If the range of the `uint32` data type is exceeded, the output data will wrap back to 0.

Resetting the Running Histogram

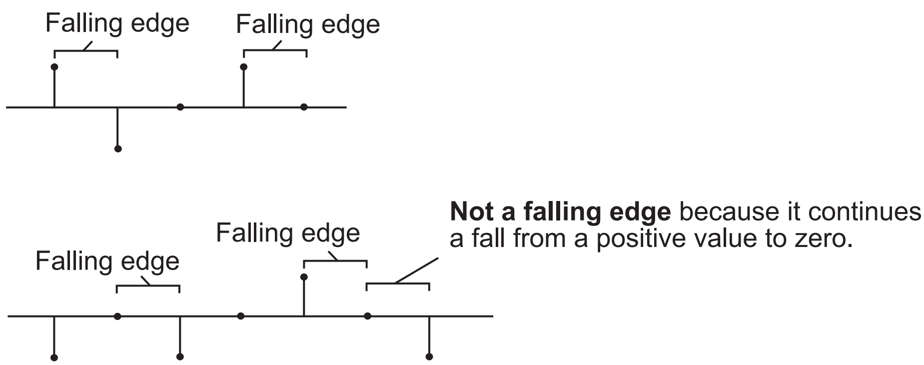
The block resets the running histogram whenever a reset event is detected at the optional Rst port. The reset signal and the input data signal must be the same rate.

You specify the reset event using the **Reset port** menu:

- `None` — Disables the Rst port
- `Rising edge` — Triggers a reset operation when the Rst input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)



- `Falling edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)



- `Either edge` — Triggers a reset operation when the `Rst` input is a `Rising edge` or `Falling edge` (as described earlier)
- `Non-zero sample` — Triggers a reset operation at each sample time that the `Rst` input is not zero

Examples

Real Input Data

The bin boundaries created by the Histogram block are determined by the data type of the input:

- Bin boundaries for real, double-precision input are cast into the data type `double`.
- Bin boundaries for real, fixed-point input are cast into the `int8` data type.

The following example shows the differences in the output of the Histogram block based on the data type of the input.

To create this model you need the following blocks.

Block	Library	Quantity
Constant	Simulink / Sources library	2
Display	Signal Processing Sinks	2
Histogram	Statistics	2

The parameter settings for the Double Precision Input Constant block are:

- **Constant value** = `double([1 2 3 4 5]')`
- **Interpret vector parameters as 1-D** = Clear this check box.
- **Sampling mode** = `Sample based`
- **Sample time** = `inf`

The parameter settings for the Fixed-Point Input Constant block are:

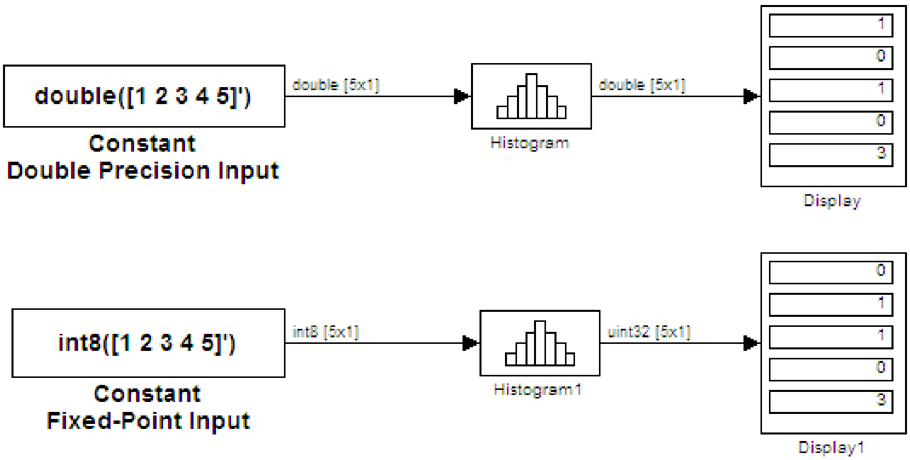
- **Constant value** = `int8([1 2 3 4 5]')`
- **Interpret vector parameters as 1-D** = Clear this check box.
- **Sampling mode** = `Sample based`

- **Sample time** = `inf`

The parameter settings for both Histogram blocks are:

- **Lower limit of histogram** = `1`
- **Upper limit of histogram** = `3`
- **Number of bins** = `5`
- **Normalized** = Clear this check box.
- **Running histogram** = Clear this check box.

Connect the blocks as shown in the following figure, and run your model.

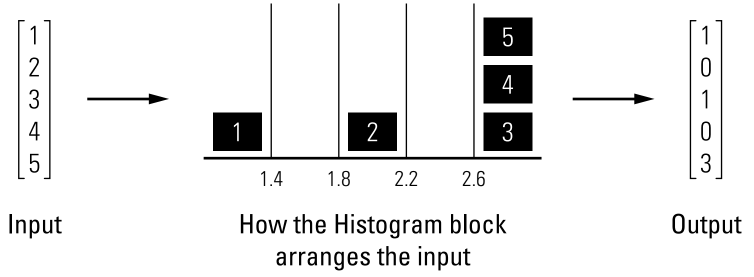


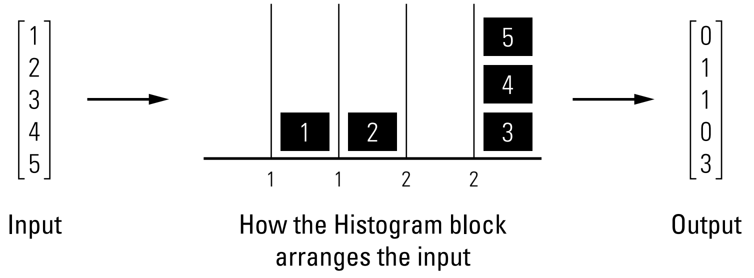
Running this model generates the following warning:

```
Warning:  The bin width resulting from the specified
parameters is less than the precision of the input data
type.  This might cause unexpected results.  Since bin
width is calculated by ((upper limit - lower limit)/number
of bins), you could increase upper limit or decrease lower
limit or number of bins.
```

This warning alerts you that it is not a good use case to have a histogram where 2 or more bin boundaries are the same. As the warning suggests, increasing the range of the limits of the histogram, or decreasing the number of bins, can correct this problem.

The following figures illustrate the different bins that are created by the Histogram block. The top figure shows the histogram for double-precision input, and the bottom figure shows the histogram for fixed-point input. The output of the histogram block differs based on the data type of the input, and the bin boundaries are duplicated in the histogram for the fixed-point input.





Complex Input Data

The bin boundaries created by the Histogram block are determined by the data type of the input.

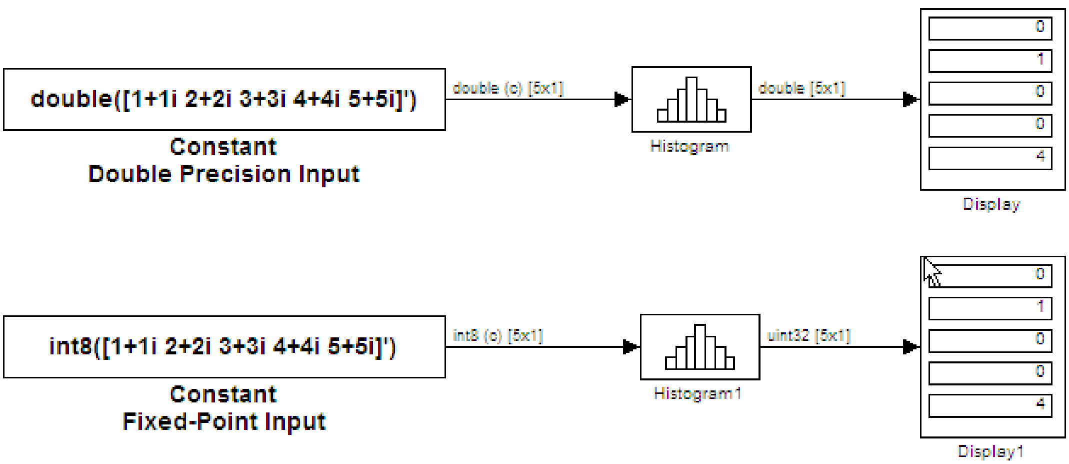
- Bin boundaries for complex, double-precision input are cast into the data type `double`. All complex, double-precision input values are placed in bins according to their normalized values.
- Bin boundaries for complex, fixed-point input are cast into the data type `double` and squared. All complex, fixed-point input values are placed in bins according to their magnitude-squared value.

The following example shows the differences in the bins created by the Histogram block based on the data type of the complex input.

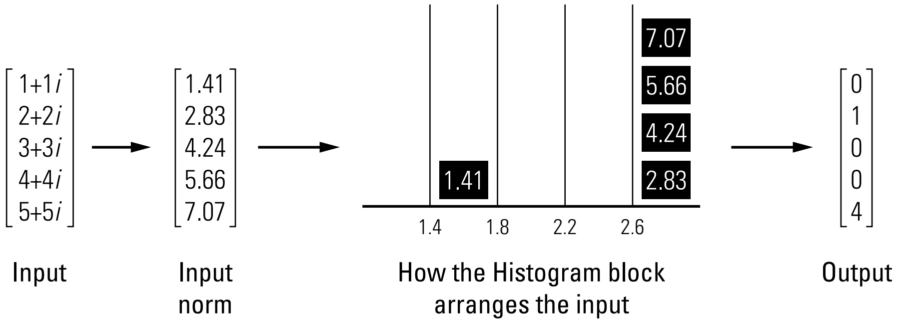
Using the same model you created for the example with real input data, modify the following parameters:

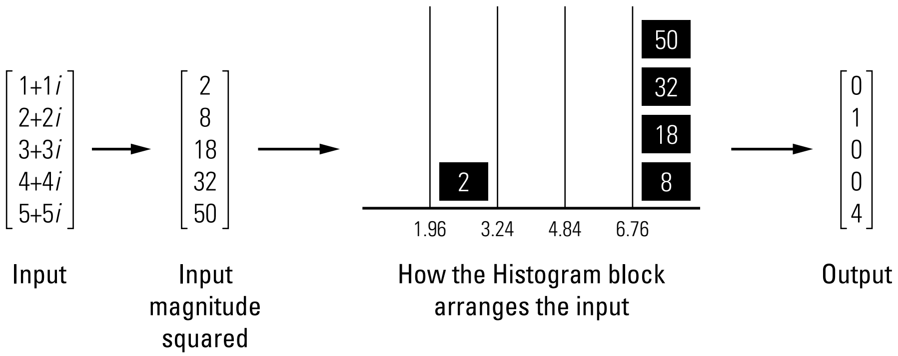
- In the Double Precision Input Constant block, set the **Constant value** parameter to `double([1+1i 2+2i 3+3i 4+4i 5+5i]')`
- In the Fixed-Point Input Constant block, set the **Constant value** parameter to `int8([1+1i 2+2i 3+3i 4+4i 5+5i]')`

Run your model. It should look similar to the following figure:



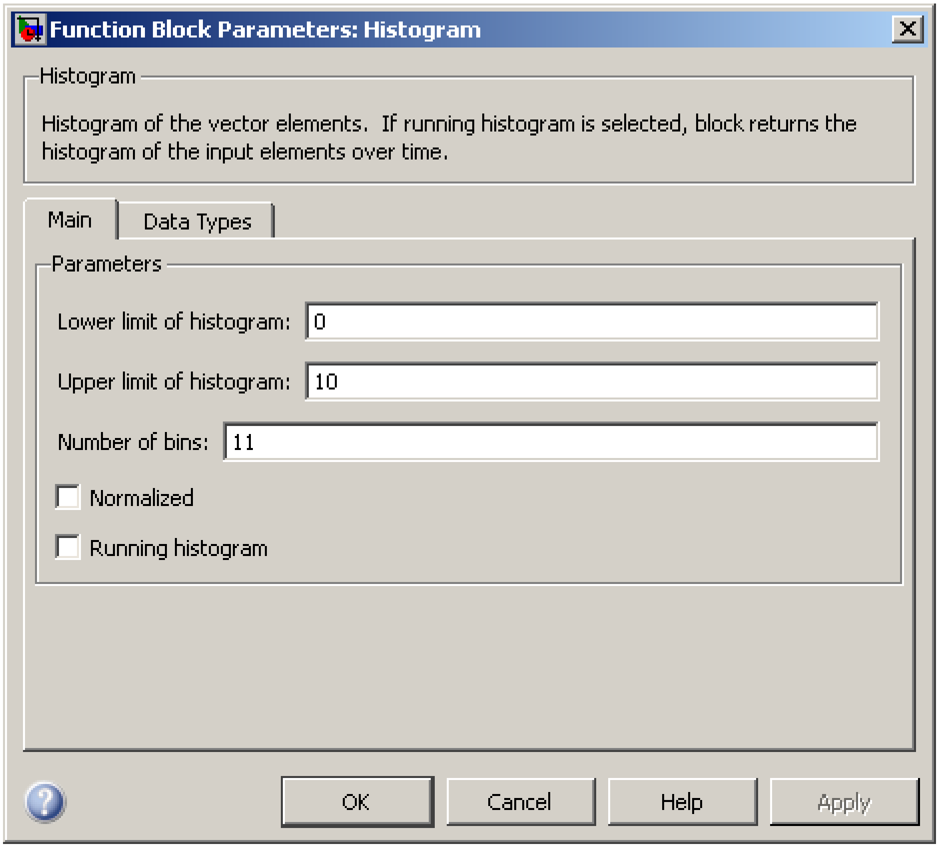
In this case, the Histogram block outputs the same result. The figures below illustrate how the Histogram block compares the input values to the bins it creates. The double-precision inputs are normalized for comparison, whereas the fixed-point inputs are placed using their magnitude squared value. The top figure shows the histogram for the double-precision input, and the bottom figure shows the histogram for the fixed-point input.





Dialog Box

The **Main** pane of the Histogram block dialog appears as follows.



Lower limit of histogram

Enter a real-valued scalar for the lower boundary, B_m, of the lowest-valued bin. `NaN` and `inf` are not valid values for B_m. Tunable.

Upper limit of histogram
: Enter a real-valued scalar for the upper boundary, B_M, of the highest-valued bin. `NaN` and `inf` are not valid values for B_M. Tunable.

Number of bins
: The number of bins, n, in the histogram.

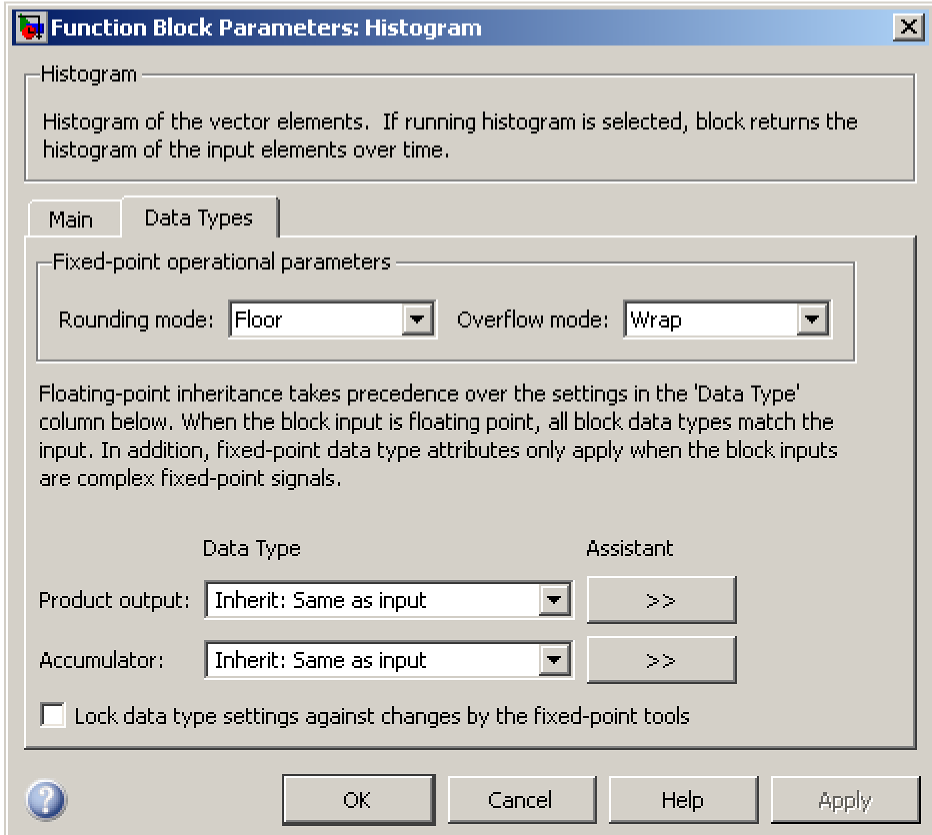
Normalized
: When selected, the output vector, v, is normalized such that `sum(v) = 1`.

  Use of this parameter is not supported for fixed-point signals.

Running histogram
: Set to enable the running histogram operation, and clear to enable basic histogram operation. For more information, see "Basic Operation" on page 2-641 and "Running Operation" on page 2-641.

Reset port
: The type of event that resets the running histogram. For more information, see "Resetting the Running Histogram" on page 2-642. The reset signal and the input data signal must be the same rate. This parameter is enabled only when you select the **Running histogram** check box. For more information, see "Running Operation" on page 2-641.

The **Data Types** pane of the Histogram block dialog appears as follows.



Note The fixed-point parameters listed are only used for fixed-point complex inputs, which are distributed by squared magnitude.

Rounding mode
: Select the rounding mode for fixed-point operations.

Overflow mode
: Select the overflow mode for fixed-point operations.

Product output data type

Specify the product output data type. See “Multiplication Data Types” for illustrations depicting the use of the product output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Accumulator data type

Specify the accumulator data type. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
In	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • 32-bit unsigned integers
Rst	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Sort	Signal Processing Blockset
`hist`	MATLAB

IDCT

Purpose

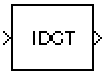
Compute inverse discrete cosine transform (IDCT) of input

Library

Transforms

`dspxfrm3`

Description



The IDCT block computes the inverse discrete cosine transform (IDCT) of each channel in the *M*-by-*N* input matrix, `u`.

```
y = idct(u)     % Equivalent MATLAB code
```

When the input is a sample-based row vector, the IDCT block computes the inverse discrete cosine transform across the vector dimension of the input. For all other sample-based N-D arrays, the block computes the IDCT across the first dimension of the input.

For both sample-based and frame-based inputs, the block assumes that each input column is a frame containing *M* consecutive samples from an independent channel. The frame size, *M*, must be a power of two. To work with other frame sizes, use the Pad block to pad or truncate the frame size to a power of two length.

When the input is an *M*-by-*N* matrix, the output is an *M*-by-*N* matrix whose *l*th column contains the length-*M* IDCT of the corresponding input column.

$$y(m,l) = \sum_{k=1}^{M} w(k)u(k,l)\cos\frac{\pi(2m-1)(k-1)}{2M}, \qquad m = 1,\ldots,M$$

where

$$w(k) = \begin{cases} \dfrac{1}{\sqrt{M}}, & k = 1 \\ \sqrt{\dfrac{2}{M}}, & 2 \le k \le M \end{cases}$$

The **Output sampling mode** parameter allows you to select the sampling mode of the output. If the input is a sample-based N-D array with 3 or more dimensions, the **Output sampling mode** must be `Sample based`. The output sample rate and data type (real/complex) are the same as those of the input.

For convenience, length-*M* 1-D vector inputs and sample-based length-*M* row vector inputs are processed as single channels (that is, as *M*-by-1 column vectors), and the output has the same dimension as the input.

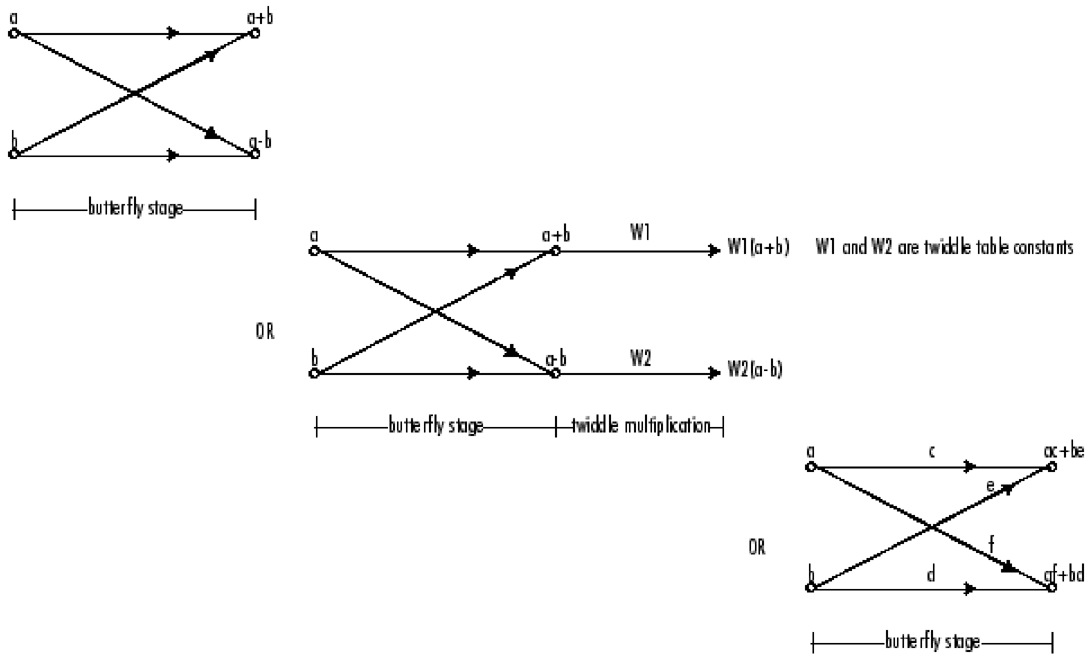
The **Sine and cosine computation** parameter determines how the block computes the necessary sine and cosine values. This parameter has two settings, each with its advantages and disadvantages, as described in the following table.

Sine and Cosine Computation Parameter Setting	Sine and Cosine Computation Method	Effect on Block Performance
`Table lookup`	The block computes and stores the trigonometric values before the simulation starts, and retrieves them during the simulation. When you generate code from the block, the processor running the generated code stores the trigonometric values computed by the block in a speed-optimized table, and retrieves the values during code execution.	The block usually runs much more quickly, but requires extra memory for storing the precomputed trigonometric values.
`Trigonometric fcn`	The block computes sine and cosine values during the simulation. When you generate code from the block, the processor running the generated code computes the sine and cosine values while the code runs.	The block usually runs more slowly, but does not need extra data memory. For code generation, the block requires a support library to emulate the trigonometric functions, increasing the size of the generated code.

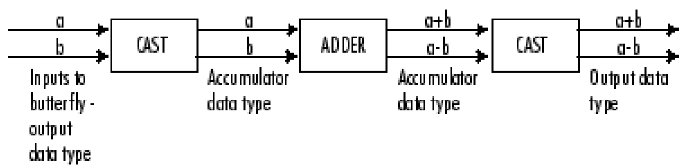
Fixed-Point Data Types

The following diagrams show the data types used within the IDCT block for fixed-point signals. You can set the sine table, accumulator, product output, and output data types displayed in the diagrams in the IDCT block dialog as discussed in “Dialog Box” on page 2-659.

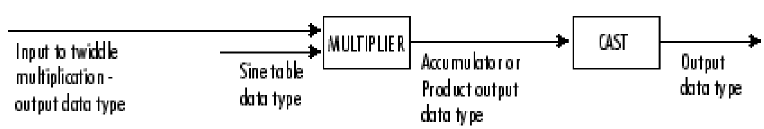
Inputs to the IDCT block are first cast to the output data type and stored in the output buffer. Each butterfly stage processes signals in the accumulator data type, with the final output of the butterfly being cast back into the output data type.



Butterfly Stage Data Types



Twiddle Multiplication Data Types



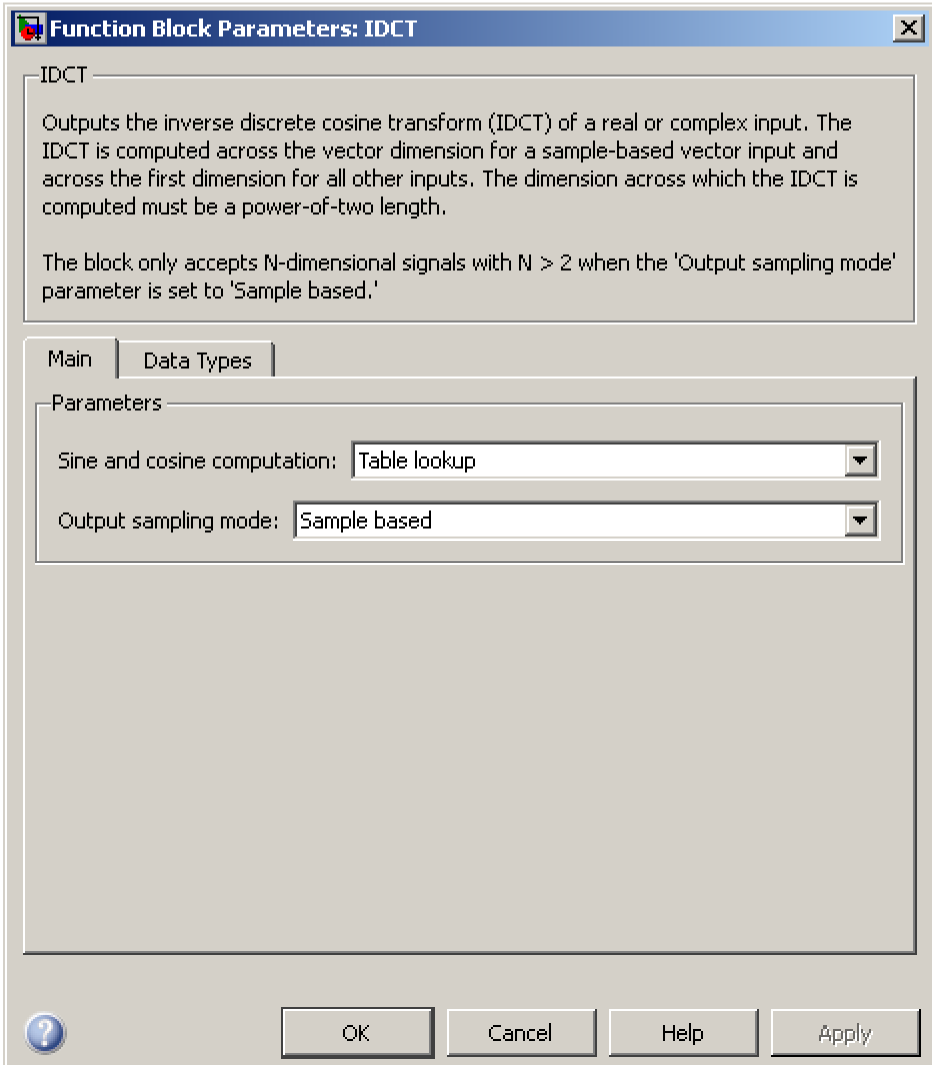
The output of the multiplier is in the product output data type when at least one of the inputs to the multiplier is real. When both of the

inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see "Multiplication Data Types".

Note When the block input is fixed point, all internal data types are signed fixed point.

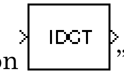
Dialog Box

The **Main** pane of the IDCT block dialog appears as follows.



Sine and cosine computation

Sets the block to compute sines and cosines by either looking up sine and cosine values in a speed-optimized table (`Table lookup`), or by making sine and cosine function calls (`Trigonometric fcn`).

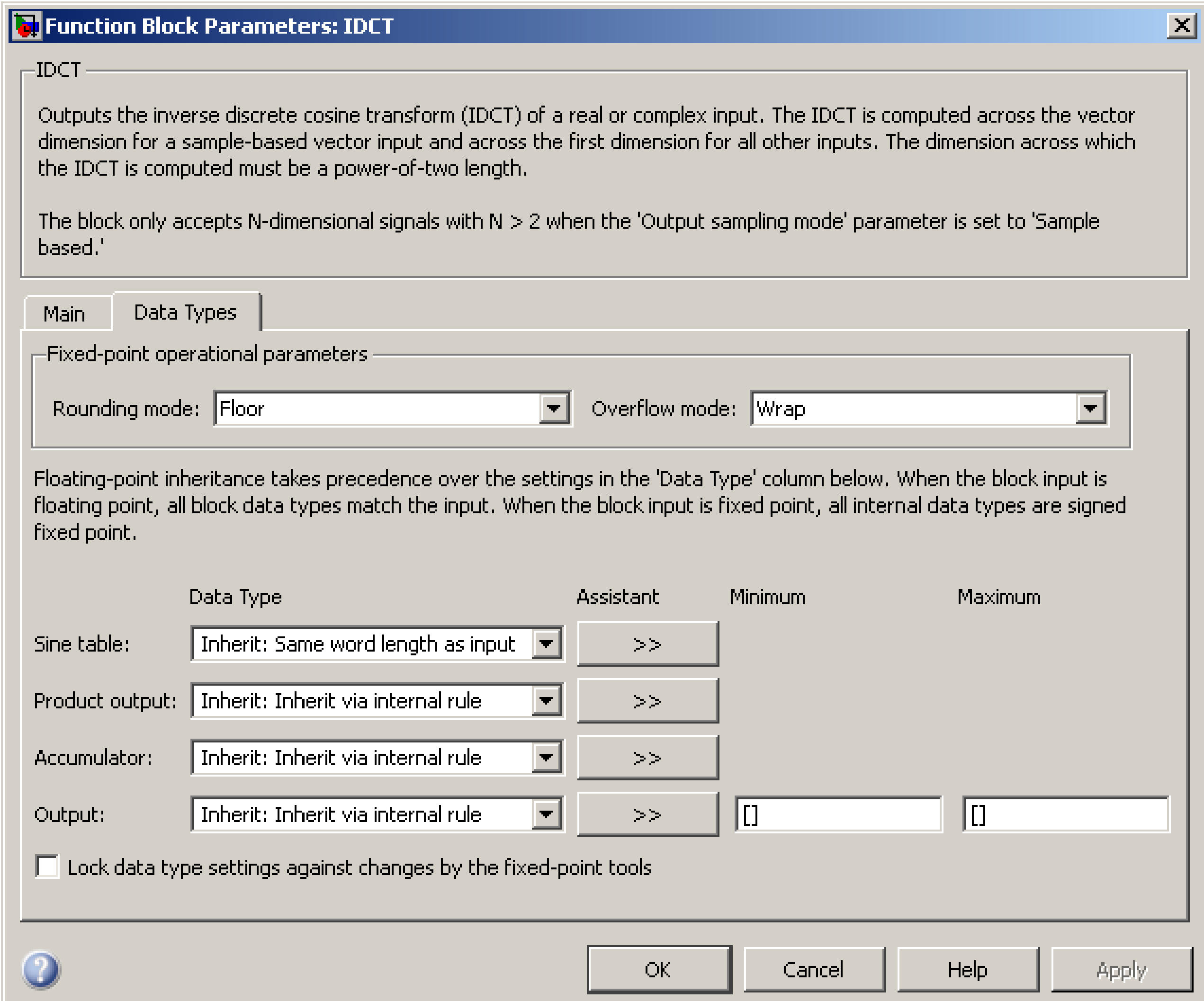
See the table in the "Description 

" on page 2-654 section.

Output sampling mode

Select `Sample based` or `Frame based` output. If the input to the IDCT block has 3 or more dimensions, you must select `Sample based` output.

The **Data Types** pane of the IDCT block dialog appears as follows.


Function Block Parameters: IDCT
IDCT
Outputs the inverse discrete cosine transform (IDCT) of a real or complex input. The IDCT is computed across the vector dimension for a sample-based vector input and across the first dimension for all other inputs. The dimension across which the IDCT is computed must be a power-of-two length.
The block only accepts N-dimensional signals with N > 2 when the 'Output sampling mode' parameter is set to 'Sample based.'
Main
Data Types
Fixed-point operational parameters
Rounding mode:
Floor
Overflow mode:
Wrap
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input. When the block input is fixed point, all internal data types are signed fixed point.
Data Type
Assistant
Minimum
Maximum
Sine table:
Inherit: Same word length as input
>>
Product output:
Inherit: Inherit via internal rule
>>
Accumulator:
Inherit: Inherit via internal rule
>>
Output:
Inherit: Inherit via internal rule
>>
[]
[]
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Rounding mode
Select the rounding mode for fixed-point operations. The sine table values do not obey this parameter; they always round to `Nearest`.

Overflow mode
Select the overflow mode for fixed-point operations. The sine table values do not obey this parameter; instead, they are always saturated.

Sine table data type
Choose how you specify the word length of the values of the sine table. The fraction length of the sine table values always equals the word length minus one. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same word length as input`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16)`

The sine table values do not obey the **Rounding mode** and **Overflow mode** parameters; instead, they are always saturated and rounded to `Nearest`.

Product output data type
Specify the product output data type. See "Fixed-Point Data Types" on page 2-656 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Accumulator data type

Specify the accumulator data type. See "Fixed-Point Data Types" on page 2-656 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Output data type

Specify the output data type. See "Fixed-Point Data Types" on page 2-656 for illustrations depicting the use of the output data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`.

  When you select `Inherit:  Inherit via internal rule`, the block calculates the output word length and fraction length automatically. The internal rule first calculates an ideal output word length and fraction length using the following equations:

  $$WL_{ideal\ output} = WL_{input} + floor(\log_2(DCT\ length - 1)) + 1$$

  $$FL_{ideal\ output} = FL_{input}$$

Using these ideal results, the internal rule then selects word lengths and fraction lengths that are appropriate for your hardware. For more information, see “Inherit via Internal Rule”.

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User’s Guide* for more information.

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

DCT	Signal Processing Blockset
IFFT	Signal Processing Blockset
`idct`	Signal Processing Toolbox

Identity Matrix

Purpose

Generate matrix with ones on main diagonal and zeros elsewhere

Library

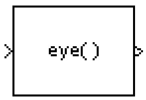
- Signal Processing Sources

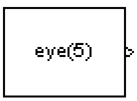
  `dspsrcs4`

- Math Functions / Matrices and Linear Algebra / Matrix Operations

  `dspmtrx3`

Description





The Identity Matrix block generates a rectangular matrix with ones on the main diagonal and zeros elsewhere.

When you select the **Inherit output port attributes from input port** check box, the input port is enabled, and an *M*-by-*N* matrix input generates a sample-based *M*-by-*N* matrix output with the same sample period. The values in the input matrix are ignored.

```
y = eye([M N])      % Equivalent MATLAB code
```

When you do not select the **Inherit output port attributes from input port** check box, the input port is disabled, and the dimensions of the output matrix are determined by the **Matrix size** parameter. A scalar value, *M*, specifies an *M*-by-*M* identity matrix, while a two-element vector, [*M N*], specifies an *M*-by-*N* unit-diagonal matrix. The output is sample based, and has the sample period specified by the **Sample time** parameter.

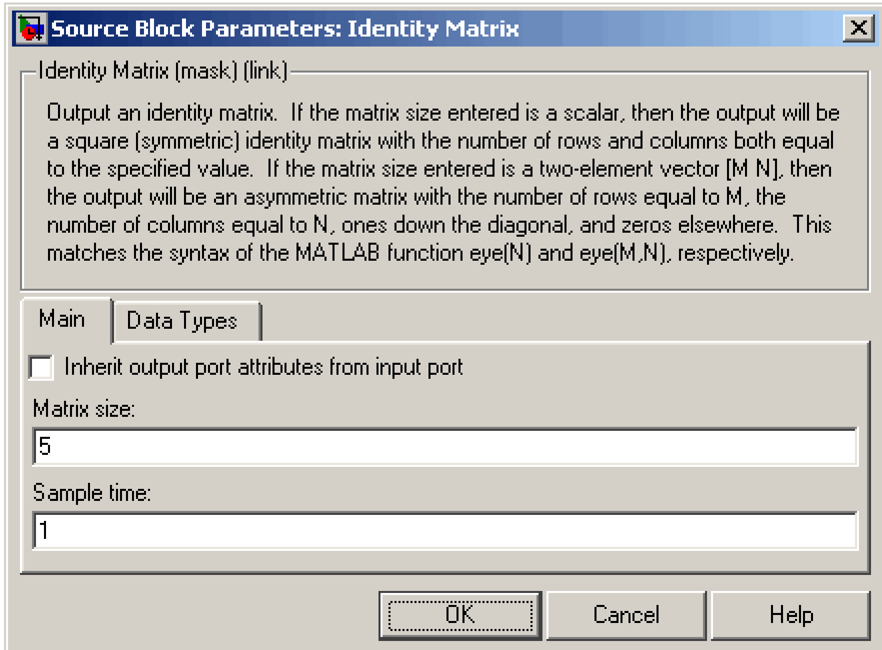
Examples

Set **Matrix size** to `[3 6]` to generate the 3-by-6 unit-diagonal matrix below.

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

Dialog Box

The **Main** pane of the Identity Matrix block dialog appears as follows.



Inherit output port attributes from input port

Enables the input port when selected. In this mode, the output inherits its dimensions, sample period, and data type from the input. The output is always real.

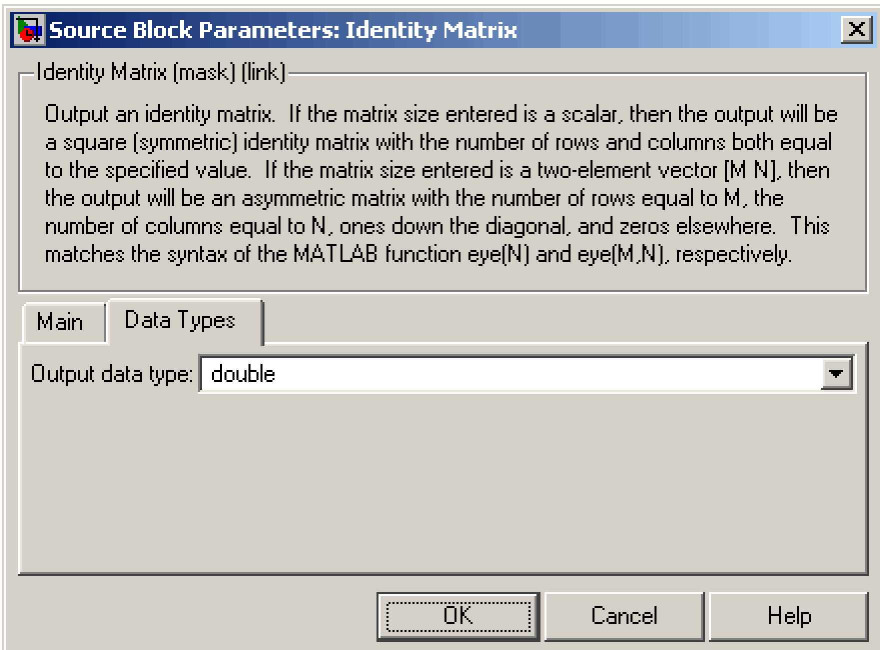
Matrix size

The number of rows and columns in the output matrix: a scalar *M* for a square *M*-by-*M* output, or a vector [*M N*] for an *M*-by-*N* output. This parameter is disabled when you select **Inherit input port attributes from input port**.

Sample time

The discrete sample period of the output. This parameter is disabled when you select **Inherit input port attributes from input port**.

The **Data Types** pane of the Identity Matrix block dialog appears as follows.



Output data type

Specify the output data type in one of the following ways:

- Choose one of the built-in data types from the list.
- Choose `Fixed-point` to specify the output data type and scaling in the **Signed**, **Word length**, **Set fraction length in output to,** and **Fraction length** parameters.

- Choose `User-defined` to specify the output data type and scaling in the **User-defined data type**, **Set fraction length in output to,** and **Fraction length** parameters.
- Choose `Inherit via back propagation` to set the output data type and scaling to match the following block

Signed
Select to output a signed fixed-point signal. Otherwise, the signal is unsigned. This parameter is visible only when you select `Fixed-point` for the **Output data type** parameter.

Word length
Specify the word length, in bits, of the fixed-point output data type. This parameter is visible only when you select `Fixed-point` for the **Output data type** parameter.

User-defined data type
Specify any built-in or fixed-point data type. You can specify fixed-point data types using the following Simulink Fixed Point functions: `sfix`, `ufix`, `sint`, `uint`, `sfrac`, and `ufrac`. This parameter is visible only when you select `User-defined` for the **Output data type** parameter.

Set fraction length in output to
Specify the scaling of the fixed-point output by either of the following two methods:

- Choose `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.
- Choose `User-defined` to specify the output scaling in the **Fraction length** parameter.

This parameter is visible only when you select `Fixed-point` for the **Output data type** parameter, or when you select `User-defined` and the specified output data type is a fixed-point data type.

Fraction length

For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is visible only when you select `Fixed-point` or `User-defined` for the **Output data type** parameter and `User-defined` for the **Set fraction length in output to** parameter.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Constant Diagonal Matrix	Signal Processing Blockset
Constant	Simulink
`eye`	MATLAB

Purpose

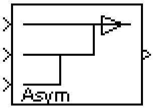
Inverse discrete wavelet transform (IDWT) of input or reconstruct signals from subbands with smaller bandwidths and slower sample rates

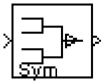
Library

Transforms

`dspxfrm3`

Description





The IDWT block is the same as the Dyadic Synthesis Filter Bank block in the Multirate Filters library, but with different default settings. See the Dyadic Synthesis Filter Bank block reference page for more information on how to use the block.

Purpose

Compute inverse fast Fourier transform (IFFT) of input

Library

Transforms

`dspxfrm3`

Description

The IFFT block computes the inverse fast Fourier transform (IFFT) of each row of a sample-based 1-by-P input vector, u, or across the first dimension (P) of an N-D input array, u. When you select the **Inherit FFT length from input dimensions** check box, P must be an integer power of two and the **FFT length**, M, gets set equal to P. If you do not select the check box, P can be any length, and the value of the **FFT length** parameter, M, must be a positive integer power of two. For user-specified FFT lengths, when M is not equal to P, zero padding or modulo-M data wrapping happens before the IFFT operation, as per Orfanidis [1].

```
y = ifft(u,M)  % P   M

y(:,l) = ifft(datawrap(u(:,l),M))  % P > M; l = 1,...,N
```

When the input length, P, is greater than the FFT length, M, you may see magnitude increases in your IFFT output. These magnitude increases occur because the IFFT block uses modulo-M data wrapping to preserve all available input samples.

To avoid such magnitude increases, you can truncate the length of your input sample, P, to the FFT length, M. To do so, place a Pad block before the IFFT block in your model.

The kth entry of the lth output channel, $y(k, l)$, is equal to the kth point of the M-point inverse discrete Fourier transform (IDFT) of the lth input channel:

$$y(k,l) = \frac{1}{M}\sum_{p=1}^{P} u(p,l)e^{j2\pi(p-1)(k-1)/M} \qquad k = 1,\ldots,M$$

This block supports real and complex floating-point and fixed-point inputs.

Input and Output Characteristics

The following table describes the input and output characteristics of the IFFT block. The table's columns provide the following information:

- Valid inputs to the IFFT block. They can be real- or complex-valued, and they can be in linear or bit-reversed order.
- The dimension along which the block computes the IDFT.
- The corresponding block output characteristics. The output port rate must equal the input port rate.

<table>
<tr><th>Valid Block Inputs</th><th>Dimension Along Which Block Computes IDFT</th><th>Corresponding Block Output Characteristics</th></tr>
<tr><td>N-D array</td><td>First dimension</td><td rowspan="3">The following output characteristics apply to all valid block inputs:

• Frame based
• Complex valued. If you input conjugate symmetric data and select the Input is conjugate symmetric check box, the block outputs a real-valued result.
• Same dimension as input (for 1-D inputs, output is a length-M column).
• Each output column (each row for sample-based row inputs) contains the M-point IDFT of the corresponding input channel in linear order. If you do not select the Divide output by FFT length check box, the block computes a modified version of</td></tr>
<tr><td>Sample-based 1-by-P row vector</td><td>Row</td></tr>
<tr><td>1-D length-P vector</td><td>Vector</td></tr>
</table>

Valid Block Inputs	Dimension Along Which Block Computes IDFT	Corresponding Block Output Characteristics
		the IDFT that does not include the multiplication factor of $1/M$.

Selecting the Twiddle Factor Computation Method

The **Twiddle factor computation** parameter determines how the block computes the necessary sine and cosine terms to calculate the term $e^{j2\pi(p-1)(k-1)/M}$, shown in the first equation of this block reference page. This parameter has two settings, each with its advantages and disadvantages, as described in the following table. For fixed-point signals, only `Table lookup` mode is supported.

Twiddle Factor Computation Parameter Setting	Sine and Cosine Computation Method	Effect on Block Performance
`Table lookup`	The block computes and stores the trigonometric values before the simulation starts and retrieves them during the simulation. When you generate code from the block, the processor running the generated code stores the trigonometric values computed by the block and retrieves the values during code execution.	The block usually runs much more quickly, but requires extra memory for storing the precomputed trigonometric values. You can optimize the table for memory consumption or speed, as described in "Optimizing the Table of Trigonometric Values" on page 2-675.
`Trigonometric fcn`	The block computes sine and cosine values during the simulation. When you generate code from the	The block usually runs more slowly, but does not need extra data memory. For code generation,

Twiddle Factor Computation Parameter Setting	Sine and Cosine Computation Method	Effect on Block Performance
	block, the processor running the generated code computes the sine and cosine values while the code runs.	the block requires a support library to emulate the trigonometric functions, increasing the size of the generated code.

Optimizing the Table of Trigonometric Values

When you set the **Twiddle factor computation** parameter to `Table lookup`, you also need to set the **Optimize table for** parameter. This parameter optimizes the table of trigonometric values for speed or memory by varying the number of table entries as summarized in the following table.

Optimize Table for Parameter Setting	Number of Table Entries for N-Point IFFT	Memory Required for Single-Precision 512-Point IFFT
`Speed`	$3N/4$ — floating point N — fixed point	$\left(\frac{3\times 512}{4}\text{table entries}\right)\times\left(4\frac{\text{bytes}}{\text{table entry}}\right)=1536\text{ bytes}$
`Memory`	$N/4$ — floating point Not supported for fixed point	$\left(\frac{512}{4}\text{ table entries}\right)\times\left(4\frac{\text{bytes}}{\text{table entry}}\right)=512\text{ bytes}$

Input Order

Select or clear the **Input is in bit-reversed order** check box to designate the ordering of the column elements of the block input. If you select the **Input is in bit-reversed order** check box, the block assumes the input is in bit-reversed order. If you clear the **Input is in bit-reversed order** check box, the block assumes the input is in linear order. For more information on ordering of the output, see "Linear and Bit-Reversed Output Order".

Conjugate Symmetric Input

The FFT block yields conjugate symmetric output when you input real-valued data. Taking the IFFT of a conjugate symmetric input matrix produces real-valued output. Therefore, if you input conjugate symmetric data and select the **Input is conjugate symmetric** check box, the block produces real-valued outputs. Selecting this check box optimizes the block's computation method.

If you input conjugate symmetric data to the IFFT block and do not select the **Input is conjugate symmetric** check box, the IFFT block outputs a complex-valued signal with small imaginary parts. If you select this check box and the input is not conjugate symmetric, the block output is invalid.

Scaled Output

When you select the **Divide output by FFT length** check box, the block computes its output according to the IDFT equation, discussed in the Description section.

If you clear the **Divide output by FFT length** check box, the block computes a modified version of the IDFT, $M \cdot y(k,l)$, which is defined by the following equation:

$$M \cdot y(k,l) = \sum_{p=1}^{P} u(p,l) e^{j2\pi(p-1)(k-1)/M} \quad k = 1,\dots,M$$

Algorithms Used for IFFT Computation

Depending on whether the block input is in bit-reversed order, conjugate symmetric order, or both, the block uses one or more of the following algorithms as summarized in the subsequent table:

- Bit-reversal operation
- Double-signal algorithm
- Half-length algorithm
- Radix-2 decimation-in-time (DIT) algorithm

Input Complexity	Other Parameter Settings	Algorithms Used for IFFT Computation
Real or complex	☐ Input is in bit-reversed order ☐ Input is conjugate symmetric	Bit-reversal operation and radix-2 DIT
Real or complex	☑ Input is in bit-reversed order ☐ Input is conjugate symmetric	Radix-2 DIT
Real or complex	☐ Input is in bit-reversed order ☑ Input is conjugate symmetric	Bit-reversal operation and radix-2 DIT in conjunction with the half-length and double-signal algorithms
Real or complex	☑ Input is in bit-reversed order ☑ Input is conjugate symmetric	Radix-2 DIT in conjunction with the half-length and double-signal algorithms

When the **Input is conjugate symmetric** check box is selected, the efficiency of the IFFT algorithm can be enhanced in two ways:

- By forming one length-M complex-valued sequence from two length-M conjugate symmetric input sequences
- By forming one length-M complex-valued sequence from one length-$2M$ conjugate symmetric input sequence

These optimizations correspond to the optimizations used in the FFT computation of real input signals.

When there are $2N+1$ conjugate symmetric input channels and the **Input is conjugate symmetric** check box is selected, the IFFT block applies the double-signal algorithm to the first $2N$ input channels and the half-length algorithm to the last odd-numbered channel. If the input signals have fixed-point data types, it is possible to see different numerical results in the output of the last odd channel, even if all input channels are identical. This numerical difference results from differences in the double-signal algorithm and the half-length algorithm.

You can eliminate this numerical difference in two ways:

- Using full precision arithmetic for fixed-point input signals
- Changing the input data type to floating point

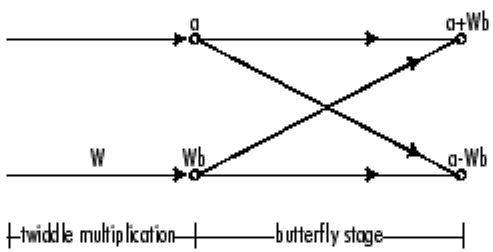
For more information on the double-signal and half-length algorithms, see Proakis [2]. “Efficient Computation of the DFT of Two Real Sequences” on page 475 describes the double signal algorithm. “Efficient Computation of the DFT of a 2N-Point Real Sequence” on page 476 describes the half-length algorithm.

Fixed-Point Data Types

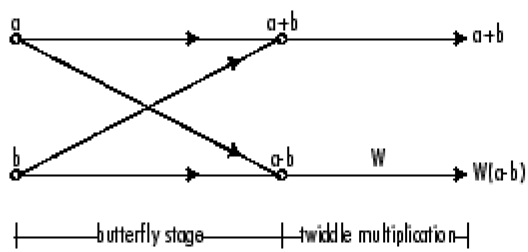
The following diagrams show the data types used within the IFFT block for fixed-point signals. You can set the sine table, accumulator, product output, and output data types displayed in the diagrams in the IFFT block dialog, as discussed in “Dialog Box” on page 2-681.

The IFFT block first casts input to the output data type and then stores it in the output buffer. Each butterfly stage then processes signals in the accumulator data type, with the final output of the butterfly being cast back into the output data type. The block multiplies in a twiddle factor before each butterfly stage in a decimation-in-time IFFT, and after each butterfly stage in a decimation-in-frequency IFFT.

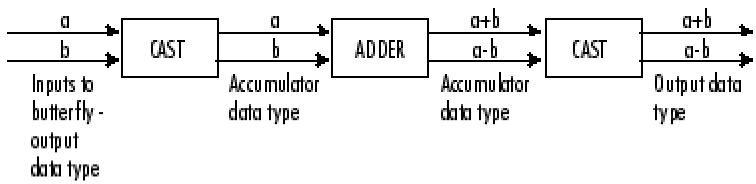
Decimation-in-time IFFT



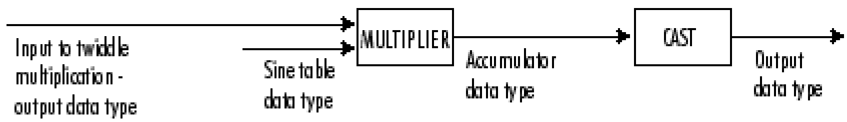
Decimation-in-frequency IFFT



Butterfly stage data types



Twiddle multiplication data types



The output of the multiplier is in the accumulator data type because both of the inputs to the multiplier are complex. For details on the complex multiplication performed, see “Multiplication Data Types”.

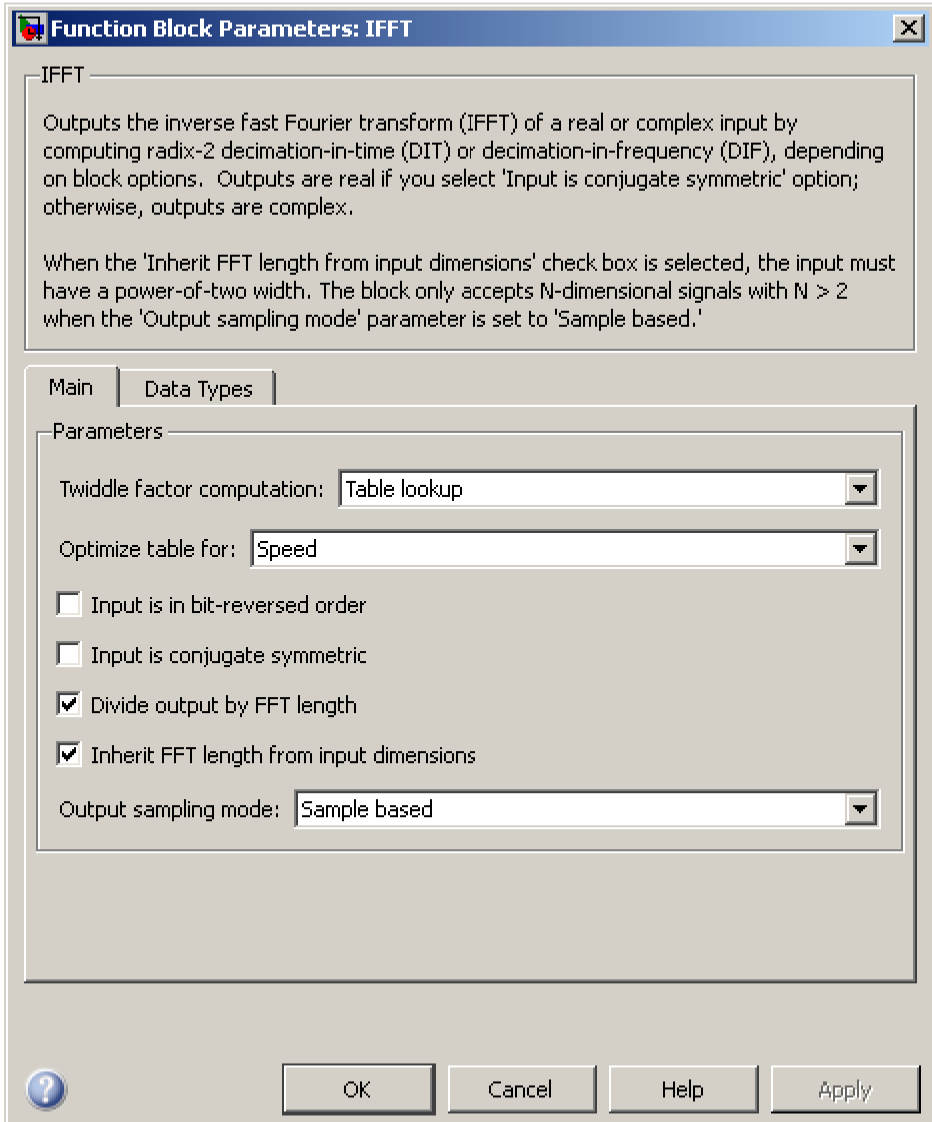
Note When the block input is fixed point, all internal data types are signed fixed point.

Examples

See “Transforming Frequency-Domain Data into the Time Domain” in the *Signal Processing Blockset User’s Guide*.

Dialog Box

The **Main** pane of the IFFT block dialog appears as follows.



Twiddle factor computation

Specify the computation method of the term $e^{j2\pi(p-1)(l-1)/M}$ shown in the first equation of this block reference page.

In `Table lookup` mode, the block computes and stores the sine and cosine values before the simulation starts.

In `Trigonometric fcn` mode, the block computes the sine and cosine values during the simulation. See "Selecting the Twiddle Factor Computation Method" on page 2-674.

This parameter must be set to `Table lookup` for fixed-point signals.

Optimize table for

Select this option to optimize the table of sine and cosine values for either `Speed` or `Memory`. This parameter becomes available only when you set the **Twiddle factor computation** parameter to `Table lookup`. See "Optimizing the Table of Trigonometric Values" on page 2-675 for more information.

This parameter must be set to `Speed` for fixed-point signals.

Input is in bit-reversed order

Designate the order of the input channel elements. Select this check box when the input is in bit-reversed order, and clear this check box when the input is in linear order. The block yields invalid outputs when you do not set this parameter correctly. See "Input Order" on page 2-676 for more information.

You cannot select this parameter if you have cleared the **Inherit FFT length from input dimensions** parameter, and you are specifying the FFT length using the **FFT length** parameter.

Input is conjugate symmetric

Select this option when the input to the block is conjugate symmetric and you want real-valued outputs. If you select this option and the input is not conjugate symmetric, the block output is invalid.

You cannot select this parameter if you have cleared the **Inherit FFT length from input dimensions** parameter, and you are specifying the FFT length using the **FFT length** parameter.

Divide output by FFT length

When you select this check box, the block computes the output according to the IDFT equation discussed in the Description section.

When you clear this check box, the block computes the output using a modified version of the IDFT. The modified IDFT equation does not include the multiplication factor of $1/M$. For the full equation, see "Scaled Output" on page 2-676.

Inherit FFT length from input dimensions

Select to inherit the FFT length from the input dimensions. When you select this parameter, the input length P must be a power of two. If you do not select this parameter, the **FFT length** parameter becomes available.

You cannot clear this parameter when you select either the **Input is in bit-reversed order** or the **Input is conjugate symmetric** parameter.

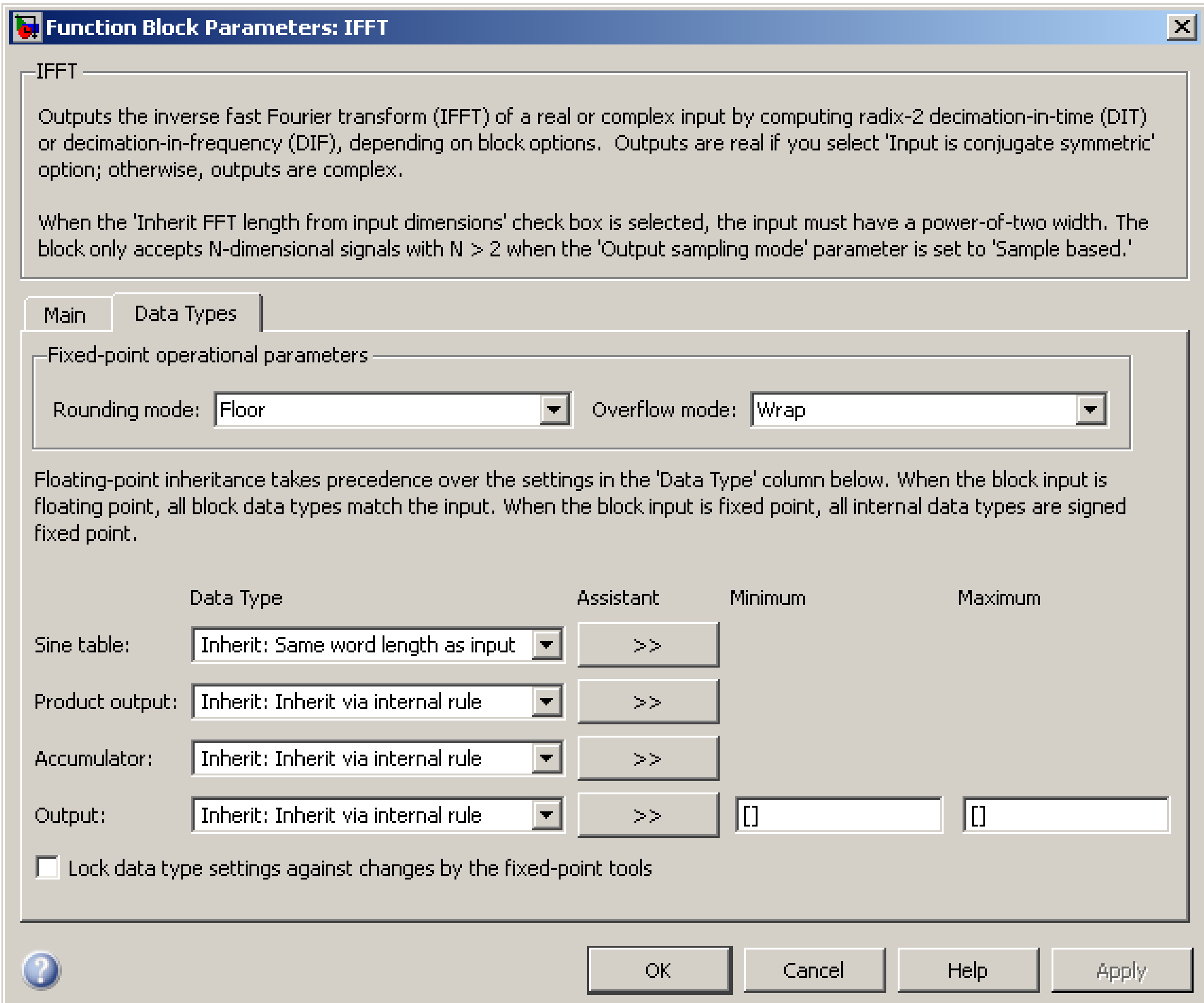
FFT length

Specify a power-of-two FFT length. This parameter only becomes available if you do not select the **Inherit FFT length from input dimensions** parameter.

Output sampling mode

Select `Sample based` or `Frame based` output. If the input to the IFFT block has 3 or more dimensions, you must select `Sample based` output.

The **Data Types** pane of the IFFT block dialog appears as follows.


Function Block Parameters: IFFT
IFFT
Outputs the inverse fast Fourier transform (IFFT) of a real or complex input by computing radix-2 decimation-in-time (DIT) or decimation-in-frequency (DIF), depending on block options. Outputs are real if you select 'Input is conjugate symmetric' option; otherwise, outputs are complex.
When the 'Inherit FFT length from input dimensions' check box is selected, the input must have a power-of-two width. The block only accepts N-dimensional signals with N > 2 when the 'Output sampling mode' parameter is set to 'Sample based.'
Main
Data Types
Fixed-point operational parameters
Rounding mode:
Floor
Overflow mode:
Wrap
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input. When the block input is fixed point, all internal data types are signed fixed point.
Data Type
Assistant
Minimum
Maximum
Sine table:
Inherit: Same word length as input
>>
Product output:
Inherit: Inherit via internal rule
>>
Accumulator:
Inherit: Inherit via internal rule
>>
Output:
Inherit: Inherit via internal rule
>>
[]
[]
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Rounding mode

Select the rounding mode for fixed-point operations. The sine table values do not obey this parameter; instead, they always round to `Nearest`.

Overflow mode

Select the overflow mode for fixed-point operations. The sine table values do not obey this parameter; instead, they are always saturated.

Sine table data type

Choose how you specify the word length of the values of the sine table. The fraction length of the sine table values always equals the word length minus one. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same word length as input`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16)`

The sine table values do not obey the **Rounding mode** and **Overflow mode** parameters; instead, they are always saturated and rounded to `Nearest`.

Product output data type

Specify the product output data type. See "Fixed-Point Data Types" on page 2-678 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Accumulator data type

Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-678 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Output data type

Specify the output data type. See “Fixed-Point Data Types” on page 2-678 for illustrations depicting the use of the output data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`.

  When you select `Inherit:  Inherit via internal rule`, the block calculates the output word length and fraction length automatically. The internal rule first calculates an ideal output word length and fraction length using the following equations:

  $$WL_{ideal\ output} = WL_{input} + floor(\log_2(FFT\ length - 1)) + 1$$

  $$FL_{ideal\ output} = FL_{input}$$

Using these ideal results, the internal rule then selects word lengths and fraction lengths that are appropriate for your hardware. For more information, see “Inherit via Internal Rule”.

- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User’s Guide* for more information.

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

References

[1] Orfanidis, S. J. *Introduction to Signal Processing*. Upper Saddle River, NJ: Prentice Hall, 1996, p. 497.

[2] Proakis, John G. and Dimitris G. Manolakis. *Digital Signal Processing*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 1996.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

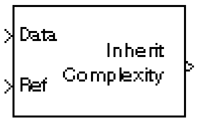
FFT	Signal Processing Blockset
IDCT	Signal Processing Blockset
Pad	Signal Processing Blockset
`bitrevorder`	Signal Processing Toolbox
`fft`	MATLAB
`ifft`	MATLAB

Purpose Change complexity of input to match reference signal

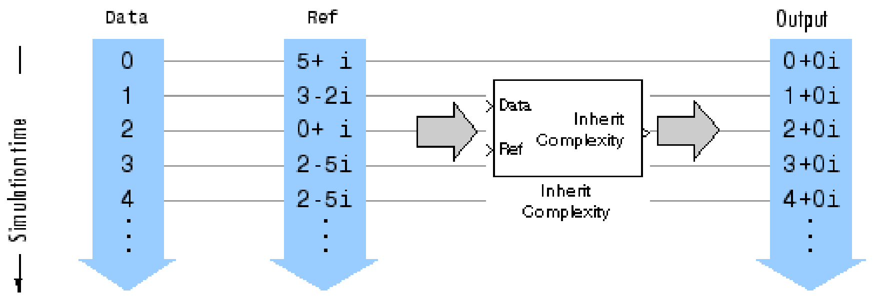
Library Signal Management / Signal Attributes

`dspsigattribs`

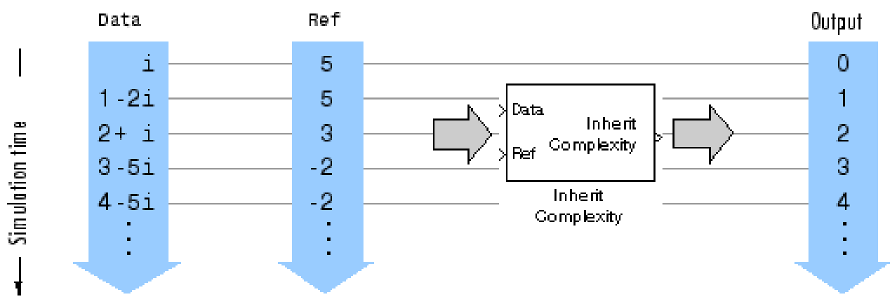
Description



The Inherit Complexity block alters the input data at the Data port to match the complexity of the reference input at the Ref port. When the Data input is real, and the Ref input is complex, the block appends a zero-valued imaginary component, `0i`, to each element of the Data input.

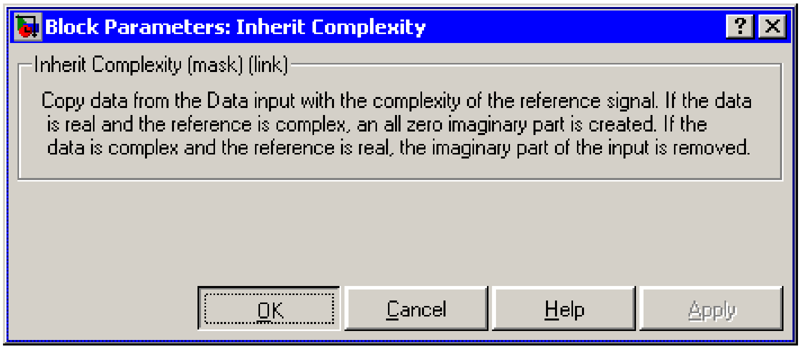


When the `Data` input is complex, and the `Ref` input is real, the block outputs the real component of the `Data` input.



When both the `Data` input and `Ref` input are real, or when both the `Data` input and `Ref` input are complex, the block propagates the `Data` input with no change.

Dialog Box



Supported Data Types

Port	Supported Data Types
Data	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Ref	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Check Signal Attributes	Signal Processing Blockset
Complex to Magnitude-Angle	Simulink
Complex to Real-Imag	Simulink
Magnitude-Angle to Complex	Simulink
Real-Imag to Complex	Simulink

Integer Delay (Obsolete)

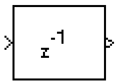
Purpose

Delay input by integer number of sample periods

Library

`dspobslib`

Description



Note The Integer Delay block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the Delay block.

The Integer Delay block delays a discrete-time input by the number of sample intervals specified in the **Delay** parameter. Noninteger delay values are rounded to the nearest integer, and negative delays are clipped at 0.

Sample-Based Operation

When the input is a sample-based M-by-N matrix, the block treats each of the $M*N$ matrix elements as an independent channel. The **Delay** parameter, v, can be an M-by-N matrix of positive integers that specifies the number of sample intervals to delay each channel of the input, or a scalar integer by which to equally delay all channels.

For example, when the input is M-by-1 and v is the matrix `[v(1) v(2) ... v(M)]'`, the first channel is delayed by `v(1)` sample intervals, the second channel is delayed by `v(2)` sample intervals, and so on. Note that when a channel is delayed for Δ sample-time units, the output sample at time t is the input sample at time $t - \Delta$. When $t - \Delta$ is negative, then the output is the corresponding value specified by the **Initial conditions** parameter.

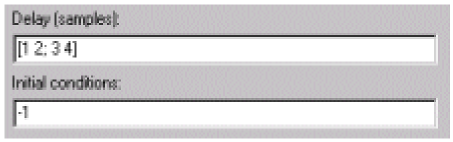
A 1-D vector of length M is treated as an M-by-1 matrix, and the output is 1-D.

The **Initial conditions** parameter specifies the output of the block during the initial delay in each channel. The initial delay for a particular channel is the time elapsed from the start of the simulation until the first input in that channel is propagated to the output. Both fixed and time-varying initial conditions can be specified in a variety of ways to suit the dimensions of the input.

Fixed Initial Conditions

A fixed initial condition in sample-based mode can be specified as one of the following:

- Scalar value to be repeated at each sample time of the initial delay (for every channel). For a 2-by-2 input with the parameter settings below,

  the block generates the following sequence of matrices at the start of the simulation,

  $$\begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix}, \begin{bmatrix} u_{11}^1 & -1 \\ -1 & -1 \end{bmatrix}, \begin{bmatrix} u_{11}^2 & u_{12}^1 \\ -1 & -1 \end{bmatrix}, \begin{bmatrix} u_{11}^3 & u_{12}^2 \\ u_{21}^1 & -1 \end{bmatrix}, \begin{bmatrix} u_{11}^4 & u_{12}^3 \\ u_{21}^2 & u_{22}^1 \end{bmatrix}, \ldots$$

  where u_{ij}^k is the i,jth element of the kth matrix in the input sequence.

- Array of size M-by-N-by-d. In this case, you can set different fixed initial conditions for each element of a sample-based input. This setting is explained further in the Array bullet in "Time-Varying Initial Conditions" on page 2-693.

Initial conditions cannot be specified by full matrices.

Time-Varying Initial Conditions

A time-varying initial condition in sample-based mode can be specified in one of the following ways:

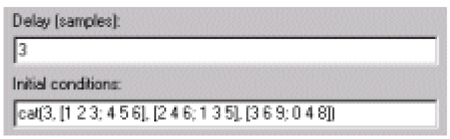
- Vector of length d, where d is the maximum value specified for any channel in the **Delay** parameter. The vector can be a L-by-d, 1-by-d, or 1-by-1-by-d. The d elements of the vector are output in sequence, one at each sample time of the initial delay.

  For a scalar input and the parameters shown below,

  the block outputs the sequence `-1, -1, -1, 0, 1,...` at the start of the simulation.

- Array of dimension M-by-N-by-d, where d is the value specified for the **Delay** parameter (the maximum value when the **Delay** is a vector) and M and N are the number of rows and columns, respectively, in the input matrix. The d pages of the array are output in sequence, one at each sample time of the initial delay. For a 2-by-3 input, and the parameters below,

  the block outputs the matrix sequence

  $$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 5 \end{bmatrix}, \begin{bmatrix} 3 & 6 & 9 \\ 0 & 4 & 8 \end{bmatrix}$$

  at the start of the simulation. Note that setting **Initial conditions** to an array with the same matrix for each entry implements constant initial conditions; a different constant initial condition for each input matrix element (channel).

Initial conditions cannot be specified by full matrices.

Frame-Based Operation

When the input is a frame-based M-by-N matrix, the block treats each of the N columns as an independent channel, and delays each channel as specified by the **Delay** parameter.

For frame-based inputs, the **Delay** parameter can be a scalar integer by which to equally delay all channels. It can also be a 1-by-N row vector, each element of which serves as the delay for the corresponding channel of the N-channel input. Likewise, it can also be an M-by-1 column vector, each element of which serves as the delay for one of the corresponding M samples for each channel. The **Delay** parameter can be an M-by-N matrix of positive integers as well; in this case, each element of each channel is delayed by the corresponding element in the delay matrix. For instance, if the fifth element of the third column of the delay matrix was 3, then the fifth element of the third channel of the input matrix is always delayed by three sample-time units.

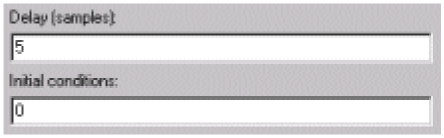
When a channel is delayed for Δ sample-time units, the output sample at time t is the input sample at time $t - \Delta$. When $t - \Delta$ is negative, then the output is the corresponding value specified in the **Initial conditions** parameter.

The **Initial conditions** parameter specifies the output during the initial delay. Both fixed and time-varying initial conditions can be specified. The initial delay for a particular channel is the time elapsed from the start of the simulation until the first input in that channel is propagated to the output.

Fixed Initial Conditions

The settings shown below specify fixed initial conditions. The value entered in the **Initial conditions** parameter is repeated at the output for each sample time of the initial delay. A fixed initial condition in frame-based mode can be one of the following:

- Scalar value to be repeated for all channels of the output at each sample time of the initial delay. For a general M-by-N input with the parameter settings below,



the first five samples in each of the N channels are zero. Notice that when the frame size is larger than the delay, all of these zeros are all included in the first output from the block.

- Array of size 1-by-N-by-D. In this case, you can also specify different fixed initial conditions for each channel. See “Time-Varying Initial Conditions” on page 2-696 for details.

Initial conditions cannot be specified by full matrices.

Time-Varying Initial Conditions

The following settings specify time-varying initial conditions. For time-varying initial conditions, the values specified in the **Initial conditions** parameter are output in sequence during the initial delay. A time-varying initial condition in frame-based mode can be specified in the following ways:

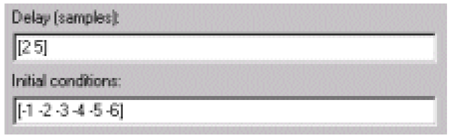
- Vector of length D, where each of the N channels have the same initial conditions sequence specified in the vector. D is defined as follows:
  - When an element of the delay entry is less than the frame size,

    $D = d + 1$

    where d is the maximum delay.
  - When the all elements of the delay entry are greater than the input frame size,

    $D = d +$ `input frame size` $- 1$

  Only the first d entries of the initial condition vector are used; the rest of the values are ignored, but you must include them nonetheless. For a two-channel ramp input `[1:100; 1:100]'` with a frame size of 4 and the parameter settings below,

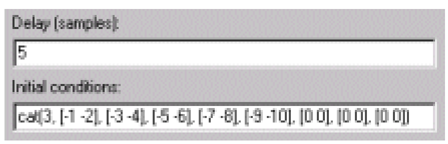


the block outputs the following sequence of frames at the start of the simulation.

$$\begin{bmatrix} -4 & -1 \\ -5 & -2 \\ 1 & -3 \\ 2 & -4 \end{bmatrix}, \begin{bmatrix} 3 & -5 \\ 4 & 1 \\ 5 & 2 \\ 6 & 3 \end{bmatrix}, \begin{bmatrix} 7 & 4 \\ 8 & 5 \\ 9 & 6 \\ 10 & 7 \end{bmatrix}, \ldots$$

Note that since one of the delays, 2, is less than the frame size of the input, 4, the length of the **Initial conditions** vector is the sum of the maximum delay and 1 (5+1), which is 6. The first five entries of the initial conditions vector are used by the channel with the maximum delay, and the rest of the entries are ignored. Since the first channel is delayed for less than the maximum delay (2 sample time units), it only makes use of two of the initial condition entries.

- Array of size 1-by-N-by-D, where D is defined in "Time-Varying Initial Conditions" on page 2-696. In this case, the kth entry of each 1-by-N entry in the array corresponds to an initial condition for the kth channel of the input matrix. Thus, a 1-by-N-by-D initial conditions input allows you to specify different initial conditions for each channel. For instance, for a two-channel ramp input `[1:100; 1:100]'` with a frame size of 4 and the parameter settings below,



the block outputs the following sequence of frames at the start of the simulation.

$$\begin{bmatrix} -1 & -2 \\ -3 & -4 \\ -5 & -6 \\ -7 & -8 \end{bmatrix}, \begin{bmatrix} -9 & -10 \\ 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 \\ 5 & 5 \\ 6 & 6 \\ 7 & 7 \end{bmatrix}, \ldots$$

Note that the channels have distinct time varying initial conditions; the initial conditions for channel 1 correspond to the first entry of each length-2 row vector in the initial conditions array, and the initial conditions for channel 2 correspond to the second entry of each row vector in the initial conditions array. Only the first five entries in the initial conditions array are used; the rest are ignored.

The 1-by-N-by-D array entry can also specify different fixed initial conditions for every channel; in this case, every 1-by-N entry in the array would be identical, so that the initial conditions for each column are fixed over time.

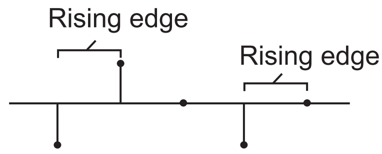
Initial conditions cannot be specified by full matrices.

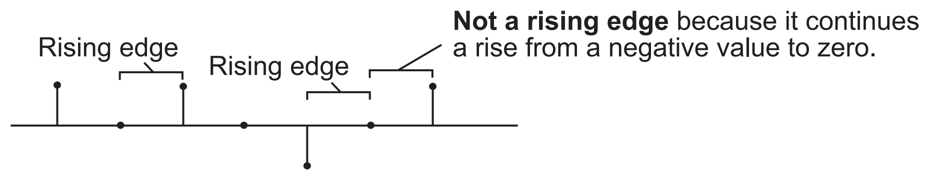
Resetting the Delay

The block resets the delay whenever it detects a reset event at the optional Rst port. The reset sample time must be a positive integer multiple of the input sample time.

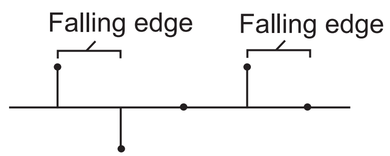
You specify the reset event in the **Reset port** parameter:

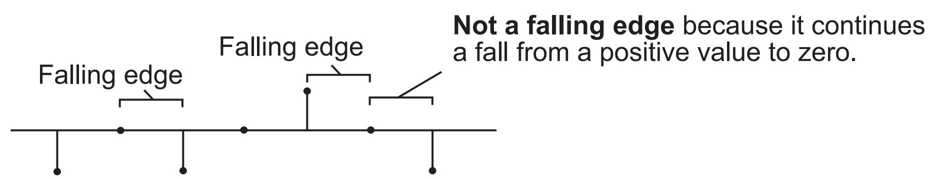
- `None` disables the Rst port.
- `Rising edge` — Triggers a reset operation when the Rst input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)





- `Falling edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)





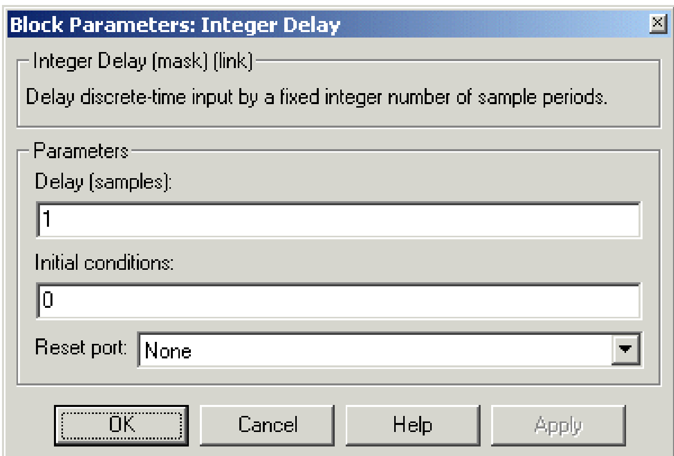
- `Either edge` — Triggers a reset operation when the `Rst` input is a `Rising edge` or `Falling edge` (as described above).
- `Non-zero sample` — Triggers a reset operation at each sample time that the `Rst` input is not zero.

> **Note** When running simulations in the Simulink `MultiTasking` mode, sample-based reset signals have a one-sample latency, and frame-based reset signals have one frame of latency. Thus, there is a one-sample or one-frame delay between the time the block detects a reset event, and when it applies the reset. For more information on latency and the Simulink tasking modes, see “Excess Algorithmic Delay (Tasking Latency)” and “Scheduling Considerations” in the *Real-Time Workshop User’s Guide*.

Examples

The `dspafxr` demo illustrates an audio reverberation system built around the Integer Delay block.

Dialog Box



Delay
The number of sample periods to delay the input signal.

Initial conditions
The value of the block’s output during the initial delay.

Reset port

Determines the reset event that causes the block to reset the delay. For more information, see "Resetting the Delay" on page 2-698.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed only)
- Boolean — The block accepts Boolean inputs to the Rst port, which is enabled by the **Reset port** parameter.
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers

See Also

Unit Delay	Simulink
Variable Fractional Delay	Signal Processing Blockset
Variable Integer Delay	Signal Processing Blockset

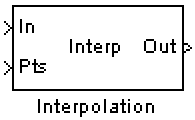
Interpolation

Purpose Interpolate values of real input samples

Library Signal Operations

`dspsigops`

Description



The Interpolation block interpolates discrete, real, inputs using linear or FIR interpolation. The block accepts both sample- and frame-based input data in the form of a vector, matrix, or sample-based N-D array. The block outputs a scalar, vector, matrix, or N-D array of the interpolated values, which has the same frame status as the input data.

You must specify the interpolation points (times at which to interpolate values) in a one-based interpolation array, I_{Pts}. An entry of 1 in I_{Pts} refers to the first sample of the input data, an entry of 2.5 refers to the sample half-way between the second and third input sample, and so on. Depending on the dimensions of the input data, I_{Pts} can be a scalar, a length-P row or column vector, a P-by-N frame-based matrix, or a sample-based N-D array where P is the size of the first dimension of the N-D array. In most cases, P can be any positive integer. For more information about valid interpolation arrays, refer to the tables in "How the Block Applies Interpolation Arrays to Inputs" on page 2-703.

In most cases, the block applies I_{Pts} across the first dimension of an N-D input array, or to each input vector. You can set the block to apply the same interpolation array for all input data (static interpolation points entered on the block mask) or to use a different interpolation array for each N-D array, matrix, or vector input (time-varying interpolation points received via the Pts input port).

Sections of This Reference Page

Specifying Static Interpolation Points

To supply the block with a static interpolation array (an interpolation array applied to every vector or N-D array of input data), perform the following steps:

- Set the **Source of interpolation points** parameter to `Specify via dialog`.
- Enter the interpolation array in the **Interpolation points** parameter. To learn about interpolation arrays, see “How the Block Applies Interpolation Arrays to Inputs” on page 2-703.

Specifying Time-Varying Interpolation Points

To supply the block with time-varying interpolation arrays (where the block uses a different interpolation array for each vector or N-D array input), perform the following steps:

1. Set the **Source of interpolation points** parameter to `Input port`, the **Pts** port appears on the block.
2. Generate a signal of interpolation arrays, and supply it to the Pts port. The block uses the input to this port as the interpolation points. To learn about interpolation arrays, see “How the Block Applies Interpolation Arrays to Inputs” on page 2-703.

How the Block Applies Interpolation Arrays to Inputs

The interpolation array I_{Pts} represents the points in time at which to interpolate values of the input signal. An entry of 1 in I_{Pts} refers to the first sample of the input, an entry of 2.5 refers to the sample half-way between the second and third input sample, and so on. In most cases, when I_{Pts} is a vector, it can be of any length.

Valid values in the interpolation array, I_{Pts}, range from 1 to the number of samples in each channel of the input. To learn how the block handles out of range interpolation values, see "Handling Out-of-Range Interpolation Points" on page 2-709.

Depending on the dimension and frame status of the input and the dimension of I_{Pts}, the block usually applies I_{Pts} to the input in one of the following ways:

- Applies the I_{Pts} array across the first dimension of a sample-based N-D array or frame-based matrix input, resulting in a sample-based N-D array or frame-based matrix output.
- Applies the vector I_{Pts} to each input vector (as if the input vector were a single channel), resulting in a vector output with the same orientation as the input (row or column).

The following tables summarize how the block applies the interpolation array I_{Pts} to all the possible types of sample- and frame-based inputs, and show the resulting output dimensions.

The first table describes the block's behavior when the **Source of interpolation points** is `Specify via dialog` and the input is sample based.

Input Dimensions (Sample Based)	Valid Dimensions of Interpolation Array I_{Pts}	How Block Applies I_{Pts} to Input	Output Dimensions (Sample Based)
N-D Array (ex. M-by-N-by-Q)	1-by-P row	Applies I_{Pts} to the first dimension of the input array	P-by-N-by-Q array
	P-by-1 column		
	P-by-N-by-Q array	Applies the columns of I_{Pts} to the corresponding	

Input Dimensions (Sample Based)	Valid Dimensions of Interpolation Array I_{Pts}	How Block Applies I_{Pts} to Input	Output Dimensions (Sample Based)
		columns of the input array	
M-by-1 column	1-by-P row (block treats I_n as a column)	Applies I_{Pts} to the input column	P-by-1 column
	P-by-1 column		
1-by-N row	1-by-P row	Applies I_n to the input row	1-by-P row
	P-by-1 column (block treats I_{Pts} as a row)		

The next table describes the block's behavior when the **Source of interpolation points** is `Specify via dialog` and the input is frame based.

Input Dimensions (Frame Based)	Valid Dimensions of Interpolation Array I_{Pts}	How Block Applies I_{Pts} to Input	Output Dimensions (Frame Based)
M-by-N matrix	1-by-N row	Applies each column of I_{Pts} (each element of I_{Pts}) to the corresponding column of the input matrix	1-by-N row

Input Dimensions (Frame Based)	Valid Dimensions of Interpolation Array I_{Pts}	How Block Applies I_{Pts} to Input	Output Dimensions (Frame Based)
	P-by-1 column	Applies I_{Pts} to each input column	P-by-N matrix
	P-by-N matrix	Applies the columns of I_{Pts} to the corresponding columns of the input matrix	
M-by-1 column	1-by-P row (block treats I_{Pts} as a column)	Applies I_{Pts} to the input column	P-by-1 column
	P-by-1 column		
1-by-N row (not recommended)	1-by-N row	Not Applicable. Block copies input vector	1-by-N row, a copy of the input vector
	P-by-1 column		P-by-N matrix where each row is a copy of the input vector
	P-by-N matrix		

The next table describes the block's behavior when the **Source of interpolation points** is `Input port` and the input is sample based.

Input Dimensions (Sample Based)	Valid Dimensions of Interpolation Array I_{Pts}	How Block Applies I_{Pts} to Input	Output Dimensions (Sample Based)
N-D Array (ex. M-by-N-by-Q)	Sample-based 1-by-P row	Applies I_{Pts} to the first dimension of the input array	P-by-N-by-Q array
	Sample- or frame-based P-by-1 column		
	Sample-based P-by-N-by-Q array	Applies the columns of I_{Pts} to the corresponding columns of the input array	
M-by-1 column	Sample-based 1-by-P row	Applies I_{Pts} to the input column	P-by-1 column
	Sample- or frame-based P-by-1 column		
1-by-N row	Sample-based 1-by-P row	Applies I_{Pts} to the input row	1-by-P row
	Sample or frame-based P-by-1 column		

The next table describes the block's behavior when the **Source of interpolation points** is `Input port` and the input is frame based.

<table>
<tr><th>Input Dimensions (Frame Based)</th><th>Valid Dimensions of Interpolation Array I_{Pts}</th><th>How Block Applies I_{Pts} to Input</th><th>Output Dimensions (Frame Based)</th></tr>
<tr><td rowspan="4">M-by-N matrix</td><td>Frame-based 1-by-N row</td><td>Applies each column of I_{Pts} (each element of I_{Pts}) to the corresponding column of the input matrix</td><td>1-by-N row</td></tr>
<tr><td>Sample-based 1-by-P row</td><td rowspan="2">Applies I_{Pts} to each input column</td><td rowspan="3">P-by-N matrix</td></tr>
<tr><td>Frame- or sample-based P-by-1 column</td></tr>
<tr><td>Frame- or sample-based P-by-N matrix</td><td>Applies the columns of I_{Pts} to the corresponding columns of the input matrix</td></tr>
<tr><td rowspan="2">M-by-1 column</td><td>Sample-based 1-by-P row</td><td rowspan="2">Applies I_{Pts} to the input column</td><td rowspan="2">P-by-1 column</td></tr>
<tr><td>Frame- or sample-based P-by-1 column</td></tr>
</table>

Input Dimensions (Frame Based)	Valid Dimensions of Interpolation Array I_{Pts}	How Block Applies I_{Pts} to Input	Output Dimensions (Frame Based)
1-by-N row (not recommended)	Frame-based 1-by-N row	Not Applicable. Block copies input vector	1-by-N row, a copy of the input vector
	Sample-based 1-by-P row		P-by-N matrix where each row is a copy of the input vector
	Frame- or sample-based P-by-1 column		
	Frame- or sample-based P-by-N matrix		

Handling Out-of-Range Interpolation Points

Valid values in the interpolation array I_{Pts} range from 1 to the number of samples in each channel of the input. For instance, given a length-5 input vector `D`, all entries of I_{Pts} must range from 1 to 5. I_{Pts} cannot contain entries such as 7 or -9, since there is no 7th or -9th entry in `D`.

The **Out of range interpolation points** parameter sets how the block handles interpolation points that are fall outside the valid range, and has the following settings:

- `Clip` — The block replaces any out-of-range values in I_{Pts} with the closest value in the valid range (from 1 to the number of input samples), and then proceeds with computations using the clipped version of I_{Pts}.
- `Clip and warn` — In addition to `Clip`, the block issues a warning at the MATLAB command line every time clipping occurs.

- `Error` — When the block encounters an out-of-range value in I_{Pts}, the simulation stops, and the block issues an error at the MATLAB command line.

Example of Clipping

Suppose the block is set to clip out-of-range interpolation points, and gets the following input vector and interpolation points:

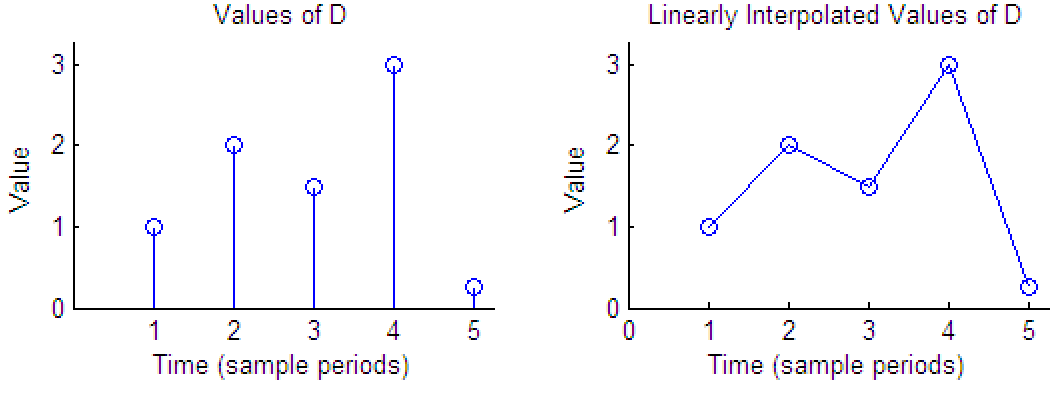
- `D = [11 22 33 44]'`
- $\mathtt{I_{Pts}}$ `= [10 2.6 -3]'`

Because D has four samples, valid interpolation points range from 1 to 4. The block clips the interpolation point 10 to 4 and the point -3 to 1, resulting in the clipped interpolation vector $\mathtt{I_{PtsClipped}}$ `= [4 2.6 1]'`.

Linear Interpolation Mode

When **Interpolation Mode** is set to `Linear`, the block interpolates data values by assuming that the data varies linearly between samples taken at adjacent sample times.

For instance, if the input signal D = [1 2 1.5 3 0.25]', the following plot on the left shows the samples in D, and the plot on the right shows the linearly interpolated values between the samples in D.

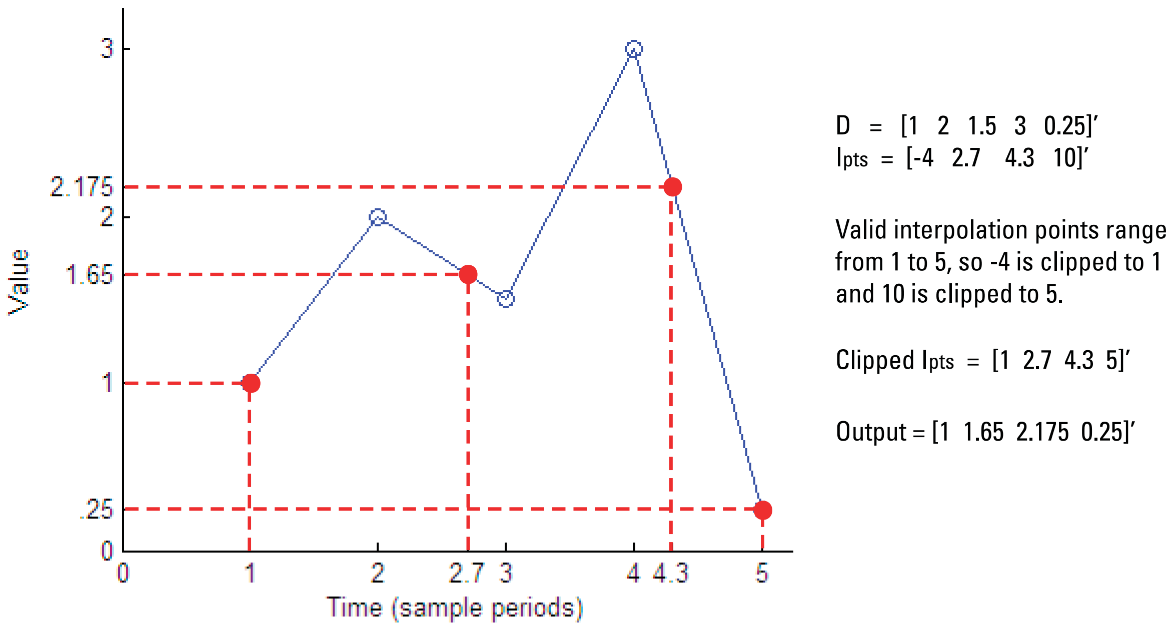


The following figure illustrates the case of a block in linear interpolation mode that is set to clip out-of-range interpolation points. The vector D supplies the input data and the vector I_{Pts} supplies the interpolation points:

- `D = [1 2 1.5 3 0.25]'`
- $\mathtt{I_{Pts}}$ `= [-4 2.7 4.3 10]'`

The block clips the invalid interpolation points, and outputs the linearly interpolated values in a vector, `[1 1.65 2.175 0.25]'`.

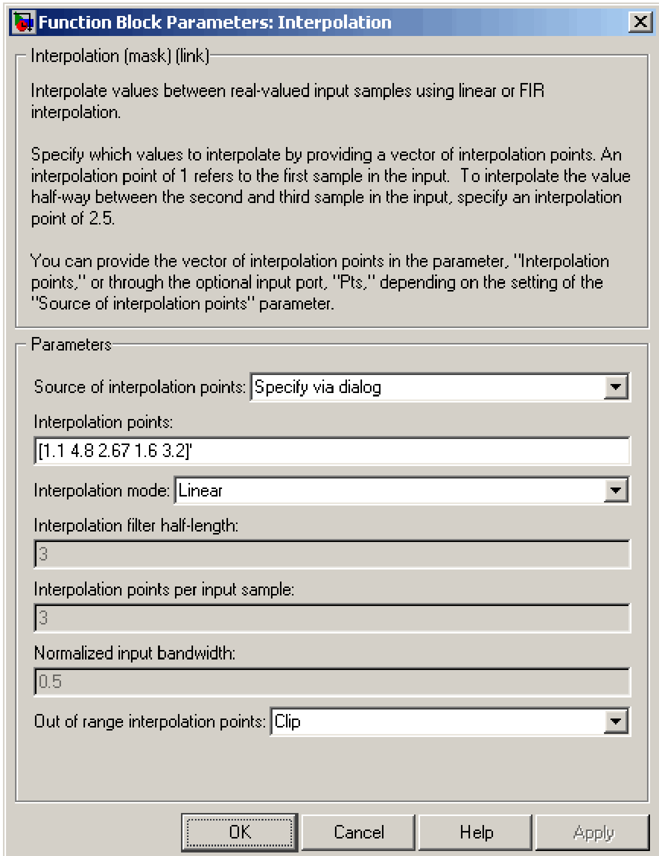
Interpolated Values of D at Clipped Interpolation Points



FIR Interpolation Mode

When **Interpolation Mode** is set to `FIR`, the block interpolates data values using an FIR interpolation filter, specified by various block parameters. See FIR Interpolation Mode on page 1412 in the Variable Fractional Delay block reference for more information.

Dialog Box



Source of interpolation points

Choose how you want to specify the interpolation points. If you select `Specify via dialog`, the **Interpolation points** parameter become available. Use this option for static interpolation points. If you select `Input port`, the Pts port appears on the block. The block uses the input to this port as the interpolation points. Use this option for time-varying interpolation points. For more information, see "Specifying Static Interpolation Points" on page 2-703 and "Specifying Time-Varying Interpolation Points" on page 2-703.

Interpolation points

The array of points in time at which to interpolate the input signal (I_{Pts}). An entry of 1 in I_{Pts} refers to the first sample of the input, an entry of 2.5 refers to the sample half-way between the second and third input sample, and so on. See "How the Block Applies Interpolation Arrays to Inputs" on page 2-703. Tunable.

Interpolation mode

Sets the block to interpolate by either linear or FIR interpolation. For more information, see "Linear Interpolation Mode" on page 2-710 and "FIR Interpolation Mode" on page 2-711.

Interpolation filter half-length

Specify the half-length of the FIR interpolation filter (`P`). To perform the interpolation in `FIR` mode, the block uses the nearest 2*`P` low-rate samples. In most cases, `P` low-rate samples must appear below and above each interpolation point. However, if you interpolate at a low-rate sample point, the block includes that low-rate sample in the required 2*`P` samples and requires only 2*`P`–1 neighboring low-rate samples. If an interpolation point does not have the required number of neighboring low-rate samples, the block interpolates that point using linear interpolation.

This parameter becomes available only when the **Interpolation mode** is set to `FIR`. For more information, see "FIR Interpolation Mode" on page 2-711.

Interpolation points per input sample

Also known as the *upsampling factor*, this parameter defines the number of points per input sample (`L`) at which the block computes a unique FIR interpolation filter. To perform the FIR Interpolation, the block uses a polyphase structure with `L` filter arms of length 2*`P`.

For example, if L=4, the block constructs a polyphase filter with four arms. The block then interpolates at points corresponding to 1 +*i*/L, 2 +*i*/L, 3 +*i*/L..., where the integers 1, 2, and 3 represent the low-rate samples, and *i*=0,1,2,3. To interpolate at a point that does not directly correspond to an arm of the polyphase filter requires an extra computation. The block first rounds that point down to the nearest value that does correspond to an arm of the polyphase filter. Thus, to interpolate at the point 2.2, the block rounds 2.2 down to 2, and computes the FIR interpolation using the first arm of the polyphase filter structure. Similarly, to interpolate the point 2.65, the block rounds the value down to 2.5 and uses the third arm of the polyphase filter structure.

This parameter becomes available only when the **Interpolation mode** is set to FIR. For more information, see "FIR Interpolation Mode" on page 2-711.

Normalized input bandwidth

The bandwidth of the input divided by Fs/2 (half the input sample frequency).

This parameter is only available when the **Interpolation mode** is set to FIR. For more information, see "FIR Interpolation Mode" on page 2-711.

Out of range interpolation points

When an interpolation point is out of range, this parameter sets the block to either clip the interpolation point, clip the value and issue a warning at the MATLAB command line, or stop the simulation and issue an error at the MATLAB command line. For more information, see "Handling Out-of-Range Interpolation Points" on page 2-709.

Supported Data Types

Port	Supported Data Types
In	• Double-precision floating point • Single-precision floating point
Pts	• Double-precision floating point • Single-precision floating point
Out	• Double-precision floating point • Single-precision floating point

Purpose

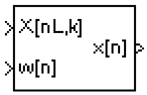
Recover time-domain signals by performing inverse short-time, fast Fourier transform (FFT)

Library

Transforms

`dspxfrm3`

Description



Inverse Short-Time FFT

The Inverse Short-Time FFT block reconstructs the time-domain signal from the frequency-domain output of the Short-Time FFT block using a two-step process. First, the block performs the overlap add algorithm shown below.

$$x[n] = \frac{L}{W(0)} \sum_{p=-\infty}^{\infty} \left[\frac{1}{N} \sum_{k=0}^{N-1} X[pL,k] e^{j2\pi kn/N} \right]$$

Then, the block rebuffers the signal in order to reconstruct the frame-based time-domain signal. Depending on the analysis window used by the Short-Time FFT block, the Inverse Short-Time FFT block might or might not achieve perfect reconstruction of the time domain signal.

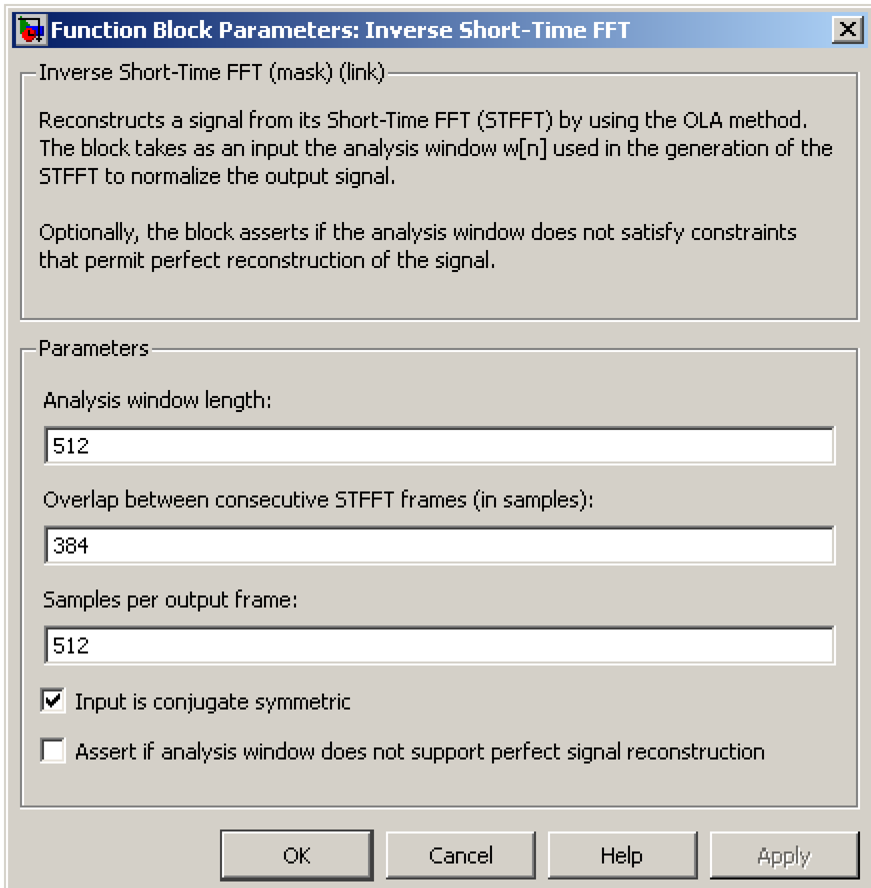
Connect your complex-valued, sample-based, single-channel or multichannel input signal to the X(n,k) port. The block uses the **Overlap between consecutive STFFT frames (in samples)** and **Samples per output frame** parameters as well as the **Input is conjugate symmetric** check box to reconstruct the original time-domain signal. The real or complex-valued, frame-based, single-channel or multichannel inverse short-time FFT is output at port x(n).

Connect your complex-valued, sample-based or frame-based, single-channel analysis window to the w(n) port. When you select the **Assert if analysis window does not support perfect signal reconstruction** check box, the block displays an error when the input signal cannot be perfectly reconstructed. The block uses the values you enter for the **Analysis window length (W)** and **Reconstruction error tolerance**, or maximum amount of allowable error in the reconstruction process, to determine if the signal can be perfectly reconstructed.

Examples

The `dspstsa` demo illustrates how to use the Short-Time FFT and Inverse Short-Time FFT blocks to remove the background noise from a speech signal.

Dialog Box



Analysis window length
Enter the length of the analysis window. This parameter is visible when you select the **Assert if analysis window does not support perfect signal reconstruction** check box.

Overlap between consecutive STFFT frames (in samples)
Enter the number of samples of overlap for each frame of the Short-Time FFT block's input signal. This value should be the same as the **Overlap between consecutive windows (in samples)** parameter in the Short-Time FFT block parameters dialog.

Samples per output frame
Enter the desired frame length of the frame-based output signal.

Input is conjugate symmetric
Select this check box when the input to the block is both floating point and conjugate symmetric, and you want real-valued outputs. When you select this check box when the input is not conjugate symmetric, the output of the block is invalid. This parameter cannot be used for fixed-point signals.

Assert if analysis window does not support perfect signal reconstruction
Select this check box to display an error when the analysis window used by the Short-Time FFT block does not support perfect signal reconstruction.

Reconstruction error tolerance
Enter the amount of acceptable error in the reconstruction of the original signal. This parameter is visible when you select the **Assert if analysis window does not support perfect signal reconstruction** check box.

References

Quatieri, Thomas E. *Discrete-Time Speech Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 2001.

Supported Data Types

Port	Supported Data Types
X(n,k)	• Double-precision floating point • Single-precision floating point
w(n)	• Double-precision floating point • Single-precision floating point
x(n)	• Double-precision floating point • Single-precision floating point

See Also

Burg Method	Signal Processing Blockset
Magnitude FFT	Signal Processing Blockset
Periodogram	Signal Processing Blockset
Short-Time FFT	Signal Processing Blockset
Spectrum Scope	Signal Processing Blockset
Window Function	Signal Processing Blockset
Yule-Walker Method	Signal Processing Blockset
`pwelch`	Signal Processing Toolbox

Purpose

Design inverse sinc filter

Library

Filtering / Filter Designs

`dspfdesign`

Description

This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. You must have a Filter Design Toolbox license to design filters with this block. However, you can run models containing this block without a license. This allows you to run a model sent to you by a colleague who has designed a filter using this block, even if you do not have the Filter Design Toolbox product.

Dialog Box

See “Inverse Sinc Filter Design Dialog Box — Main Pane” in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Kalman Adaptive Filter (Obsolete)

Purpose

Compute filter estimates for inputs using Kalman adaptive filter algorithm

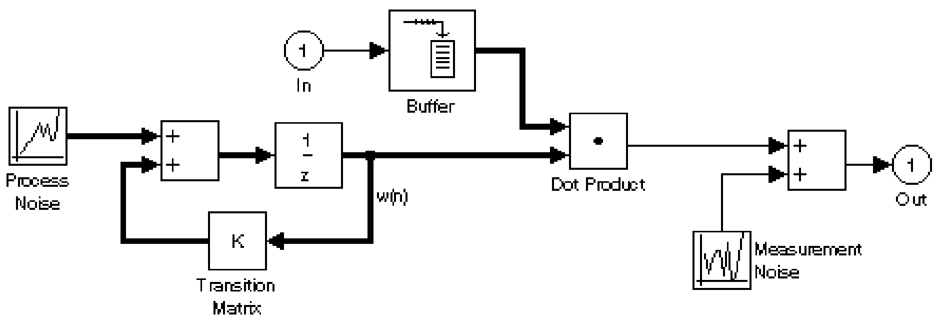
Library

`dspobslib`

Description

Note The Kalman Adaptive Filter block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the Kalman Filter block.

The Kalman Adaptive Filter block computes the optimal linear minimum mean-square estimate (MMSE) of the FIR filter coefficients using a one-step predictor algorithm. This Kalman filter algorithm is based on the following physical realization of a dynamic system.



The Kalman filter assumes that there are no deterministic changes to the filter taps over time (that is, the transition matrix is identity), and that the only observable output from the system is the filter output with additive noise. The corresponding Kalman filter is expressed in matrix form as

$$g(n) = \frac{K(n-1)u(n)}{u^H(n)K(n-1)u(n) + Q_M}$$

$$y(n) = u^H(n)\hat{w}(n)$$

$$e(n) = d(n) - y(n)$$

$$\hat{w}(n+1) = \hat{w}(n) + e(n)g(n)$$

$$K(n) = K(n-1) - g(n)u^H(n)K(n-1) + Q_p$$

The variables are as follows

Variable	Description
n	The current algorithm iteration
$u(n)$	The buffered input samples at step n
$K(n)$	The correlation matrix of the state estimation error
$g(n)$	The vector of Kalman gains at step n
$\hat{w}(n)$	The vector of filter-tap estimates at step n
$y(n)$	The filtered output at step n
$e(n)$	The estimation error at step n
$d(n)$	The desired response at step n
Q_M	The correlation matrix of the measurement noise
Q_P	The correlation matrix of the process noise

The correlation matrices, Q_M and Q_P, are specified in the parameter dialog by scalar variance terms to be placed along the matrix diagonals, thus ensuring that these matrices are symmetric. The filter algorithm based on this constraint is also known as the random-walk Kalman filter.

The implementation of the algorithm in the block is optimized by exploiting the symmetry of the input covariance matrix $K(n)$. This decreases the total number of computations by a factor of two.

The block icon has port labels corresponding to the inputs and outputs of the Kalman algorithm. Note that inputs to the In and Err ports must be sample-based scalars with the same complexity. The signal at the Out port is a scalar, while the signal at the Taps port is a sample-based vector.

Block Ports	Corresponding Variables
In	u, the scalar input, which is internally buffered into the vector $u(n)$
Out	$y(n)$, the filtered scalar output
Err	$e(n)$, the scalar estimation error
Taps	$\hat{w}(n)$, the vector of filter-tap estimates

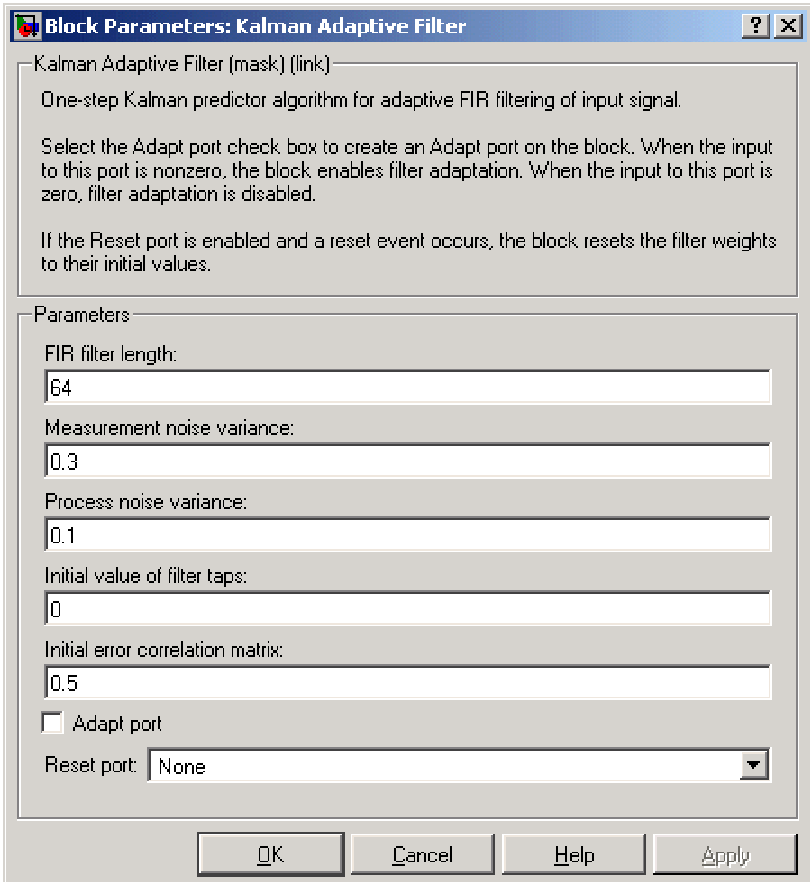
An optional Adapt input port is added when you select the **Adapt port** check box in the dialog. When this port is enabled, the block continuously adapts the filter coefficients while the Adapt input is nonzero. A zero-valued input to the `Adapt` port causes the block to stop adapting, and to hold the filter coefficients at their current values until the next nonzero Adapt input.

The **FIR filter length** parameter specifies the length of the filter that the Kalman algorithm estimates. The **Measurement noise variance** and the **Process noise variance** parameters specify the correlation matrices of the measurement and process noise, respectively. The **Measurement noise variance** must be a scalar, while the **Process noise variance** can be a vector of values to be placed along the diagonal, or a scalar to be repeated for the diagonal elements.

The **Initial value of filter taps** specifies the initial value $\hat{w}(0)$ as a vector, or as a scalar to be repeated for all vector elements. The **Initial**

error correlation matrix specifies the initial value *K(0)*, and can be a diagonal matrix, a vector of values to be placed along the diagonal, or a scalar to be repeated for the diagonal elements.

Dialog Box



FIR filter length

The length of the FIR filter.

Measurement noise variance
The value to appear along the diagonal of the measurement noise correlation matrix. Tunable.

Process noise variance
The value to appear along the diagonal of the process noise correlation matrix. Tunable.

Initial value of filter taps
The initial FIR filter coefficients.

Initial error correlation matrix
The initial value of the error correlation matrix.

Adapt port
Enables the `Adapt` port.

References

Haykin, S. *Adaptive Filter Theory*. 3rd ed. Englewood Cliffs, NJ: Prentice Hall, 1996.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

LMS Adaptive Filter (Obsolete)	Signal Processing Blockset
RLS Adaptive Filter (Obsolete)	Signal Processing Blockset

See "Adaptive Filters" for related information.

Purpose

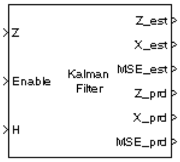
Predict or estimate states of dynamic systems

Library

Filtering/Adaptive Filters

`dspadpt3`

Description



Kalman Filter

Use the Kalman Filter block to predict or estimate the state of a dynamic system from a series of incomplete and/or noisy measurements. Suppose you have a noisy linear system that is defined by the following equations:

$$x_k = Ax_{k-1} + w_{k-1}$$
$$z_k = Hx_k + v_k$$

This block can use the previously estimated state, $\hat{x}_{k-1}$, to predict the current state at time k, x_k^-, as shown by the following equation:

$$x_k^- = A\hat{x}_{k-1}$$
$$P_k^- = A\hat{P}_{k-1}A^T + Q$$

The block can also use the current measurement, z_k, and the predicted state, x_k^-, to estimate the current state value at time k, $\hat{x}_k$, so that it is a more accurate approximation:

$$K_k = P_k^- H^T \left(HP_k^- H^T + R\right)^{-1}$$
$$\hat{x}_k = x_k^- + K_k\left(z_k - Hx_k^-\right)$$
$$\hat{P}_k = \left(I - K_k H\right)P_k^-$$

The variables in the previous equations are defined in the following table.

Variable	Definition	Default Value or Initial Condition
x	State	N/A
$\hat{x}$	Estimated state	`zeros([6, 1])`
x^{-}	Predicted state	N/A
A	State transition matrix	$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
w	Process noise	N/A
z	Measurement	N/A
H	Measurement matrix	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
v	Measurement noise	N/A
$\hat{P}$	Estimated error covariance	`10*eye(6)`
P^{-}	Predicted error covariance	N/A
Q	Process noise covariance	`0.05*eye(6)`
K	Kalman gain	N/A

Variable	Definition	Default Value or Initial Condition
R	Measurement noise covariance	`eye(4)`
I	Identity matrix	N/A

In the previous equations, z is a vector of measurement values. Most of the time, the block processes Z, an M-by-N matrix, where M is the number of measurement values and N is the number of filters.

Use the **Number of filters** parameter to specify the number of filters to use to predict or estimate the current value.

Use the **Enable filters** parameter to specify which filters are enabled or disabled at each time step. If you select `Always`, the filters are always enabled. If you choose `Specify via input port <Enable>`, the Enable port appears on the block. The input to this port must be a row vector of 1s and 0s whose length is equal to the number of filters. For example, if there are 3 filters and the input to the Enable port is [1 0 1], only the first and third filter are enabled at this time step. If you select the **Reset the estimated state and estimated error covariance when filters are disabled** check box, the estimated and predicted states as well as the estimated error covariance that correspond to the disabled filters are reset to their initial values.

Note All filters have the same state transition matrix, measurement matrix, initial conditions, and noise covariance, but their state, measurement, enable, and MSE signals are unique. Within the state, measurement, enable, and MSE signals, each column corresponds to a filter.

Use the **Measurement matrix source** parameter to specify how to enter the measurement matrix values. If you select `Specify via dialog`, the **Measurement matrix** parameter appears in the dialog box. If you select `Input port <H>`, the H port appears on the block. Use this port to specify your measurement matrix.

See the Radar Tracking demo for an example of how to use this block. You can open this demo by typing

```
aero_radmod_dsp
```

at the MATLAB command prompt.

If you have the Video and Image Processing Blockset product installed, you can also explore the People Tracking demo. Open this demo by typing

```
viptrackpeople
```

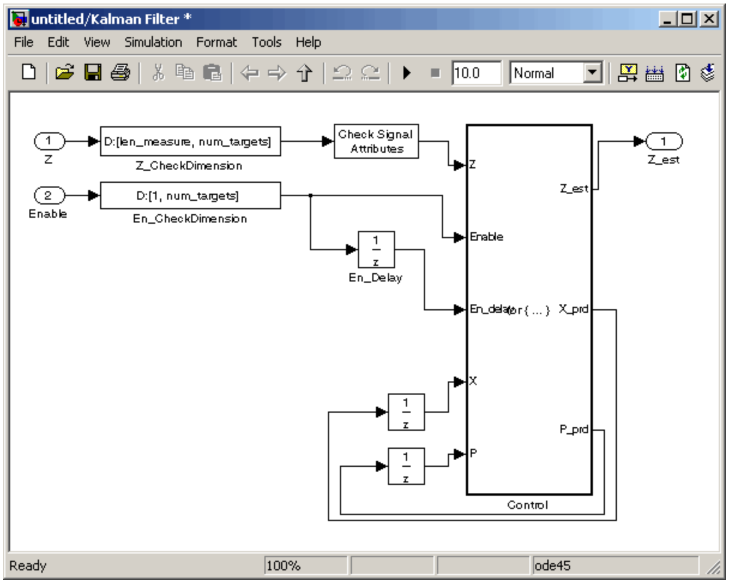
at the MATLAB command prompt.

Fixed-Point Data Types

This block can process fixed-point data types if you make several modifications to its subsystem:

1. Add a Kalman Filter block to a Simulink model.
2. Right-click the block and select **Link Options > Disable Link**.
3. Right-click the block and select **Look Under Mask**.

   The subsystem that implements the Kalman filter opens.

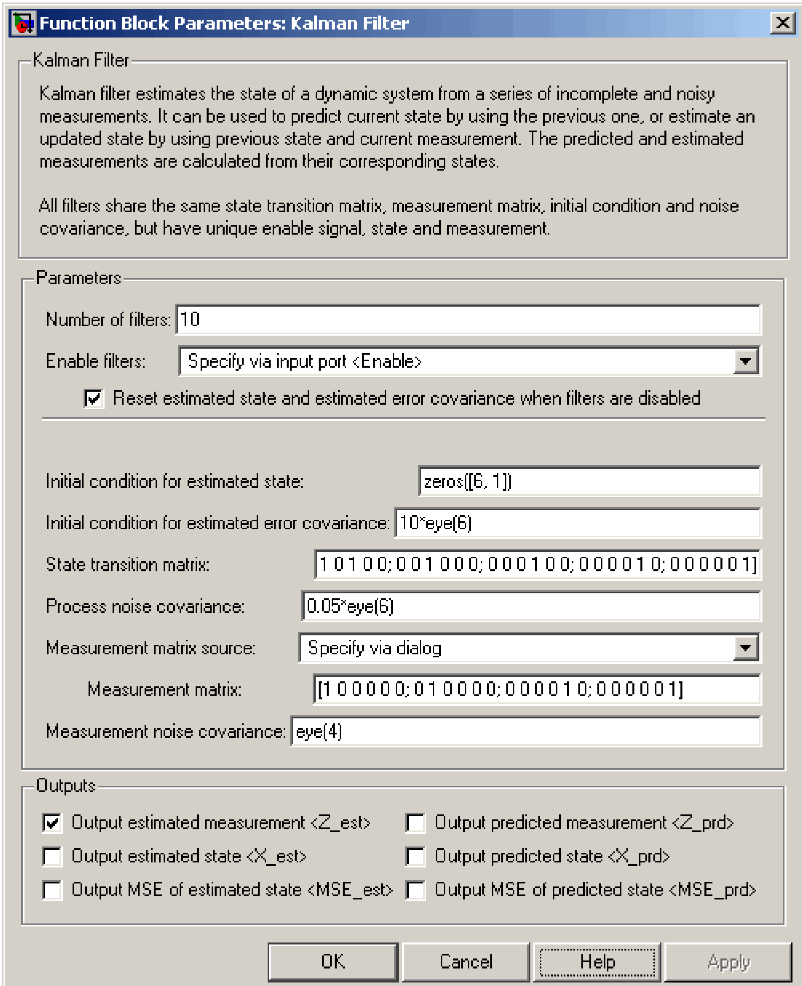


4 Double-click the Check Signal Attributes block and change the **Data type** parameter to `Ignore`.

If you have the Video and Image Processing Blockset product installed, you can see an example by typing `viptrackpeople_fixpt_all` at the MATLAB command prompt. This opens the fixed-point version of the People Tracking demo. The Kalman Filter block is located inside the Enabled Subsystem block called Tracking.

Dialog Box

The Kalman Filter dialog box appears as shown in the following figure.



Number of filters

Specify the number of filters to use to predict or estimate the current value.

Enable filters
Specify which filters are enabled or disabled at each time step. If you select `Always`, the filters are always enabled. If you choose `Specify via input port <Enable>`, the Enable port appears on the block.

Reset the estimated state and estimated error covariance when filters are disabled
If you select this check box, the estimated and predicted states as well as the estimated error covariance that correspond to the disabled filters are reset to their initial values. This parameter is visible if, for the **Enable filters** parameter, you select `Specify via input port <Enable>`.

Initial condition for estimated state
Enter the initial condition for the estimated state.

Initial condition for estimated error covariance
Enter the initial condition for the estimated error covariance.

State transition matrix
Enter the state transition matrix.

Process noise covariance
Enter the process noise covariance.

Measurement matrix source
Specify how to enter the measurement matrix values. If you select `Specify via dialog`, the **Measurement matrix** parameter appears in the dialog box. If you select `Input port <H>`, the H port appears on the block.

Measurement matrix
Enter the measurement matrix values. This parameter is visible if you select `Specify via dialog` for the **Measurement matrix source** parameter.

Measurement noise covariance
Enter the measurement noise covariance.

Output estimated measurement <Z_est>
Select this check box if you want the block to output the estimated measurement.

Output estimated state <X_est>
Select this check box if you want the block to output the estimated state.

Output MSE of estimated state <MSE_est>
Select this check box if you want the block to output the mean-squared error of the estimated state.

Output predicted measurement <Z_prd>
Select this check box if you want the block to output the predicted measurement.

Output predicted state <X_prd>
Select this check box if you want the block to output the predicted state.

Output MSE of predicted state <MSE_prb>
Select this check box if you want the block to output the mean-squared error of the predicted state.

References

[1] Haykin, Simon. *Adaptive Filter Theory*. Upper Saddle River, NJ: Prentice Hall, 1996.

[2] Welch, Greg and Gary Bishop, "An Introduction to the Kalman Filter," TR 95–041, Department of Computer Science, University of North Carolina.

Supported Data Types

Port	Input/Output	Supported Data Types
Z	M-by-N measurement where M is the length of the measurement vector and N is the number of filters.	• Double-precision floating point • Single-precision floating point
Enable	1-by-N vector of 1s and 0s where N is the number of filters.	• Double-precision floating point • Single-precision floating point • Boolean
H	M-by-P measurement matrix where M is the length of the measurement vector and P is the length of the filter state vectors.	Same as Z port
Z_est	M-by-N estimated measurement matrix where M is the length of the measurement vector and N is the number of filters.	Same as Z port
X_est	P-by-N estimated state matrix where P is the length of the filter state vectors and N is the number of filters.	Same as Z port
MSE_est	1-by-N vector that represents the mean-squared-error of the estimated state. N is the number of filters.	Same as Z port

Port	Input/Output	Supported Data Types
Z_prd	M-by-N predicted measurement matrix where M is the length of the measurement vector and N is the number of filters.	Same as Z port
X_prd	P-by-N predicted state matrix where P is the length of the filter state vectors and N is the number of filters.	Same as Z port
MSE_prd	1-by-N vector that represents the mean-squared-error of the predicted state. N is the number of filters.	Same as Z port

See Also

LDL Solver	Signal Processing Blockset

Purpose

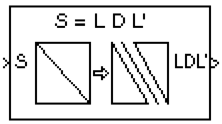
Factor square Hermitian positive definite matrices into lower, upper, and diagonal components

Library

Math Functions / Matrices and Linear Algebra / Matrix Factorizations

`dspfactors`

Description



The LDL Factorization block uniquely factors the square Hermitian positive definite input matrix S as

$$S = LDL^*$$

where L is a lower triangular square matrix with unity diagonal elements, D is a diagonal matrix, and L^* is the Hermitian (complex conjugate) transpose of L. Only the diagonal and lower triangle of the input matrix are used, and any imaginary component of the diagonal entries is disregarded.

The block's output is a composite matrix with lower triangle elements l_{ij} from L, diagonal elements d_{ij} from D, and upper triangle elements u_{ij} from L^*. It is always sample based. The output format is shown below for a 5-by-5 matrix.

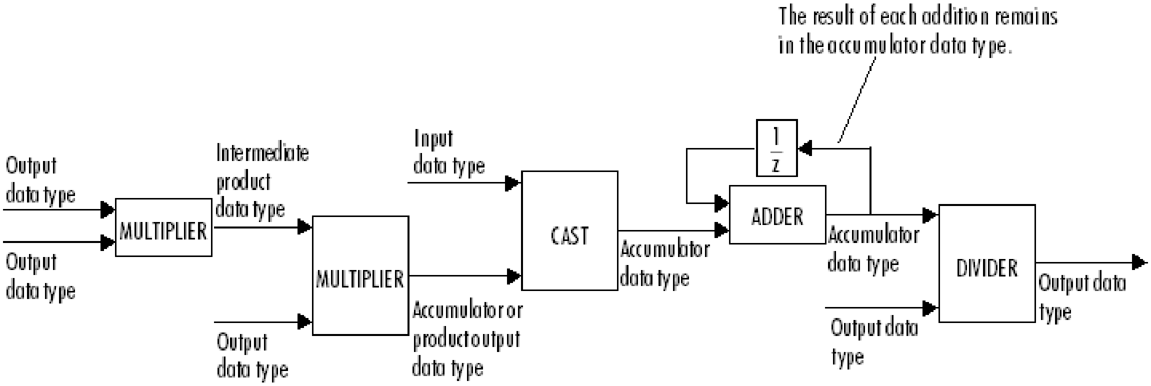
$$\begin{bmatrix} d_{11} & u_{12} & u_{13} & u_{14} & u_{15} \\ l_{21} & d_{22} & u_{23} & u_{24} & u_{25} \\ l_{31} & l_{32} & d_{33} & u_{34} & u_{35} \\ l_{41} & l_{42} & l_{43} & d_{44} & u_{45} \\ l_{51} & l_{52} & l_{53} & l_{54} & d_{55} \end{bmatrix} \qquad u_{ij} = l^*_{ji}$$

LDL factorization requires half the computation of Gaussian elimination (LU decomposition), and is always stable. It is more efficient than Cholesky factorization because it avoids computing the square roots of the diagonal elements.

The algorithm requires that the input be square and Hermitian positive definite. When the input is not positive definite, the block reacts with the behavior specified by the **Non-positive definite input** parameter.

Fixed-Point Data Types

The following diagram shows the data types used within the LDL Factorization block for fixed-point signals.

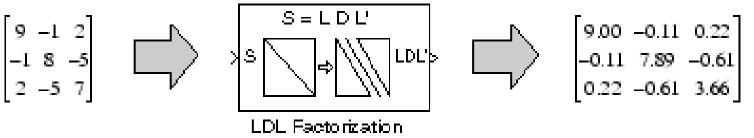


You can set the intermediate product, product output, accumulator, and output data types in the block dialog as discussed below.

The output of the second multiplier is in the product output data type when the input is real. When the input is complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see "Multiplication Data Types".

Examples

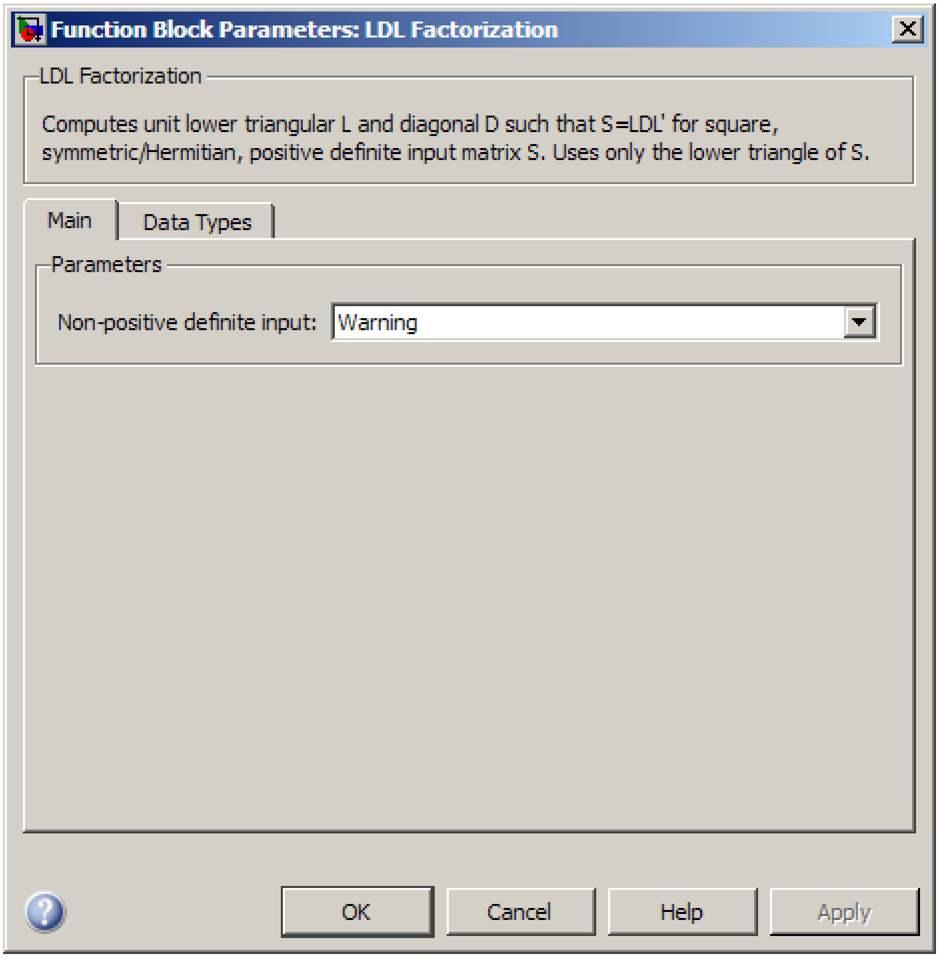
LDL decomposition of a 3-by-3 Hermitian positive definite matrix:



$$L = \begin{bmatrix} 1 & 0 & 0 \\ -0.11 & 1 & 0 \\ 0.22 & -0.61 & 1 \end{bmatrix} \qquad D = \begin{bmatrix} 9.00 & 0 & 0 \\ 0 & 7.89 & 0 \\ 0 & 0 & 3.66 \end{bmatrix} \qquad L' = \begin{bmatrix} 1 & -0.11 & 0.22 \\ 0 & 1 & -0.61 \\ 0 & 0 & 1 \end{bmatrix}$$

Dialog Box

The **Main** pane of the LDL Factorization block dialog appears as follows.

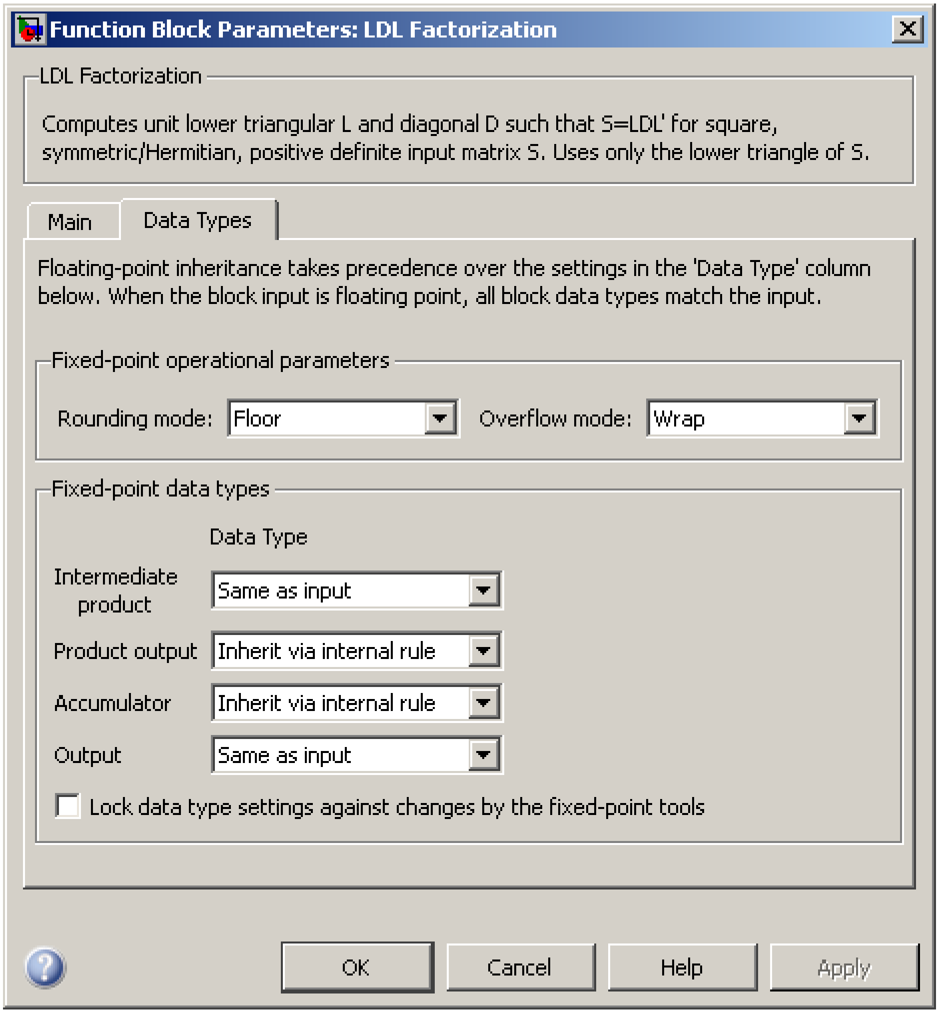


Non-positive definite input

Specify the action when nonpositive definite matrix inputs occur:

- `Ignore` — Proceed with the computation and do not issue an alert. The output is not a valid factorization. A partial factorization is present in the upper left corner of the output.
- `Warning` — Display a warning message in the MATLAB Command Window, and continue the simulation. The output is not a valid factorization. A partial factorization is present in the upper left corner of the output.
- `Error` — Display an error dialog and terminate the simulation.

The **Data Types** pane of the LDL Factorization block dialog appears as follows.



Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode for fixed-point operations.

Intermediate product

Use this parameter to specify how you would like to designate the intermediate product word and fraction lengths. See "Fixed-Point Data Types" on page 2-738 for an illustration depicting the use of the intermediate product data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the intermediate product, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the intermediate product. This block requires power-of-two slope and a bias of zero.

Product output

Use this parameter to specify how you would like to designate the product output word and fraction lengths. See "Fixed-Point Data Types" on page 2-738 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block:

- When you select `Inherit via internal rule`, the product output word length and fraction length are calculated automatically. For information about how the product output word and fraction lengths are calculated when an internal rule is used, see "Inherit via Internal Rule".
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator

Use this parameter to specify how you would like to designate the accumulator word and fraction lengths. See "Fixed-Point Data Types" on page 2-738 and "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block.

- When you select `Inherit via internal rule`, the accumulator word length and fraction length are calculated automatically. For information about how the accumulator word and fraction lengths are calculated when an internal rule is used, see "Inherit via Internal Rule".
- When you select `Same as product output`, these characteristics match those of the product output.
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Use this parameter to specify how you would like to designate the output word and fraction lengths. See "Fixed-Point Data Types" on page 2-738 for an illustration depicting the use of the output data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

References

Golub, G. H., and C. F. Van Loan. *Matrix Computations*. 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

Supported Data Types

Port	Supported Data Types
S	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
LDL'	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

Cholesky Factorization	Signal Processing Blockset
LDL Inverse	Signal Processing Blockset
LDL Solver	Signal Processing Blockset
LU Factorization	Signal Processing Blockset
QR Factorization	Signal Processing Blockset

See “Matrix Factorizations” for related information.

LDL Inverse

Purpose

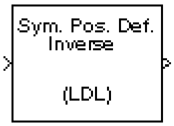
Compute inverse of Hermitian positive definite matrix using LDL factorization

Library

Math Functions / Matrices and Linear Algebra / Matrix Inverses

`dspinverses`

Description



The LDL Inverse block computes the inverse of the Hermitian positive definite input matrix S by performing an LDL factorization.

$$S^{-1} = \left(LDL^{*}\right)^{-1}$$

L is a lower triangular square matrix with unity diagonal elements, D is a diagonal matrix, and L^* is the Hermitian (complex conjugate) transpose of L. Only the diagonal and lower triangle of the input matrix are used, and any imaginary component of the diagonal entries is disregarded. The output is always sample based.

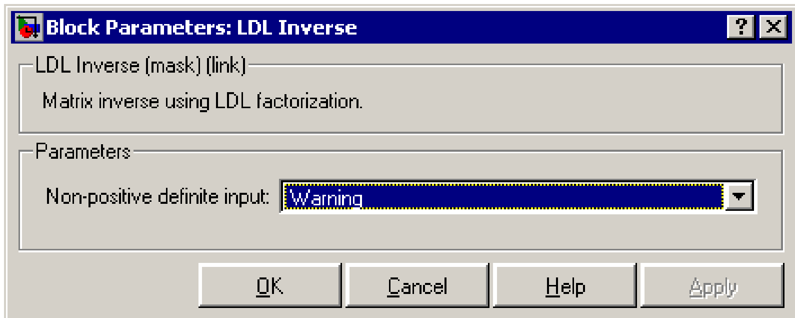
LDL factorization requires half the computation of Gaussian elimination (LU decomposition), and is always stable. It is more efficient than Cholesky factorization because it avoids computing the square roots of the diagonal elements.

The algorithm requires that the input be Hermitian positive definite. When the input is not positive definite, the block reacts with the behavior specified by the **Non-positive definite input** parameter. The following options are available:

- `Ignore` — Proceed with the computation and do not issue an alert. The output is not a valid inverse.
- `Warning` — Display a warning message in the MATLAB command window, and continue the simulation. The output is not a valid inverse.
- `Error` — Display an error dialog and terminate the simulation.

Note The **Non-positive definite input** parameter is a diagnostic parameter. Like all diagnostic parameters on the Configuration Parameters dialog, it is set to `Ignore` in the code generated for this block by Real-Time Workshop code generation software.

Dialog Box



Non-positive definite input
Response to nonpositive definite matrix inputs.

References

Golub, G. H., and C. F. Van Loan. *Matrix Computations*. 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Cholesky Inverse	Signal Processing Blockset
LDL Factorization	Signal Processing Blockset
LDL Solver	Signal Processing Blockset
LU Inverse	Signal Processing Blockset

Pseudoinverse	Signal Processing Blockset
`inv`	MATLAB

See “Matrix Inverses” for related information.

Purpose

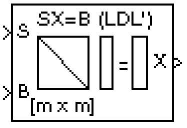
Solve $SX=B$ for X when S is square Hermitian positive definite matrix

Library

Math Functions / Matrices and Linear Algebra / Linear System Solvers

`dspsolvers`

Description



The LDL Solver block solves the linear system SX=B by applying LDL factorization to the matrix at the S port, which must be square (M-by-M) and Hermitian positive definite. Only the diagonal and lower triangle of the matrix are used, and any imaginary component of the diagonal entries is disregarded. The input to the B port is the right side M-by-N matrix, B. The output is the unique solution of the equations, M-by-N matrix X, and is always sample based.

A length-M 1-D vector input for right side B is treated as an M-by-1 matrix.

When the input is not positive definite, the block reacts with the behavior specified by the **Non-positive definite input** parameter. The following options are available:

- `Ignore` — Proceed with the computation and do not issue an alert. The output is not a valid solution.
- `Warning` — Proceed with the computation and display a warning message in the MATLAB Command Window. The output is not a valid solution.
- `Error` — Display an error dialog and terminate the simulation.

Note The **Non-positive definite input** parameter is a diagnostic parameter. Like all diagnostic parameters on the Configuration Parameters dialog, it is set to `Ignore` in the code generated for this block by Real-Time Workshop code generation software.

Algorithm

The LDL algorithm uniquely factors the Hermitian positive definite input matrix S as

$$S = LDL^*$$

where L is a lower triangular square matrix with unity diagonal elements, D is a diagonal matrix, and L^* is the Hermitian (complex conjugate) transpose of L.

The equation

$$LDL^*X = B$$

is solved for X by the following steps:

1. Substitute

$$Y = DL^*X$$

2. Substitute

$$Z = L^*X$$

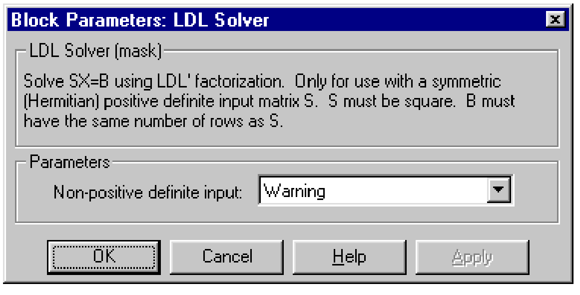
3. Solve one diagonal and two triangular systems.

$$LY = B$$

$$DZ = Y$$

$$L^*X = Z$$

Dialog Box



Non-positive definite input
Response to nonpositive definite matrix inputs.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Autocorrelation LPC	Signal Processing Blockset
Cholesky Solver	Signal Processing Blockset
LDL Factorization	Signal Processing Blockset
LDL Inverse	Signal Processing Blockset
Levinson-Durbin	Signal Processing Blockset
LU Solver	Signal Processing Blockset
QR Solver	Signal Processing Blockset

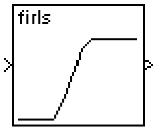
See "Linear System Solvers" for related information.

Least Squares FIR Filter Design (Obsolete)

Purpose Design and implement least-squares FIR filter

Library `dspobslib`

Description



Note The Least Squares FIR Filter Design block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the Digital Filter block.

The Least Squares FIR Filter Design block designs an FIR filter and applies it to a discrete-time input using the Direct Form II Transpose Filter block. The filter design uses the Signal Processing Toolbox `firls` function to minimize the integral of the squared error between the desired frequency response and the actual frequency response.

An *M*-by-*N* sample-based matrix input is treated as *M**N* independent channels, and an *M*-by-*N* frame-based matrix input is treated as *N* independent channels. In both cases, the block filters each channel independently over time, and the output has the same size and frame status as the input.

The **Filter type** parameter allows you to specify one of the following filters:

- `Multiband` — The `Multiband` filter designs a linear-phase filter with an arbitrary magnitude response.
- `Differentiator` — The `Differentiator` filter approximates the ideal differentiator. Differentiators are antisymmetric FIR filters with approximately linear magnitude responses. To obtain the correct derivative, scale the **Gains at these frequencies** vector by πF_s rad/s, where F_s is the sample frequency in Hertz.

- `Hilbert Transformer` — The `Hilbert Transformer` filter approximates the ideal Hilbert transformer. Hilbert transformers are antisymmetric FIR filters with approximately constant magnitude.

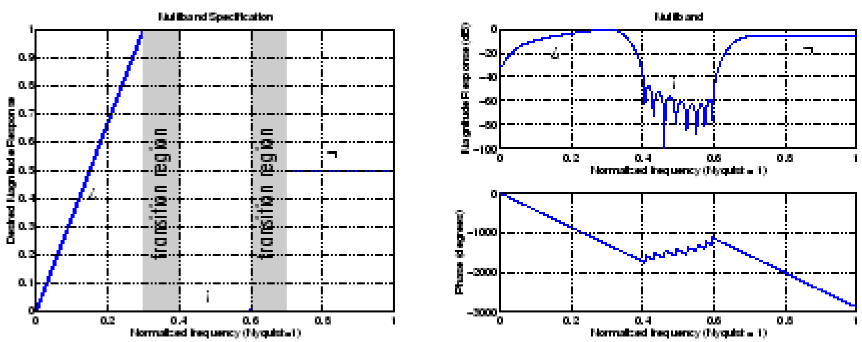
The **Band-edge frequency vector** parameter is a vector of frequency points in the range `0` to `1`, where `1` corresponds to half the sample frequency. This vector must have even length, and intermediate points must appear in ascending order. The **Gains at these frequencies** parameter is a vector containing the desired magnitude response at the corresponding points in the **Band-edge frequency vector**.

Each odd-indexed frequency-amplitude pair defines the left endpoint of a line segment representing the desired magnitude response in that frequency band. The corresponding even-indexed frequency-amplitude pair defines the right endpoint. Between the frequency bands specified by these end-points, there may be undefined sections of the specified frequency response. These are called “don’t care” or “transition” regions, and the magnitude response in these areas is a result of the optimization in the other (specified) frequency ranges.

Band edge frequency = `[0 0.3 0.4 0.6 0.7 1]`

Gains = `[0 1 0 0 0.5 0.5]`

Band: ¿ ¡ ¬



The **Weights** parameter is a vector that specifies the emphasis to be placed on minimizing the error in certain frequency bands relative to others. This vector specifies one weight per band, so it is half the length of the **Band-edge frequency vector** and **Gains at these frequencies** vectors.

In most cases, differentiators and Hilbert transformers have only a single band, so the weight is a scalar value that does not affect the final filter. However, the **Weights** parameter is useful when using the block to design an antisymmetric multiband filter, such as a Hilbert transformer with stopbands.

For more information on the **Band-edge frequency vector**, **Gains at these frequencies**, and **Weights** parameters, see "Filter Designs and Implementation" in the Signal Processing Toolbox documentation. For more on the FIR filter algorithm, see the description of the `firls` function in the Signal Processing Toolbox documentation.

Examples

Example 1: Multiband

Consider a lowpass filter with a transition band in the normalized frequency range 0.4 to 0.5, and 10 times more error minimization in the stopband than the passband. In this case,

- **Filter type** = `Multiband`
- **Band-edge frequency vector** = `[0 0.4 0.5 1]`
- **Gains at these frequencies** = `[1 1 0 0]`
- **Weights** = `[1 10]`

Example 2: Differentiator

Assume the specifications for a differentiator filter require it to have order 21. The "ramp" response extends over the entire frequency range. In this case, specify:

- **Filter type** = `Differentiator`
- **Filter order** = `21`

- **Band-edge frequency vector** = `[0 1]`
- **Gains at these frequencies** = `[0 pi*Fs]`

For a type III (even order) filter, the differentiation band should stop short of half the sample frequency. For example, if the filter order is 20, you could specify the block parameters as follows:

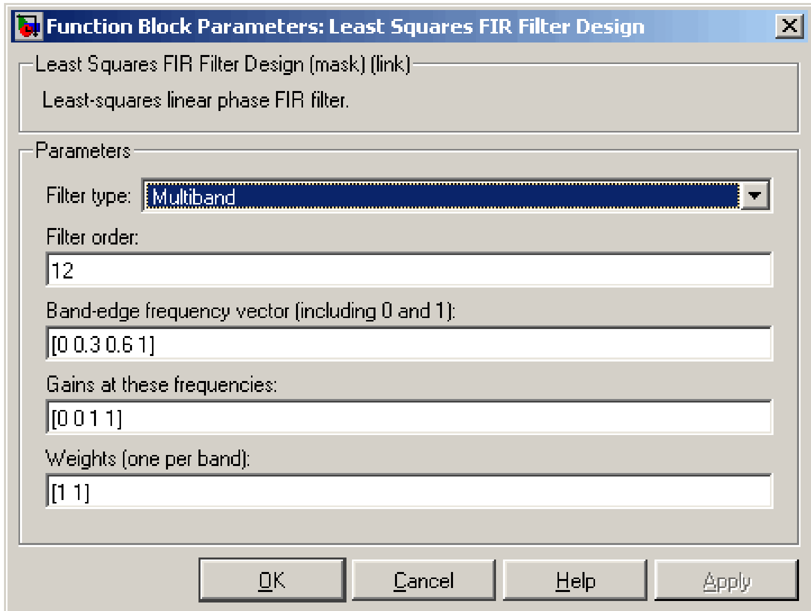
- **Filter type** = `Differentiator`
- **Filter order** = `20`
- **Band-edge frequency vector** = `[0 0.9]`
- **Gains at these frequencies** = `[0 0.9*pi*Fs]`

Example 3: Hilbert Transformer

Assume the specifications for a Hilbert transformer filter require it to have order 21. The passband extends over approximately the entire frequency range. In this case, specify:

- **Filter type** = `Hilbert Transform`
- **Filter order** = `21`
- **Band-edge frequency vector** = `[0.1 1]`
- **Gains at these frequencies** = `[1 1]`

Dialog Box



Filter type
  The filter type. Tunable.

Filter order
  The filter order.

Band-edge frequency vector
  A vector of frequency points, in ascending order, in the range `0` to `1`. The value `1` corresponds to half the sample frequency. This vector must have even length. Tunable.

Gains at these frequencies
  A vector of frequency-response amplitudes corresponding to the points in the **Band-edge frequency vector**. This vector must be the same length as the **Band-edge frequency vector**. Tunable.

Weights

A vector containing one weight for each frequency band. This vector must be half the length of the **Band-edge frequency vector** and **Gains at these frequencies** vectors. Tunable.

References

Oppenheim, A. V. and R. W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

Proakis, J. and D. Manolakis. *Digital Signal Processing*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

Least Squares Polynomial Fit

Purpose

Compute polynomial coefficients that best fit input data in least-squares sense

Library

Math Functions / Polynomial Functions

`dsppolyfun`

Description

The Least Squares Polynomial Fit block computes the coefficients of the nth order polynomial that best fits the input data in the least-squares sense, where you specify n in the **Polynomial order** parameter. A distinct set of n+1 coefficients is computed for each column of the M-by-N input, u.

For a given input column, the block computes the set of coefficients, c_1, c_2, ..., c_{n+1}, that minimizes the quantity

$$\sum_{i=1}^{M}(u_i - \hat{u}_i)^2$$

where u_i is the ith element in the input column, and

$$\hat{u}_i = f(x_i) = c_1 x_i^n + c_2 x_i^{n-1} + \ \dots\ + c_{n+1}$$

The values of the independent variable, x_1, x_2, ..., x_M, are specified as a length-M vector by the **Control points** parameter. The same M control points are used for all N polynomial fits, and can be equally or unequally spaced. The equivalent MATLAB code is shown below.

```
c = polyfit(x,u,n)      % Equivalent MATLAB code
```

Inputs can be frame based or sample based. For convenience, a length-M 1-D vector input is treated as an M-by-1 matrix.

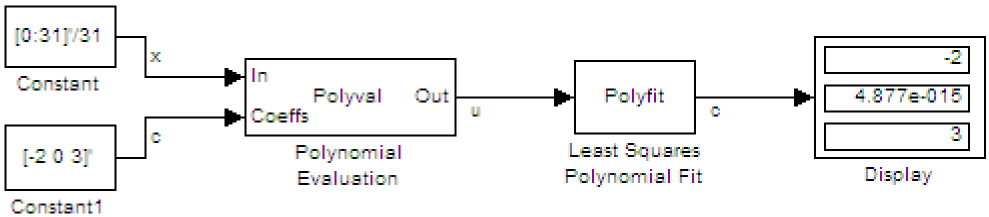
Each column of the (n+1)-by-N output matrix, c, represents a set of n+1 coefficients describing the best-fit polynomial for the corresponding column of the input. The coefficients in each column are arranged in order of descending exponents, c_1, c_2, ..., c_{n+1}. The output is always sample based.

Examples

In the model below, the Polynomial Evaluation block uses the second-order polynomial

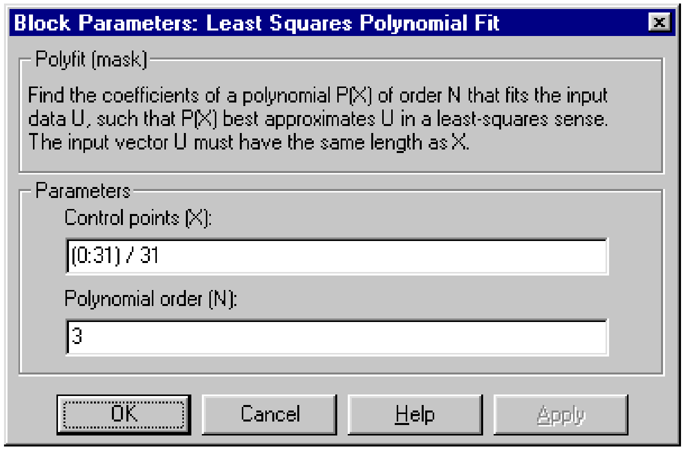
$$y = -2u^2 + 3$$

to generate four values of dependent variable y from four values of independent variable u, received at the top port. The polynomial coefficients are supplied in the vector `[-2 0 3]` at the bottom port. Note that the coefficient of the first-order term is zero.



The **Control points** parameter of the Least Squares Polynomial Fit block is configured with the same four values of independent variable u that are used as input to the Polynomial Evaluation block, `[1 2 3 4]`. The Least Squares Polynomial Fit block uses these values together with the input values of dependent variable y to reconstruct the original polynomial coefficients.

Dialog Box



Control points

The values of the independent variable to which the data in each input column correspond. For an *M*-by-*N* input, this parameter must be a length-*M* vector. Tunable.

Polynomial order

The order, *n*, of the polynomial to be used in constructing the best fit. The number of coefficients is *n*+1.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Detrend	Signal Processing Blockset
Polynomial Evaluation	Signal Processing Blockset
Polynomial Stability Test	Signal Processing Blockset
`polyfit`	MATLAB

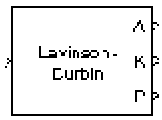
Levinson-Durbin

Purpose Solve linear system of equations using Levinson-Durbin recursion

Library Math Functions / Matrices and Linear Algebra / Linear System Solvers

`dspsolvers`

Description



The Levinson-Durbin block solves the nth-order system of linear equations

$$Ra = b$$

in the cases where:

- R is a Hermitian, positive-definite, Toeplitz matrix.
- b is identical to the first column of R shifted by one element and with the opposite sign.

$$\begin{bmatrix} r(1) & r^*(2) & \cdots & r^*(n) \\ r(2) & r(1) & \cdots & r^*(n-1) \\ \vdots & \vdots & \ddots & \vdots \\ r(n) & r(n-1) & \cdots & r(1) \end{bmatrix} \begin{bmatrix} a(2) \\ a(3) \\ \vdots \\ a(n+1) \end{bmatrix} = \begin{bmatrix} -r(2) \\ -r(3) \\ \vdots \\ -r(n+1) \end{bmatrix}$$

The input to the block, r `= [`r`(1)` r`(2) ...` r`(`n`+1)]`, can be a 1-D or 2-D row or column vector or a sample- or frame-based matrix. If the input is a matrix, the block treats each column as an independent channel and solves it separately. Each channel of the input contains lags *0* through n of an autocorrelation sequence, which appear in the matrix R.

The block can output the polynomial coefficients, A, the reflection coefficients, K, and the prediction error power, P, in various combinations. The **Output(s)** parameter allows you to enable the A and K outputs by selecting one of the following settings:

- `A` — For each channel, port A outputs A`=[1` a`(2)` a`(3) ...` a`(`n`+1)]`, the solution to the Levinson-Durbin equation. A has the same dimension as the input. You can also view the elements of each output channel as the coefficients of an nth-order autoregressive (AR) process.

- `K` — For each channel, port K outputs K=`[`k`(1)` k`(2)` `...` k`(`n`)]`, which contains n reflection coefficients and has the same dimension as the input, less one element. A scalar input channel causes an error when you select `K`. You can use reflection coefficients to realize a lattice representation of the AR process described later in this page.
- `A and K` — The block outputs both representations at their respective ports. A scalar input channel causes an error when you select `A and K`.

A and K are matrices when the input is a matrix. Otherwise, A and K are 1-D vectors.

Select the **Output prediction error power (P)** check box to output the prediction error power for each channel, P. For each channel, P represents the power of the output of an FIR filter with taps A and input autocorrelation described by r, where A represents a prediction error filter and r is the input to the block. In this case, A is a whitening filter. P has one element per input channel.

When you select the **If the value of lag 0 is zero, A=[1 zeros], K=[zeros], P=0** check box (default), an input channel whose r`(1)` element is zero generates a zero-valued output. When you clear this check box, an input with r`(1) = 0` generates `NaN`s in the output. In general, an input with r`(1) = 0` is invalid because it does not construct a positive-definite matrix R. Often, however, blocks receive zero-valued inputs at the start of a simulation. The check box allows you to avoid propagating `NaN`s during this period.

Applications

One application of the Levinson-Durbin formulation implemented by this block is in the Yule-Walker AR problem, which concerns modeling an unknown system as an autoregressive process. You would model such a process as the output of an all-pole IIR filter with white Gaussian noise input. In the Yule-Walker problem, the use of the signal's autocorrelation sequence to obtain an optimal estimate leads to an $Ra = b$ equation of the type shown above, which is most efficiently solved by Levinson-Durbin recursion. In this case, the input to the block represents the autocorrelation sequence, with `r(1)` being the zero-lag value. The output at the block's A port then contains the coefficients of the autoregressive process that optimally models the system. The coefficients are ordered in descending powers of z, and the AR process is minimum phase. The prediction error, G, defines the gain for the unknown system, where $G = \sqrt{P}$:

$$H(z) = \frac{G}{A(z)} = \frac{G}{1 + a(2)z^{-1} + \ \dots \ + a(n+1)z^{-n}}$$

The output at the block's K port contains the corresponding reflection coefficients, `[`k`(1)` k`(2) ...` k`(n)]`, for the lattice realization of this IIR filter. The Yule-Walker AR Estimator block implements this autocorrelation-based method for AR model estimation, while the Yule-Walker Method block extends the method to spectral estimation.

Another common application of the Levinson-Durbin algorithm is in linear predictive coding, which is concerned with finding the coefficients of a moving average (MA) process (or FIR filter) that predicts the next value of a signal from the current signal sample and a finite number of past samples. In this case, the input to the block represents the signal's autocorrelation sequence, with r`(1)` being the zero-lag value, and the output at the block's A port contains the coefficients of the predictive MA process (in descending powers of z).

$$H(z) = A(z) = 1 + a(2)z^{-1} + \ \dots \ a(n+1)z^{-n}$$

These coefficients solve the following optimization problem:

$$\min_{\{a_i\}}$$

$$E\left[\left|x_n - \sum_{i=1}^{N} a_i x_{n-i}\right|^2\right]$$

Again, the output at the block's K port contains the corresponding reflection coefficients, `[k(1) k(2) ... k(n)]`, for the lattice realization of this FIR filter. The Autocorrelation LPC block in the Linear Prediction library implements this autocorrelation-based prediction method.

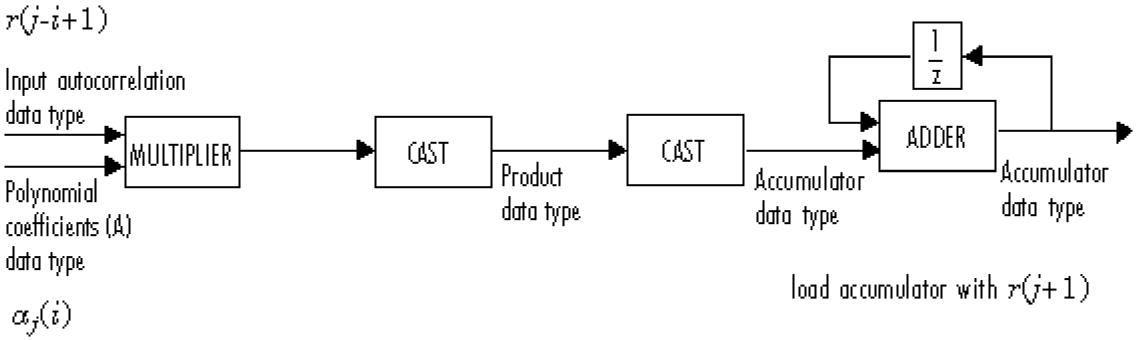
Fixed-Point Data Types

The diagrams in this section show the data types used within the Levinson-Durbin block for fixed-point signals.

After initialization the block performs n updates. At the (j+1) update,

$$\text{value in accumulator } = r(j+1) + \sum a_j(i) \times r(j-i+1)$$

The following diagram displays the fixed-point data types used in this calculation:



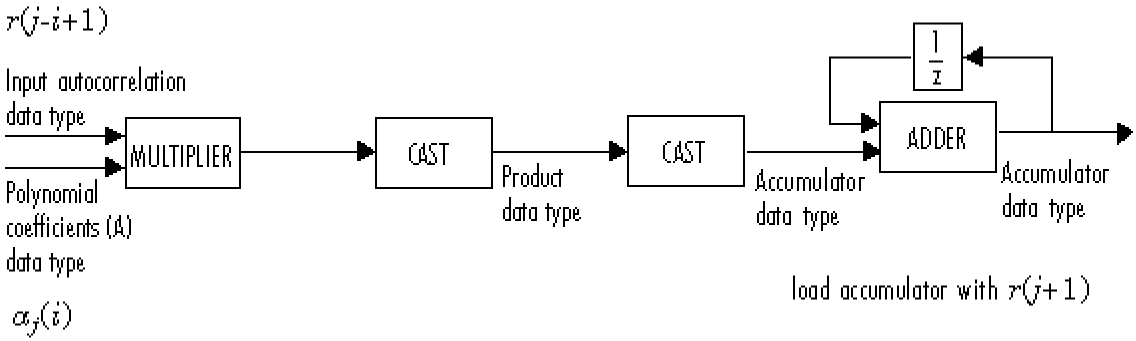
The block then updates the reflection coefficients K according to

$$K_{j+1} = \text{ value in accumulator} / P_j$$

The block then updates the prediction error power P according to

$$P_{j+1} = P_j - P_j \times K_{j+1} \times \text{conj}\left(K_{j+1}\right)$$

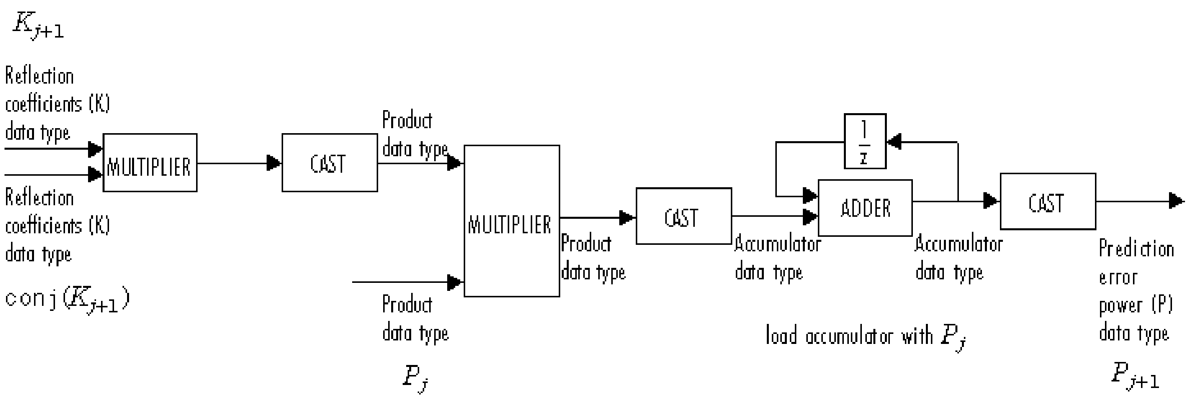
The next diagram displays the fixed-point data types used in this calculation:



The polynomial coefficients A are then updated according to

$$a_{j+1}(i) = a_j(i) + K_{j+1} \times \text{conj}\left(\mathrm{a_j}(j-1+i)\right)$$

This diagram displays the fixed-point data types used in this calculation:

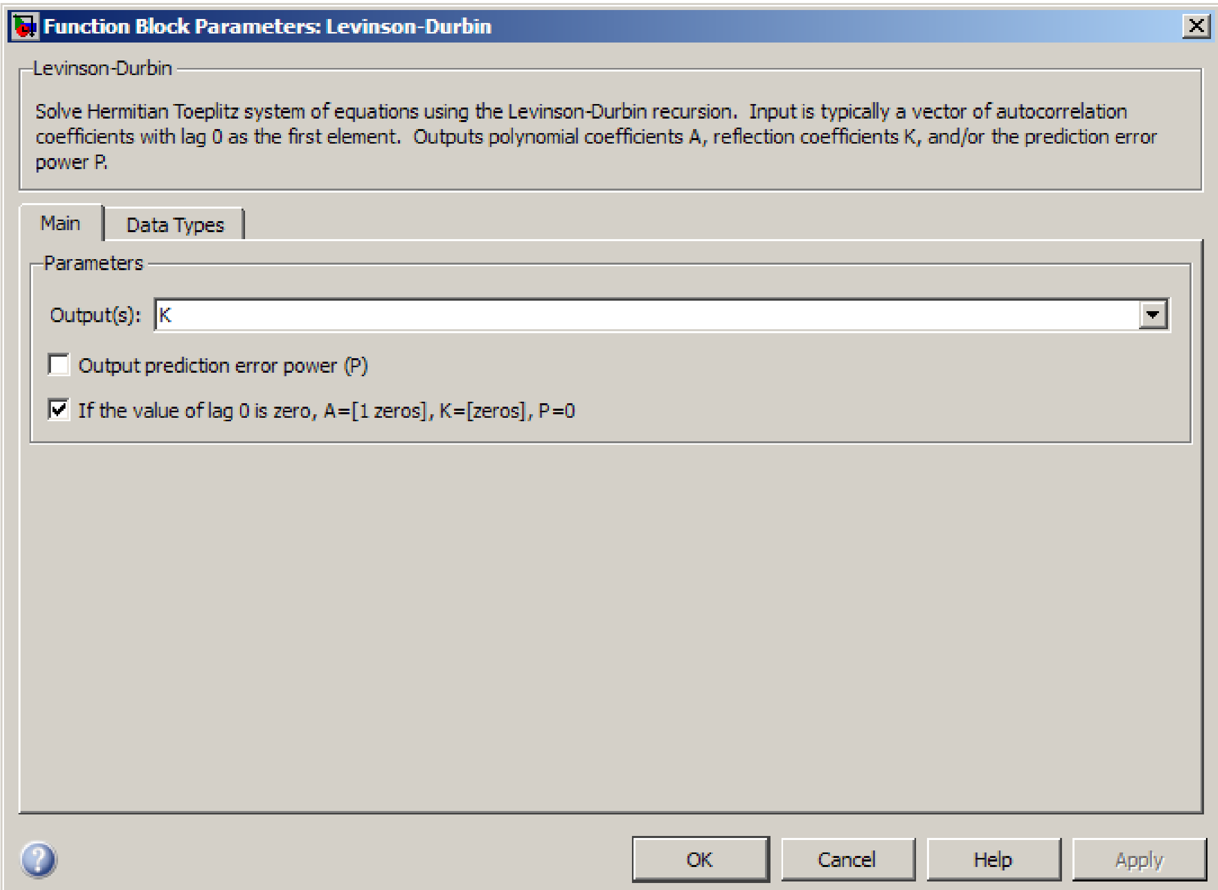


Algorithm

The algorithm requires $O(n^2)$ operations for each input channel. This implementation is therefore much more efficient for large n than standard Gaussian elimination, which requires $O(n^3)$ operations per channel.

Dialog Box

The **Main** pane of the Levinson-Durbin block dialog box appears as follows.



Output(s)

Specify the solution representation of $Ra = b$ to output: model coefficients (`A`), reflection coefficients (`K`), or both (`A and K`). For scalar and frame-based row vector inputs, this parameter must be set to `A`.

Output prediction error power (P)

Select to output the prediction error at port P.

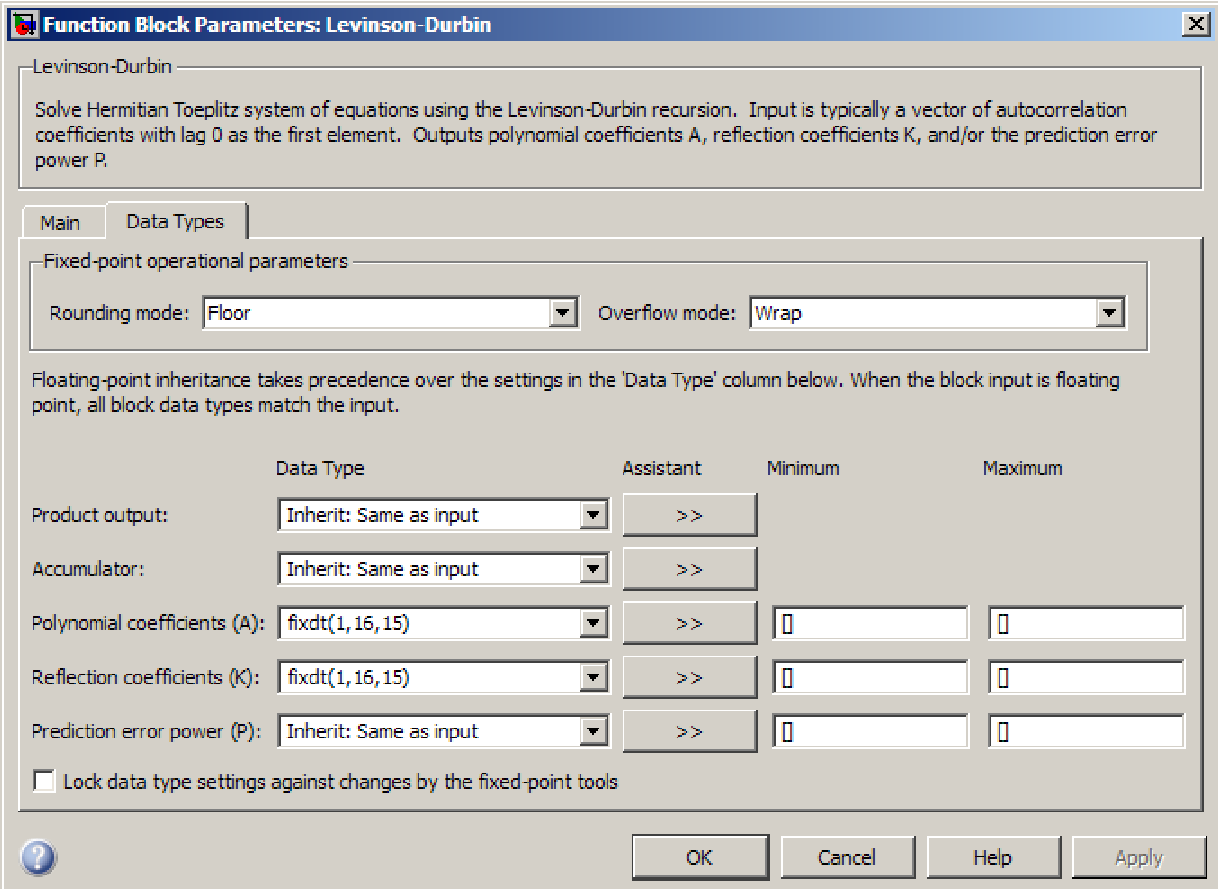
If the value of lag 0 is zero, A=[1 zeros], K=[zeros], P=0

When you select this check box and the first element of the input, `r(1)`, is zero, the block outputs the following vectors, as appropriate:

- `A = [1 zeros(1,n)]`
- `K = [zeros(1,n)]`
- `P = 0`

When you clear this check box, the block outputs a vector of `NaN`s for each channel whose `r(1)` element is zero.

The **Data Types** pane of the Levinson-Durbin block dialog box appears as follows.



Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode for fixed-point operations.

Product output data type
Specify the product output data type. See “Fixed-Point Data Types” on page 2-764 and “Multiplication Data Types” for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Accumulator data type
Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-764 for illustrations depicting the use of the accumulator data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Polynomial coefficients (A)

Specify the polynomial coefficients (*A*) data type. See "Fixed-Point Data Types" on page 2-764 for illustrations depicting the use of the A data type in this block. You can set it to an expression that evaluates to a valid data type, for example, `fixdt(1,16,15)`.

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **A** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Reflection coefficients (K)

Specify the polynomial coefficients (*A*) data type. See "Fixed-Point Data Types" on page 2-764 for illustrations depicting the use of the K data type in this block. You can set it to an expression that evaluates to a valid data type, for example, `fixdt(1,16,15)`.

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **K** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Prediction error power (P)

Specify the prediction error power (*P*) data type. See "Fixed-Point Data Types" on page 2-764 for illustrations depicting the use of the P data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **P** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Minimum

Specify the minimum values that the polynomial coefficients, reflection coefficients, or prediction error power should have. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see “Checking Parameter Values”)
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum values that the polynomial coefficients, reflection coefficients, or prediction error power should have. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Parameter range checking (see “Checking Parameter Values”)
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

References

Golub, G. H. and C. F. Van Loan. Sect. 4.7 in *Matrix Computations*. 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

Ljung, L. *System Identification: Theory for the User*. Englewood Cliffs, NJ: Prentice Hall, 1987. Pgs. 278–280.

Kay, Steven M. *Modern Spectral Estimation: Theory and Application*. Englewood Cliffs, NJ: Prentice Hall, 1988.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed only)
- 8-, 16-, and 32-bit signed integers

See Also

Cholesky Solver	Signal Processing Blockset
LDL Solver	Signal Processing Blockset
Autocorrelation LPC	Signal Processing Blockset
LU Solver	Signal Processing Blockset
QR Solver	Signal Processing Blockset
Yule-Walker AR Estimator	Signal Processing Blockset
Yule-Walker Method	Signal Processing Blockset
`levinson`	Signal Processing Toolbox

See "Linear System Solvers" for related information.

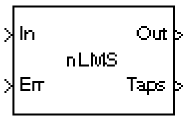
Purpose

Compute filter estimates for input using LMS adaptive filter algorithm

Library

`dspobslib`

Description



Note The LMS Adaptive Filter block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the LMS Filter block.

The LMS Adaptive Filter block implements an adaptive FIR filter using the stochastic gradient algorithm known as the normalized least mean-square (LMS) algorithm.

$$y(n) = \hat{w}^H(n-1)u(n)$$

$$e(n) = d(n) - y(n)$$

$$\hat{w}(n) = \hat{w}(n-1) + \frac{u(n)}{a + u^H(n)u(n)}\mu e^*(n)$$

The variables are as follows.

Variable	Description
n	The current algorithm iteration
$u(n)$	The buffered input samples at step n
$\hat{w}(n)$	The vector of filter-tap estimates at step n
$y(n)$	The filtered output at step n
$e(n)$	The estimation error at step n
$d(n)$	The desired response at step n
μ	The adaptation step size

To overcome potential numerical instability in the tap-weight update, a small positive constant (a = 1e-10) has been added in the denominator.

To turn off normalization, clear the **Use normalization** check box in the parameter dialog. The block then computes the filter-tap estimate as

$$\hat{w}(n) = \hat{w}(n-1) + u(n)\mu e^*(n)$$

The block icon has port labels corresponding to the inputs and outputs of the LMS algorithm. Note that inputs to the In and Err ports must be sample-based scalars. The signal at the Out port is a scalar, while the signal at the Taps port is a sample-based vector.

Block Ports	Corresponding Variables
`In`	u, the scalar input, which is internally buffered into the vector $u(n)$
`Out`	$y(n)$, the filtered scalar output
`Err`	$e(n)$, the scalar estimation error
`Taps`	$\hat{w}(n)$, the vector of filter-tap estimates

An optional `Adapt` input port is added when you select the **Adapt input** check box in the dialog. When this port is enabled, the block continuously adapts the filter coefficients while the `Adapt` input is nonzero. A zero-valued input to the Adapt port causes the block to stop adapting, and to hold the filter coefficients at their current values until the next nonzero `Adapt` input.

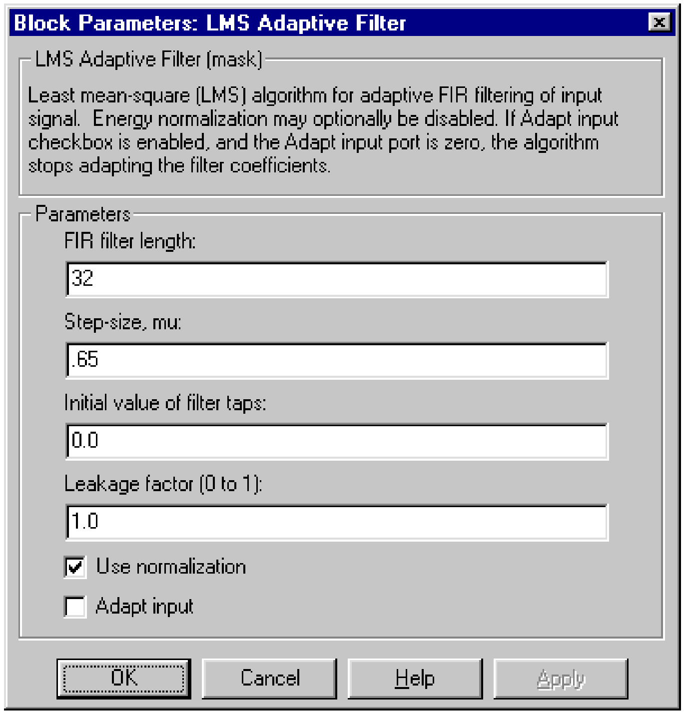
The **FIR filter length** parameter specifies the length of the filter that the LMS algorithm estimates. The **Step size** parameter corresponds to μ in the equations. Typically, for convergence in the mean square, μ must be greater than 0 and less than 2. The **Initial value of filter taps** specifies the initial value $\hat{w}(0)$ as a vector, or as a scalar to be repeated for all vector elements. The **Leakage factor** specifies the value of the leakage factor, 1 –μ α, in the leaky LMS algorithm below. This parameter must be between 0 and 1.

$$\hat{w}(n+1) = (1-\mu\alpha)\hat{w}(n) + \frac{u(n)}{u^H(n)u(n)}\mu e^*(n)$$

Examples

See the `lmsadeq` and `lmsadtde` demos.

Dialog Box



FIR filter length
The length of the FIR filter.

Step-size
The step-size, usually in the range (0, 2). Tunable.

Initial value of filter taps
The initial FIR filter coefficients.

Leakage factor
The leakage factor, in the range [0, 1]. Tunable.

Use normalization
Select this check box to compute the filter-tap estimate using the normalized equations.

Adapt input
Enables the Adapt port when selected.

References

Haykin, S. *Adaptive Filter Theory*. 3rd ed. Englewood Cliffs, NJ: Prentice Hall, 1996.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Kalman Adaptive Filter (Obsolete)	Signal Processing Blockset
RLS Adaptive Filter (Obsolete)	Signal Processing Blockset

See "Adaptive Filters" for related information.

LMS Filter

Purpose

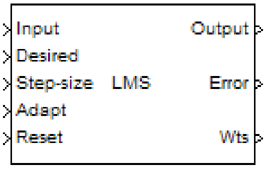
Compute output, error, and weights using LMS adaptive algorithm

Library

Filtering / Adaptive Filters

`dspadpt3`

Description



The LMS Filter block can implement an adaptive FIR filter using five different algorithms. The block estimates the filter weights, or coefficients, needed to minimize the error, *e(n)*, between the output signal *y(n)* and the desired signal, *d(n)*. Connect the signal you want to filter to the Input port. This input signal can be a sample-based scalar or a single-channel frame-based signal. Connect the desired signal to the Desired port. The desired signal must have the same data type, frame status, complexity, and dimensions as the input signal. The Output port outputs the filtered input signal, which is the estimate of the desired signal. The output of the Output port has the same frame status as the input signal. The Error port outputs the result of subtracting the output signal from the desired signal.

When you select `LMS` for the **Algorithm** parameter, the block calculates the filter weights using the least mean-square (LMS) algorithm. This algorithm is defined by the following equations.

$$y(n) = \mathbf{w}^T(n-1)\mathbf{u}(n)$$
$$e(n) = d(n) - y(n)$$
$$\mathbf{w}(n) = \mathbf{w}(n-1) + f(\mathbf{u}(n), e(n), \mu)$$

The weight update function for the LMS adaptive filter algorithm is defined as

$$f(\mathbf{u}(n), e(n), \mu) = \mu e(n)\mathbf{u}^*(n)$$

The variables are as follows.

Variable	Description
n	The current time index
$\mathbf{u}(n)$	The vector of buffered input samples at step n
$\mathbf{u^*}(n)$	The complex conjugate of the vector of buffered input samples at step n
$\mathbf{w}(n)$	The vector of filter weight estimates at step n
$y(n)$	The filtered output at step n
$e(n)$	The estimation error at step n
$d(n)$	The desired response at step n
μ	The adaptation step size

When you select `Normalized LMS` for the **Algorithm** parameter, the block calculates the filter weights using the normalized LMS algorithm. The weight update function for the normalized LMS algorithm is defined as

$$f(\mathbf{u}(n), e(n), \mu) = \mu e(n)\frac{\mathbf{u}^*(n)}{\varepsilon + \mathbf{u}^H(n)\mathbf{u}(n)}$$

To overcome potential numerical instability in the update of the weights, a small positive constant, epsilon, has been added in the denominator. For double-precision floating-point input, epsilon is 2.2204460492503131e-016. For single-precision floating-point input, epsilon is 1.192092896e-07. For fixed-point input, epsilon is 0.

When you select `Sign-Error LMS` for the **Algorithm** parameter, the block calculates the filter weights using the LMS algorithm equations.

However, each time the block updates the weights, it replaces the error term *e(n)* with `+1` when the error term is positive, `-1` when it is negative, or `0` when it is zero.

When you select `Sign-Data LMS` for the **Algorithm** parameter, the block calculates the filter weights using the LMS algorithm equations. However, each time the block updates the weights, it replaces each sample of the input vector $\boldsymbol{u}$ (n) with `+1` when the input sample is positive, `-1` when it is negative, or `0` when it is zero.

When you select `Sign-Sign LMS` for the **Algorithm** parameter, the block calculates the filter weights using the LMS algorithm equations. However, each time the block updates the weights, it replaces the error term *e(n)* with `+1` when the error term is positive, `-1` when it is negative, or `0` when it is zero. It also replaces each sample of the input vector $\boldsymbol{u}$ (n) with `+1` when the input sample is positive, `-1` when it is negative, or `0` when it is zero.

Use the **Filter length** parameter to specify the length of the filter weights vector.

The **Step size (mu)** parameter corresponds to μ in the equations. For convergence of the normalized LMS equations, $0<\mu<2$. You can either specify a step size using the input port, Step-size, or by entering a value in the Block Parameters: LMS Filter dialog.

Use the **Leakage factor (0 to 1)** parameter to specify the leakage factor $1 - \mu\alpha$ where $0 < 1 - \mu\alpha \leq 1$ in the leaky LMS algorithm shown below.

$$\mathbf{w}(n) = (1 - \mu\alpha)\mathbf{w}(n-1) + f\left(\mathbf{u}(n), e(n), \mu\right)$$

When you select `LMS` from the **Algorithm** list, the weight update function in the above equation is the LMS weight update function. When you select `Normalized LMS` from the **Algorithm** list, the weight update function in the above equation is the normalized LMS weight update function.

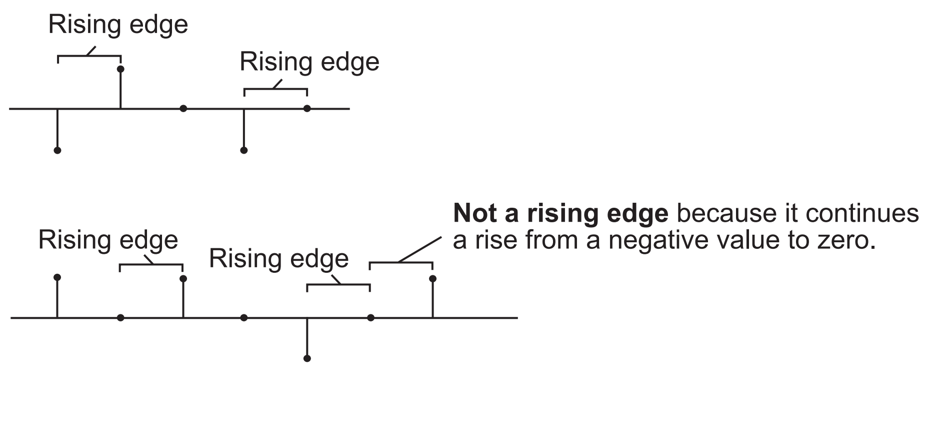
Enter the initial filter weights $\mathbf{w}(0)$ as a vector or a scalar in the **Initial value of filter weights** text box. When you enter a scalar, the block uses the scalar value to create a vector of filter weights. This vector has length equal to the filter length and all of its values are equal to the scalar value.

When you select the **Adapt port** check box, an Adapt port appears on the block. When the input to this port is greater than zero, the block continuously updates the filter weights. When the input to this port is less than or equal to zero, the filter weights remain at their current values.

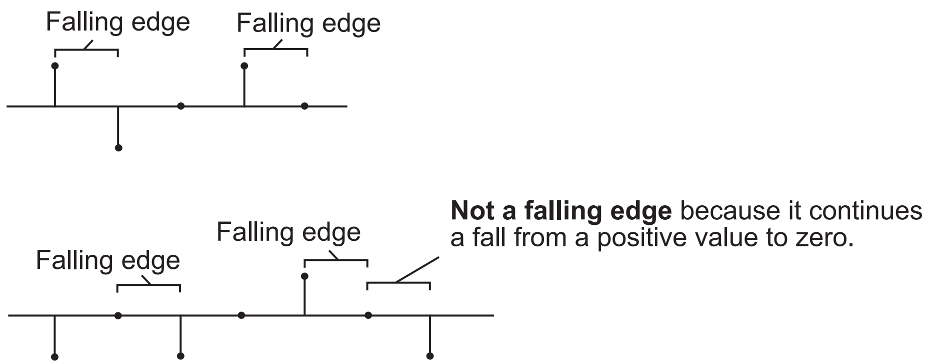
When you want to reset the value of the filter weights to their initial values, use the **Reset port** parameter. The block resets the filter weights whenever a reset event is detected at the Reset port. The reset signal rate must be the same rate as the data signal input.

From the **Reset port** list, select `None` to disable the Reset port. To enable the Reset port, select one of the following from the **Reset port** list:

- `Rising edge` — Triggers a reset operation when the Reset input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)



- `Falling edge` — Triggers a reset operation when the Reset input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)



- `Either edge` — Triggers a reset operation when the Reset input is a `Rising edge` or `Falling edge` (as described above)
- `Non-zero sample` — Triggers a reset operation at each sample time that the Reset input is not zero

Select the **Output filter weights** check box to create a Wts port on the block. For each iteration, the block outputs the current updated filter weights from this port.

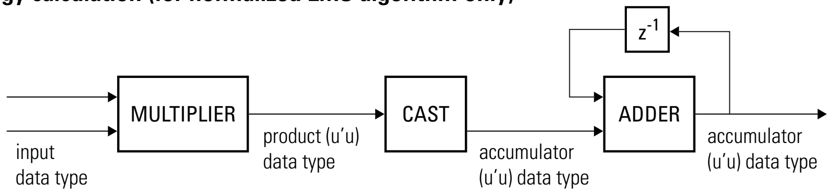
Fixed-Point Data Types

The following diagrams show the data types used within the LMS Filter block for fixed-point signals; the table summarizes the definitions of variables used in the diagrams:

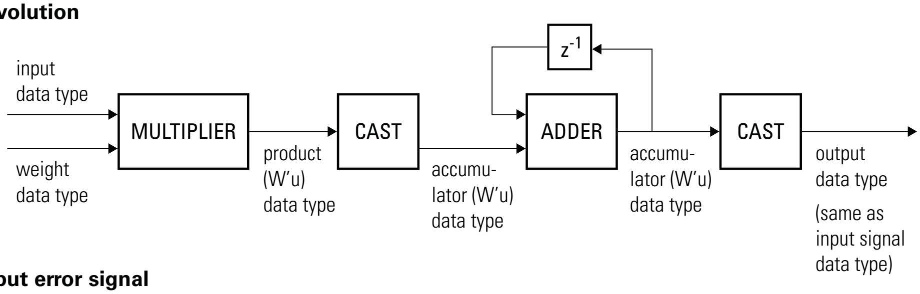
Variable	Definition
u	Input vector
W	Vector of filter weights
μ	Step size
e	Error
Q	Quotient, $Q = \frac{\mu \cdot e}{\mathbf{u'u}}$
Product **u'u**	Product data type in Energy calculation diagram
Accumulator **u'u**	Accumulator data type in Energy calculation diagram
Product **W'u**	Product data type in Convolution diagram
Accumulator **W'u**	Accumulator data type in Convolution diagram
Product $\mu \cdot e$	Product data type in Product of step size and error diagram
Product $Q \cdot \mathbf{u}$	Product and accumulator data type in Weight update diagram. [1]

[1]The accumulator data type for this quantity is automatically set to be the same as the product data type. The minimum, maximum, and overflow information for this accumulator is logged as part of the product information. Autoscaling treats this product and accumulator as one data type.

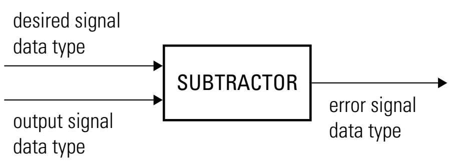
Energy calculation (for normalized LMS algorithm only)



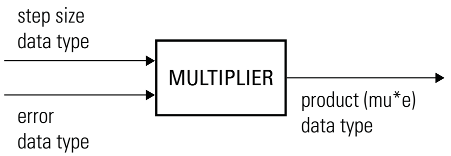
Convolution



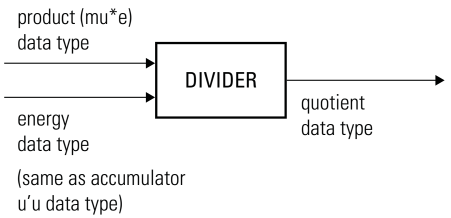
Output error signal



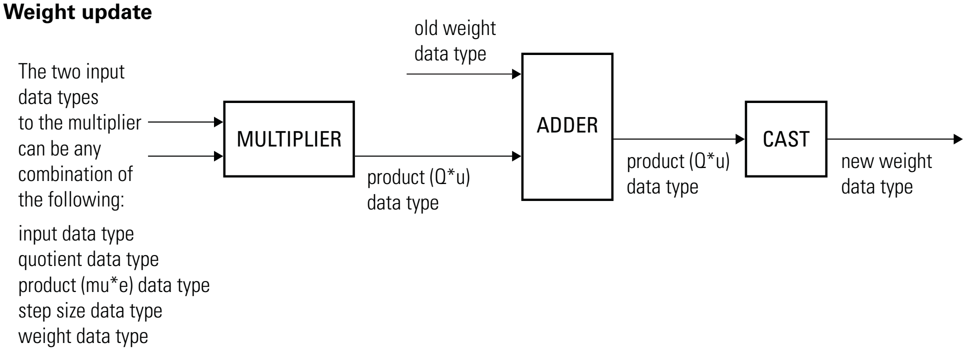
Product of step size and error (for LMS and Sign-Data LMS algorithms only)



Quotient (for normalized LMS only)



Weight update



You can set the data type of the parameters, weights, products, quotient, and accumulators in the block mask. Fixed-point inputs, outputs, and mask parameters of this block must have the following characteristics:

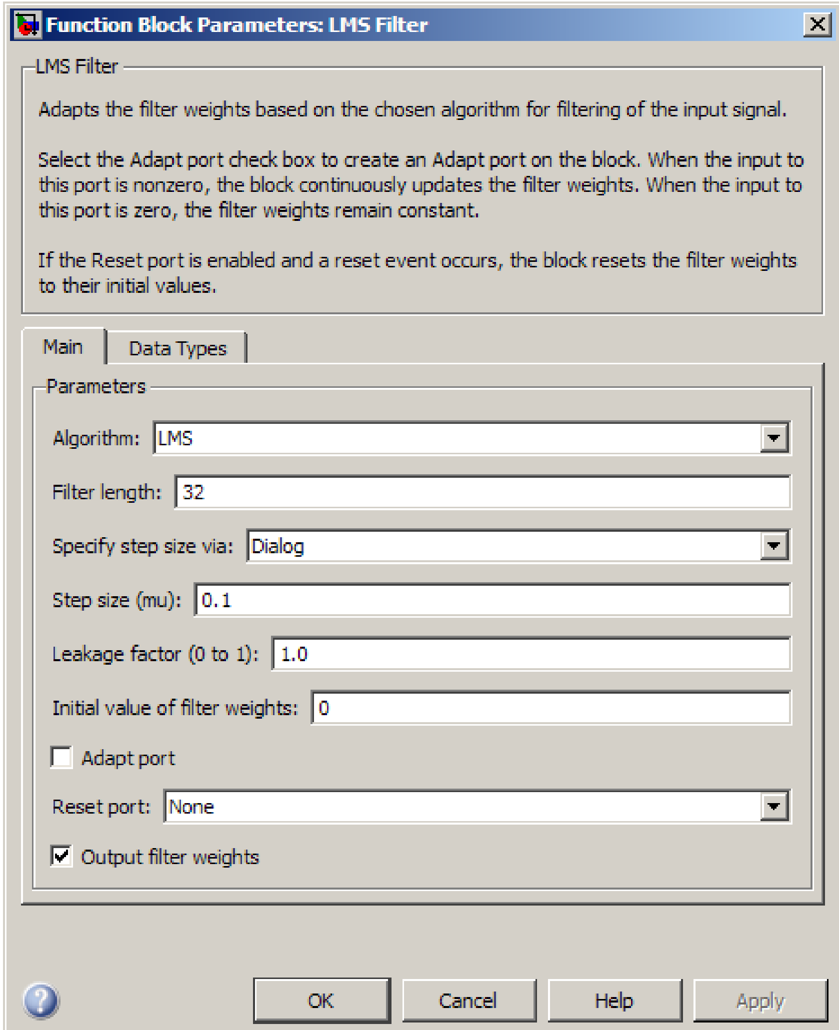
- The input signal and the desired signal must have the same word length, but their fraction lengths can differ.

- The step size and leakage factor must have the same word length, but their fraction lengths can differ.
- The output signal and the error signal have the same word length and the same fraction length as the desired signal.
- The quotient and the product output of the **u'u**, **W'u**, $\mu \cdot e$, and $Q \cdot \mathbf{u}$ operations must have the same word length, but their fraction lengths can differ.
- The accumulator data type of the **u'u** and **W'u** operations must have the same word length, but their fraction lengths can differ.

The output of the multiplier is in the product output data type if at least one of the inputs to the multiplier is real. If both of the inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see "Multiplication Data Types".

Dialog Box

The **Main** pane of the LMS Filter block dialog appears as follows.



Algorithm
Choose the algorithm used to calculate the filter weights.

Filter length
Enter the length of the FIR filter weights vector.

Specify step size via
Select `Dialog` to enter a value for step size in the Block parameters: LMS Filter dialog. Select `Input port` to specify step size using the Step-size input port.

Step size (mu)
Enter the step size μ. Tunable.

Leakage factor (0 to 1)
Enter the leakage factor, $0 < 1 - \mu\alpha \leq 1$. Tunable.

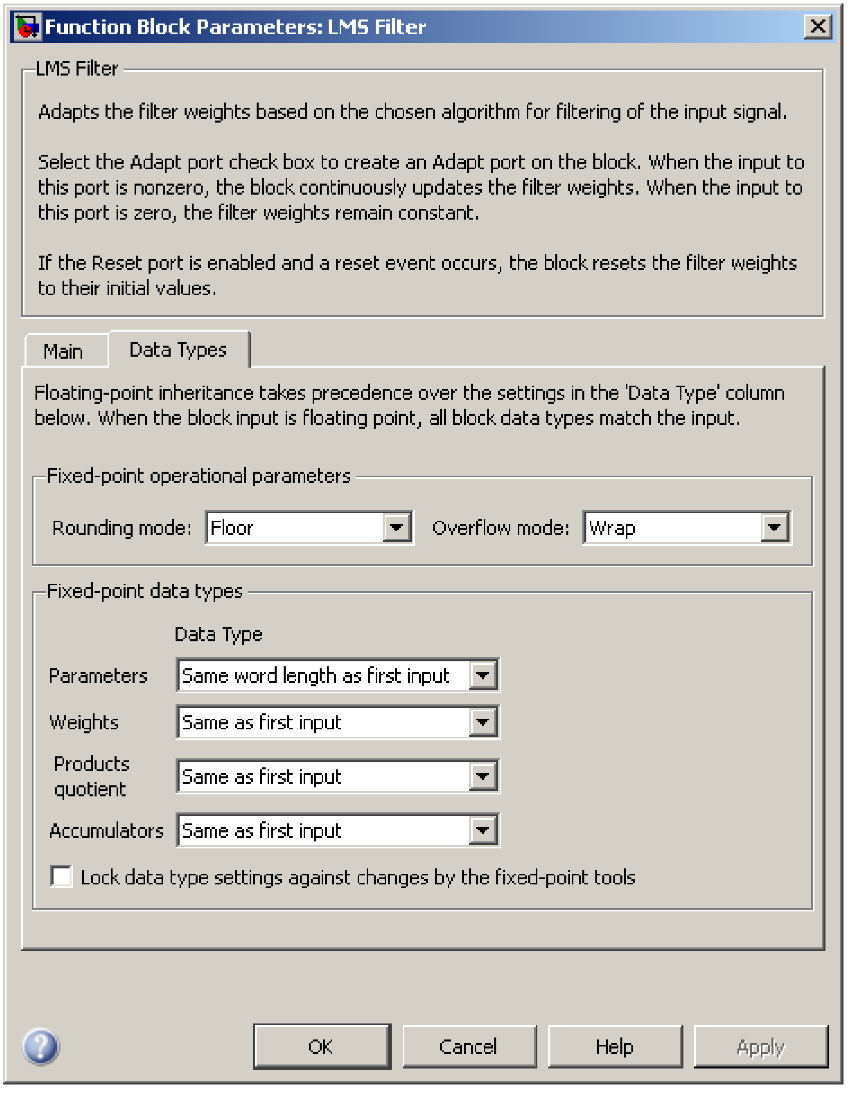
Initial value of filter weights
Specify the initial values of the FIR filter weights.

Adapt port
Select this check box to enable the Adapt input port.

Reset port
Select this check box to enable the Reset input port.

Output filter weights
Select this check box to export the filter weights from the Wts port.

The **Data Types** pane of the LMS Filter block dialog appears as follows.



Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode for fixed-point operations.

Parameters

This parameter is visible if, for the **Specify step size via** parameter, you choose `Dialog`. Choose how you specify the word length and the fraction length of the leakage factor and step size:

- When you select `Same word length as first input`, the word length of the leakage factor and step size match that of the first input to the block. In this mode, the fraction length of the leakage factor and step size is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.
- When you select `Specify word length`, you can enter the word length of the leakage factor and step size, in bits. In this mode, the fraction length of the leakage factor and step size is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the leakage factor and step size, in bits. The leakage factor and the step size must have the same word length, but the fraction lengths can differ.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the leakage factor and step size. The leakage factor and the step size must have the same word length, but the slopes can differ. This block requires a power-of-two slope and a bias of zero.

If, for the **Specify step size via** parameter, you choose `Input port`, the word length of the leakage factor is the same as the word length of the step size input at the Step size port. The fraction length of the leakage factor is automatically set to the best precision possible based on the word length of the leakage factor.

Weights

Choose how you specify the word length and fraction length of the filter weights of the block:

- When you select `Same as first input`, the word length and fraction length of the filter weights match those of the first input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the filter weights, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the filter weights. This block requires a power-of-two slope and a bias of zero.

Products & quotient

Choose how you specify the word length and fraction length of **u'u**, **W'u**, $\mu \cdot e$, $Q \cdot \mathbf{u}$, and the quotient, Q. Here, **u** is the input vector, **W** is the vector of filter weights, μ is the step size, e is the error, and Q is the quotient, which is defined as $Q = \frac{\mu \cdot e}{\mathbf{u'u}}$

- When you select `Same as first input`, the word length and fraction length of these quantities match those of the first input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of these quantities, in bits. The word length of the quantities must be the same, but the fraction lengths can differ.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of these quantities. The word length of the quantities must be the same, but the slopes can differ. This block requires a power-of-two slope and a bias of zero.

Accumulators

Use this parameter to specify how you would like to designate the word and fraction lengths of the accumulators for the **u'u** and **W'u** operations.

Note This parameter is *not* used to designate the word and fraction lengths of the accumulator for the $Q \cdot \mathbf{u}$ operation. The accumulator data type for this quantity is automatically set to be the same as the product data type. The minimum, maximum, and overflow information for this accumulator is logged as part of the product information. Autoscaling treats this product and accumulator as one data type.

See "Fixed-Point Data Types" on page 2-782 and "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block:

- When you select `Same as first input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulators, in bits. The word length of both the accumulators must be the same, but the fraction lengths can differ.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulators. The word length of both the accumulators must be the same, but the slopes can differ. This block requires a power-of-two slope and a bias of zero.

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

References

Hayes, M.H. *Statistical Digital Signal Processing and Modeling.* New York: John Wiley & Sons, 1996.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Signed fixed point
Desired	• Must be the same as Input for floating-point signals • Must be any signed fixed-point data type when Input is fixed point
Step-size	• Must be the same as Input for floating-point signals • Must be any signed fixed-point data type when Input is fixed point
Adapt	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers
Reset	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers
Output	• Must be the same as Input for floating-point signals • Must be the same as Desired for fixed-point signals

Port	Supported Data Types
Error	• Must be the same as Input for floating-point signals • Must be the same as Desired for fixed-point signals
Wts	• Must be the same as Input for floating-point signals • Obeys the **Weights** parameter for fixed-point signals

See Also

Kalman Adaptive Filter (Obsolete)	Signal Processing Blockset
RLS Filter	Signal Processing Blockset
Block LMS Filter	Signal Processing Blockset
Fast Block LMS Filter	Signal Processing Blockset

See "Adaptive Filters" for related information.

Purpose

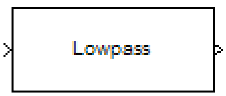
Design lowpass Filter

Library

Filtering / Filter Designs

`dspfdesign`

Description



This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. Without a Filter Design Toolbox license, you can run models that contain this block, and can edit some, but not all, block parameters. To enable the full filter design functionality of this block, you must have a Filter Design Toolbox license.

Dialog Box

See “Lowpass Filter Design Dialog Box — Main Pane” in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Purpose

Convert linear prediction coefficients to line spectral pairs or line spectral frequencies

Library

Estimation / Linear Prediction

`dsplp`

Description

The LPC to LSF/LSP Conversion block takes a vector or matrix of linear prediction coefficients (LPCs) and converts it to a vector or matrix of line spectral pairs (LSPs) or line spectral frequencies (LSFs). When converting LPCs to LSFs, the block outputs match those of the `poly2lsf` function.

The block input must be a sample-based row vector, which is treated as a single channel, or a matrix, which is treated as a single channel per column.

The input LPCs for each channel, 1, a_1, a_2, ..., a_m, must be the denominator of the transfer function of a stable all-pole filter with the form given in the first equation of "Requirements for Valid Outputs" on page 2-797. A length-M+1 input channel yields a length-M output channel. Inputs can be sample based or frame based, but outputs are always sample based.

See other sections of this reference page to learn about how to ensure that you get valid outputs, how to detect invalid outputs, how the block computes the LSF/LSP values, and more.

Requirements for Valid Outputs

To get valid outputs, your inputs and the **Root finding coarse grid points** parameter value must meet these requirements:

- The input LPCs for each channel, 1, a_1, a_2, ..., a_m, must come from the denominator of the following transfer function, $H(z)$, of a stable all-pole filter (all roots of $H(z)$ must be inside the unit circle). Note that the first term in $H(z)$'s denominator must be 1. When the input LPCs do not come from a transfer function of the following form, the block outputs are invalid.

$$H(z) = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2} + \ \dots \ + a_m z^{-m}}$$

- The **Root finding coarse grid points** parameter value must be large enough so that the block can find all the LSP or LSF values. (The output LSFs and LSPs are roots of polynomials related to the input LPC polynomial; the block looks for these roots to produce the output. For details, see "LSF and LSP Computation Method: Chebyshev Polynomial Method for Root Finding" on page 2-805.) When you do not set **Root finding coarse grid points** to a high enough value relative to the number of LPCs, the block might not find all the LSPs or LSFs and yield invalid outputs as described in "Root Finding Method Limitations: Failure to Find Roots" on page 2-809.

To learn about recognizing invalid inputs and outputs and parameters for dealing with them, see "Handling and Recognizing Invalid Inputs and Outputs" on page 2-800.

Setting Outputs to LSFs or LSPs

Set the **Output** parameter to one of the following settings to determine whether the block outputs LSFs or LSPs:

- `LSF in radians (0 pi)` — Block outputs the LSF values between 0 and π radians in increasing order. The block does not output the guaranteed LSF values, 0 and π.
- `LSF normalized in range (0 0.5)` — Block outputs normalized LSF values in increasing order, computed by dividing the LSF values between 0 and π radians by 2π. The block does not output the guaranteed normalized LSF values, 0 and 0.5.
- `LSP in range (-1 1)` — Block outputs LSP values in decreasing order, equal to the cosine of the LSF values between 0 and π radians. The block does not output the guaranteed LSP values, -1 and 1.

Adjusting Output Computation Time and Accuracy with Root Finding Parameters

The values n and k determine the block's output computation time and accuracy, where

- n is the value of the **Root finding coarse grid points** parameter (choose this value with care; see the note below).
- k is the value of the **Root finding bisection refinement** parameter.
- Decreasing the values of n and k decreases the output computation time, but also decreases output accuracy:
  - The upper bound of block's computation time is proportional to $k \cdot (n-1)$.
  - Each LSP output is within $1/(n \cdot 2^k)$ of the actual LSP value.
  - Each LSF output is within ΔLSF of the actual LSF value, LSF_{act}, where

$$\Delta LSF = \left| a\cos(LSF_{act}) - a\cos\left(LSF_{act} + 1/\left(n \cdot 2^k\right)\right)\right|$$

Note When the value of the **Root finding coarse grid points** parameter is too small relative to the number of LPCs, the block might output invalid data as described in "Requirements for Valid Outputs" on page 2-797. Also see "Handling and Recognizing Invalid Inputs and Outputs" on page 2-800.

Notable Input and Output Properties

- To get valid outputs, your input LPCs and the value of the **Root finding coarse grid points** parameter must meet the requirements described in "Requirements for Valid Outputs" on page 2-797.
- Length-L+1 input channel yields length-L output channel

- Output is always sample based
- **Output** parameter determines the output type (see "Setting Outputs to LSFs or LSPs" on page 2-798):
  - LSFs — frequencies, w_k, where $0 < w_k < \pi$ and $w_k < w_{k+1}$
  - Normalized LSFs — $w_k / 2\pi$
  - LSPs — $\cos(w_k)$

Handling and Recognizing Invalid Inputs and Outputs

The block outputs invalid data when your input LPCs and the value of the **Root finding coarse grid points** parameter do not meet the requirements described in "Requirements for Valid Outputs" on page 2-797. The following topics describe what invalid outputs look like, and how to set the block parameters provided for handling invalid inputs and outputs:

- "What Invalid Outputs Look Like" on page 2-800
- "Parameters for Handling Invalid Inputs and Outputs" on page 2-801

What Invalid Outputs Look Like

The channels of an invalid output have the same dimensions, sizes, and frame statues as the channels of a valid output. However, invalid output channels do not contain all the LSP or LSF values. Instead, they contain none or some of the LSP and LSF values and the rest of the output is filled with place holder values (-1, 0.5, or π) depending on the **Output** parameter setting).

In short, all invalid outputs in a channel end in one of the place holder values (-1, 0.5, or π) as illustrated in the following table. To learn how to use the block's parameters for handling invalid inputs and outputs, see the next section.

Output Parameter Setting	Place Holder	Sample Invalid Outputs
LSF in radians (0 pi)	π	$\begin{bmatrix} w_1 & w_2 & w_3 & \pi & \pi & \pi & \pi & \pi \end{bmatrix}$
LSF normalized in range (0 0.5)	0.5	$\begin{bmatrix} w_1 \\ w_2 \\ 0.5 \end{bmatrix}$
LSP in range (-1 1)	-1	$\begin{bmatrix} \cos(w_{13}) \\ \cos(w_{23}) \\ -1 \\ -1 \\ -1 \end{bmatrix}$

Parameters for Handling Invalid Inputs and Outputs

You must set how the block handles invalid inputs and outputs by setting these parameters:

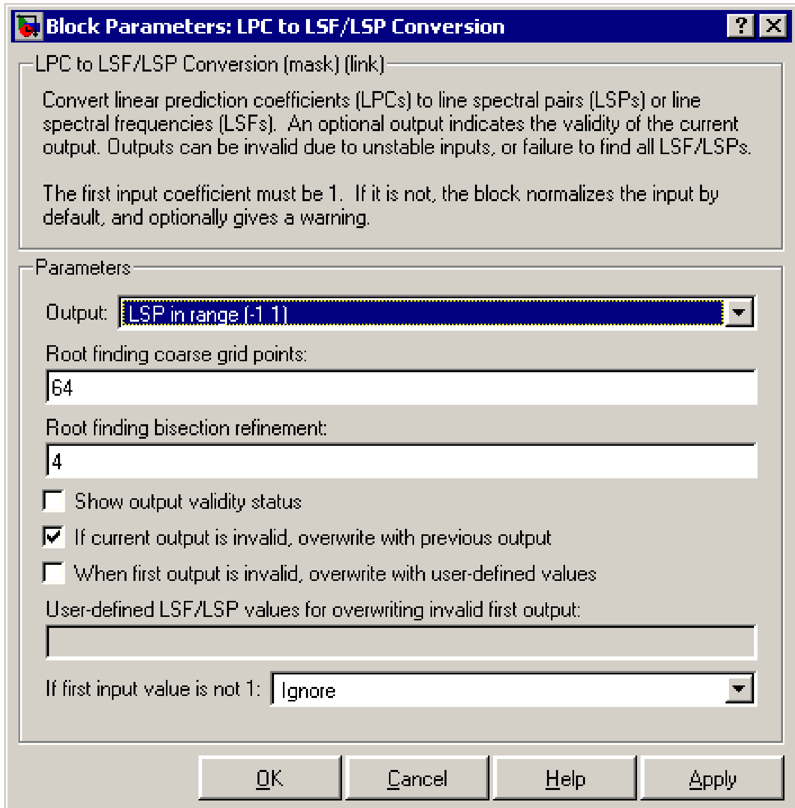
- **Show output validity status (1=valid, 0=invalid)** — Set this parameter to activate a second output port that outputs a vector with one Boolean element per channel; 1 when the output of the corresponding channel is valid, and 0 when the output is invalid. The LSF and LSP outputs are invalid when the block fails to find all the LSF or LSP values or when the input LPCs are unstable (for details, see “Requirements for Valid Outputs” on page 2-797). See the previous section to learn how to recognize invalid outputs.
- **If current output is invalid, overwrite with previous output** — Select this check box to cause the block to overwrite invalid outputs with the previous output. When you set this parameter you also need to consider these parameters:

- **When first output is invalid, overwrite with user-defined values** — When the first input is unstable, you can overwrite the invalid first output with either
  - The default values, by clearing this check box
  - Values you specify, by selecting this check box

  The default initial overwrite values are the LSF or LSP representations of an all-pass filter. The vector that is used to overwrite invalid output is stored as an internal state.
- **User-defined LSP/LSF values for overwriting invalid first output** — Specify a vector of values for overwriting an invalid first output if you selected the **When first output is invalid, overwrite with user-defined values** parameter. For multichannel inputs, provide a matrix with the same number of channels as the input, or one vector that will be applied to every channel. The vector or matrix of LSP/LSF values you specify should have the same dimension, size, and frame status as the other outputs.

• **If first input value is not 1** — The block output in any channel is invalid when the first coefficient in an LPC vector is not 1; this parameter determines what the block does when given such inputs:
  - `Ignore` — Proceed with computations as if the first coefficient is 1.
  - `Normalize` — Divide the input LPCs by the value of the first coefficient before computing the output.
  - `Normalize and warn` — In addition to `Normalize`, display a warning message at the MATLAB command line.
  - `Error` — Stop the simulation and display an error message at the MATLAB command line.

Dialog Box



Output

Specifies whether to convert the input linear prediction polynomial coefficients (LPCs) to `LSP in range (-1 1)`, `LSF in radians (0 pi)`, or `LSF normalized in range (0 0.5)`. See “Setting Outputs to LSFs or LSPs” on page 2-798 for descriptions of the three settings.

Root finding coarse grid points

The value *n*, where the block divides the interval (-1, 1) into *n* subintervals of equal length, and looks for roots (LSP values) in each subinterval. You must pick *n* large enough or the block output might be invalid as described in "Requirements for Valid Outputs" on page 2-797. To learn how the block uses this parameter to compute the output, see "LSF and LSP Computation Method: Chebyshev Polynomial Method for Root Finding" on page 2-805. Also see "Adjusting Output Computation Time and Accuracy with Root Finding Parameters" on page 2-799. Tunable.

Root finding bisection refinement

The value k, where each LSP output is within $1/(n \cdot 2^k)$ of the actual LSP value, where n is the value of the **Root finding coarse grid points** parameter. To learn how the block uses this parameter to compute the output, see "LSF and LSP Computation Method: Chebyshev Polynomial Method for Root Finding" on page 2-805. Also see "Adjusting Output Computation Time and Accuracy with Root Finding Parameters" on page 2-799. Tunable.

Show output validity status

Set this parameter to activate a second output port that outputs a vector with one Boolean element per channel; `1` when the output of the corresponding channel is valid, and `0` when the output is invalid. The LSF and LSP outputs are invalid when the block fails to find all the LSF or LSP values or when the input LPCs are unstable (for details, see "Requirements for Valid Outputs" on page 2-797).

If current output is invalid, overwrite with previous output

Selecting this check box causes the block to overwrite invalid outputs with the previous output. Setting this parameter activates other parameters for taking care of initial overwrite values (when the very first output of the block is invalid). For more information, see "Parameters for Handling Invalid Inputs and Outputs" on page 2-801.

When first output is invalid, overwrite with user-defined values

When the first input is unstable, you can overwrite the invalid first output with either

- The default values, by clearing this check box

- Values you specify, by selecting this check box

  The default initial overwrite values are the LSF or LSP representations of an all-pass filter. The vector that is used to overwrite invalid output is stored as an internal state. For more information, see "Parameters for Handling Invalid Inputs and Outputs" on page 2-801.

User-defined LSP/LSF values for overwriting invalid first output

Specify a vector of values for overwriting an invalid first output if you selected the **When first output is invalid, overwrite with user-defined values** parameter. For multichannel inputs, provide a matrix with the same number of channels as the input, or one vector that will be applied to every channel. The vector or matrix of LSP/LSF values you specify should have the same dimension, size, and frame status as the other outputs.

If first input value is not 1

Determines what the block does when the first coefficient of an input is not `1`. The block can either proceed with computations as when the first coefficient is 1 (`Ignore`); divide the input LPCs by the value of the first coefficient before computing the output (`Normalize`); in addition to `Normalize`, display a warning message at the MATLAB command line (`Normalize and warn`); stop the simulation and display an error message at the MATLAB command line (`Error`). For more information, see "Parameters for Handling Invalid Inputs and Outputs" on page 2-801.

Theory

LSF and LSP Computation Method: Chebyshev Polynomial Method for Root Finding

---

Note To learn the principles on which the block's LSP and LSF computation method is based, see the reference listed in "References" on page 2-811.

To compute LSP outputs for each channel, the block relies on the fact that LSP values are the roots of two particular polynomials related to the input LPC polynomial; the block finds these roots using the Chebyshev polynomial root finding method, described next. To compute LSF outputs, the block computes the arc cosine of the LSPs, outputting values ranging from 0 to π radians.

Root Finding Method

LSPs, which are the roots of two particular polynomials, always lie in the range (-1, 1). (The guaranteed roots at 1 and -1 are factored out.) The block finds the LSPs by looking for a sign change of the two polynomials' values between points in the range (-1, 1). The block searches a maximum of $k(n-1)$ points, where

- n is the value of the **Root finding coarse grid points** parameter.
- k is the value of the **Root finding bisection refinement** parameter.

The block's method for choosing which points to check consists of the following two steps:

1. **Coarse Root Finding** —- The block divides the interval [-1, 1] into n intervals, each of length $2/n$, and checks the signs of both polynomials' values at the endpoints of the intervals. The block starts checking signs at 1, and continues checking signs at $1 - 4/n$, $1 - 6/n$, and so on at steps of length $2/n$, outputting any point if it is a root. The block stops searching in these situations:

   a. The block finds a sign change of a polynomial's values between two adjacent points. An interval containing a sign change is guaranteed to contain a root, so the block further searches the interval as described in Step 2, Root Finding Refinement.

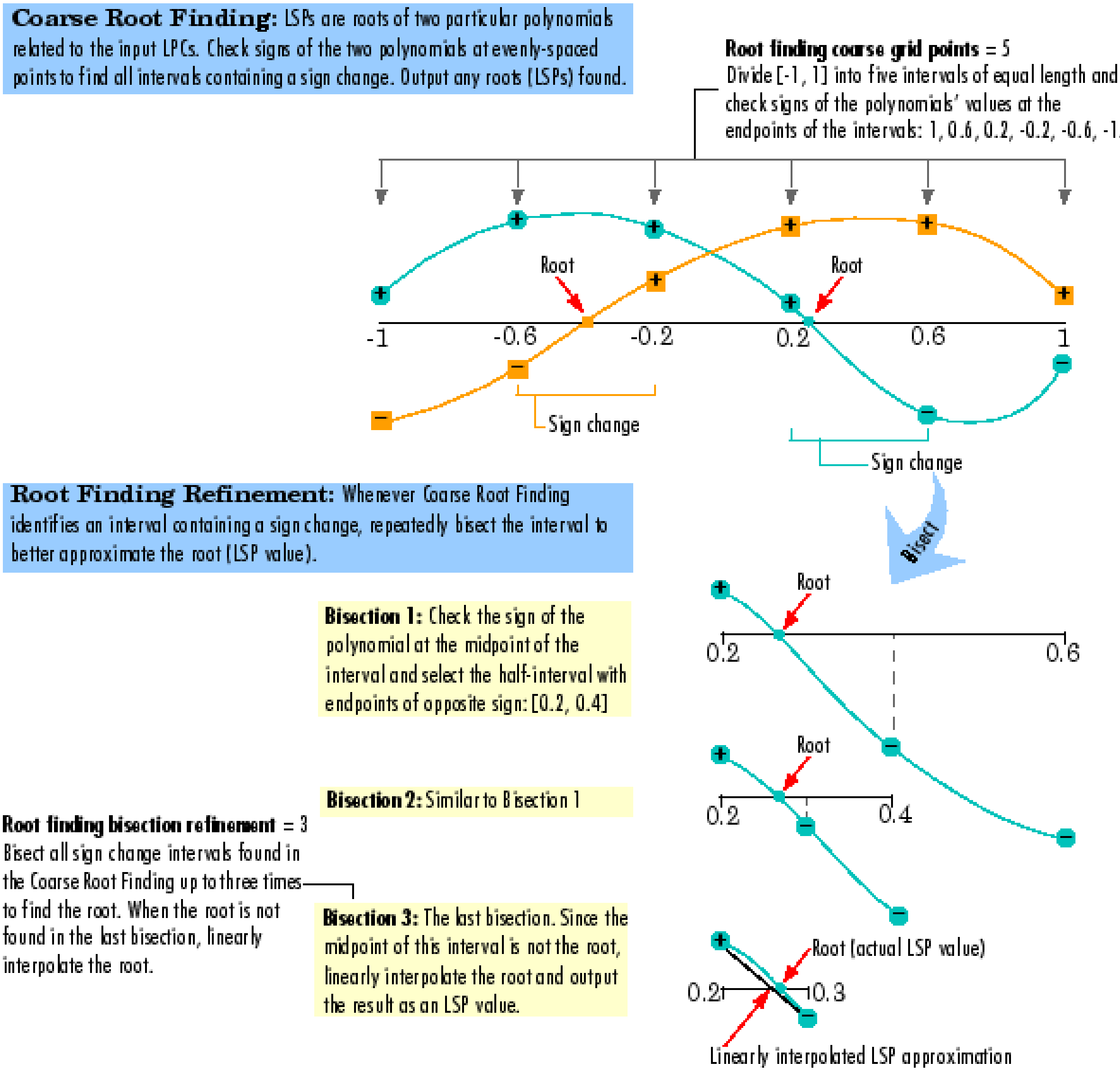
   b. The block finds and outputs all M roots (given a length-M+1 LPC input).

   c. The block fails to find all M roots and yields invalid outputs as described in "Handling and Recognizing Invalid Inputs and Outputs" on page 2-800.

2. **Root Finding Refinement** — When the block finds a sign change in an interval, [*a*, *b*], it searches for the root guaranteed to lie in the interval by following these steps:

   a. **Check if Midpoint Is a Root** — The block checks the sign of the midpoint of the interval [*a*, *b*]. The block outputs the midpoint if it is a root, and continues Step 1, Coarse Root Finding, at the next point, $a - 2/n$. Otherwise, the block selects the half-interval with endpoints of opposite sign (either $[a, (a + b)/2]$ or $[(a + b)/2, b]$) and executes Step 2b, Stop or Continue Root Finding Refinement.

   b. **Stop or Continue Root Finding Refinement** — When the block has repeated Step 2a *k* times (*k* is the value of the **Root finding bisection refinement** parameter), the block linearly interpolates the root by using the half-interval's endpoints, outputs the result as an LSP value, and returns to Step 1, Coarse Root Finding. Otherwise, the block repeats Step 2a using the half-interval.



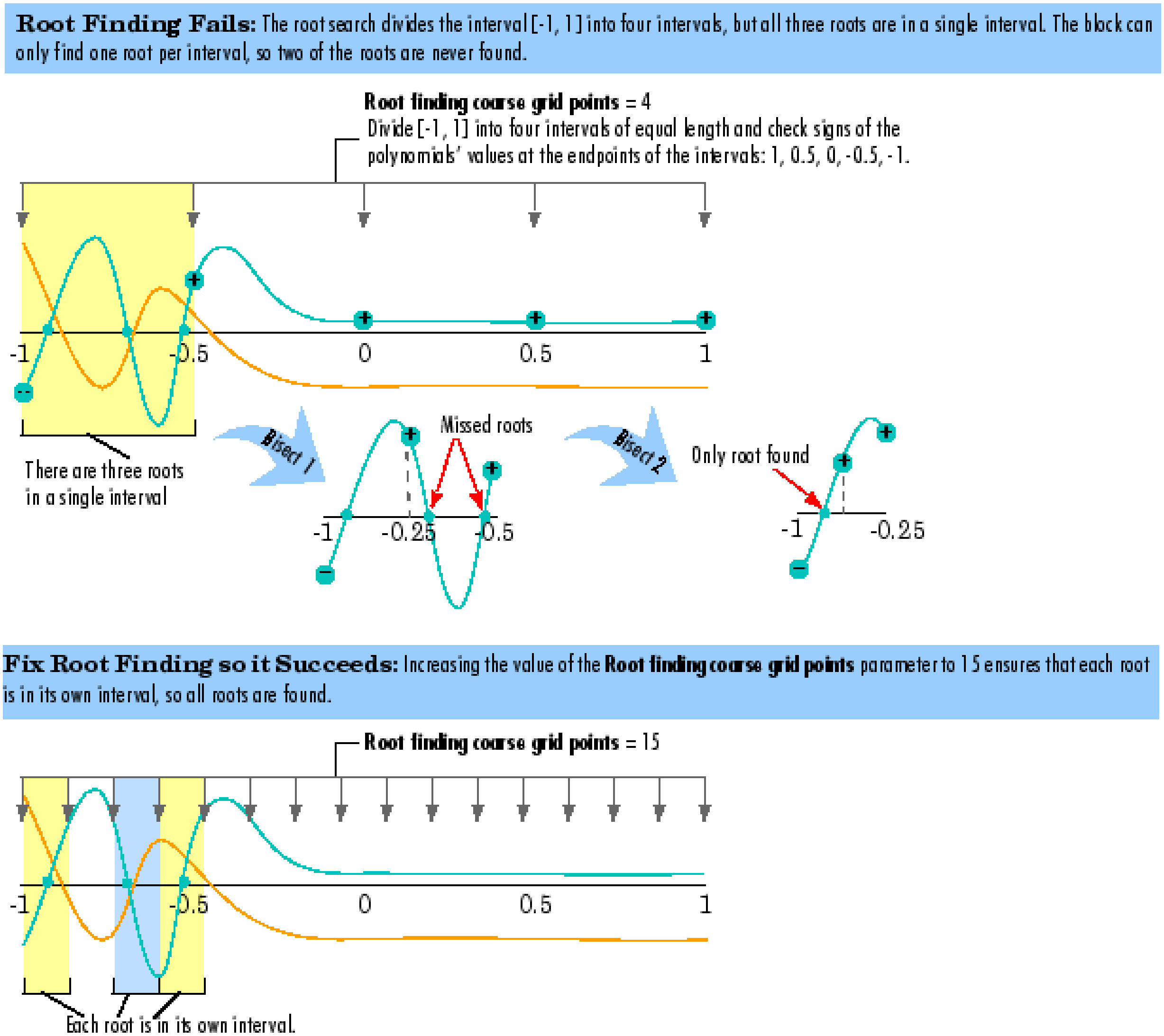
Coarse Root Finding and Root Finding Refinement

Root Finding Method Limitations: Failure to Find Roots

The block root finding method described above can fail, causing the block to produce invalid outputs (for details on invalid outputs, see "Handling and Recognizing Invalid Inputs and Outputs" on page 2-800).

In particular, the block can fail to find some roots if the value of the **Root finding coarse grid points** parameter, n, is too small. If the polynomials oscillate quickly and have roots that are very close together, the root finding might be too coarse to identify roots that are very close to each other, as illustrated in Fixing a Failed Root Finding on page 2-810.

For higher-order input LPC polynomials, you should increase the **Root finding coarse grid points** value to ensure the block finds all the roots and produces valid outputs.



Fixing a Failed Root Finding

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Boolean — Supported only by the optional output port that appears when you set the parameter, **Show output validity status (1=valid, 0=invalid)**

References

Kabal, P. and Ramachandran, R. “The Computation of Line Spectral Frequencies Using Chebyshev Polynomials.“*IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-34 No. 6, December 1986. pp. 1419-1426.

See Also

LSF/LSP to LPC Conversion	Signal Processing Blockset
LPC to/from RC	Signal Processing Blockset
LPC/RC to Autocorrelation	Signal Processing Blockset
`poly2lsf`	Signal Processing Toolbox

LSF/LSP to LPC Conversion

Purpose

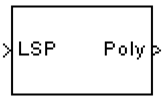
Convert line spectral frequencies or line spectral pairs to linear prediction coefficients

Library

Estimation / Linear Prediction

`dsplp`

Description

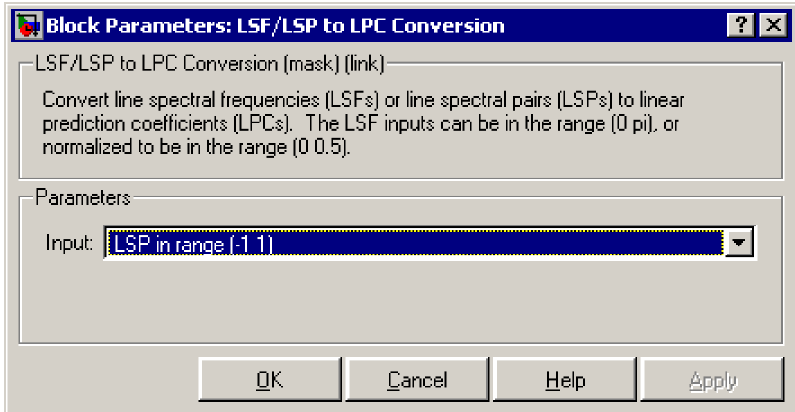


The LSF/LSP to LPC Conversion block takes a vector or matrix of line spectral pairs (LSPs) or line spectral frequencies (LSFs) and converts it to a vector or matrix of linear prediction polynomial coefficients (LPCs). When converting LSFs to LPCs, the block outputs match those of the `lsf2poly` function.

The block input must be a sample-based row vector, which is treated as a single channel, or a matrix, which is treated as a single channel per column. Each input channel must be in the same format, which you specify in the **Input** parameter:

- `LSF in range (0 pi)` — Vector of LSF values between 0 and π radians in increasing order. Do not include the guaranteed LSF values, 0 and π.
- `LSF normalized in range (0 0.5)` — Vector of normalized LSF values in increasing order, (compute by dividing the LSF values between 0 and π radians by 2π). Do not include the guaranteed normalized LSF values, 0 and 0.5.
- `LSP in range (-1 1)` — Vector of LSP values in decreasing order, equal to the cosine of the LSF values between 0 and π radians. Do not include the guaranteed LSP values, -1 and 1.

Dialog Box



Input

Specifies whether to convert `LSP in range (-1 1)`, `LSF in range (0 pi)`, or `LSF normalized in range (0 0.5)` to linear prediction coefficients (LPCs).

Supported Data Types

- Double-precision floating point
- Single-precision floating point

References

Kabal, P. and Ramachandran, R. "The Computation of Line Spectral Frequencies Using Chebyshev Polynomials." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-34 No. 6, December 1986. pp. 1419-1426.

See Also

LPC to LSF/LSP Conversion	Signal Processing Blockset
LPC to/from RC	Signal Processing Blockset
LPC/RC to Autocorrelation	Signal Processing Blockset
`lsf2poly`	Signal Processing Toolbox

LPC to/from Cepstral Coefficients

Purpose

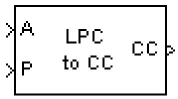
Convert linear prediction coefficients to cepstral coefficients or cepstral coefficients to linear prediction coefficients

Library

Estimation / Linear Prediction

`dsplp`

Description



LPC to/from Cepstral Coefficients

The LPC to/from Cepstral Coefficients block either converts linear prediction coefficients (LPCs) to cepstral coefficients (CCs) or cepstral coefficients to linear prediction coefficients. Set the **Type of conversion** parameter to `LPCs to cepstral coefficients` or `Cepstral coefficients to LPCs` to select the domain into which you want to convert your coefficients. The LPC port corresponds to LPCs, and the CC port corresponds to the CCs. For more information, see “Algorithm” on page 2-815.

The block input must be a sample-based row vector, which is treated as a single channel, or a matrix, which is treated as a single channel per column.

Consider a signal $x(n)$ as the input to an FIR analysis filter represented by LPCs. The output of this analysis filter, $e(n)$, is known as the prediction error signal. The power of this error signal is denoted by P, the prediction error power.

When you select `LPCs to cepstral coefficients` from the **Type of conversion** list, you can specify the prediction error power in two ways. From the **Specify P** list, choose `via input port` to input the prediction error power using input port P. The input to the port must be a vector with length equal to the number of input channels. Select `assume P equals 1` to set the prediction error power equal to 1 for all channels.

When you select `LPCs to cepstral coefficients` from the **Type of conversion** list, the **Output size same as input size** check box appears. When you select this check box, the length of the input vector of LPCs is equal to the output vector of CCs. When you do not select this check box, enter a positive scalar for the **Length of output cepstral coefficients** parameter.

When you select `LPCs to cepstral coefficients` from the **Type of conversion** list, you can use the **If first input value is not 1** parameter to specify the behavior of the block when the first coefficient of the LPC vector is not 1. The following options are available:

- `Replace it with 1` —- Changes the first value of the coefficient vector to 1. The other coefficient values are unchanged.
- `Normalize` — Divides the entire vector of coefficients by the first coefficient so that the first coefficient of the LPC vector is 1.
- `Normalize and Warn` — Divides the entire vector of coefficients by the first coefficient so that the first coefficient of the LPC vector is 1. The block displays a warning message telling you that your vector of coefficients has been normalized.
- `Error` — Displays an error telling you that the first coefficient of the LPC vector is not 1.

When you select `Cepstral coefficients to LPCs` from the **Type of conversion** list, the **Output P** check box appears on the block. Select this check box when you want to output the prediction error power from output port P.

Algorithm

The cepstral coefficients are the coefficients of the Fourier transform representation of the logarithm magnitude spectrum. Consider a sequence, $x(n)$, having a Fourier transform $X(\omega)$. The cepstrum, $c_x(n)$, is defined by the inverse Fourier transform of $C_x(\omega)$, where $C_x(\omega) = \log_e X(\omega)$. See the Real Cepstrum block reference page for information on computing cepstrum coefficients from time-domain signals.

LPC to CC

When in this mode, this block uses a recursion technique to convert LPCs to CCs. The LPC vector is defined by $\begin{bmatrix} a_0 & a_1 & a_2 & \ldots & a_p \end{bmatrix}$ and the CC vector is defined by $\begin{bmatrix} c_0 & c_1 & c_2 & \ldots & c_p & \ldots & c_{n-1} \end{bmatrix}$. The recursion is defined by the following equations:

$$c_0 = \log_e P$$

$$c_m = -a_m + \frac{1}{m}\sum_{k=1}^{m-1}\left[-(m-k)\cdot a_k \cdot c_{(m-k)}\right], 1 \le m \le p$$

$$c_m = \sum_{k=1}^{p}\left[\frac{-(m-k)}{m}\cdot a_k \cdot c_{(m-k)}\right], p < m < n$$

CC to LPC

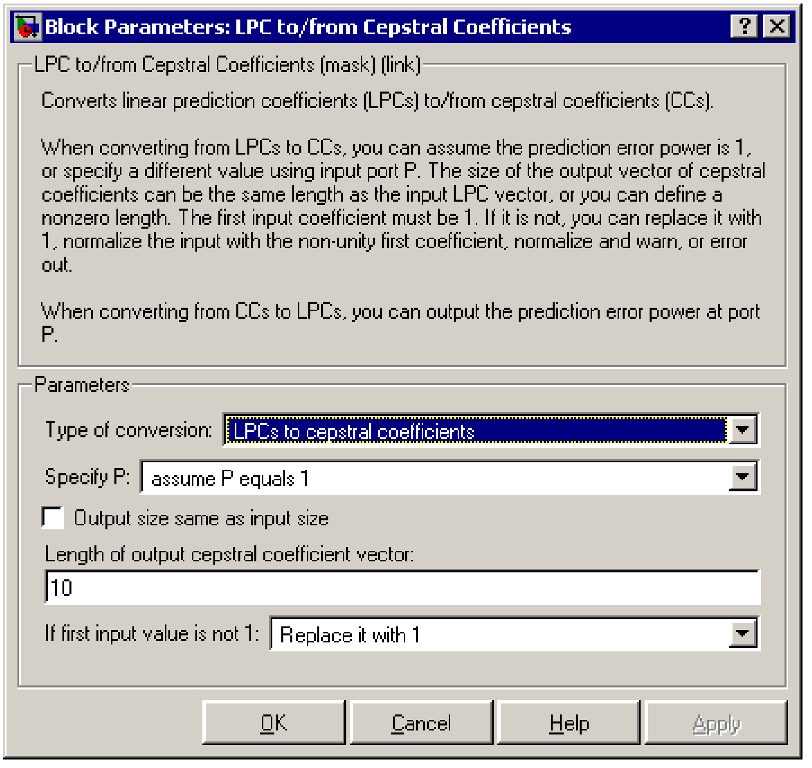
When in this mode, this block uses a recursion technique to convert CCs to LPCs. The CC vector is defined by $\begin{bmatrix} c_0 & c_1 & c_2 & \ldots & c_p & \ldots & c_n \end{bmatrix}$ and the LPC vector is defined by $\begin{bmatrix} a_0 & a_1 & a_2 & \ldots & a_p \end{bmatrix}$. The recursion is defined by the following equations

$$a_m = -c_m - \frac{1}{m}\sum_{k=1}^{m-1}\left[(m-k)\cdot c_{(m-k)} \cdot a_k\right]$$

$$P = \exp(C_0)$$

where $m = 1, 2, \ldots, p$.

Dialog Box



Type of conversion
: Choose `LPCs to cepstral coefficients` or `Cepstral coefficients to LPCs` to specify the domain into which you want to convert your coefficients.

Specify P
: Choose `via input port` to input the values of prediction error power using input port P. Select `assume P equals 1` to set the prediction error power equal to 1.

Output size same as input size
: When you select this check box, the length of the input vector of LPCs is equal to the output vector of CCs.

Length of output cepstral coefficients
Enter a positive scalar that is the length of each output channel of CCs.

If first input value is not 1
Select what you would like the block to do when the first coefficient of the LPC vector is not 1. You can choose `Replace it with 1`, `Normalize`, `Normalize and Warn`, and `Error`.

Output P
Select this check box to output the prediction error power for each channel from output port P.

References

Papamichalis, Panos E. *Practical Approaches to Speech Coding*. Englewood Cliffs, NJ: Prentice Hall, 1987.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Levinson-Durbin	Signal Processing Blockset
LPC to LSF/LSP Conversion	Signal Processing Blockset
LSF/LSP to LPC Conversion	Signal Processing Blockset
LPC to/from RC	Signal Processing Blockset
LPC/RC to Autocorrelation	Signal Processing Blockset
Real Cepstrum	Signal Processing Blockset
Complex Cepstrum	Signal Processing Blockset

Purpose

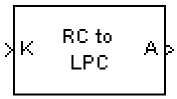
Convert linear prediction coefficients to reflection coefficients or reflection coefficients to linear prediction coefficients

Library

Estimation / Linear Prediction

`dsplp`

Description



LPC to/from RC

The LPC to/from RC block either converts linear prediction coefficients (LPCs) to reflection coefficients (RCs) or reflection coefficients to linear prediction coefficients. Set the **Type of conversion** parameter to `LPC to RC` or `RC to LPC` to select the domain into which you want to convert your coefficients. The A port corresponds to LPC coefficients, and the K port corresponds to the RC coefficients. For more information, see “Algorithm” on page 2-820.

The block input must be a sample-based row vector, which is treated as a single channel, or a matrix, which is treated as a single channel per column.

Consider a signal $x(n)$ as the input to an FIR analysis filter represented by LPC coefficients. The output of the this analysis filter, $e(n)$, is known as the prediction error signal. The power of this error signal is denoted by P. When the zero lag autocorrelation coefficient of $x(n)$ is one, the autocorrelation sequence and prediction error power are said to be normalized.

Select the **Output normalized prediction error power** check box to enable port P. The normalized prediction error power output at `P` is a vector with one element per input channel. Each element varies between zero and one.

Select the **Output LPC filter stability** check box to output the stability of the filter represented by the LPCs or RCs. The synthesis filter represented by the LPCs is stable when the absolute value of each of the roots of the LPC polynomial is less than one. The lattice filter represented by the RCs is stable when the absolute value of each reflection coefficient is less than 1. When the filter is stable, the block outputs a Boolean value of `1` for each input channel at the S port. When the filter is unstable, the block outputs a Boolean value of `0` for each input channel at the S port.

If first input value is not 1 parameter specifies the behavior of the block when the first coefficient of the LPC coefficient vector in any channel is not 1. The following options are available:

- `Replace it with 1` — Changes the first value of the coefficient channel to 1. The other coefficient values are unchanged.
- `Normalize` — Divides the entire channel of coefficients by the first coefficient so that the first coefficient of the LPC coefficient vector is 1.
- `Normalize and Warn` — Divides the entire channel of coefficients by the first coefficient so that the first coefficient of the LPC coefficient vector is 1. The block displays a warning message telling you that your vector of coefficients has been normalized.
- `Error` — Displays an error telling you that the first coefficient of the LPC coefficient channel is not 1.

Algorithm

LPC to RC

When in this mode, this block uses backward Levinson recursion to convert linear prediction coefficients (LPCs) to reflection coefficients (RCs). For a given Nth order LPC vector

$LPC_N = [\ 1 \quad a_{N1} \quad a_{N2} \quad \dots \quad a_{NN}\ ]$, the block calculates the Nth reflection coefficient value using the formula $\gamma_N = -a_{NN}$. The block then finds the lower order LPC vectors , $LPC_{N-1}, LPC_{N-2}, \ldots, LPC_1$, using the following recursion.

for $p = N, N-1, \ldots, 2,$

$$\gamma_p = a_{pp}$$
$$F = 1 - \gamma_p^{\,2}$$
$$a_{p-1,m} = \frac{a_{p,m}}{F} - \frac{\gamma_p a_{p,p-m}}{F}, \quad 1 \le m < p$$

end

Finally, $\gamma_1 = -a_{11}$. The reflection coefficient vector is $[\gamma_1, \ \gamma_2, \ \ldots, \ \gamma_N]$.

RC to LPC

When in this mode, this block uses Levinson recursion to convert reflection coefficients (RCs) to linear prediction coefficients (LPCs).

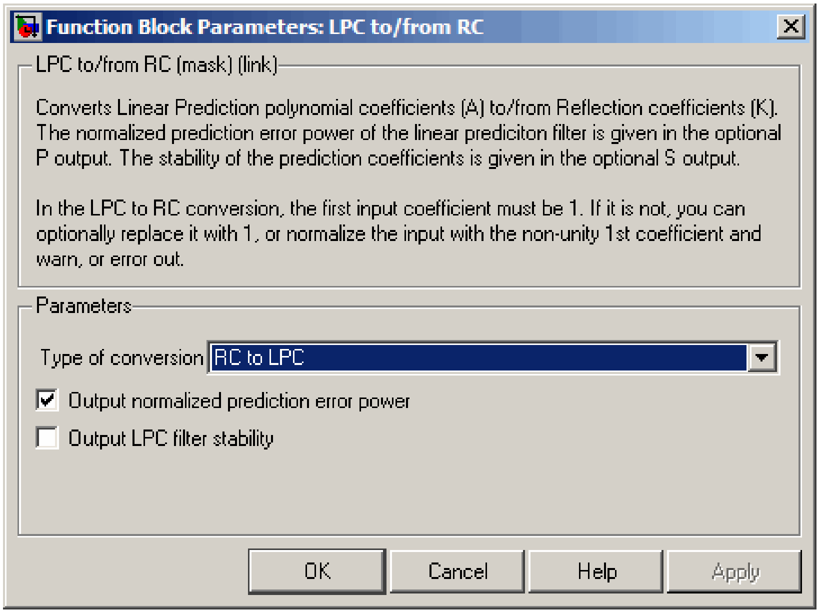
In this case, the input to the block is $RC = [\gamma_1 \quad \gamma_2 \quad \ldots \quad \gamma_N]$. The zeroth order LPC vector term is 1. Starting with this term, the block uses recursion to calculate the higher order LPC vectors, $LPC_2, LPC_3, \ldots LPC_N$, until it has calculated the entire LPC matrix.

$$LPC_{matrix} = \begin{bmatrix} LPC_0 \\ LPC_1 \\ LPC_2 \\ \cdots \\ LPC_N \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & a_{11} & 0 & 0 & \cdots & 0 \\ 1 & a_{21} & a_{22} & 0 & \cdots & 0 \\ 1 & a_{31} & a_{32} & a_{33} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & a_{N1} & a_{N2} & a_{N3} & \cdots & a_{NN} \end{bmatrix}$$

This LPC matrix consists of LPC vectors of order 0 through N found by using the Levinson recursion. The following are the formulas for the recursion steps, for $p = 0, 1, \ldots, N-1$.

$$a_{p+1,m} = a_{p,m} + \gamma_{p+1} a_{p,p+1-m}, \quad 1 \le m \le p$$
$$a_{p+1,p+1} = \gamma_{p+1}$$

Dialog Box



Type of conversion

Select `LPC to RC` or `RC to LPC` to select the domain into which you want to convert your coefficients.

Output normalized prediction error power

Select this check box to output the normalized prediction error power at port P.

Output LPC filter stability

Select this check box to output the stability of the filter. When the filter represented by the LPCs or RCs is stable, the block outputs a Boolean value of `1` for each input channel at the S port. When the filter represented by the LPCs or RCs is unstable, the block outputs a Boolean value of `0` for each input channel at the S port.

If first input value is not 1

Select what you would like the block to do when the first coefficient of the LPC coefficient vector is not 1. You can choose `Replace it with 1`, `Normalize`, `Normalize and Warn`, and `Error`.

References

Makhoul, J *Linear Prediction: A tutorial review*. Proc. IEEE. 63, 63, 56 (1975).

Markel, J.D. and A. H. Gray, Jr., *Linear Prediction of Speech*. New York, Springer-Verlag, 1976.

Supported Data Types

- Double-precision floating-point
- Single-precision floating-point

See Also

Levinson-Durbin	Signal Processing Blockset
LPC to LSF/LSP Conversion	Signal Processing Blockset
LSF/LSP to LPC Conversion	Signal Processing Blockset
LPC/RC to Autocorrelation	Signal Processing Blockset

Purpose

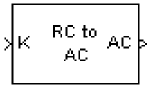
Convert linear prediction coefficients or reflection coefficients to autocorrelation coefficients

Library

Estimation / Linear Prediction

`dsplp`

Description



LPC/RC to Autocorrelation

The LPC/RC to Autocorrelation block either converts linear prediction coefficients (LPCs) to autocorrelation coefficients (ACs) or reflection coefficients (RCs) to autocorrelation coefficients (ACs). Set the **Type of conversion** parameter to `LPC to autocorrelation` or `RC to autocorrelation` to select the domain from which you want to convert your coefficients. The A port corresponds to LPC coefficients, and the K port corresponds to the RC coefficients.

The block input must be a sample-based row vector, which is treated as a single channel, or a matrix, which is treated as a single channel per column.

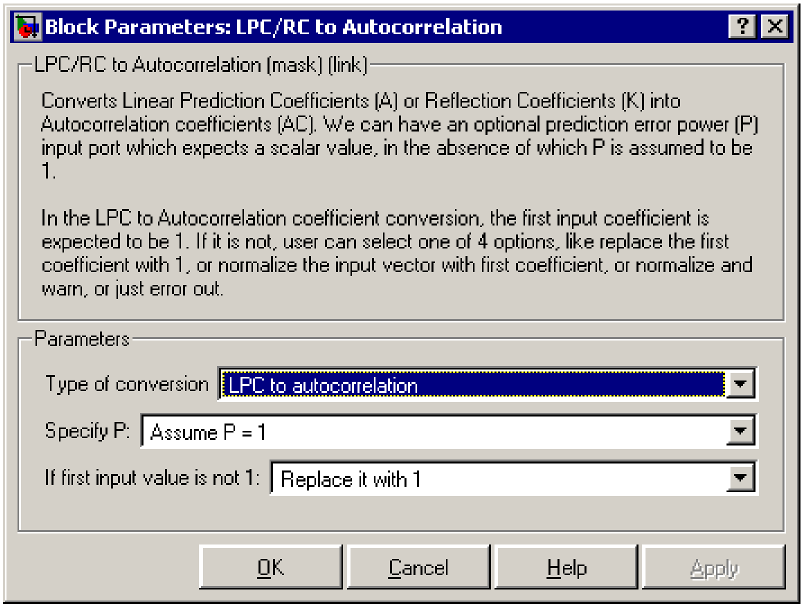
Use the **Specify P** parameter to set the value of the prediction error power. You can set this parameter to 1 by selecting `Assume P=1`. When you select `Via input port`, a P port appears on the block. You can use this port to input the value of the actual, non-unity prediction error power for each channel. The length of this vector must equal the number of channels in the input.

The **If first input value is not 1** parameter specifies the behavior of the block when the first coefficient of the LPC coefficient vector is not 1. The following options are available:

- `Replace it with 1` — The block changes the first value of the coefficient vector to 1. The rest of the coefficient values are unchanged.
- `Normalize` — The block divides the entire vector of coefficients by the first coefficient so that the first coefficient of the LPC coefficient vector is 1.

- `Normalize and Warn` — The block divides the entire vector of coefficients by the first coefficient so that the first coefficient of the LPC coefficient vector is 1. The block displays a warning message telling you that your vector of coefficients has been normalized.
- `Error` — The block displays an error telling you that the first coefficient of the LPC coefficient vector is not 1.

Dialog Box



Type of conversion

From the list select `LPC to autocorrelation` or `RC to autocorrelation` to specify the domain from which you want to convert your coefficients.

Specify P

From the list select `Assume P=1` or `Via input port` to specify the value of prediction error power.

If first input value is not 1

Select what you would like the block to do when the first coefficient of the LPC coefficient vector is not 1. You can choose `Replace it with 1`, `Normalize`, `Normalize and Warn`, and `Error`.

References

Orfanidis, S.J. *Optimum Signal Processing*. New York, McGraw-Hill, 1988.

Makhoul, J. *Linear Prediction: A tutorial review*. Proc. IEEE. 63, 63, 56 (1975).

Markel, J.D. and A. H. Gray, Jr., *Linear Prediction of Speech*. New York, Springer-Verlag, 1976.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Levinson-Durbin	Signal Processing Blockset
LPC to LSF/LSP Conversion	Signal Processing Blockset
LSF/LSP to LPC Conversion	Signal Processing Blockset
LPC to/from RC	Signal Processing Blockset

Purpose

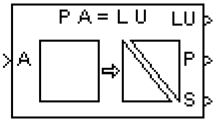
Factor square matrix into lower and upper triangular components

Library

Math Functions / Matrices and Linear Algebra / Matrix Factorizations

`dspfactors`

Description



The LU Factorization block factors a row-permuted version of the square input matrix A as $A_p = L*U$, where L is a unit-lower triangular matrix, U is an upper triangular matrix, and A_p contains the rows of A permuted as indicated by the permutation index vector P. The block uses the pivot matrix A_p instead of the exact input matrix A because it improves the numerical accuracy of the factorization. You can determine the singularity of the input matrix A by enabling the optional output port S. When A is singular, the block outputs a `1` at port S; when A is nonsingular, it outputs a `0`.

To improve efficiency, the output of the LU Factorization block at port LU is a composite matrix containing both the lower triangle elements of L and the upper triangle elements of U. Thus, the output is in a different format than the output of the MATLAB `lu` function, which returns L and U as separate matrices. To convert the output from the block's `LU` port to separate L and U matrices, use the following code:

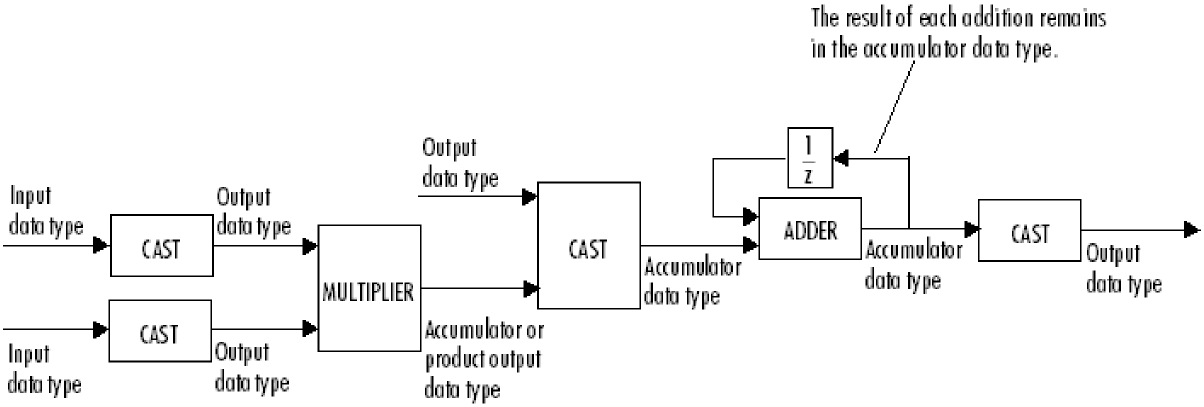
```
L = tril(LU,-1)+eye(size(LU));
U = triu(LU);
```

If you compare the results produced by these equations to the actual output of the MATLAB `lu` function, you may see slightly different values. These differences are due to rounding error, and are expected.

See the `lu` function reference page in the MATLAB documentation for more information about LU factorizations.

Fixed-Point Data Types

The following diagram shows the data types used within the LU Factorization block for fixed-point signals.



You can set the product output, accumulator, and output data types in the block dialog as discussed below.

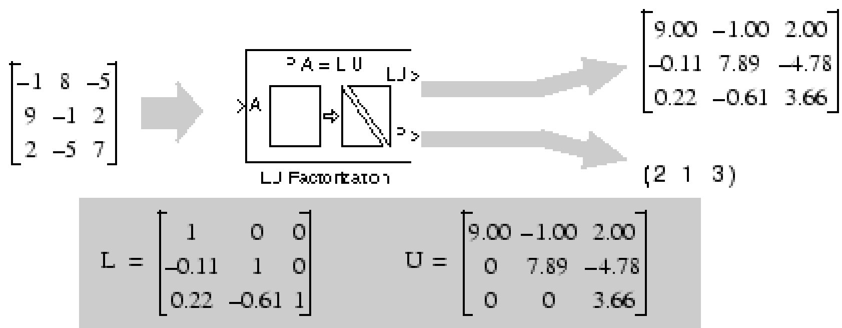
The output of the multiplier is in the product output data type when the input is real. When the input is complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see "Multiplication Data Types".

Examples

The row-pivoted matrix A_p and permutation index vector P computed by the block are shown below for 3-by-3 input matrix A.

$$A = \begin{bmatrix} -1 & 8 & -5 \\ 9 & -1 & 2 \\ 2 & -5 & 7 \end{bmatrix} \qquad P = (2 \quad 1 \quad 3) \qquad A_P = \begin{bmatrix} 9 & -1 & 2 \\ -1 & 8 & -5 \\ 2 & -5 & 7 \end{bmatrix}$$

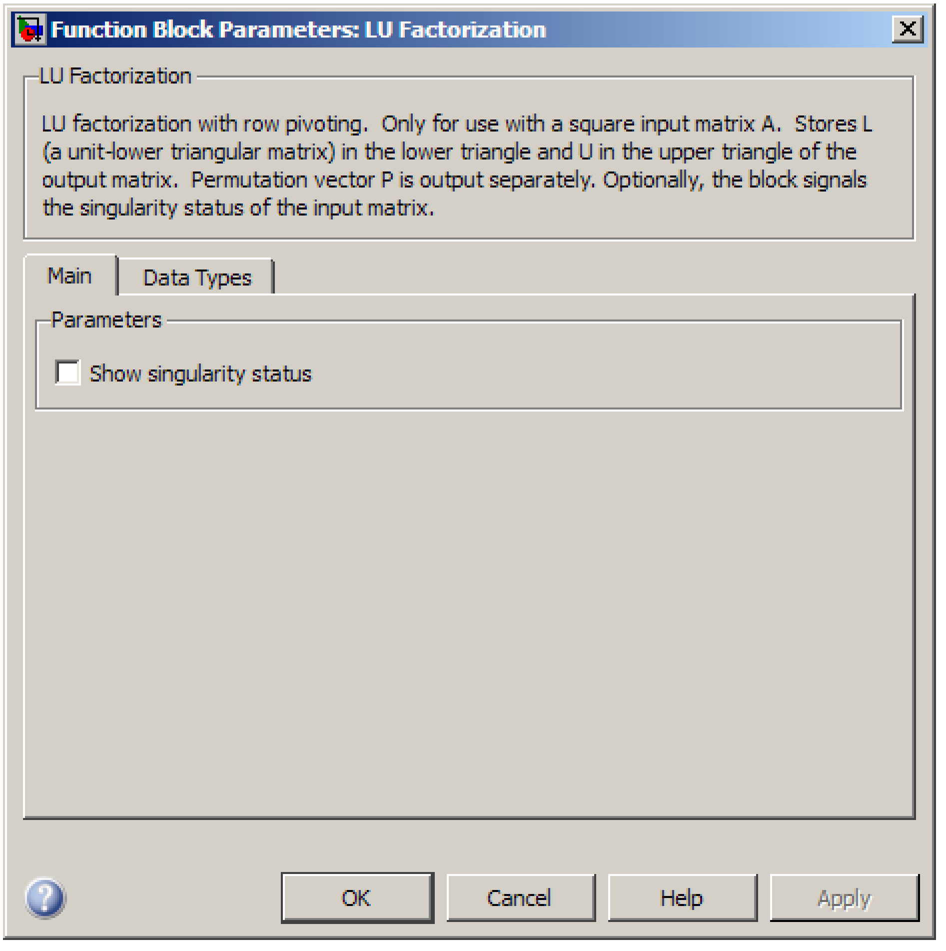
The `LU` output is a composite matrix whose lower subtriangle forms L and whose upper triangle forms U.



See "Example: LU Factorization" in the *Signal Processing Blockset User's Guide* for another example using the LU Factorization block.

Dialog Box

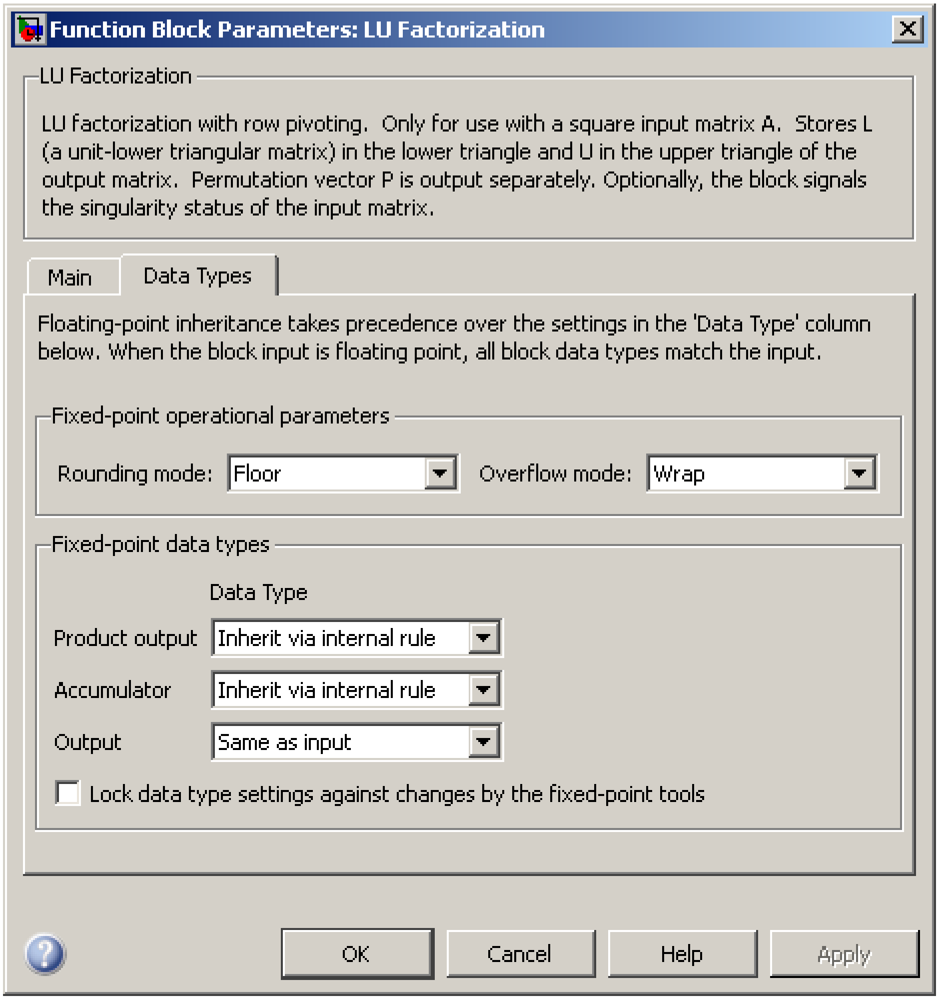
The **Main** pane of the LU Factorization block dialog appears as follows.



Show singularity status

Select to output the singularity of the input at port S, which outputs Boolean data type values of 1 or 0. An output of 1 indicates that the current input is singular, and an output of 0 indicates the current input is nonsingular.

The **Data Types** pane of the LU Factorization block dialog appears as follows.



Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode for fixed-point operations.

Product output

Use this parameter to specify how you would like to designate the product output word and fraction lengths. See "Fixed-Point Data Types" on page 2-827 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block:

- When you select `Inherit via internal rule`, the product output word length and fraction length are calculated automatically. For information about how the product output word and fraction lengths are calculated when an internal rule is used, see "Inherit via Internal Rule".
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator

Use this parameter to specify how you would like to designate the accumulator word and fraction lengths. See "Fixed-Point Data Types" on page 2-827 and "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block.

- When you select `Inherit via internal rule`, the accumulator word length and fraction length are calculated automatically. For information about how the accumulator word and fraction lengths are calculated when an internal rule is used, see "Inherit via Internal Rule".

- When you select `Same as product output`, these characteristics match those of the product output.
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Use this parameter to specify how you would like to designate the output word and fraction lengths. See "Fixed-Point Data Types" on page 2-827 for an illustration depicting the use of the output data type in this block:

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

References

Golub, G. H., and C. F. Van Loan. *Matrix Computations*. 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

Supported Data Types

Port	Supported Data Types
A	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
LU	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
P	• Double-precision floating point • Single-precision floating point • 32-bit unsigned integers
S	• Boolean

See Also

Autocorrelation LPC	Signal Processing Blockset
Cholesky Factorization	Signal Processing Blockset
LDL Factorization	Signal Processing Blockset
LU Inverse	Signal Processing Blockset
LU Solver	Signal Processing Blockset
Permute Matrix	Signal Processing Blockset
QR Factorization	Signal Processing Blockset
`lu`	MATLAB

See "Matrix Factorizations" for related information.

Purpose

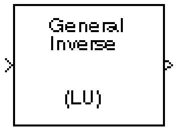
Compute inverse of square matrix using LU factorization

Library

Math Functions / Matrices and Linear Algebra / Matrix Inverses

`dspinverses`

Description



The LU Inverse block computes the inverse of the square input matrix A by factoring and inverting row-pivoted variant A_p.

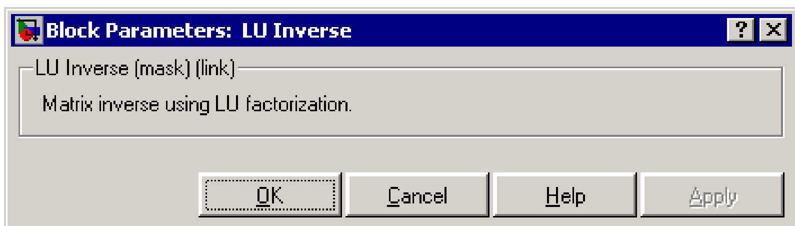
$$A_p^{-1} = (LU)^{-1}$$

L is a lower triangular square matrix with unity diagonal elements, and U is an upper triangular square matrix. The block's output is A^{-1}, and is always sample based.

Examples

See "Example: LU Inverse" in the *Signal Processing Blockset User's Guide*.

Dialog Box



References

Golub, G. H., and C. F. Van Loan. *Matrix Computations*. 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Cholesky Inverse	Signal Processing Blockset
LDL Inverse	Signal Processing Blockset
LU Factorization	Signal Processing Blockset
LU Solver	Signal Processing Blockset
`inv`	MATLAB

See “Matrix Inverses” for related information.

Purpose

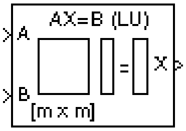
Solve AX=B for X when A is square matrix

Library

Math Functions / Matrices and Linear Algebra / Linear System Solvers

`dspsolvers`

Description



The LU Solver block solves the linear system AX=B by applying LU factorization to the M-by-M matrix at the A port. The input to the B port is the right side M-by-N matrix, B. The output is the unique solution of the equations, M-by-N matrix X, and is always sample based.

A length-M 1-D vector input for right side B is treated as an M-by-1 matrix.

Algorithm

The LU algorithm factors a row-permuted variant (A_p) of the square input matrix A as

$$A_p = LU$$

where L is a lower triangular square matrix with unity diagonal elements, and U is an upper triangular square matrix.

The matrix factors are substituted for A_p in

$$A_pX = B_p$$

where B_p is the row-permuted variant of B, and the resulting equation

$$LUX = B_p$$

is solved for X by making the substitution $Y = UX$, and solving two triangular systems.

$$LY = B_p$$
$$UX = Y$$

Examples

See "Example: LU Solver" in the *Signal Processing Blockset User's Guide*.

Dialog Box

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Autocorrelation LPC	Signal Processing Blockset
Cholesky Solver	Signal Processing Blockset
LDL Solver	Signal Processing Blockset
Levinson-Durbin	Signal Processing Blockset
LU Factorization	Signal Processing Blockset
LU Inverse	Signal Processing Blockset
QR Solver	Signal Processing Blockset

See "Linear System Solvers" for related information.

Purpose

Compute nonparametric estimate of spectrum using periodogram method

Library

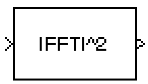
- Estimation / Power Spectrum Estimation

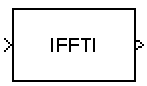
  `dspspect3`

- Transforms

  `dspxfrm3`

Description





The Magnitude FFT block computes a nonparametric estimate of the spectrum using the periodogram method.

When the **Output** parameter is set to `Magnitude squared`, the block output for an M-by-N input u is equivalent to

```
y = abs(fft(u,nfft)).^2       % M   nfft
```

When the **Output** parameter is set to `Magnitude`, the block output for an input u is equivalent to

```
y = abs(fft(u,nfft))         % M   nfft
```

When $M > N_{fft}$, the block wraps the input to N_{fft} before computing the FFT using one of the above equations:

```
y(:,k)=datawrap(u(:,k),nfft)     % 1   k   N
```

Both an M-by-N frame-based matrix input and an M-by-N sample-based matrix input are treated as M sequential time samples from N independent channels. The block computes a separate estimate for each of the N independent channels and generates an N_{fft}-by-N matrix output. When you select **Inherit FFT length from input dimensions**, N_{fft} is specified by the frame size of the input, which must be a power of 2. When you do not select **Inherit FFT length from input dimensions**, N_{fft} is specified as a power of 2 by the **FFT length** parameter, and the block zero pads or wraps the input to N_{fft} before computing the FFT.

Each column of the output matrix contains the estimate of the corresponding input column's power spectral density at N_{fft} equally spaced frequency points in the range $[0,F_s)$, where F_s is the signal's sample frequency. The output is always sample based.

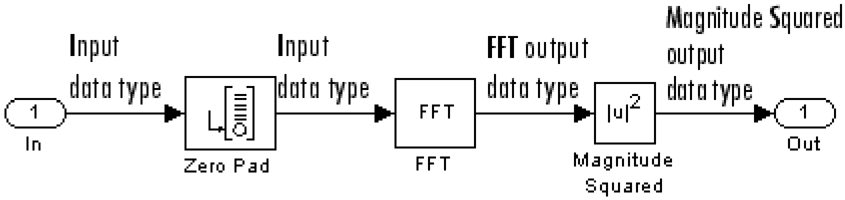
The block does not accept sample-based 1-by-N row vector inputs.

The Magnitude FFT block supports real and complex floating-point inputs. The block also supports real fixed-point inputs in both `Magnitude` and `Magnitude squared` modes, and complex fixed-point inputs in the `Magnitude squared` mode.

The Magnitude FFT block supports real and complex floating-point inputs. The block also supports real fixed-point inputs in both `Magnitude` and `Magnitude squared` modes, and complex fixed-point inputs in the `Magnitude squared` mode.

Fixed-Point Data Types

The following diagram shows the data types used within the Magnitude FFT subsystem block for fixed-point signals.



The settings for the fixed-point parameters of the FFT block in the diagram above are as follows:

- Sine table — `Same word length as input`
- Integer rounding mode — `Floor`
- Saturate on integer overflow — unchecked
- Product output — `Inherit via internal rule`
- Accumulator — `Inherit via internal rule`
- Output — `Inherit via internal rule`

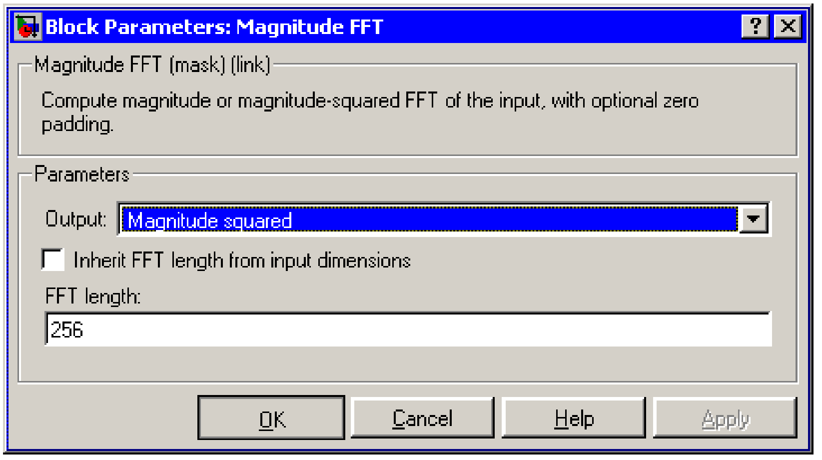
The settings for the fixed-point parameters of the Magnitude Squared block in the diagram above are as follows:

- Integer rounding mode — `Floor`
- Saturate on integer overflow — checked
- Output — `Inherit via internal rule`

Examples

The `dspsacomp` demo compares the periodogram method with several other spectral estimation methods.

Dialog Box



Output

Specify whether the block computes the magnitude FFT or magnitude-squared FFT of the input.

Inherit FFT length from input dimensions

Select to use the input frame size as the number of data points, N_{fft}, on which to perform the FFT.

FFT length

Enter the number of data points on which to perform the FFT, N_{fft}. When N_{fft} is larger than the input frame size, each frame is zero-padded as needed. When N_{fft} is smaller than the input frame size, each frame is wrapped as needed. This parameter is enabled when you clear the **Inherit FFT length from input dimensions** check box.

References

Oppenheim, A. V. and R. W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

Orfanidis, S. J. *Introduction to Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

Proakis, J. and D. Manolakis. *Digital Signal Processing*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

Burg Method	Signal Processing Blockset
Short-Time FFT	Signal Processing Blockset
Spectrum Scope	Signal Processing Blockset
Yule-Walker Method	Signal Processing Blockset
`pwelch`	Signal Processing Toolbox

See “Power Spectrum Estimation” for related information.

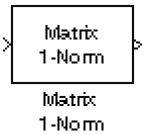
Matrix 1-Norm

Purpose Compute 1-norm of matrix

Library Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description

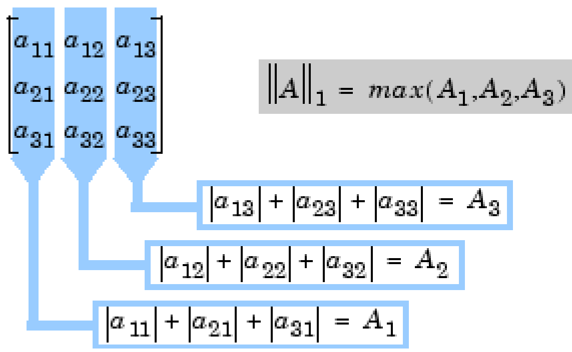


The Matrix 1-Norm block computes the 1-norm, or maximum column-sum, of an M-by-N input matrix, A.

$$y = \|A\|_1 = \max_{1 \le j \le N} \sum_{i=1}^{M} |a_{ij}|$$

This is equivalent to

```
y = max(sum(abs(A)))      % Equivalent
MATLAB code
```

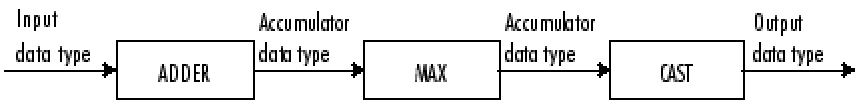


A length-M 1-D vector input is treated as an M-by-1 matrix. The output, y, is always a scalar.

The Matrix 1-Norm block supports real and complex floating-point inputs, and real fixed-point inputs.

Fixed-Point Data Types

The following diagram shows the data types used within the Matrix 1-Norm block for fixed-point signals.

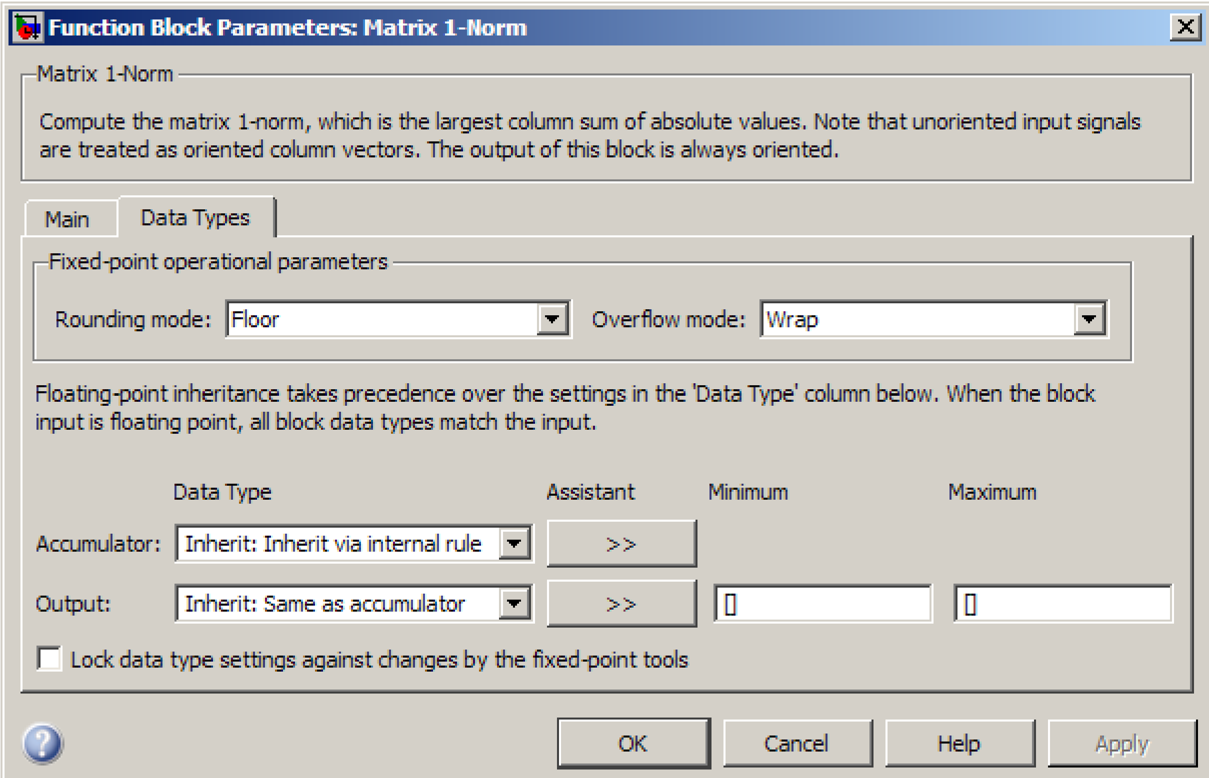


The block calculations are all done in the accumulator data type until the `max` is performed. The result is then cast to the output data type. You can set the accumulator and output data types in the block dialog as discussed in "Dialog Box" on page 2-845 below.

Dialog Box

There are no parameters on the **Main** pane of this dialog.

The **Data types** pane of the Matrix 1-Norm block dialog appears as follows.



Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Rounding mode
: Select the rounding mode for fixed-point operations.

Overflow mode
: Select the overflow mode for fixed-point operations.

Accumulator data type

Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-844 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Output data type

Specify the output data type. See “Fixed-Point Data Types” on page 2-844 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as accumulator`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User’s Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

References

Golub, G. H., and C. F. Van Loan. *Matrix Computations*. 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Normalization	Signal Processing Blockset
Reciprocal Condition	Signal Processing Blockset
`norm`	MATLAB

Purpose

Concatenate input signals of same data type to create contiguous output signal

Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description

The Matrix Concatenate block is an implementation of the Simulink Matrix Concatenate block. See Matrix Concatenate for more information.

Purpose

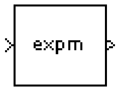
Compute matrix exponential

Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

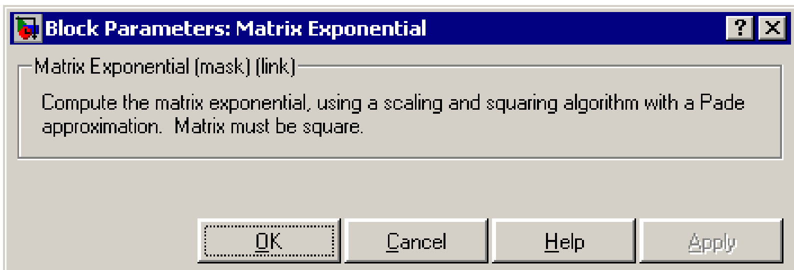
Description



Matrix Exponential

The Matrix Exponential block computes the matrix exponential using a scaling and squaring algorithm with a Pade approximation. The input matrix must be square.

Dialog Box



Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Array-Vector Multiply	Signal Processing Blockset
`expm`	MATLAB
Dot Product	Simulink

Matrix Product	Signal Processing Blockset
Product	Simulink

Purpose

Multiply or divide inputs

Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description

The Matrix Multiply block is an implementation of the Simulink Product block. See Product for more information.

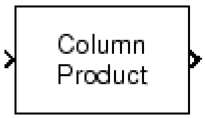
Matrix Product

Purpose Multiply matrix elements along rows, columns, or entire input

Library Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description



The Matrix Product block multiplies the elements of an M-by-N input matrix u along its rows, its columns, or over all its elements.

When the **Multiply over** parameter is set to `Rows`, the block multiplies across the elements of each row and outputs the resulting M-by-1 matrix. A length-N 1-D vector input is treated as a 1-by-N matrix.

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} \left(\prod_{j=1}^{3} u_{1j}\right) \\ \left(\prod_{j=1}^{3} u_{2j}\right) \\ \left(\prod_{j=1}^{3} u_{3j}\right) \end{bmatrix}$$

When the **Multiply over** parameter is set to `Columns`, the block multiplies down the elements of each column and outputs the resulting 1-by-N matrix. A length-M 1-D vector input is treated as a M-by-1 matrix.

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} = \begin{bmatrix} \left(\prod_{i=1}^{3} u_{i1}\right) & \left(\prod_{i=1}^{3} u_{i2}\right) & \left(\prod_{i=1}^{3} u_{i3}\right) \end{bmatrix}$$

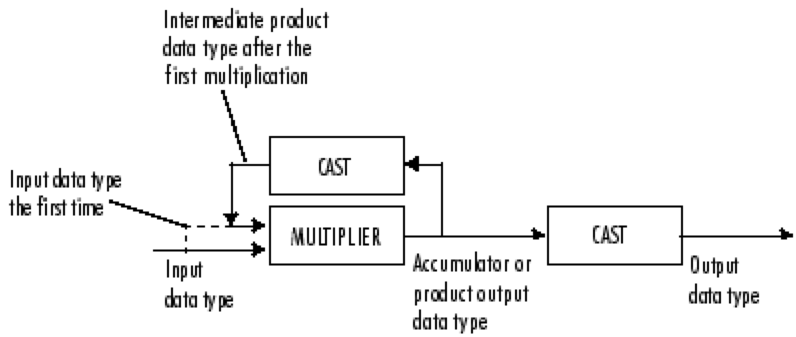
When the **Multiply over** parameter is set to `Entire input`, the block multiplies all the elements of the input together and outputs the resulting scalar.

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \Rightarrow y = \left(\prod_{i=1}^{3}\prod_{j=1}^{3} u_{ij}\right)$$

The output of the Matrix Product block has the same frame status as the input. This block accepts real and complex fixed-point and floating-point inputs except for complex unsigned fixed-point inputs.

Fixed-Point Data Types

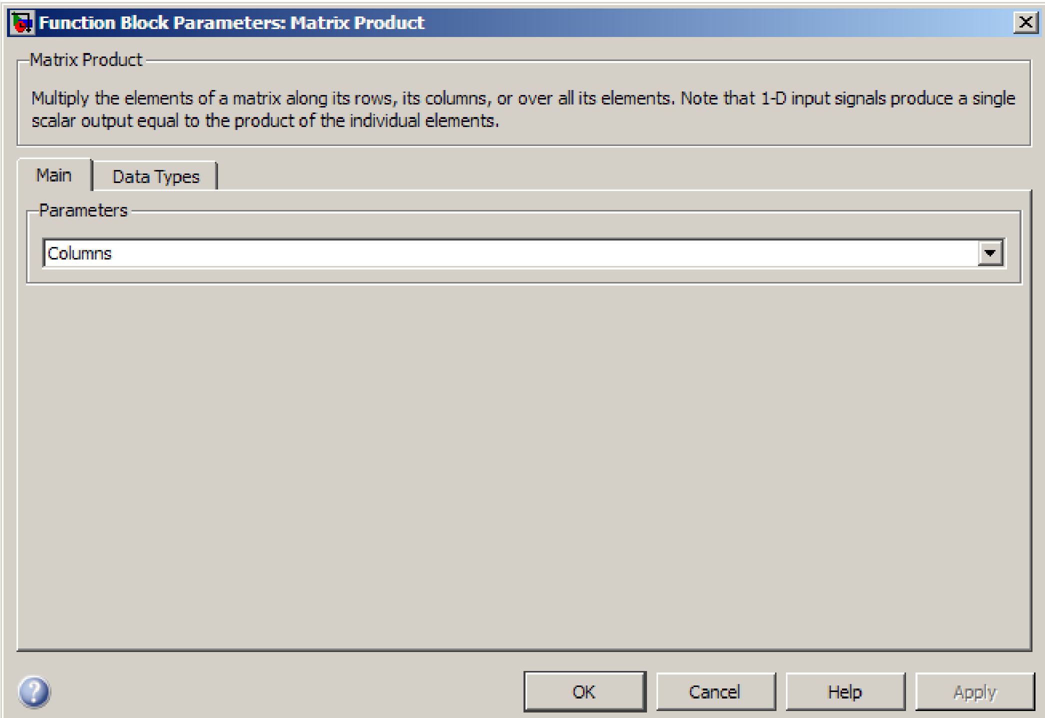
The following diagram shows the data types used within the Matrix Product block for fixed-point signals.



The output of the multiplier is in the product output data type when at least one of the inputs to the multiplier is real. When both of the inputs to the multiplier are complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see "Multiplication Data Types". You can set the accumulator, product output, intermediate product, and output data types in the block dialog as discussed in "Dialog Box" on page 2-856 below.

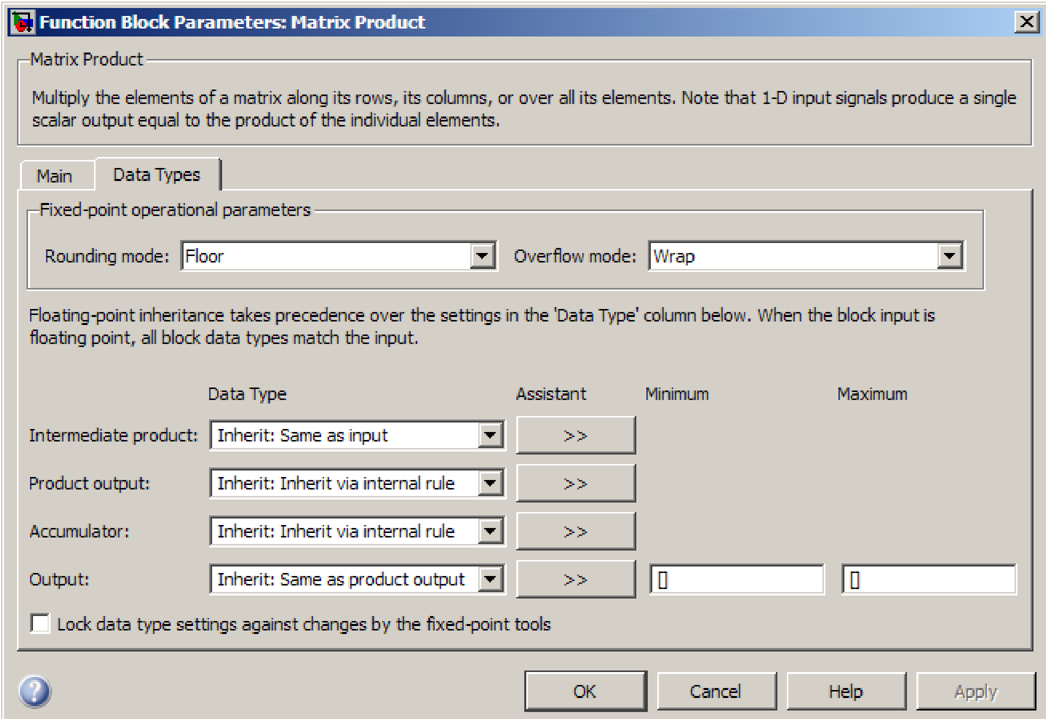
Dialog Box

The **Main** pane of the Matrix Product block dialog appears as follows.



Multiply over
: Indicate whether to multiply together the elements of each row, each column, or the entire input.

The **Data Types** pane of the Matrix Product block dialog appears as follows.



Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Rounding mode
Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode for fixed-point operations.

Intermediate product

Specify the intermediate product data type. As shown in "Fixed-Point Data Types" on page 2-855, the output of the multiplier is cast to the intermediate product data type before the next element of the input is multiplied into it. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Product output data type

Specify the product output data type. See "Fixed-Point Data Types" on page 2-855 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Accumulator data type

Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-855 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Output data type

Specify the output data type. See “Fixed-Point Data Types” on page 2-855 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as product output`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User’s Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Array-Vector Multiply	Signal Processing Blockset
Matrix Square	Signal Processing Blockset
Matrix Sum	Signal Processing Blockset
`prod`	MATLAB

Matrix Square

Purpose

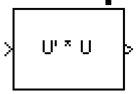
Compute square of input matrix

Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description



The Matrix Square block computes the square of an M-by-N input matrix, u, by premultiplying with the Hermitian transpose.

```
y = u' * u     % Equivalent MATLAB code
```

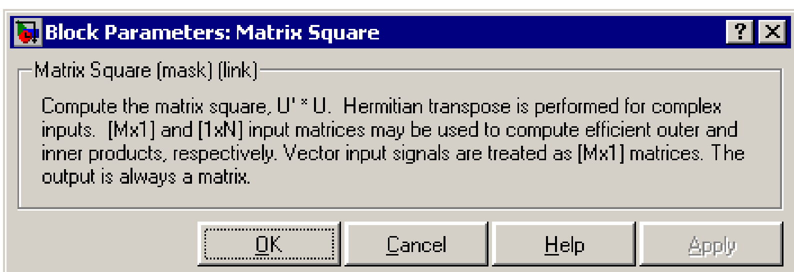
A length-M 1-D vector input is treated as an M-by-1 matrix. For both sample-based and frame-based inputs, output y is sample based with dimension N-by-N.

Applications

The Matrix Square block is useful in a variety of applications:

- General matrix squares — The Matrix Square block computes the output matrix, y, without explicitly forming u'. It is therefore more efficient than other methods for computing the matrix square.
- Sum of squares — When the input is a column vector (N=1), the block's operation is equivalent to a multiply-accumulate (MAC) process, or inner product. The output is the sum of the squares of the input, and is always a real scalar.
- Correlation matrix — When the input is a row vector (M=1), the output, y, is the symmetric autocorrelation matrix, or outer product.

Dialog Box



Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Matrix Multiply	Signal Processing Blockset
Matrix Product	Signal Processing Blockset
Matrix Sum	Signal Processing Blockset
Transpose	Signal Processing Blockset

Matrix Sum

Purpose

Sum matrix elements along rows, columns, or entire input

Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description

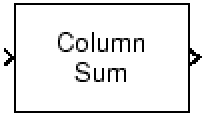
The Matrix Sum block is an implementation of the Simulink Sum block. See Sum for more information.

Purpose Sum matrix elements along rows, columns, or entire input

Library Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspobslib`

Description



The Matrix Sum block sums the elements of an M-by-N input matrix u along its rows, its columns, or over all its elements.

When the **Sum over** parameter is set to `Rows`, the block sums across the elements of each row and outputs the resulting M-by-1 matrix. A length-N 1-D vector input is treated as a 1-by-N matrix.

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} \left(\sum_{j=1}^{3} u_{1j}\right) \\ \left(\sum_{j=1}^{3} u_{2j}\right) \\ \left(\sum_{j=1}^{3} u_{3j}\right) \end{bmatrix}$$

When the **Sum over** parameter is set to `Columns`, the block sums down the elements of each column and outputs the resulting 1-by-N matrix. A length-M 1-D vector input is treated as a M-by-1 matrix.

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix}$$
$$\Downarrow$$
$$\begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} = \begin{bmatrix} \left(\sum_{i=1}^{3} u_{i1}\right) & \left(\sum_{i=1}^{3} u_{i2}\right) & \left(\sum_{i=1}^{3} u_{i3}\right) \end{bmatrix}$$

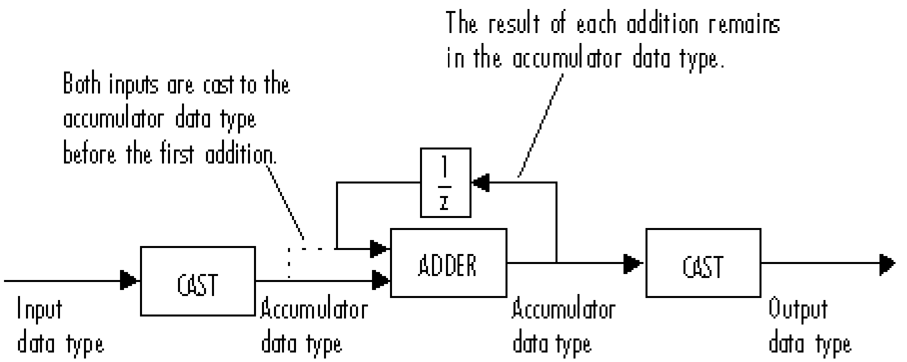
When the **Sum over** parameter is set to `Entire input`, the block sums all the elements of the input together and outputs the resulting scalar.

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \Rightarrow y = (\sum_{i=1}^{3} \sum_{j=1}^{3} u_{ij})$$

The output of the Matrix Sum block has the same frame status as the input. This block accepts real and complex fixed-point and floating-point inputs except for complex unsigned fixed-point inputs.

Fixed-Point Data Types

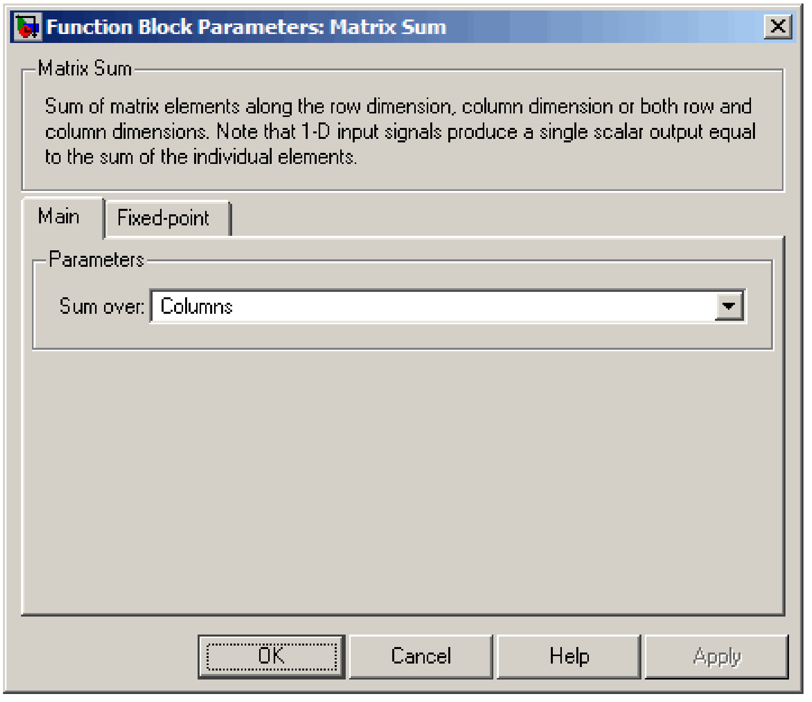
The following diagram shows the data types used within the Matrix Sum block for fixed-point signals.



You can set the accumulator and output data types in the block dialog as discussed in “Dialog Box” on page 2-867 below.

Dialog Box

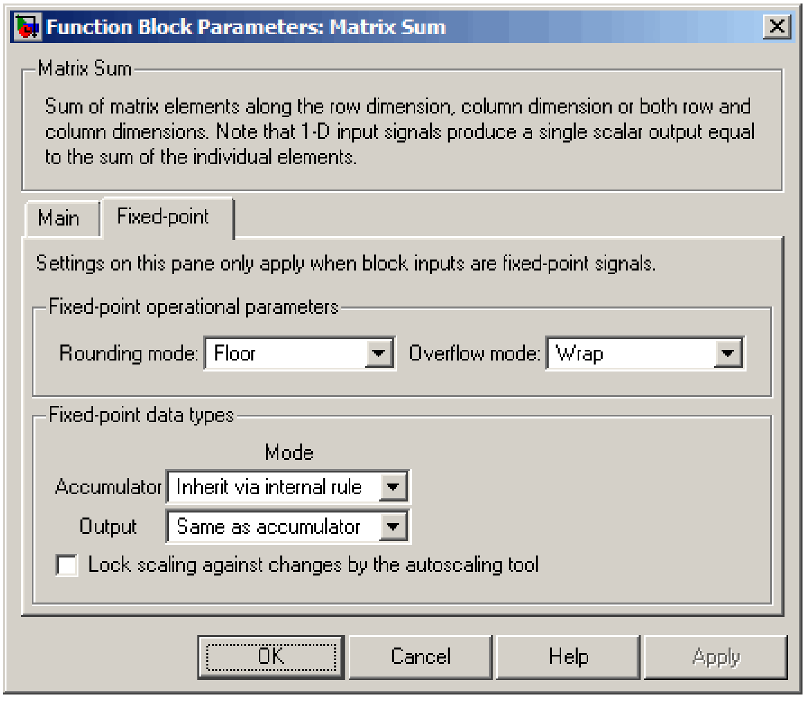
The **Main** pane of the Matrix Sum block dialog appears as follows.



Sum over

Indicate whether to sum the elements of each row, each column, or of the entire input.

The **Fixed-point** pane of the Matrix Sum block dialog appears as follows.



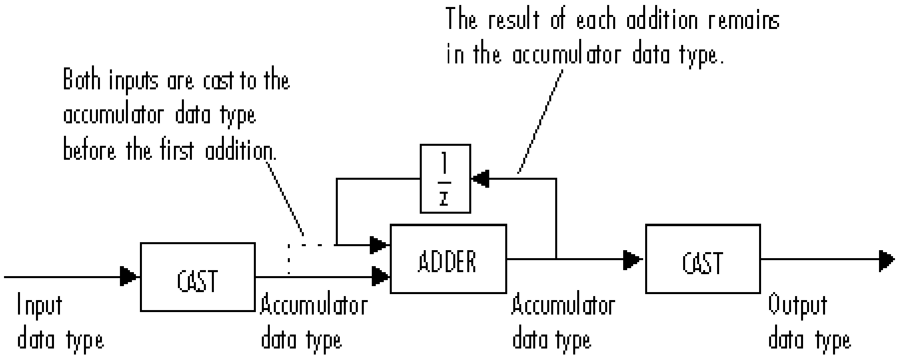
Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode for fixed-point operations.

Accumulator



As depicted above, the elements of the block input are cast to the accumulator data type before they are added together. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how you would like to designate this accumulator word and fraction lengths:

- When you select `Inherit via internal rule`, the accumulator word length and fraction length are calculated automatically. For information about how the accumulator word and fraction lengths are calculated when an internal rule is used, see "Inherit via Internal Rule".
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Choose how you specify the output word length and fraction length:

- When you select `Same as accumulator`, these characteristics match those of the accumulator.
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock scaling against changes by the autoscaling tool
Select this parameter to prevent any fixed-point scaling you specify in this block mask from being overridden by the autoscaling feature of the Fixed-Point Tool. See the `fxptdlg` reference page for more information.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Matrix Product	Signal Processing Blockset
Matrix Multiply	Signal Processing Blockset
`sum`	MATLAB

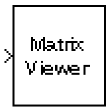
Matrix Viewer

Purpose Display matrices as color images

Library Signal Processing Sinks

`dspsnks4`

Description



The Matrix Viewer block displays an M-by-N matrix input by mapping the matrix element values to a specified range of colors. The display is updated as each new input is received. This block treats a length M 1-D vector input as an M-by-1 matrix.

Image Properties

Select the **Image Properties** tab to show the image property parameters, which control the colormap and display.

You specify the mapping of matrix element values to colors in the **Colormap matrix**, **Minimum input value**, and **Maximum input value** parameters. For a colormap with L colors, the colormap matrix has dimension L-by-3, with one row for each color and one column for each element of the RGB triple that defines the color. Examples of RGB triples are

```
[ 1   0   0 ]   (red)
[ 0   0   1 ]   (blue)
[0.8 0.8 0.8]   (light gray)
```

See the `ColorSpec` property in the MATLAB documentation for complete information about defining RGB triples.

MATLAB provides a number of functions for generating predefined colormaps, such as `hot`, `cool`, `bone`, and `autumn`. Each of these functions accepts the colormap size as an argument, and can be used in the **Colormap matrix** parameter. For example, when you specify `gray(128)` for the **Colormap matrix** parameter, the matrix is displayed in 128 shades of gray. The color in the first row of the colormap matrix represents the value specified by the **Minimum input value** parameter, and the color in the last row represents the value specified by the **Maximum input value** parameter. Values between the minimum and maximum are quantized and mapped to the intermediate rows of the colormap matrix.

The documentation for the MATLAB `colormap` function provides complete information about specifying colormap matrices, and includes a complete list of the available colormap functions.

Axis Properties

Select the **Axis Properties** tab to show the axis property parameters, which control labeling and positioning.

The **Axis origin** parameter determines where the first element of the input matrix, $U(1,1)$, is displayed. When you specify `Upper left corner`, the matrix is displayed in matrix orientation, with $U(1,1)$ in the upper-left corner.

$$\begin{bmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ U_{21} & U_{22} & U_{23} & U_{24} \\ U_{31} & U_{32} & U_{33} & U_{34} \\ U_{41} & U_{42} & U_{43} & U_{44} \end{bmatrix}$$

When you specify `Lower left corner`, the matrix is flipped vertically to image orientation, with $U(1,1)$ in the lower-left corner.

$$\begin{bmatrix} U_{41} & U_{42} & U_{43} & U_{44} \\ U_{31} & U_{32} & U_{33} & U_{34} \\ U_{21} & U_{22} & U_{23} & U_{24} \\ U_{11} & U_{12} & U_{13} & U_{14} \end{bmatrix}$$

Axis zoom, when selected, causes the image display to completely fill the figure window. Axis titles are not displayed. This option can also be selected from the pop-up menu that is displayed when you right-click in the figure window. When **Axis zoom** is cleared, the axis labels and titles are displayed in a gray border surrounding the image axes.

Figure Window

The image title in the figure title bar is the same as the block title. The axis tick marks reflect the size of the input matrix; the x-axis is numbered from 1 to N (number of columns), and the y-axis is numbered from 1 to M (number of rows).

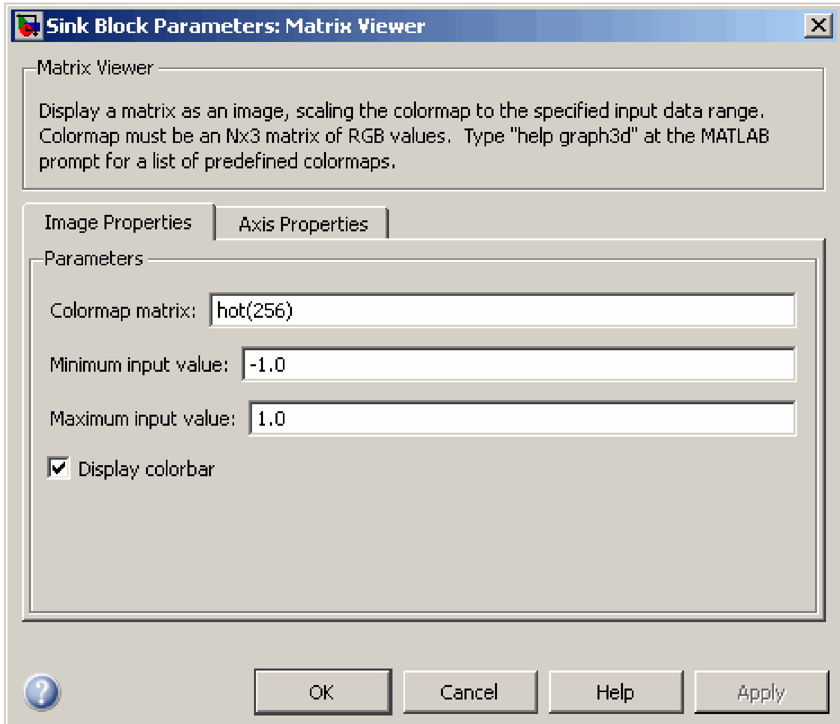
Right-click the image in the figure window to access the following menu items:

- **Refresh** erases all data on the scope display except for the most recent image.
- **Autoscale** recomputes the minimum and maximum input values to fit the range of values observed in a series of 10 consecutive inputs. The numerical limits selected by the autoscale feature are shown in the **Minimum input value** and **Maximum input value** parameters, where you can make further adjustments to them manually.
- **Axis zoom**, when selected, causes the image to completely fill the figure window. Axis titles are not displayed. When **Axis zoom** is cleared, the axis labels and titles are displayed in a gray border surrounding the scope axes. This option can also be set in the Axis Properties pane of the parameter dialog.
- **Colorbar**, when selected, displays a bar with the specified colormap to the right of the image axes.
- **Save Position** automatically updates the **Figure position** parameter in the **Axis Properties** pane to reflect the figure window's current position and size on the screen. To make the scope window open at a particular location on the screen when the simulation runs, drag the window to the desired location, resize it, and select **Save Position**. The parameter dialog must be closed when you select **Save Position** for the **Figure position** parameter to be updated.

Examples

See the Spectral Analysis Using the Periodogram Method demo for an example of using the Matrix Viewer block to create a moving spectrogram, or time-frequency plot, of a speech signal by updating just one column of the input matrix at each sample time.

Dialog Box



Colormap matrix
A 3-column matrix defining the colormap as a set of RGB triples, or a call to a colormap-generating function such as `hot` or `spring`. See the `ColorSpec` property for complete information about defining RGB triples, and the MATLAB `colormap` function for a list of colormap-generating functions. Tunable.

Minimum input value
The input value to be mapped to the color defined in the first row of the colormap matrix. Right-click in the figure window and

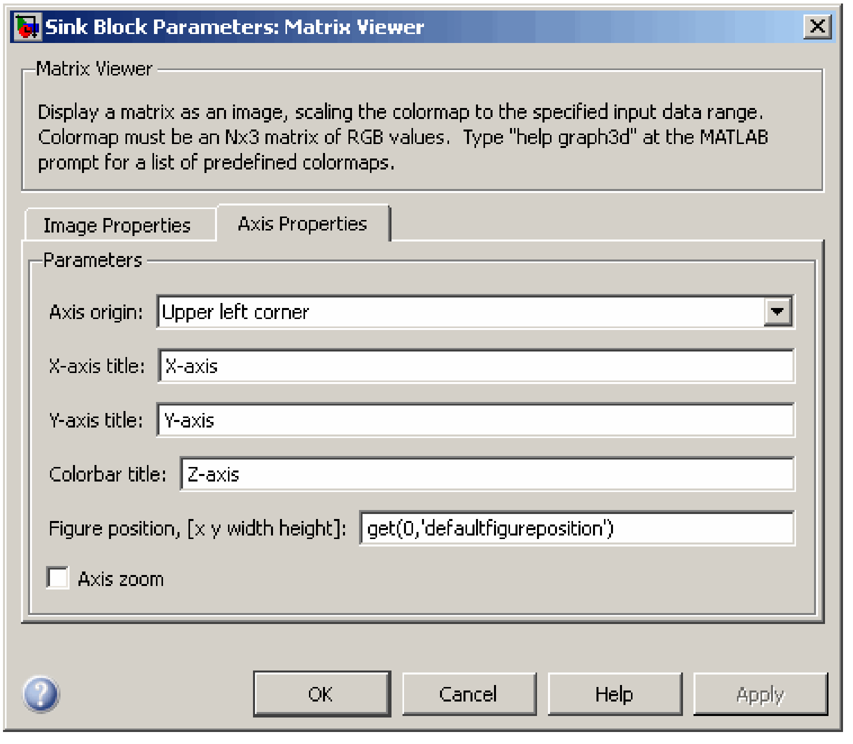
select `Autoscale` from pop-up menu to set this parameter to the minimum value observed in a series of 10 consecutive matrix inputs. Tunable.

Maximum input value

The input value to be mapped to the color defined in the last row of the colormap matrix. Right-click in the figure window and select `Autoscale` from the pop-up menu to set this parameter to the maximum value observed in a series of 10 consecutive matrix inputs. Tunable.

Display colorbar

Select to display a bar with the selected colormap to the right of the image axes. Tunable.



Axis origin
The position within the axes where the first element of the input matrix, $U(1,1)$, is plotted; bottom left or top left. Tunable.

X-axis title
The text to be displayed below the x-axis. Tunable.

Y-axis title
The text to be displayed to the left of the y-axis. Tunable.

Colorbar title
The text to be displayed to the right of the color bar, when **Display colorbar** is currently selected. Tunable.

Figure position, [x y width height]
A 4-element vector of the form `[x y width height]` specifying the position of the figure window, where `(0,0)` is the lower-left corner of the display. Tunable.

Axis zoom
Resizes the image to fill the figure window. Tunable.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Spectrum Scope	Signal Processing Blockset
Vector Scope	Signal Processing Blockset
`colormap`	MATLAB

`ColorSpec`	MATLAB
`image`	MATLAB

Purpose

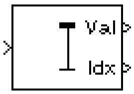
Find maximum values in input or sequence of inputs

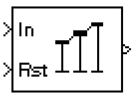
Library

Statistics

`dspstat3`

Description





The Maximum block identifies the value and/or position of the largest element in each row or column of the input, along vectors of a specified dimension of the input, or of the entire input. The Maximum block can also track the maximum values in a sequence of inputs over a period of time. The **Mode** parameter specifies the block's mode of operation and can be set to `Value`, `Index`, `Value and Index`, or `Running`.

The Maximum block supports real and complex floating-point, fixed-point, and Boolean inputs. Real fixed-point inputs can be either signed or unsigned, while complex fixed-point inputs must be signed. The data type of the maximum values output by the block match the data type of the input. The index values output by the block are `double` when the input is `double`, and `uint32` otherwise.

The frame status of the block output is the same as that of the input, except when the **Find the maximum value of** parameter is set to `Entire input`. The output is always sample based when `Entire input` is selected.

For the `Value`, `Index`, and `Value and Index` modes, the Maximum block produces identical results as the MATLAB `max` function when it is called as `[y I] = max(u,[],D)`, where u and y are the input and output, respectively, D is the dimension, and I is the index.

Value Mode

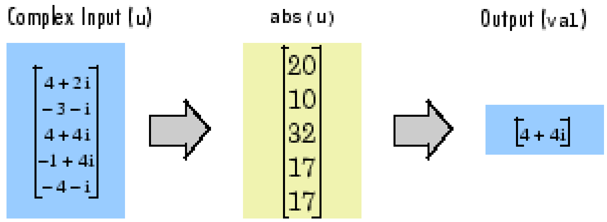
When the **Mode** parameter is set to `Value`, the block computes the maximum value in each row or column of the input, along vectors of a specified dimension of the input, or of the entire input at each sample time, and outputs the array y. Each element in y is the maximum value in the corresponding column, row, vector, or entire input. The output y depends on the setting of the **Find the maximum value**

over parameter. For example, consider a 3-dimensional input signal of size *M*-by-*N*-by-*P*:

- `Each row` — The output at each sample time consists of an *M*-by-1-by-*P* array, where each element contains the maximum value of each vector over the second dimension of the input. For an input that is an *M*-by-*N* matrix, the output at each sample time is an *M*-by-1 column vector. In this mode, the frame status of the output is the same as that of the input.
- `Each column` — The output at each sample time consists of a 1-by-*N*-by-*P* array, where each element contains the maximum value of each vector over the first dimension of the input. For an input that is an *M*-by-*N* matrix, the output at each sample time is a 1-by-*N* row vector. In this mode, the frame status of the output is the same as that of the input.

  For convenience, length-*M* 1-D vector inputs are treated as *M*-by-1 column vectors when the block is in this mode. Sample-based length-*M* row vector inputs are also treated as *M*-by-1 column vectors when the **Treat sample-based row input as a column** check box is selected.
- `Entire input` — The output at each sample time is a scalar that contains the maximum value in the *M*-by-*N*-by-*P* input matrix. In this mode, the block output is always sample based.
- `Specified dimension` — The output at each sample time depends on **Dimension**. If **Dimension** is set to `1`, the output is the same as that when you select `Each column`. If **Dimension** is set to `2`, the output is the same as when you select `Each row`. If **Dimension** is set to `3`, the output at each sample time is an *M*-by-*N* matrix containing the maximum value of each vector over the third dimension of the input. In this mode, the frame status of the output is the same as that of the input.

For complex inputs, the block selects the value in each row or column of the input, along vectors of a specified dimension of the input, or of the entire input that has the maximum magnitude squared as shown below. For complex value $u = a + bi$, the magnitude squared is $a^2 + b^2$.



Index Mode

When **Mode** is set to `Index`, the block computes the maximum value in each row or column of the input, along vectors of a specified dimension of the input, or of the entire input, and outputs the index array I. Each element in I is an integer indexing the maximum value in the corresponding column, row, vector, or entire input. The output I depends on the setting of the **Find the maximum value over** parameter. For example, consider a 3-dimensional input signal of size M-by-N-by-P:

- `Each row` — The output at each sample time consists of an M-by-1-by-P array, where each element contains the index of the maximum value of each vector over the second dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is an M-by-1 column vector. In this mode, the frame status of the output is the same as that of the input.
- `Each column` — The output at each sample time consists of a 1-by-N-by-P array, where each element contains the index of the maximum value of each vector over the first dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is a 1-by-N row vector. In this mode, the frame status of the output is the same as that of the input.

  For convenience, length-M 1-D vector inputs are treated as M-by-1 column vectors when the block is in this mode. Sample-based length-M row vector inputs are also treated as M-by-1 column vectors when the **Treat sample-based row input as a column** check box is selected.

- `Entire input` — The output at each sample time is a 1-by-3 vector that contains the location of the maximum value in the M-by-N-by-P input matrix. In this mode, the block output is always sample based. For an input that is an M-by-N matrix, the output will be a 1-by-2 vector.
- `Specified dimension` — The output at each sample time depends on **Dimension**. If **Dimension** is set to 1, the output is the same as when you select `Each column`. If **Dimension** is set to 2, the output is the same as when you select `Each row`. If **Dimension** is set to 3, the output at each sample time is an M-by-N matrix containing the indices of the maximum values of each vector over the third dimension of the input. In this mode, the frame status of the output is the same as that of the input.

When a maximum value occurs more than once, the computed index corresponds to the first occurrence. For example, when the input is the column vector `[3 2 1 2 3]'`, the computed one-based index of the maximum value is `1` rather than `5` when `Each column` is selected.

When inputs to the block are double-precision values, the index values are double-precision values. Otherwise, the index values are 32-bit unsigned integer values.

Value and Index Mode

When **Mode** is set to `Value and Index`, the block outputs both the maxima and the indices.

Running Mode

When **Mode** is set to `Running`, the block tracks the maximum value of each channel in a time sequence of M-by-N inputs. For sample-based inputs, the output is a sample-based M-by-N array with each element y_{ij} containing the maximum value observed in element u_{ij} for all inputs since the last reset. For frame-based inputs, the output is a frame-based M-by-N matrix with each element y_{ij} containing the maximum value observed in the jth column of all inputs since the last reset, up to and including element u_{ij} of the current input.

N-D signals cannot be frame based. When the block is set to `Running` mode, each element of the N-D signal is treated as a separate channel.

There are $\prod d_i$ channels, where d_i is the size of the ith dimension.

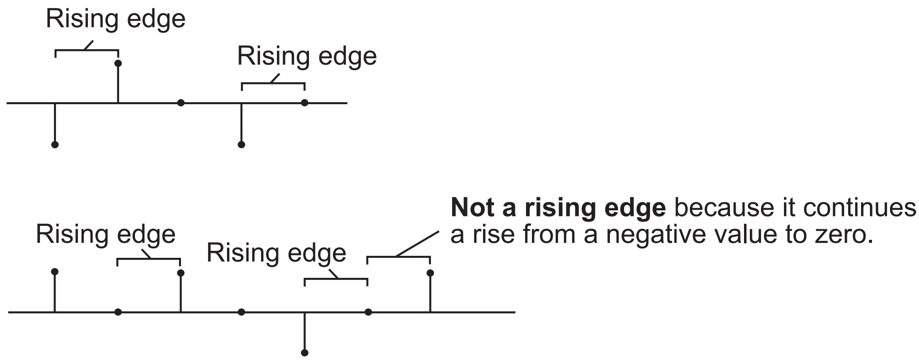
Resetting the Running Maximum

The block resets the running maximum whenever a reset event is detected at the optional Rst port. The reset sample time must be a positive integer multiple of the input sample time.

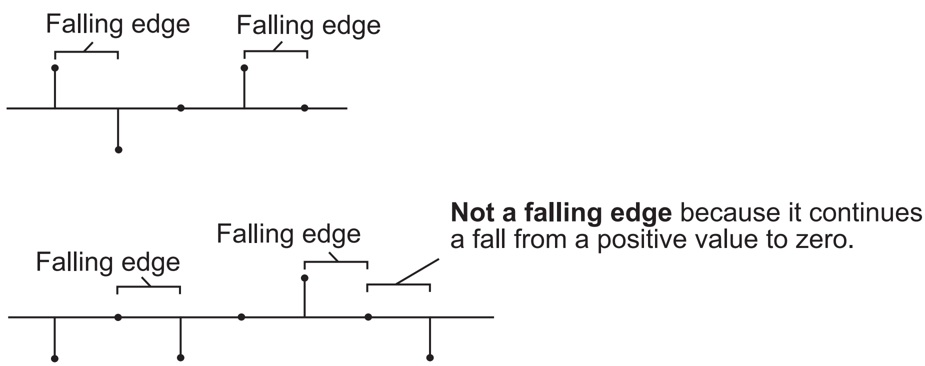
For sample-based inputs, a reset event causes the running maximum for each channel to be initialized to the value in the corresponding channel of the current input. For frame-based inputs, a reset event causes the running maximum for each channel to be initialized to the earliest value in each channel of the current input.

You specify the reset event in the **Reset port** menu:

- `None` — Disables the Rst port.
- `Rising edge` — Triggers a reset operation when the Rst input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)



- `Falling edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)



- `Either edge` — Triggers a reset operation when the `Rst` input is a `Rising edge` or `Falling edge` (as described above)
- `Non-zero sample` — Triggers a reset operation at each sample time that the `Rst` input is not zero

---

Note When running simulations in the Simulink `MultiTasking` mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset. For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

ROI Processing

To calculate the statistical value within a particular region of interest (ROI) of the input, select the **Enable ROI processing** check box. This option is only available when the **Find the maximum value over** parameter is set to `Entire input` and the **Enable ROI processing** check box is selected. ROI processing is only supported for 2-D inputs.

Note Full ROI processing is only available to users who have a Video and Image Processing Blockset license. If you only have a Signal Processing Blockset license, you can still use ROI processing, but are limited to the **ROI type** `Rectangles`.

Use the **ROI type** parameter to specify whether the ROI is a rectangle, line, label matrix, or binary mask. A binary mask is a binary image that enables you to specify which pixels to highlight, or select. In a label matrix, pixels equal to 0 represent the background, pixels equal to 1 represent the first object, pixels equal to 2 represent the second object, and so on. When the **ROI type** parameter is set to `Label matrix`, the Label and Label Numbers ports appear on the block. Use the Label Numbers port to specify the objects in the label matrix for which the block calculates statistics. The input to this port must be a vector of scalar values that correspond to the labeled regions in the label matrix. For more information about the format of the input to the ROI port when the ROI is a rectangle or a line, see the Draw Shapes block reference page.

For rectangular ROIs, use the **ROI portion to process** parameter to specify whether to calculate the statistical value for the entire ROI or just the ROI perimeter.

Use the **Output** parameter to specify the block output. The block can output separate statistical values for each ROI or the statistical value for all specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select `Binary mask`.

If, for the **ROI type** parameter, you select `Rectangles` or `Lines`, the **Output flag indicating if ROI is within image bounds** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual statistics for each ROI

Flag Port Output	Description
0	ROI is completely outside the input image.
1	ROI is completely or partially inside the input image.

Output = Single statistic for all ROIs

Flag Port Output	Description
0	All ROIs are completely outside the input image.
1	At least one ROI is completely or partially inside the input image.

If the ROI is partially outside the image, the block only computes the statistical values for the portion of the ROI that is within the image.

If, for the **ROI type** parameter, you select `Label matrix`, the **Output flag indicating if input label numbers are valid** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual statistics for each ROI

Flag Port Output	Description
0	Label number is not in the label matrix.
1	Label number is in the label matrix.

Output = Single statistic for all ROIs

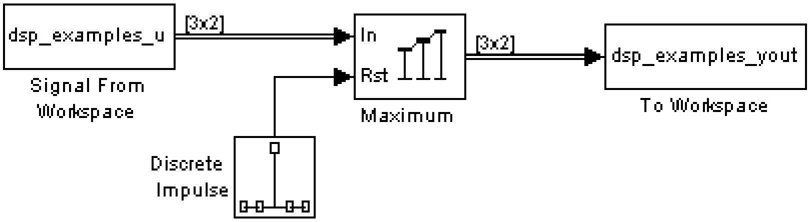
Flag Port Output	Description
0	None of the label numbers are in the label matrix.
1	At least one of the label numbers is in the label matrix.

Fixed-Point Data Types

The parameters on the **Data Types** pane of the block dialog are only used for complex fixed-point inputs. The sum of the squares of the real and imaginary parts of such an input are formed before a comparison is made, as described in "Value Mode" on page 2-879. The results of the squares of the real and imaginary parts are placed into the product output data type. The result of the sum of the squares is placed into the accumulator data type. These parameters are ignored for other types of inputs.

Examples

The Maximum block in the following model calculates the running maximum of a frame-based 3-by-2 (two-channel) matrix input, `dsp_examples_u`. The running maximum is reset at t=2 by an impulse to the block's Rst port.



The Maximum block has the following settings:

- **Mode** = `Running`
- **Reset port** = `Non-zero sample`

The Signal From Workspace block has the following settings:

- **Signal** = `dsp_examples_u`
- **Sample time** = `1/3`
- **Samples per frame** = `3`

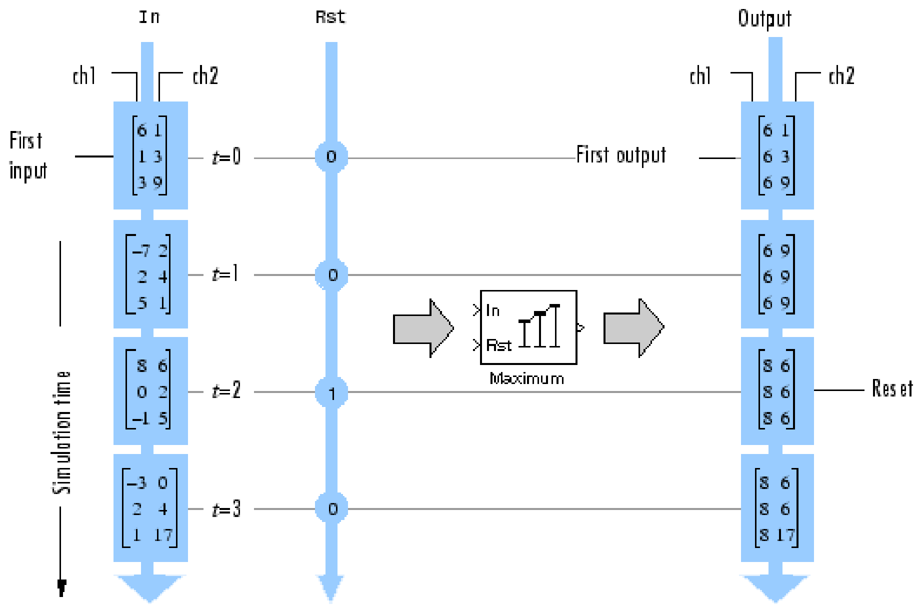
where

```
dsp_examples_u
= [6 1 3 -7 2 5 8 0 -1 -3 2 1;1 3 9 2 4 1 6 2 5 0 4 17]'
```

The Discrete Impulse block has the following settings:

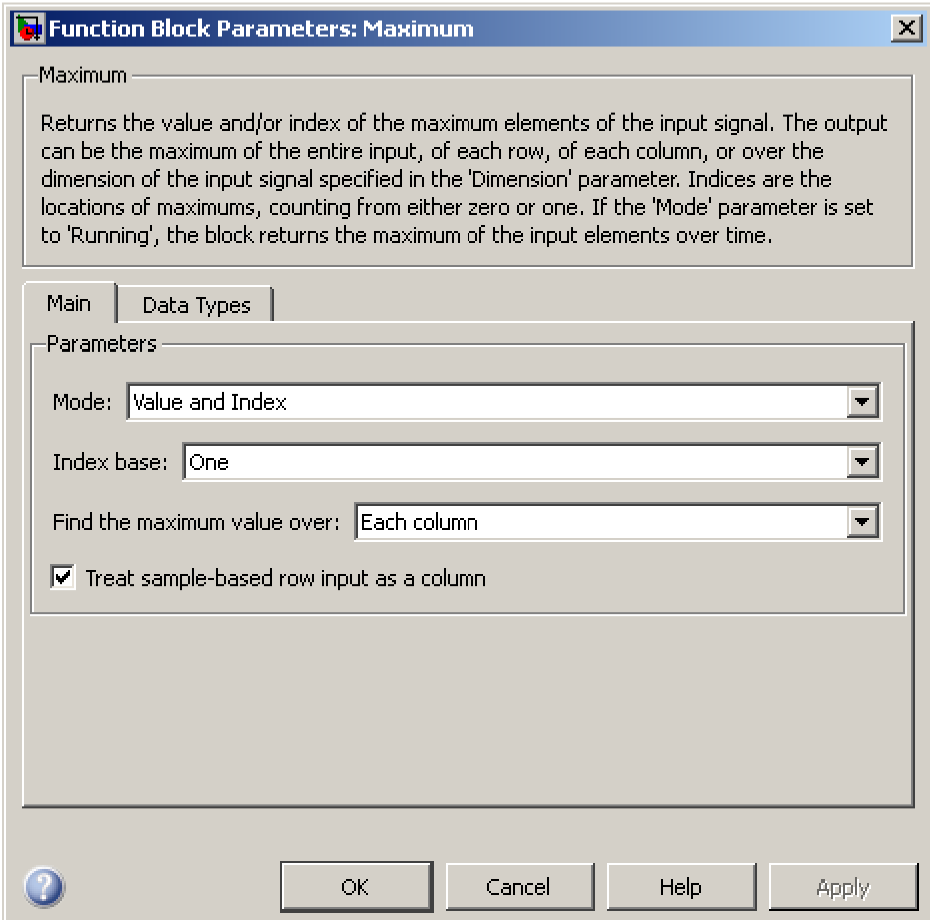
- **Delay (samples)** = `2`
- **Sample time** = `1`
- **Samples per frame** = `1`

The block's operation is shown in the figure below.


In
Rst
Output
ch1
ch2
ch1
ch2
First input
First output
Reset
Simulation time
t=0
t=1
t=2
t=3
Maximum
In
Rst

Dialog Box

The **Main** pane of the Maximum block dialog appears as follows.



Mode

Specify the block's mode of operation:

- `Value` — Output the maximum value of each input
- `Index` — Output the index of the maximum value

- `Value and index` — Output both the value and the index
- `Running` — Track the maximum value of the input sequence over time

For more information, see Description.

Index base
Specify whether the index of the maximum value is reported using one-based or zero-based numbering. This parameter is only visible when the **Mode** parameter is set to `Index` or `Value and index`.

Find the maximum value over
Specify whether to find the maximum value along rows, columns, entire input, or the dimension specified in the **Dimension** parameter. For more information, see Description.

Treat sample-based row input as a column
Select to treat sample-based length-M row vector inputs as M-by-1 column vectors. This parameter is only visible when the **Find the maximum value of** parameter is set to `Each column`.

Reset port
Specify the reset event detected at the Rst input port when you select `Running` for the **Mode** parameter. The rate of the reset signal must be a positive integer multiple of the rate of the data signal input. For information about the possible values of this parameter, see "Resetting the Running Maximum" on page 2-883.

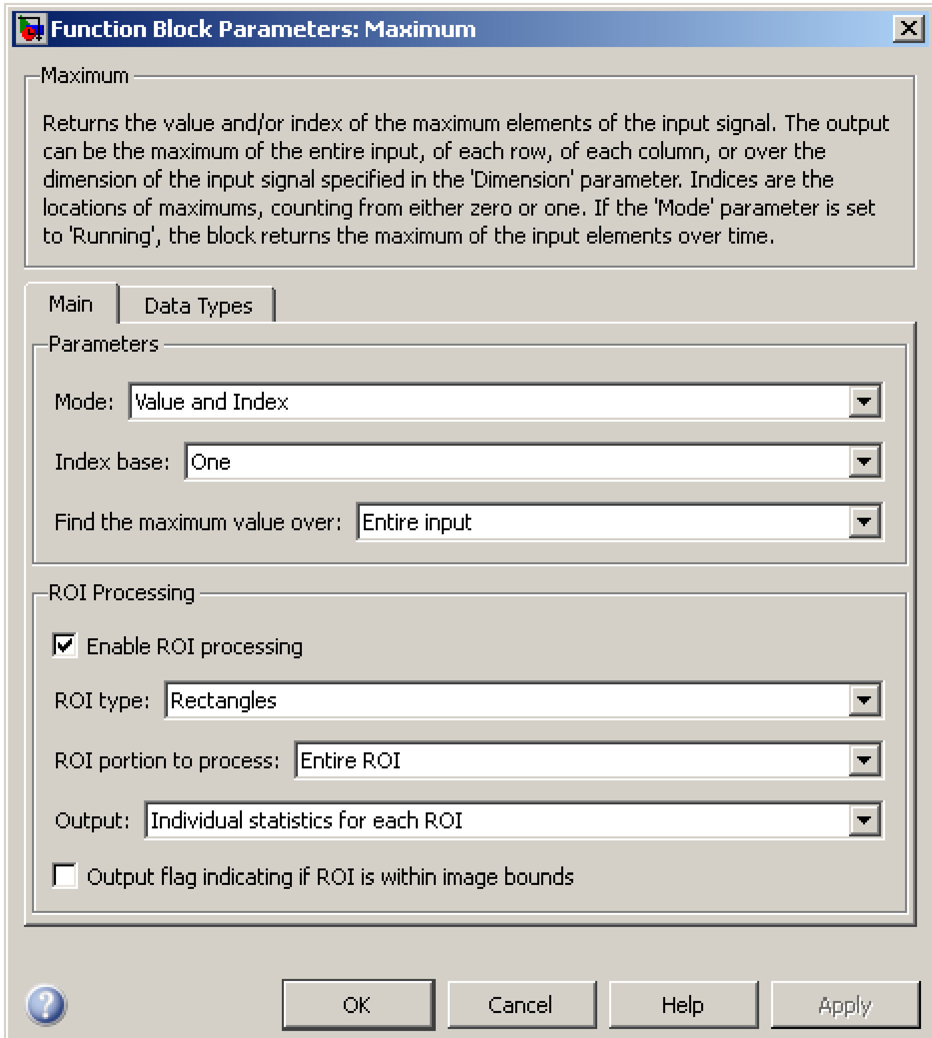
Dimension
Specify the dimension (one-based value) of the input signal, over which the maximum is computed. The value of this parameter cannot exceed the number of dimensions in the input signal. This parameter is only visible when the **Find the maximum value over** parameter is set to `Specified dimension`.

Enable ROI processing
Select this check box to calculate the statistical value within a particular region of each image. This parameter appears only

when you set the **Find the maximum value over** parameter to `Entire input`, and the block is not in running mode.

Note Full ROI processing is only available to users who have a Video and Image Processing Blockset license. If you only have a Signal Processing Blockset license, you can still use ROI processing, but are limited to the **ROI type** `Rectangles`.

The ROI processing parameters appear on the dialog box as follows.



ROI type

Specify the type of ROI you want to use. Your choices are `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`.

ROI portion to process

Specify whether you want to calculate the statistical value for the entire ROI or just the ROI perimeter. This parameter appears only if you specify an **ROI type** of `Rectangles`.

Output

Specify the block output. The block can output a vector of separate statistical values for each ROI or a scalar value that represents the statistical value for all the specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select `Binary mask`.

Output flag

☑ **Output flag indicating if ROI is within image bounds**

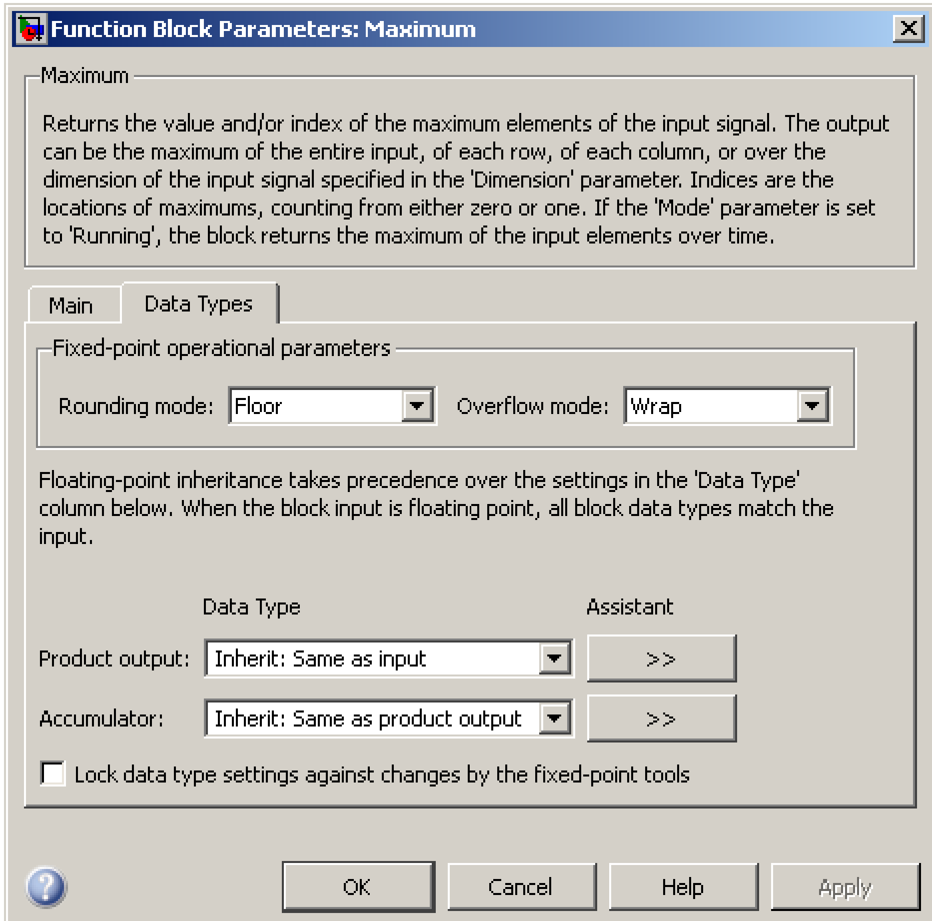
☑ **Output flag indicating if label numbers are valid**

When you select either of these check boxes, the Flag port appears on the block. For a description of the Flag port output, see the tables in "ROI Processing" on page 2-885.

The **Output flag indicating if ROI is within image bounds** check box is only visible when you select `Rectangles` or `Lines` as the **ROI type**.

The **Output flag indicating if label numbers are valid** check box is only visible when you select `Label matrix` for the **ROI type** parameter.

The **Data Types** pane of the Maximum block dialog appears as follows.


Function Block Parameters: Maximum
Maximum
Returns the value and/or index of the maximum elements of the input signal. The output can be the maximum of the entire input, of each row, of each column, or over the dimension of the input signal specified in the 'Dimension' parameter. Indices are the locations of maximums, counting from either zero or one. If the 'Mode' parameter is set to 'Running', the block returns the maximum of the input elements over time.
Main
Data Types
Fixed-point operational parameters
Rounding mode: Floor
Overflow mode: Wrap
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input.
Data Type
Assistant
Product output: Inherit: Same as input
>>
Accumulator: Inherit: Same as product output
>>
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Note The parameters on the **Data Types** pane are only used for complex fixed-point inputs. The sum of the squares of the real and imaginary parts of such an input are formed before a comparison is made, as described in "Value Mode" on page 2-879. The results of the squares of the real and imaginary parts are placed into the product output data type. The result of the sum of the squares is placed into the accumulator data type. These parameters are ignored for other types of inputs.

Rounding mode
Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode for fixed-point operations.

Product output data type
Specify the product output data type. See "Fixed-Point Data Types" on page 2-887 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Accumulator data type

Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-887 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same as product output`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Reset	• Double-precision floating point • Single-precision floating point • Boolean

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Idx	• Double-precision floating point • 32-bit unsigned integers
Val	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

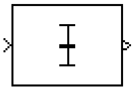
Mean	Signal Processing Blockset
Minimum	Signal Processing Blockset
MinMax	Simulink
`max`	MATLAB

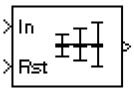
Purpose Find mean value of input or sequence of inputs

Library Statistics

`dspstat3`

Description





The Mean block computes the mean of each row or column of the input, along vectors of a specified dimension of the input, or of the entire input. The Mean block can also track the mean value in a sequence of inputs over a period of time. The **Running mean** parameter selects between basic operation and running operation.

The Mean block accepts real and complex fixed-point and floating-point inputs.

Basic Operation

When you do not select the **Running mean** check box, the block computes the mean value in each row or column of the input, along vectors of a specified dimension of the input, or of the entire input at each individual sample time. Each element in the output array `y` is the mean value of the corresponding column, row, vector, or entire input. The output array `y` depends on the setting of the **Find the mean value over** parameter. For example, consider a 3-dimensional input signal of size M-by-N-by-P:

- `Entire input` — The output at each sample time is a scalar that contains the mean value of the M-by-N-by-P input matrix. In this mode, the block output is always sample based.

  ```
  y = mean(u(:))     % Equivalent MATLAB code
  ```

- `Each row` — The output at each sample time consists of an M-by-1-by-P array, where each element contains the mean value of each vector over the second dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is an M-by-1 column vector. In this mode, the frame status of the output is the same as that of the input.

```
y = mean(u,2)     % Equivalent MATLAB code
```

- `Each column` — The output at each sample time consists of a 1-by-N-by-P array, where each element contains the mean value of each vector over the first dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is a 1-by-N row vector. In this mode, the frame status of the output is the same as that of the input.

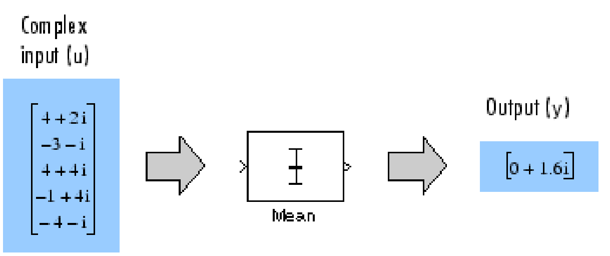
  ```
  y = mean(u)     % Equivalent MATLAB code
  ```

  For convenience, length-M 1-D vector inputs are treated as M-by-1 column vectors when the block is in this mode. Sample-based length-M row vector inputs are also treated as M-by-1 column vectors when the **Treat sample-based row input as a column** check box is selected.

- `Specified dimension` — The output at each sample time depends on **Dimension**. If **Dimension** is set to `1`, the output is the same as when you select `Each column`. If **Dimension** is set to `2`, the output is the same as when you select `Each row`. If **Dimension** is set to `3`, the output at each sample time is an M-by-N matrix containing the mean value of each vector over the third dimension of the input. In this mode, the frame status of the output is the same as that of the input.

  ```
  y = mean(u,Dimension)     % Equivalent MATLAB code
  ```

The mean of a complex input is computed independently for the real and imaginary components, as shown in the next figure.



Running Operation

When the **Running mean** check box is selected, the block tracks the mean value of each channel in a time sequence of inputs. For sample-based M-by-N inputs, the output is a sample-based M-by-N array with each element y_{ij} containing the mean value of the elements u_{ij} for all inputs since the last reset. For frame-based M-by-N inputs, the output is a frame-based M-by-N matrix with each element y_{ij} containing the mean of the values in the jth column of all inputs since the last reset, up to and including element u_{ij} of the current input.

N-D signals cannot be frame based. When the block is set to `Running` mode, each element of the N-D signal is treated as a separate channel.

There are $\prod d_i$ channels, where d_i is the size of the ith dimension.

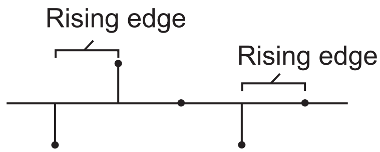
Resetting the Running Mean

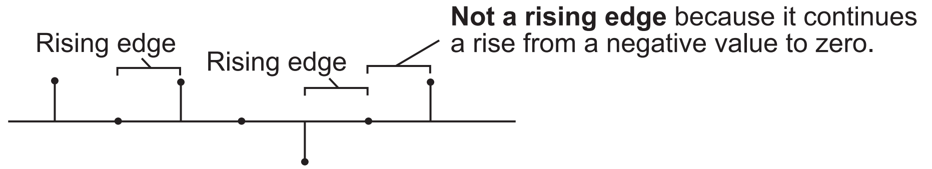
The block resets the running mean whenever a reset event is detected at the optional Rst port. The reset sample time must be a positive integer multiple of the input sample time.

When the block is reset for sample-based inputs, the running mean for each channel is initialized to the value in the corresponding channel of the current input. For frame-based inputs, the running mean for each channel is initialized to the earliest value in each channel of the current input.

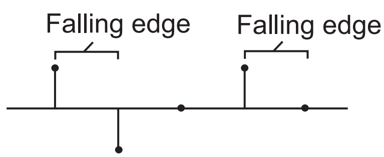
You specify the reset event by the **Reset port** parameter:

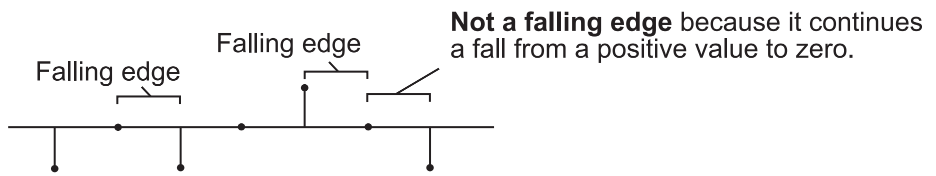
- `None` disables the Rst port.
- `Rising edge` — Triggers a reset operation when the Rst input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)





- `Falling edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)





- `Either edge` — Triggers a reset operation when the `Rst` input is a `Rising edge` or `Falling edge` (as described earlier)
- `Non-zero sample` — Triggers a reset operation at each sample time that the `Rst` input is not zero

Note When running simulations in the Simulink MultiTasking mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset. For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

ROI Processing

To calculate the statistical value within a particular region of interest (ROI) of the input, select the **Enable ROI processing** check box. This option is only available when the **Find the mean value over** parameter is set to `Entire input` and the **Running mean** check box is not selected. ROI processing is only supported for 2-D inputs.

Note Full ROI processing is only available to users who have a Video and Image Processing Blockset license. If you only have a Signal Processing Blockset license, you can still use ROI processing, but are limited to the **ROI type** `Rectangles`.

Use the **ROI type** parameter to specify whether the ROI is a rectangle, line, label matrix, or binary mask. A binary mask is a binary image that enables you to specify which pixels to highlight, or select. In a label matrix, pixels equal to 0 represent the background, pixels equal to 1 represent the first object, pixels equal to 2 represent the second object, and so on. When the **ROI type** parameter is set to `Label matrix`, the Label and Label Numbers ports appear on the block. Use the Label Numbers port to specify the objects in the label matrix for which the block calculates statistics. The input to this port must be a vector of scalar values that correspond to the labeled regions in the label matrix. For more information about the format of the input to the ROI port when the ROI is a rectangle or a line, see the Draw Shapes block reference page.

For rectangular ROIs, use the **ROI portion to process** parameter to specify whether to calculate the statistical value for the entire ROI or just the ROI perimeter.

Use the **Output** parameter to specify the block output. The block can output separate statistical values for each ROI or the statistical value for all specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select `Binary mask`.

If, for the **ROI type** parameter, you select `Rectangles` or `Lines`, the **Output flag indicating if ROI is within image bounds** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual statistics for each ROI

Flag Port Output	Description
0	ROI is completely outside the input image.
1	ROI is completely or partially inside the input image.

Output = Single statistic for all ROIs

Flag Port Output	Description
0	All ROIs are completely outside the input image.
1	At least one ROI is completely or partially inside the input image.

If the ROI is partially outside the image, the block only computes the statistical values for the portion of the ROI that is within the image.

If, for the **ROI type** parameter, you select `Label matrix`, the **Output flag indicating if input label numbers are valid** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual statistics for each ROI

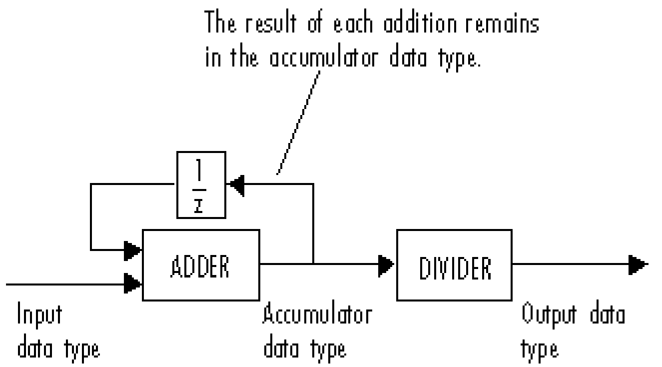
Flag Port Output	Description
0	Label number is not in the label matrix.
1	Label number is in the label matrix.

Output = Single statistic for all ROIs

Flag Port Output	Description
0	None of the label numbers are in the label matrix.
1	At least one of the label numbers is in the label matrix.

Fixed-Point Data Types

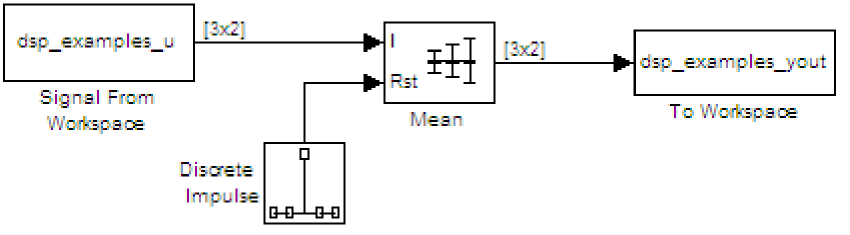
The following diagram shows the data types used within the Mean block for fixed-point signals.



You can set the accumulator and output data types in the block dialog, as discussed in "Dialog Box" on page 2-908.

Examples

The Mean block in the following model calculates the running mean of a frame-based 3-by-2 (two-channel) matrix input, u. The running mean is reset at t=2 by an impulse to the block's Rst port.



The Mean block has the following settings:

- **Running mean** = Select this check box
- **Reset port** = `Non-zero sample`

The Signal From Workspace block has the following settings:

- **Signal** = `dsp_examples_u`

- **Sample time** = 1/3
- **Samples per frame** = 3

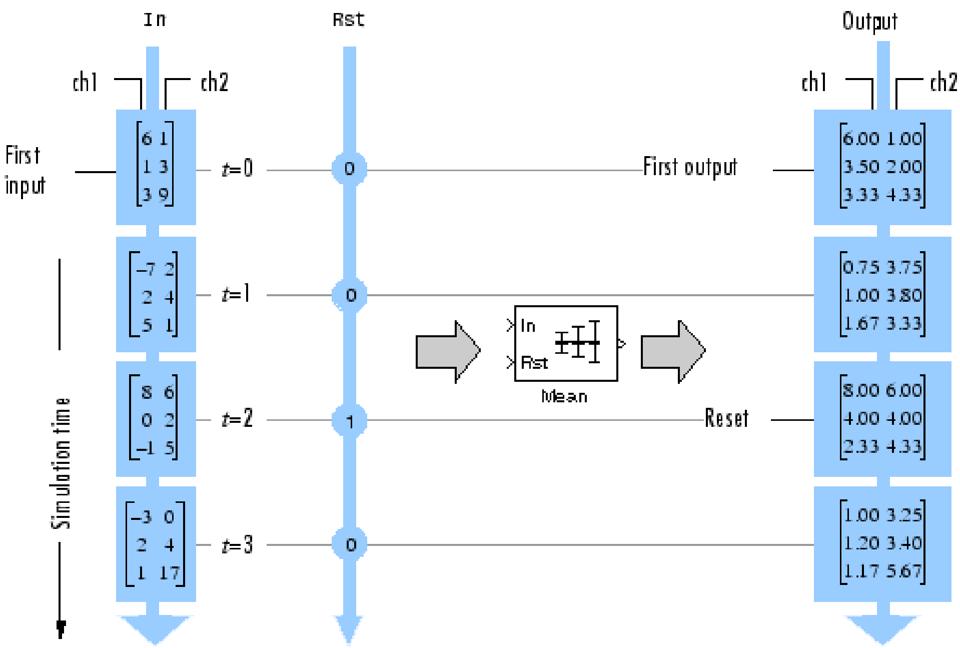
where

```
dsp_examples_u = [6 1 3 -7 2 5 8 0 -1 -3 2 1;1 3 9 2 4 1 6 2 5 0 4 17]'
```

The Discrete Impulse block has the following settings:

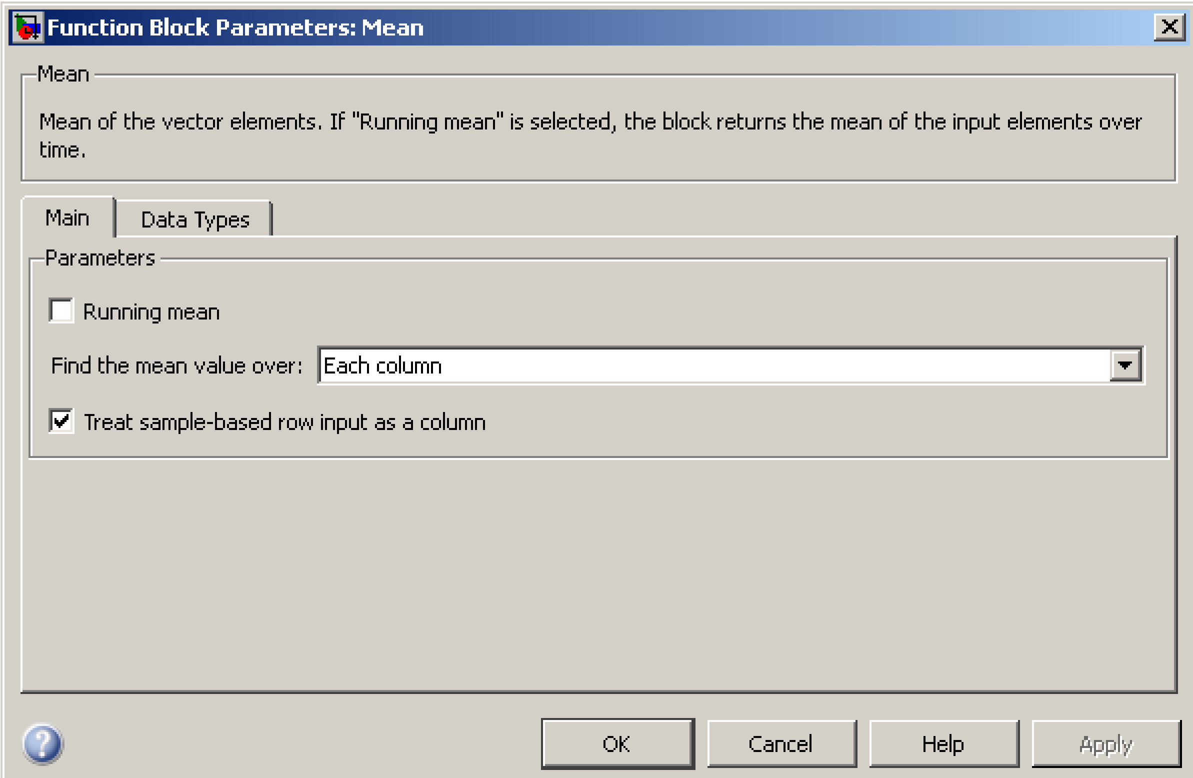
- **Delay (samples)** = 2
- **Sample time** = 1
- **Samples per frame** = 1

The block's operation is shown in the next figure.



Dialog Box

The **Main** pane of the Mean block dialog appears as follows.



Running mean
: Enables running operation when selected.

Reset port
: Determines the reset event that causes the block to reset the running mean. The rate of the reset signal must be a positive integer multiple of the rate of the data signal input. This parameter is enabled only when you set the **Running mean**

parameter. For more information, see “Resetting the Running Mean” on page 2-901.

Find the mean value over

Specify whether to find the mean value along rows, columns, entire input, or the dimension specified in the **Dimension** parameter. For more information, see “Basic Operation” on page 2-899.

Treat sample-based row input as a column

Select to treat sample-based length-M row vector inputs as M-by-1 column vectors. This parameter is only visible when the **Find the mean value over** parameter is set to `Each column`.

Dimension

Specify the dimension (one-based value) of the input signal, over which the mean is computed. The value of this parameter cannot exceed the number of dimensions in the input signal. This parameter is only visible when the **Find the mean value over** parameter is set to `Specified dimension`.

Enable ROI Processing

Select this check box to calculate the statistical value within a particular region of each image. This parameter is only available when the **Find the mean value over** parameter is set to `Entire input`, and the block is not in running mode.

Note Full ROI processing is only available to users who have a Video and Image Processing Blockset license. If you only have a Signal Processing Blockset license, you can still use ROI processing, but are limited to the **ROI type** `Rectangles`.

ROI type

Specify the type of ROI you want to use. Your choices are `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`.

ROI portion to process

Specify whether you want to calculate the statistical value for the entire ROI or just the ROI perimeter. This parameter is only visible if, for the **ROI type** parameter, you specify `Rectangles`.

Output

Specify the block output. The block can output a vector of separate statistical values for each ROI or a scalar value that represents the statistical value for all the specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select `Binary mask`.

Output flag

☑ **Output flag indicating if ROI is within image bounds**

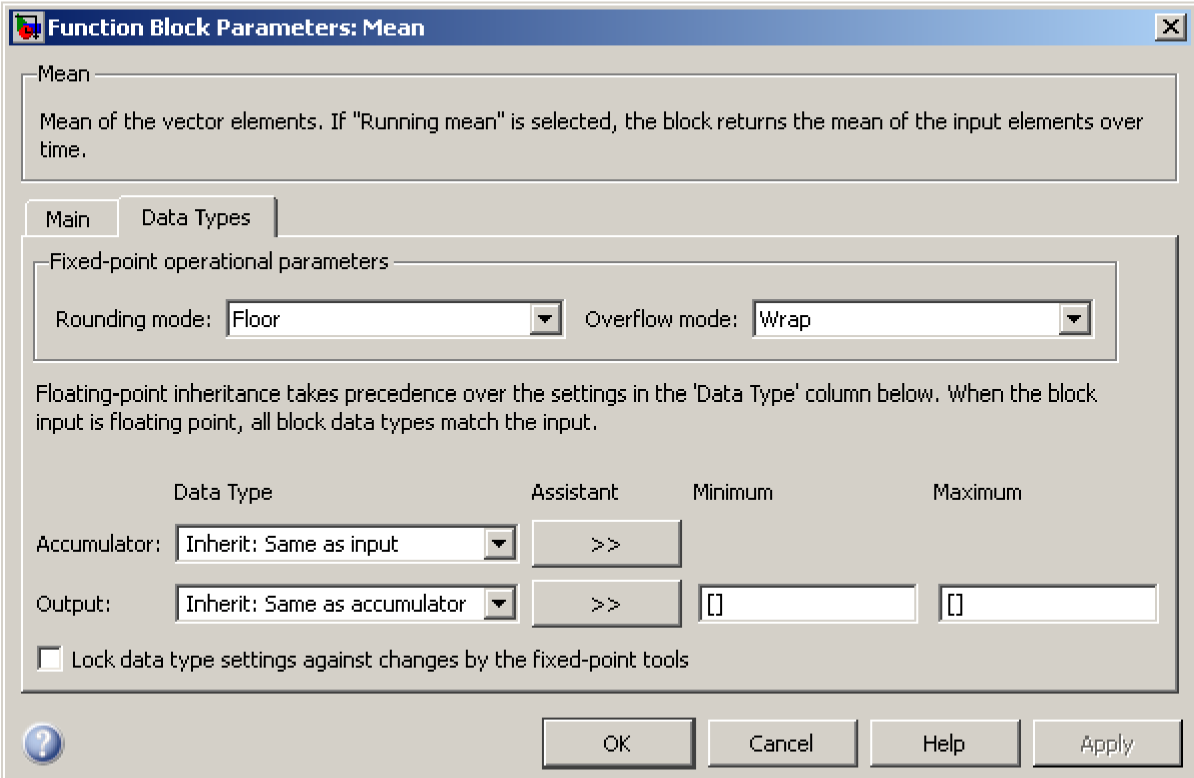
☑ **Output flag indicating if label numbers are valid**

When you select either of these check boxes, the Flag port appears on the block. For a description of the Flag port output, see the tables in "ROI Processing" on page 2-903.

The **Output flag indicating if ROI is within image bounds** check box is only visible when you select `Rectangles` or `Lines` as the **ROI type**.

The **Output flag indicating if label numbers are valid** check box is only visible when you select `Label matrix` for the **ROI type** parameter.

The **Data Types** pane of the Mean block dialog appears as follows.



Rounding mode
Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode for fixed-point operations.

Accumulator data type
Specify the accumulator data type. See "Fixed-Point Data Types" on page 2-905 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same as input`

- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Output data type

Specify the output data type. See “Fixed-Point Data Types” on page 2-905 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as accumulator`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User’s Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see “Checking Signal Ranges”)
- Automatic scaling of fixed-point data types

Maximum
Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Reset	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
ROI	Rectangles and lines: • Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers Binary Mask: • Boolean
Label	• 8-, 16-, and 32-bit unsigned integers
Label Numbers	• 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Flag	• Boolean

See Also

Maximum	Signal Processing Blockset
Median	Signal Processing Blockset
Minimum	Signal Processing Blockset
Standard Deviation	Signal Processing Blockset
`mean`	MATLAB

Purpose

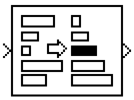
Find median value of input

Library

Statistics

`dspstat3`

Description



The Median block computes the median value of each row or column of the input, along vectors of a specified dimension of the input, or of the entire input. The median of a set of input values is calculated as follows:

1. The values are sorted.
2. If the number of values is odd, the median is the middle value.
3. If the number of values is even, the median is the average of the two middle values.

For a given input u, the size of the output array `y` depends on the setting of the **Find the median value over** parameter. For example, consider a 3-dimensional input signal of size M-by-N-by-P:

- `Entire input` — The output at each sample time is a scalar that contains the median value of the M-by-N-by-P input matrix. In this mode, the block output is always sample based.

  ```
  y = median(u(:))     % Equivalent MATLAB code
  ```

- `Each row` — The output at each sample time consists of an M-by-1-by-P array, where each element contains the median value of each vector over the second dimension of the input. For an input that is an M-by-N matrix, the output is an M-by-1 column vector. In this mode, the frame status of the output is the same as that of the input.

  ```
  y = median(u,2)     % Equivalent MATLAB code
  ```

- `Each column` — The output at each sample time consists of a 1-by-N-by-P array, where each element contains the median value of each vector over the first dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is a 1-by-N row vector. In this mode, the frame status of the output is the same as that of the input.

```
y = median(u)     % Equivalent MATLAB code
```

For convenience, length-M 1-D vector inputs are treated as M-by-1 column vectors when the block is in this mode. Sample-based length-M row vector inputs are also treated as M-by-1 column vectors when the **Treat sample-based row input as a column** check box is selected.

- `Specified dimension` — The output at each sample time depends on **Dimension**. If **Dimension** is set to `1`, the output is the same as when you select `Each column`. If **Dimension** is set to `2`, the output is the same as when you select `Each row`. If **Dimension** is set to `3`, the output at each sample time is an M-by-N matrix containing the median value of each vector over the third dimension of the input. In this mode, the frame status of the output is the same as that of the input.

  ```
  y = median(u,Dimension)     % Equivalent MATLAB code
  ```

Complex inputs are sorted by magnitude squared. For complex value $u = a + bi$, the magnitude squared is a^2+b^2.

The Median block accepts real and complex fixed-point and floating-point inputs.

Fixed-Point Data Types

For fixed-point inputs, you can specify accumulator, product output, and output data types as discussed in "Dialog Box" on page 2-917. Not all these fixed-point parameters are applicable for all types of fixed-point inputs. The following table shows when each kind of data type and scaling is used.

	Output data type	Accumulator data type	Product output data type
Even *M*	X	X	
Odd *M*	X		
Odd *M* and complex	X	X	X
Even *M* and complex	X	X	X

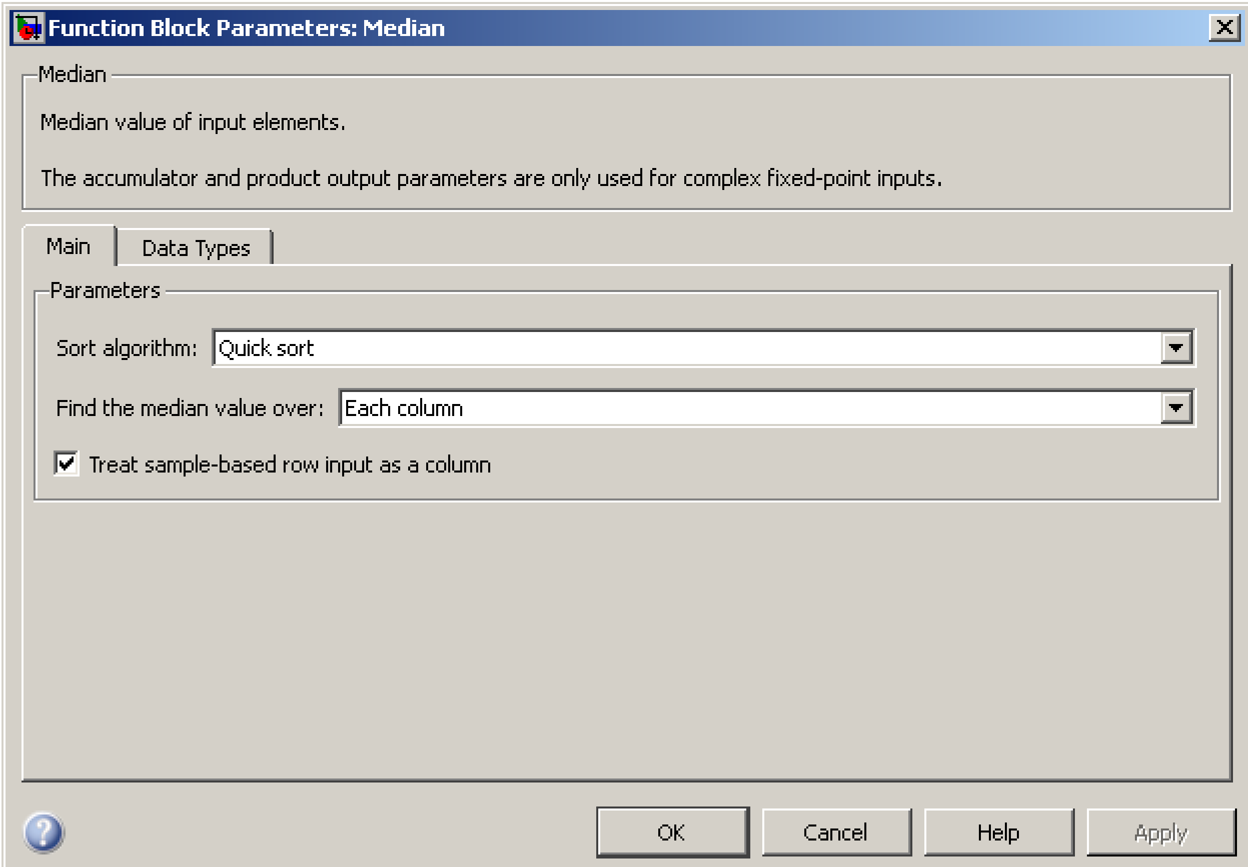
The accumulator and output data types and scalings are used for fixed-point signals when *M* is even. The result of the sum performed while calculating the average of the two central rows of the input matrix is stored in the accumulator data type and scaling. The total result of the average is then put into the output data type and scaling.

The accumulator and product output parameters are used for complex fixed-point inputs. The sum of the squares of the real and imaginary parts of such an input are formed before the input elements are sorted, as described in Description. The results of the squares of the real and imaginary parts are placed into the product output data type and scaling. The result of the sum of the squares is placed into the accumulator data type and scaling.

For fixed-point inputs that are both complex and have even *M*, the data types are used in all of the ways described. Therefore, in such cases, the accumulator type is used in two different ways.

Dialog Box

The **Main** pane of the Median block dialog appears as follows.



Sort algorithm

Specify whether to sort the elements of the input using a `Quick sort` or an `Insertion sort` algorithm.

Find the median value over

Specify whether to find the median value along rows, columns, entire input, or the dimension specified in the **Dimension** parameter. For more information, see Description.

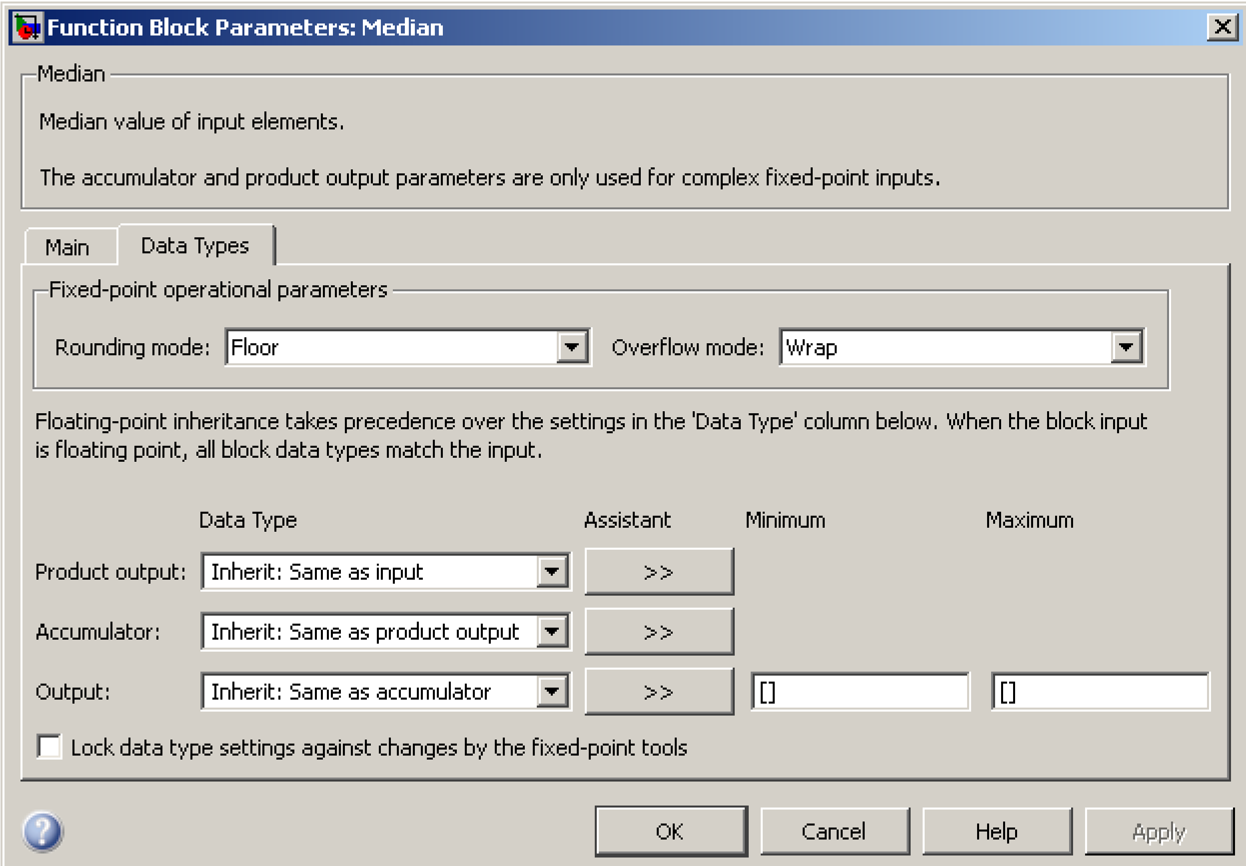
Treat sample-based row input as a column

Select to treat sample-based length-M row vector inputs as M-by-1 column vectors. This parameter is only visible when the **Find the median value over** parameter is set to `Each column`.

Dimension

Specify the dimension (one-based value) of the input signal, over which the median is computed. The value of this parameter cannot exceed the number of dimensions in the input signal. This parameter is only visible when the **Find the median value over** parameter is set to `Specified dimension`.

The **Data Types** pane of the Median block dialog appears as follows.



Note Floating-point inheritance takes precedence over the data type settings defined on this pane. When inputs are floating point, the block ignores these settings, and all internal data types are floating point.

Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode for fixed-point operations.

Product output data type
Specify the product output data type. See “Fixed-Point Data Types” on page 2-916 and “Multiplication Data Types” for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Accumulator data type
Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-916 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same as product output`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Output data type

Specify the output data type. See "Fixed-Point Data Types" on page 2-916 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as accumulator`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, 32-, and 128-bit signed integers • 8-, 16-, 32-, and 128-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, 32-, and 128-bit signed integers • 8-, 16-, 32-, and 128-bit unsigned integers

See Also

Maximum	Signal Processing Blockset
Mean	Signal Processing Blockset
Minimum	Signal Processing Blockset
Sort	Signal Processing Blockset
Standard Deviation	Signal Processing Blockset
Variance	Signal Processing Blockset
`median`	MATLAB

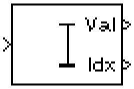
Minimum

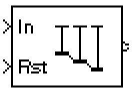
Purpose Find minimum values in input or sequence of inputs

Library Statistics

`dspstat3`

Description





The Minimum block identifies the value and/or position of the smallest element in each row or column of the input, along vectors of a specified dimension of the input, or of the entire input. The Minimum block can also track the minimum values in a sequence of inputs over a period of time. The **Mode** parameter specifies the block's mode of operation, and can be set to `Value`, `Index`, `Value and Index`, or `Running`.

The Minimum block supports real and complex floating-point, fixed-point, and Boolean inputs. Real fixed-point inputs can be either signed or unsigned, while complex fixed-point inputs must be signed. The data type of the minimum values output by the block match the data type of the input. The index values output by the block are `double` when the input is `double`, and `uint32` otherwise.

The frame status of the block output is the same as that of the input, except when the **Find the minimum value of** parameter is set to `Entire input`. The output is always sample-based when `Entire input` is selected.

For the `Value`, `Index`, and `Value and Index` modes, the Minimum block produces identical results as the MATLAB `min` function when it is called as `[y I] = min(u,[],D)`, where u and y are the input and output, respectively, D is the dimension, and I is the index.

Value Mode

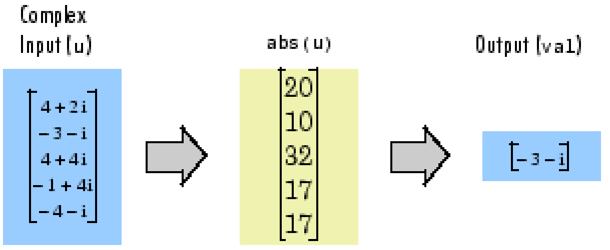
When the **Mode** parameter is set to `Value`, the block computes the minimum value in each row or column of the input, along vectors of a specified dimension of the input, or of the entire input at each sample time, and outputs the array y. Each element in y is the minimum value in the corresponding column, row, vector, or entire input. The output y depends on the setting of the **Find the minimum value**

over parameter. For example, consider a 3-dimensional input signal of size *M*-by-*N*-by-*P*:

- `Each row` — The output at each sample time consists of an *M*-by-1-by-*P* array, where each element contains the minimum value of each vector over the second dimension of the input. For an input that is an *M*-by-*N* matrix, the output at each sample time is an *M*-by-1 column vector. In this mode, the frame status of the output is the same as that of the input.
- `Each column` — The output at each sample time consists of a 1-by-*N*-by-*P* array, where each element contains the minimum value of each vector over the first dimension of the input. For an input that is an *M*-by-*N* matrix, the output at each sample time is a 1-by-*N* row vector. In this mode, the frame status of the output is the same as that of the input.

  For convenience, length-*M* 1-D vector inputs are treated as *M*-by-1 column vectors when the block is in this mode. Sample-based length-*M* row vector inputs are also treated as *M*-by-1 column vectors when the **Treat sample-based row input as a column** check box is selected.
- `Entire input` — The output at each sample time is a scalar that contains the minimum value in the *M*-by-*N*-by-*P* input matrix. In this mode, the block output is always sample based.
- `Specified dimension` — The output at each sample time depends on **Dimension**. If **Dimension** is set to `1`, the output is the same as when you select `Each column`. If **Dimension** is set to `2`, the output is the same as when you select `Each row`. If **Dimension** is set to `3`, the output at each sample time is an *M*-by-*N* matrix containing the minimum value of each vector over the third dimension of the input. In this mode, the frame status of the output is the same as that of the input.

For complex inputs, the block selects the value in each row or column of the input, along vectors of a specified dimension of the input, or of the entire input that has the minimum magnitude squared as shown below. For complex value $u = a + bi$, the magnitude squared is $a^2 + b^2$.



Index Mode

When **Mode** is set to `Index`, the block computes the minimum value in each row or column of the input, along vectors of a specified dimension of the input, or of the entire input, and outputs the index array I. Each element in I is an integer indexing the minimum value in the corresponding column, row, vector, or entire input. The output I depends on the setting of the **Find the minimum value over** parameter. For example, consider a 3-dimensional input signal of size M-by-N-by-P:

- `Each row` — The output at each sample time consists of an M-by-1-by-P array, where each element contains the index of the minimum value of each vector over the second dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is an M-by-1 column vector. In this mode, the frame status of the output is the same as that of the input.
- `Each column` — The output at each sample time consists of a 1-by-N-by-P array, where each element contains the index of the minimum value of each vector over the first dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is a 1-by-N row vector. In this mode, the frame status of the output is the same as that of the input.

  For convenience, length-M 1-D vector inputs are treated as M-by-1 column vectors when the block is in this mode. Sample-based length-M row vector inputs are also treated as M-by-1 column vectors when the **Treat sample-based row input as a column** check box is selected.

- `Entire input` — The output at each sample time is a 1-by-3 vector that contains the location of the minimum value in the M-by-N-by-P input matrix. In this mode, the block output is always sample based. For an input that is an M-by-N matrix, the output will be a 1-by-2 vector.
- `Specified dimension` — The output at each sample time depends on **Dimension**. If **Dimension** is set to 1, the output is the same as when you select `Each column`. If **Dimension** is set to 2, the output is the same as when you select `Each row`. If **Dimension** is set to 3, the output at each sample time is an M-by-N matrix containing the indices of the minimum values of each vector over the third dimension of the input. In this mode, the frame status of the output is the same as that of the input.

When a minimum value occurs more than once, the computed index corresponds to the first occurrence. For example, when the input is the column vector `[-1 2 3 2 -1]'`, the computed one-based index of the minimum value is `1` rather than `5` when `Each column` is selected.

Value and Index Mode

When **Mode** is set to `Value and Index`, the block outputs both the minima and the indices.

Running Mode

When **Mode** is set to `Running`, the block tracks the minimum value of each channel in a time sequence of M-by-N inputs. For sample-based inputs, the output is a sample-based M-by-N array with each element y_{ij} containing the minimum value observed in element u_{ij} for all inputs since the last reset. For frame-based inputs, the output is a frame-based M-by-N array with each element y_{ij} containing the minimum value observed in the jth column of all inputs since the last reset, up to and including element u_{ij} of the current input.

N-D signals cannot be frame based. When the block is set to `Running` mode, each element of the N-D signal is considered as a separate channel. There are $\prod d_i$ channels, where d_i is the size of the ith dimension.

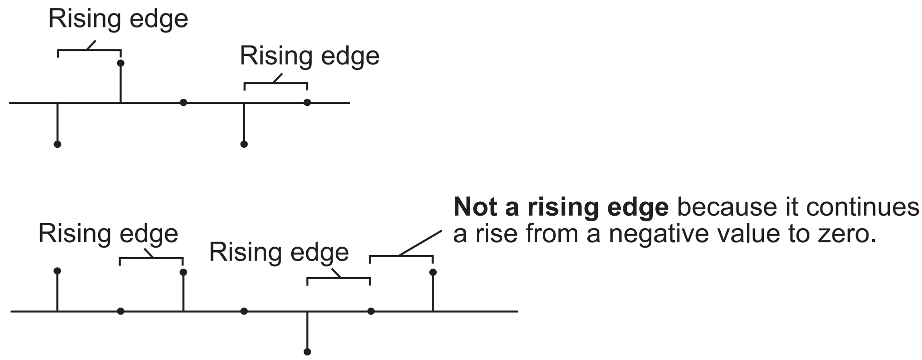
Resetting the Running Minimum

The block resets the running minimum whenever a reset event is detected at the optional Rst port. The reset sample time must be a positive integer multiple of the input sample time.

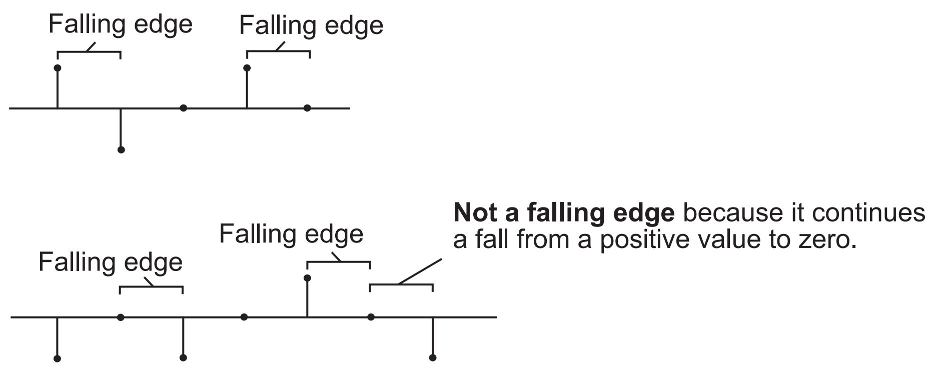
When the block is reset for sample-based inputs, the running minimum for each channel is initialized to the value in the corresponding channel of the current input. For frame-based inputs, the running minimum for each channel is initialized to the earliest value in each channel of the current input.

You specify the reset event by the **Reset port** parameter:

- `None` — Disables the Rst port
- `Rising edge` — Triggers a reset operation when the Rst input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)



- `Falling edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)



- `Either edge` — Triggers a reset operation when the `Rst` input is a `Rising edge` or `Falling edge` (as described above)
- `Non-zero sample` — Triggers a reset operation at each sample time that the `Rst` input is not zero

Note When running simulations in the Simulink `MultiTasking` mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset. For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

ROI Processing

To calculate the statistical value within a particular region of interest (ROI) of the input, select the **Enable ROI processing** check box. This option is only available when the **Find the minimum value over** parameter is set to `Entire input` and the **Enable ROI processing** check box is selected. ROI processing is only supported for 2-D inputs.

Note Full ROI processing is only available to users who have a Video and Image Processing Blockset license. If you only have a Signal Processing Blockset license, you can still use ROI processing, but are limited to the **ROI type** `Rectangles`.

Use the **ROI type** parameter to specify whether the ROI is a rectangle, line, label matrix, or binary mask. A binary mask is a binary image that enables you to specify which pixels to highlight, or select. In a label matrix, pixels equal to 0 represent the background, pixels equal to 1 represent the first object, pixels equal to 2 represent the second object, and so on. When the **ROI type** parameter is set to `Label matrix`, the Label and Label Numbers ports appear on the block. Use the Label Numbers port to specify the objects in the label matrix for which the block calculates statistics. The input to this port must be a vector of scalar values that correspond to the labeled regions in the label matrix. For more information about the format of the input to the ROI port when the ROI is a rectangle or a line, see the Draw Shapes block reference page.

For rectangular ROIs, use the **ROI portion to process** parameter to specify whether to calculate the statistical value for the entire ROI or just the ROI perimeter.

Use the **Output** parameter to specify the block output. The block can output separate statistical values for each ROI or the statistical value for all specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select `Binary mask`.

If, for the **ROI type** parameter, you select `Rectangles` or `Lines`, the **Output flag indicating if ROI is within image bounds** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual statistics for each ROI

Flag Port Output	Description
0	ROI is completely outside the input image.
1	ROI is completely or partially inside the input image.

Output = Single statistic for all ROIs

Flag Port Output	Description
0	All ROIs are completely outside the input image.
1	At least one ROI is completely or partially inside the input image.

If the ROI is partially outside the image, the block only computes the statistical values for the portion of the ROI that is within the image.

If, for the **ROI type** parameter, you select `Label matrix`, the **Output flag indicating if input label numbers are valid** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual statistics for each ROI

Flag Port Output	Description
0	Label number is not in the label matrix.
1	Label number is in the label matrix.

Output = Single statistic for all ROIs

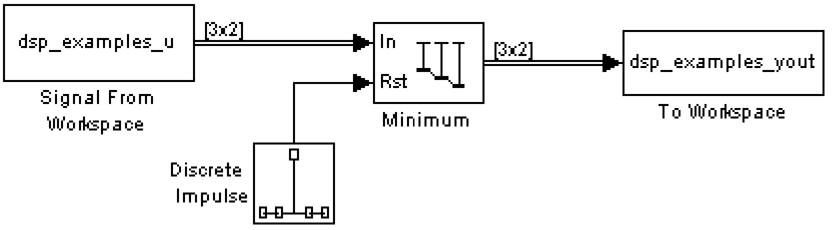
Flag Port Output	Description
0	None of the label numbers are in the label matrix.
1	At least one of the label numbers is in the label matrix.

Fixed-Point Data Types

The parameters on the **Data Types** pane of the block dialog are only used for complex fixed-point inputs. The sum of the squares of the real and imaginary parts of such an input are formed before a comparison is made, as described in "Value Mode" on page 2-924. The results of the squares of the real and imaginary parts are placed into the product output data type. The result of the sum of the squares is placed into the accumulator data type. These parameters are ignored for other types of inputs.

Examples

The Minimum block in the following model calculates the running minimum of a frame-based 3-by-2 (two-channel) matrix input. The running minimum is reset at t=2 by an impulse to the block's Rst port.



The Minimum block has the following settings:

- **Mode** = `Running`
- **Reset port** = `Non-zero sample`

The Signal From Workspace block has the following settings:

- **Signal** = `dsp_examples_u`
- **Sample time** = `1/3`
- **Samples per frame** = `3`

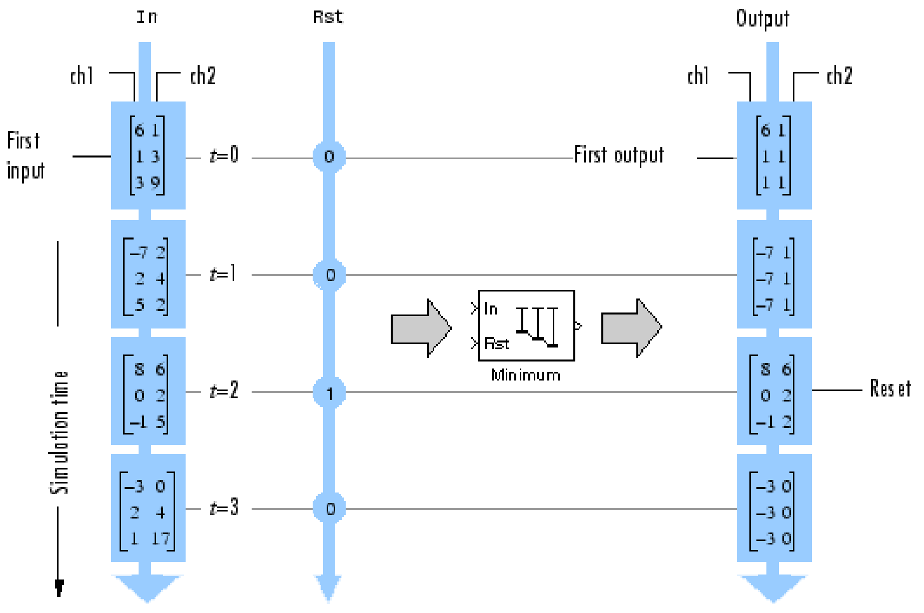
where

```
dsp_examples_u = ...
  [6 1 3 -7 2 5 8 0 -1 -3 2 1;1 3 9 2 4 2 6 2 5 0 4 17]'
```

The Discrete Impulse block has the following settings:

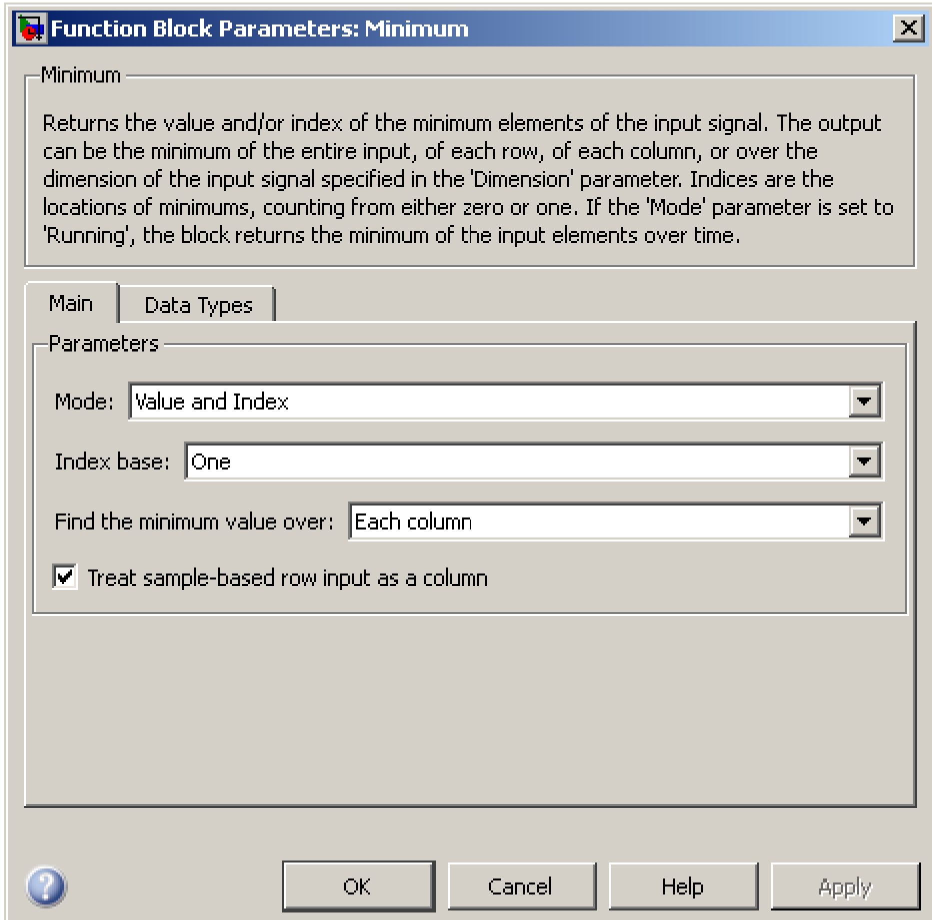
- **Delay (samples)** = `2`
- **Sample time** = `1`
- **Samples per frame** = `1`

The block's operation is shown in the figure below.


In
Rst
Output
ch1
ch2
First
input
t=0
t=1
t=2
t=3
First output
Reset
Simulation time
Minimum

Dialog Box

The **Main** pane of the Minimum block dialog appears as follows.



Mode

Specify the block's mode of operation:

- `Value` — Output the minimum value of each input
- `Index` — Output the index of the minimum value

- `Value and index` — Output both the value and the index
- `Running` — Track the minimum value of the input sequence over time

For more information, see Description.

Index base
Specify whether the index of the minimum value is reported using one-based or zero-based numbering. This parameter is only visible when the **Mode** parameter is set to `Index` or `Value and index`.

Find the minimum value over
Specify whether to find the minimum value along rows, columns, entire input, or the dimension specified in the **Dimension** parameter. For more information, see Description.

Reset port
Specify the reset event detected at the `RST` input port when you select `Running` for the **Mode** parameter. The rate of the reset signal must be a positive integer multiple of the rate of the data signal input. This parameter is enabled only when you set the **Mode** parameter to `Running`. For information about the possible values of this parameter, see “Resetting the Running Minimum” on page 2-928.

Treat sample-based row input as a column
Select to treat sample-based length-M row vector inputs as M-by-1 column vectors. This parameter is only visible when the **Find the minimum value of** parameter is set to `Each column`.

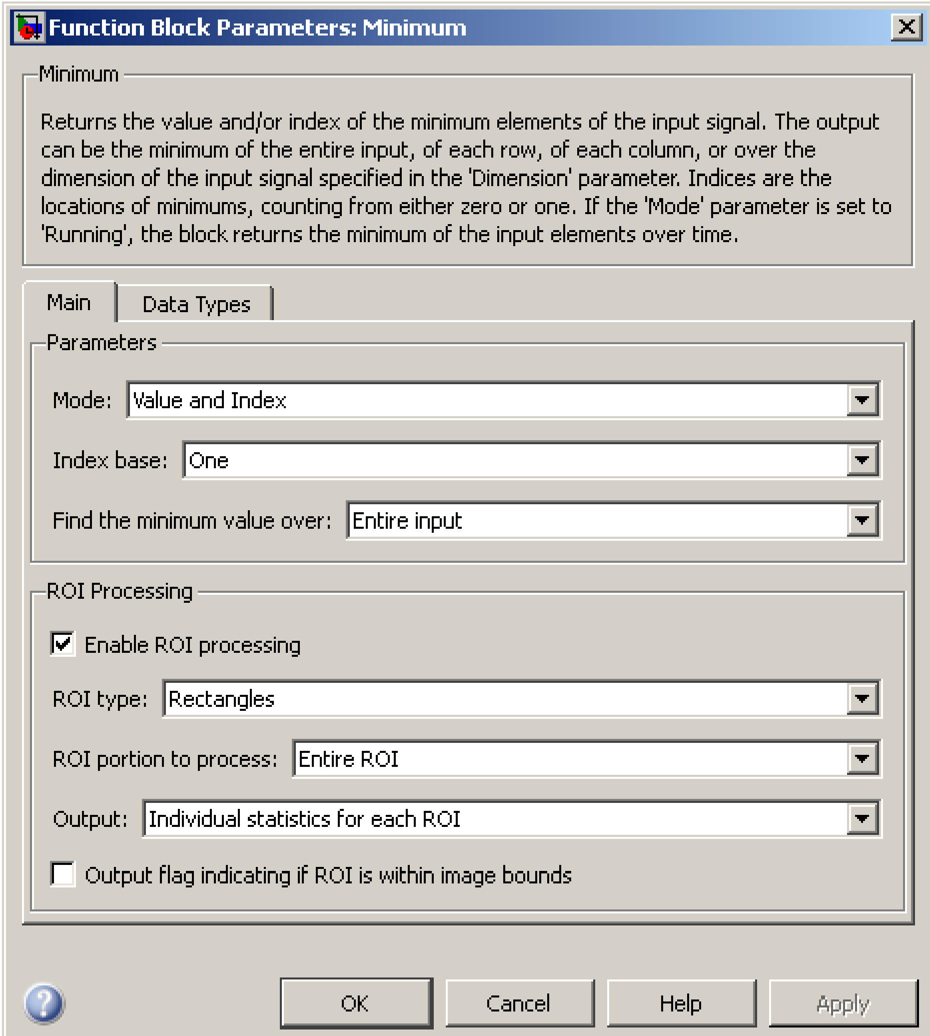
Dimension
Specify the dimension (one-based value) of the input signal, over which the minimum is computed. The value of this parameter cannot exceed the number of dimensions in the input signal. This parameter is only visible when the **Find the minimum value over** parameter is set to `Specified dimension`.

Enable ROI processing

Select this check box to calculate the statistical value within a particular region of each image. This parameter is only available when the **Find the minimum value over** parameter is set to `Entire input`, and the block is not in running mode.

Note Full ROI processing is only available to users who have a Video and Image Processing Blockset license. If you only have a Signal Processing Blockset license, you can still use ROI processing, but are limited to the **ROI type** `Rectangles`.

The ROI processing parameters appear as follows.



ROI type

Specify the type of ROI you want to use. Your choices are `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`.

ROI portion to process

Specify whether you want to calculate the statistical value for the entire ROI or just the ROI perimeter. This parameter is only visible if, for the **ROI type** parameter, you specify `Rectangles`.

Output

Specify the block output. The block can output a vector of separate statistical values for each ROI or a scalar value that represents the statistical value for all the specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select `Binary mask`.

Output flag

☑ **Output flag indicating if ROI is within image bounds**

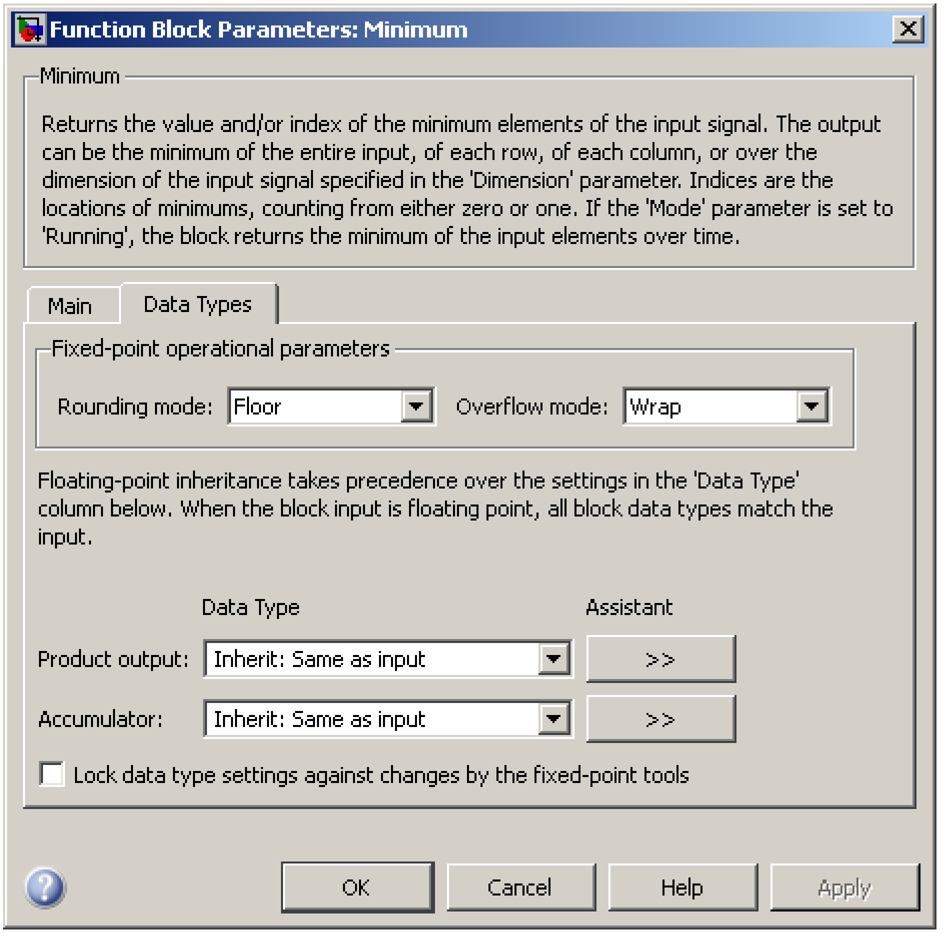
☑ **Output flag indicating if label numbers are valid**

When you select either of these check boxes, the Flag port appears on the block. For a description of the Flag port output, see the tables in "ROI Processing" on page 2-930.

The **Output flag indicating if ROI is within image bounds** check box is only visible when you select `Rectangles` or `Lines` as the **ROI type**.

The **Output flag indicating if label numbers are valid** check box is only visible when you select `Label matrix` for the **ROI type** parameter.

The **Data Types** pane of the Minimum block dialog appears as follows.


Function Block Parameters: Minimum
Minimum
Returns the value and/or index of the minimum elements of the input signal. The output can be the minimum of the entire input, of each row, of each column, or over the dimension of the input signal specified in the 'Dimension' parameter. Indices are the locations of minimums, counting from either zero or one. If the 'Mode' parameter is set to 'Running', the block returns the minimum of the input elements over time.
Main
Data Types
Fixed-point operational parameters
Rounding mode:
Floor
Overflow mode:
Wrap
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input.
Data Type
Assistant
Product output:
Inherit: Same as input
>>
Accumulator:
Inherit: Same as input
>>
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Note The parameters on the **Data Types** pane are only used for complex fixed-point inputs. The sum of the squares of the real and imaginary parts of such an input are formed before a comparison is made, as described in "Value Mode" on page 2-924. The results of the squares of the real and imaginary parts are placed into the product output data type. The result of the sum of the squares is placed into the accumulator data type. These parameters are ignored for other types of inputs.

Rounding mode
Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode for fixed-point operations.

Product output data type
Specify the product output data type. See "Fixed-Point Data Types" on page 2-932 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Accumulator data type

Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-932 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button >> to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Reset	• Double-precision floating point • Single-precision floating point • Boolean

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Idx	• Double-precision floating point • 32-bit unsigned integers
Val	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Maximum	Signal Processing Blockset
Mean	Signal Processing Blockset
MinMax	Simulink
Histogram	Signal Processing Blockset
`min`	MATLAB

Modified Covariance AR Estimator

Purpose

Compute estimate of autoregressive (AR) model parameters using modified covariance method

Library

Estimation / Parametric Estimation

`dspparest3`

Description

The Modified Covariance AR Estimator block uses the modified covariance method to fit an autoregressive (AR) model to the input data. This method minimizes the forward and backward prediction errors in the least squares sense. The input is a frame of consecutive time samples, which is assumed to be the output of an AR system driven by white noise. The block computes the normalized estimate of the AR system parameters, *A(z)*, independently for each successive input.

$$H(z) = \frac{G}{A(z)} = \frac{G}{1 + a(2)z^{-1} + \ldots + a(p+1)z^{-p}}$$

You specify the order, p, of the all-pole model in the **Estimation order** parameter. To guarantee a valid output, you must set the **Estimation order** parameter to be less than or equal to two thirds the input vector length.

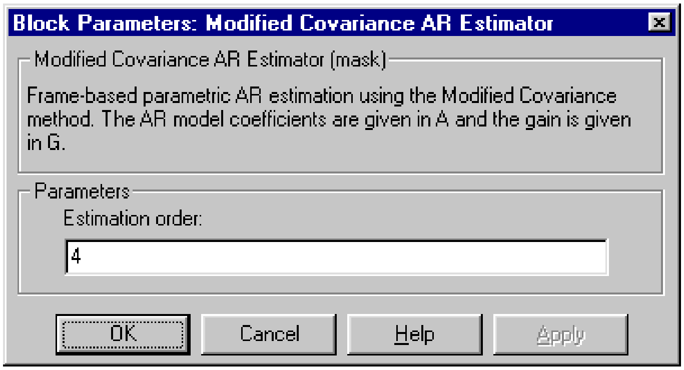
The output port labeled A outputs the normalized estimate of the AR model coefficients in descending powers of z.

```
[1 a(2) ... a(p+1)]
```

The scalar gain, G, is output from the output port labeled G.

See the Burg AR Estimator block reference page for a comparison of the Burg AR Estimator, Covariance AR Estimator, Modified Covariance AR Estimator, and Yule-Walker AR Estimator blocks.

Dialog Box



Estimation order

Specify the order of the AR model, p.

References

Kay, S. M. *Modern Spectral Estimation: Theory and Application.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

Marple, S. L., Jr., *Digital Spectral Analysis with Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1987.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point
A	• Double-precision floating point • Single-precision floating point
G	• Double-precision floating point • Single-precision floating point

The output data type is the same as the input data type.

See Also

Burg AR Estimator	Signal Processing Blockset
Covariance AR Estimator	Signal Processing Blockset
Modified Covariance Method	Signal Processing Blockset
Yule-Walker AR Estimator	Signal Processing Blockset
`armcov`	Signal Processing Toolbox

Purpose

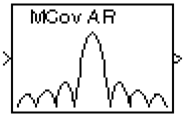
Power spectral density estimate using modified covariance method

Library

Estimation / Power Spectrum Estimation

`dspspect3`

Description



The Modified Covariance Method block estimates the power spectral density (PSD) of the input using the modified covariance method. This method fits an autoregressive (AR) model to the signal. It does so by minimizing the forward and backward prediction errors in the least squares sense. The **Estimation order** parameter value specifies the order of the all-pole model. To guarantee a valid output, the **Estimation order** parameter must be less than or equal to two thirds of the input vector length. The block computes the spectrum from the FFT of the estimated AR model parameters.

The input must be a sample-based vector (row, column, or 1-D) or frame-based vector (column only). This input represents a frame of consecutive time samples from a single-channel signal. The block outputs a column vector containing the estimate of the power spectral density of the signal at N_{fft} equally spaced frequency points. The frequency points are in the range $[0,F_s)$, where F_s is the sampling frequency of the signal.

Selecting **Inherit FFT length from estimation order**, specifies that N_{fft} is one greater than the estimation order. Clearing the **Inherit FFT length from estimation order** check box allows you to use the **FFT length** parameter to specify N_{fft} as a power of 2. The block zero-pads or wraps the input to N_{fft} before computing the FFT. The output is always sample based.

When you select the **Inherit sample time from input** check box, the block computes the frequency data from the sample period of the input signal. For the block to produce valid output, the following conditions must hold:

- The input to the block is the original signal, with no samples added or deleted (by insertion of zeros, for example).

- The sample period of the time-domain signal in the simulation equals the sample period of the original time series.

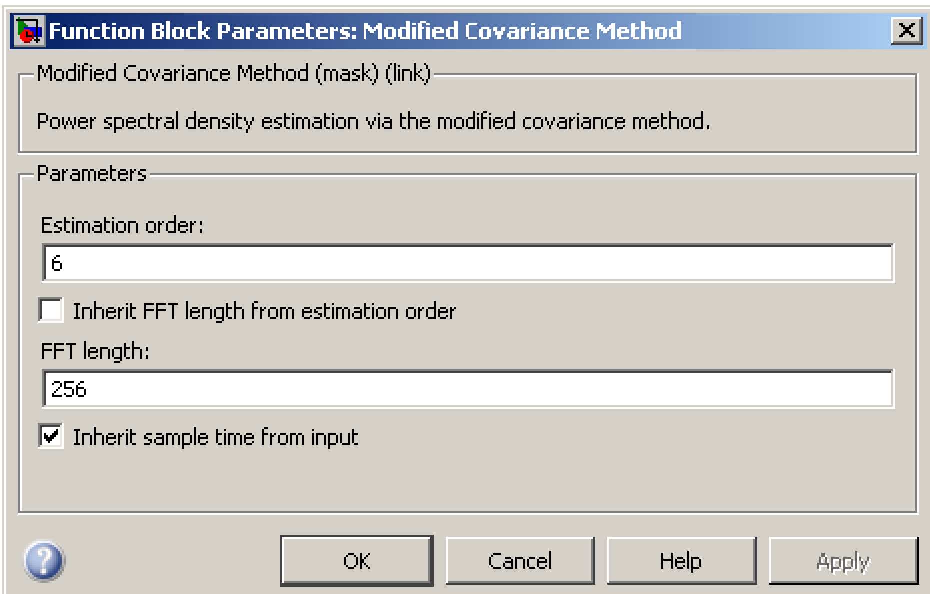
If these conditions do not hold, clear the **Inherit sample time from input** check box. You can then specify a sample time using the **Sample time of original time series** parameter.

See the Burg Method block reference for a comparison of the Burg Method, Covariance Method, Modified Covariance Method, and Yule-Walker Method blocks.

Examples

The `dspsacomp` demo compares the modified covariance method with several other spectral estimation methods.

Dialog Box



Estimation order

Specify the order of the AR model. To guarantee a valid output, the **Estimation order** parameter must be less than or equal to two thirds of the input vector length.

Inherit FFT length from estimation order

When you select this check box, the option specifies that the FFT length is one greater than the estimation order. To specify the number of points on which to perform the FFT, clear this check box. You can then specify a power of two FFT length using the **FFT length** parameter.

FFT length

Enter the number of data points, N_{fft}, on which to perform the FFT. When N_{fft} is larger than the input frame size, the block zero-pads each frame as needed. When N_{fft} is smaller than the input frame size, the block wraps each frame as needed. This parameter becomes visible only when you clear the **Inherit FFT length from estimation order** check box.

Inherit sample time from input

If you select the **Inherit sample time from input** check box, the block computes the frequency data from the sample period of the input signal. For the block to produce valid output, the following conditions must hold:

- The input to the block is the original signal, with no samples added or deleted (by insertion of zeros, for example).
- The sample period of the time-domain signal in the simulation equals the sample period of the original time series.

If these conditions do not hold, clear the **Inherit sample time from input** check box. You can then specify a sample time using the **Sample time of original time series** parameter.

Sample time of original time series

Specify the sample time of the original time-domain signal. This parameter becomes visible only when you clear the **Inherit sample time from input** check box.

References

Kay, S. M. *Modern Spectral Estimation: Theory and Application.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

Marple, S. L. Jr., *Digital Spectral Analysis with Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1987.

Orfanidis, S. J. *Introduction to Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1995.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point
Output	• Double-precision floating point • Single-precision floating point

The output data type is the same as the input data type.

See Also

Burg Method	Signal Processing Blockset
Covariance Method	Signal Processing Blockset
Modified Covariance AR Estimator	Signal Processing Blockset
Short-Time FFT	Signal Processing Blockset
Yule-Walker Method	Signal Processing Blockset
`spectrum.mcov`	Signal Processing Toolbox

See “Power Spectrum Estimation” for related information.

Purpose

Generate multiple binary clock signals

Library

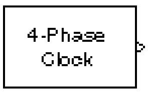
- Signal Processing Sources

  `dspsrcs4`

- Signal Management / Switches and Counters

  `dspswit3`

Description



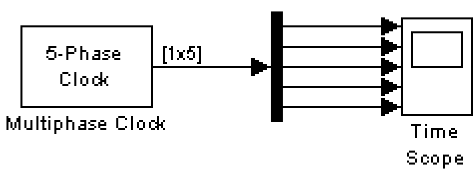
The Multiphase Clock block generates a sample-based 1-by-*N* vector of clock signals, where you specify the integer *N* in the **Number of phases** parameter. Each of the *N* phases has the same frequency, *f*, specified in hertz by the **Clock frequency** parameter.

The clock signal indexed by the **Starting phase** parameter is the first to become active, at *t*=0. The other signals in the output vector become active in turn, each one lagging the preceding signal's activation by 1/(*N**f*) seconds, the phase interval. The period of the sample-based output is therefore 1/(*N**f*) seconds.

The active level can be either high (1) or low (0), as specified by the **Active level (polarity)** parameter. The duration of the active level, *D*, is set by the **Number of phase intervals over which the clock is active**. This value, which can be an integer value between 1 and *N*-1, specifies the number of phase intervals that each signal should remain in the active state after becoming active. The active duty cycle of the signal is *D*/*N*.

Examples

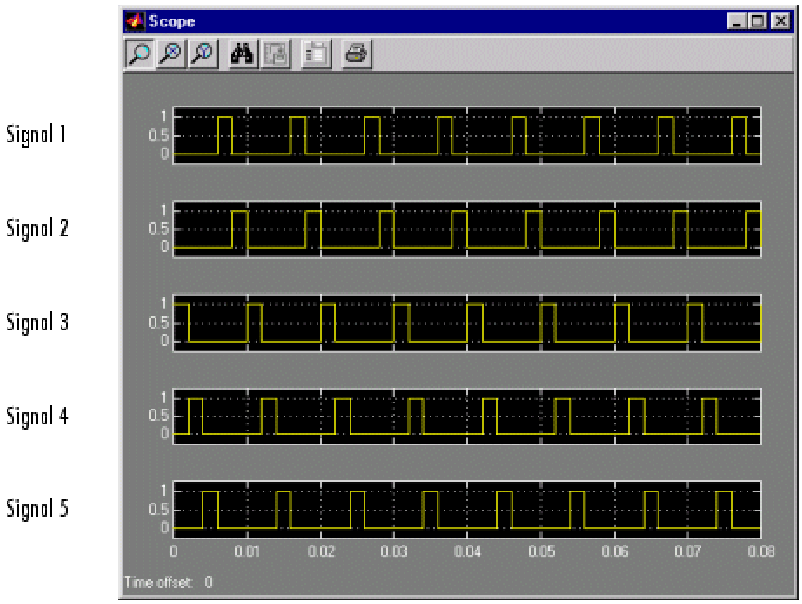
Configure the Multiphase Clock block in the model below to generate a 100 Hz five-phase output in which the third signal is first to become active. Use a high active level with a duration of one interval.



The corresponding settings are as follows:

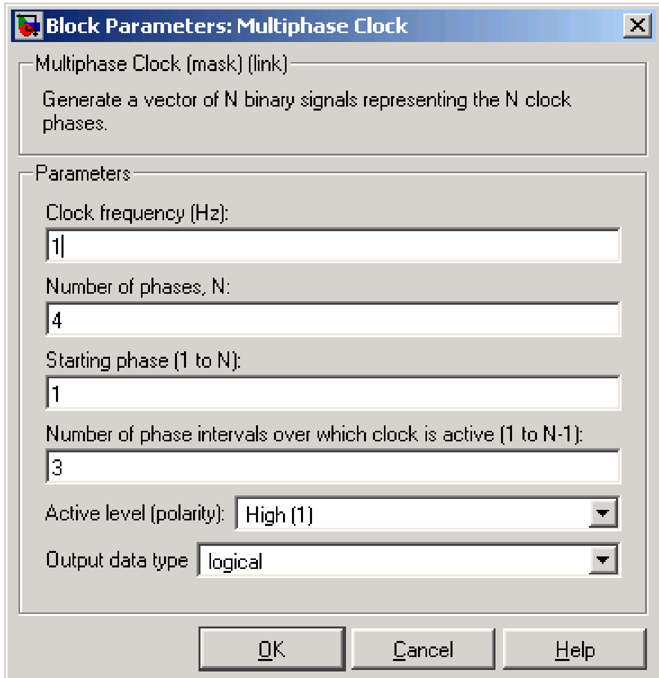
- **Clock frequency** = `100`
- **Number of phases** = `5`
- **Starting phase** = `3`
- **Number of phase intervals over which the clock is active** = `1`
- **Active level (polarity)** = `High (1)`

The Scope window below shows the Multiphase Clock block's output for these settings. Note that the first active level appears at *t*=0 on `y(3)`, the second active level appears at *t*=0.002 on `y(4)`, the third active level appears at *t*=0.004 on `y(5)`, the fourth active level appears at *t*=0.006 on `y(1)`, and the fifth active level appears at *t*=0.008 on `y(2)`. Each signal becomes active 1/(5*100) seconds after the previous signal.



To experiment further, try changing the **Number of phase intervals over which clock is active** setting to 3 so that the active-level duration is three phase intervals (60% duty cycle).

Dialog Box



Opening this dialog causes a running simulation to pause. See “Changing Source Block Parameters During Simulation” in the online Simulink documentation for details.

Clock frequency
The frequency of all output clock signals.

Number of phases
The number of different phases, *N*, in the output vector.

Starting phase
The vector index of the output signal to first become active.

Number of phase intervals over which clock is active
The duration of the active level for every output signal.

Active level
The active level, `High (1)` or `Low (0)`.

Output data type
The output data type.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Boolean

See Also

Clock	Simulink
Counter	Signal Processing Blockset
Pulse Generator	Simulink
Event-Count Comparator	Signal Processing Blockset

Purpose

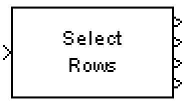
Distribute arbitrary subsets of input rows or columns to multiple output ports

Library

Signal Management / Indexing

`dspindex`

Description



The Multiport Selector block extracts multiple subsets of rows or columns from M-by-N input matrix u, and propagates each new submatrix to a distinct output port. A length-M 1-D vector input is treated as an M-by-1 matrix.

The **Indices to output** parameter is a cell array whose kth cell contains a one-dimensional indexing expression specifying the subset of input rows or columns to be propagated to the kth output port. The total number of cells in the array determines the number of output ports on the block.

When the **Select** parameter is set to `Rows`, the specified one-dimensional indices are used to select matrix rows, and all elements on the chosen rows are included. When the **Select** parameter is set to `Columns`, the specified one-dimensional indices are used to select matrix columns, and all elements on the chosen columns are included. A given input row or column can appear any number of times in any of the outputs, or not at all.

When an index references a nonexistent row or column of the input, the block reacts with the behavior specified by the **Invalid index** parameter. The following options are available:

- `Clip index` — Clip the index to the nearest valid value, and do not issue an alert.

  Example: For a 64-by-4 input with **Select** = `Rows`, an index of 72 is clipped to 64; with **Select** = `Columns`, an index of 72 is clipped to 4. In both cases, an index of -2 is clipped to 1.

- `Clip and warn` — Display a warning message in the MATLAB Command Window, and clip as above.

- `Generate error` — Display an error dialog and terminate the simulation.

Examples

Example 1

Consider the following **Indices to output** cell array:

```
{4,[1:2 5],[7;8],10:-1:6}
```

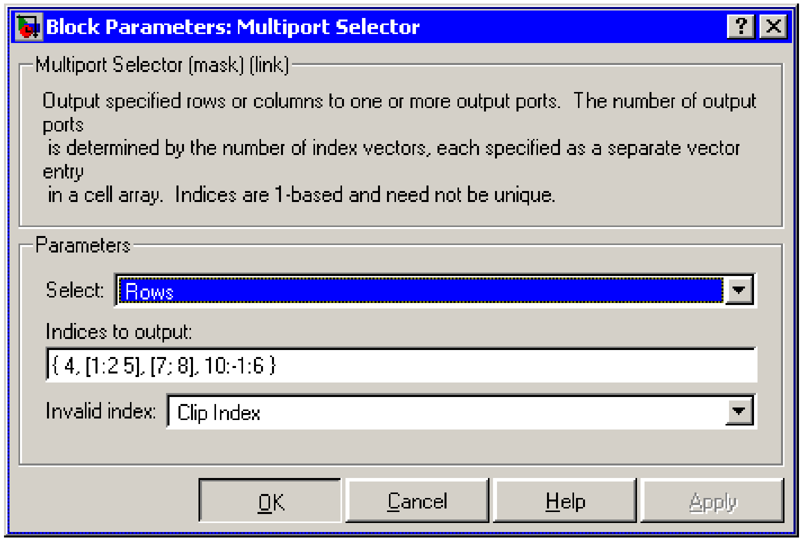
This is a four-cell array, which requires the block to generate four independent outputs (each at a distinct port). The table below shows the dimensions of these outputs when **Select** = `Rows` and the input dimension is M-by-N.

Cell	Expression	Description	Output Size
1	`4`	Row 4 of input	1-by-N
2	`[1:2 5]`	Rows 1, 2, and 5 of input	3-by-N
3	`[7;8]`	Rows 7 and 8 of input	2-by-N
4	`10:-1:6`	Rows 10, 9, 8, 7, and 6 of input	5-by-N

Example 2

To see the Multiport Selector block used in a model, see "Splitting Multichannel Sample-Based Signals into Individual Signals", and "Splitting Multichannel Frame-Based Signals into Individual Signals" in the *Signal Processing Blockset User's Guide*.

Dialog Box



Select

The dimension of the input to select, `Rows` or `Columns`.

Indices to output

A cell array specifying the row- or column-subsets to propagate to each of the output ports. The number of cells in the array determines the number of output ports on the block.

Invalid index

Response to an invalid index value.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated
Outputs	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated

See Also

Permute Matrix	Signal Processing Blockset
Selector	Simulink
Submatrix	Signal Processing Blockset
Variable Selector	Signal Processing Blockset

Purpose

Output ones or zeros for specified number of sample times

Library

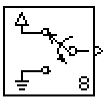
- Signal Processing Sources

  `dspsrcs4`

- Signal Management / Switches and Counters

  `dspswit3`

Description



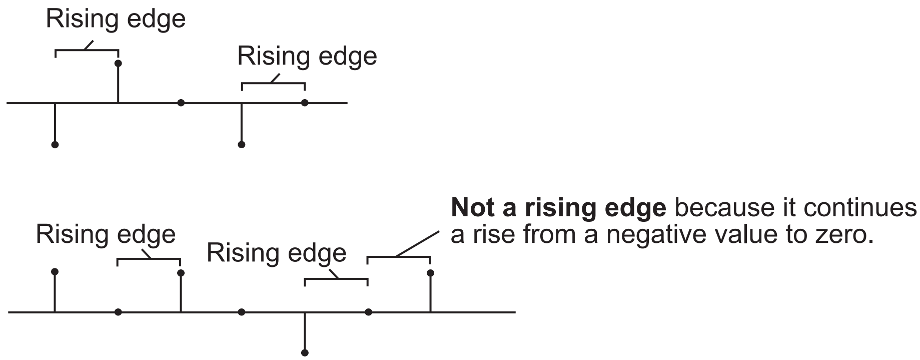
The N-Sample Enable block outputs the inactive value (`0` or `1`, whichever is not selected in the **Active level** parameter) during the first *N* sample times, where *N* is the **Trigger count** value. Beginning with output sample *N*+1, the block outputs the active value (`1` or `0`, whichever you select in the **Active level** parameter) until a reset event occurs or the simulation terminates.

The output is always sample based.

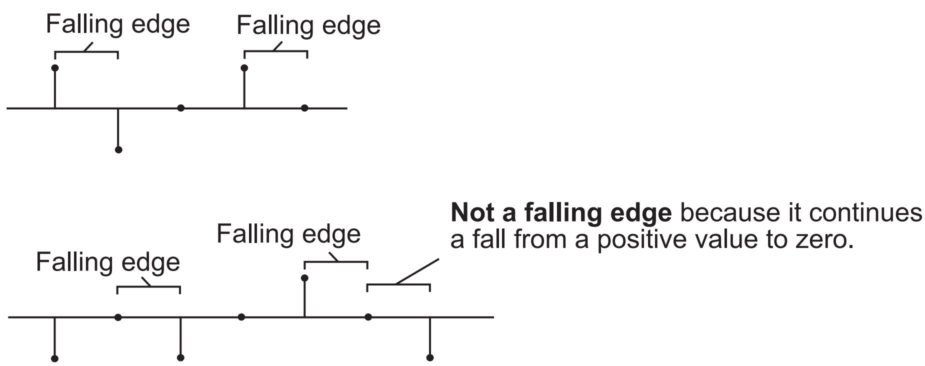
The **Reset input** check box enables the Rst input port. At any time during the count, a trigger event at the input port resets the counter to its initial state. The reset sample time must be a positive integer multiple of the input sample time. This block supports triggered subsystems when you select the **Reset input** check box.

You specify the triggering event in the **Trigger type** pop-up menu:

- `Rising edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)



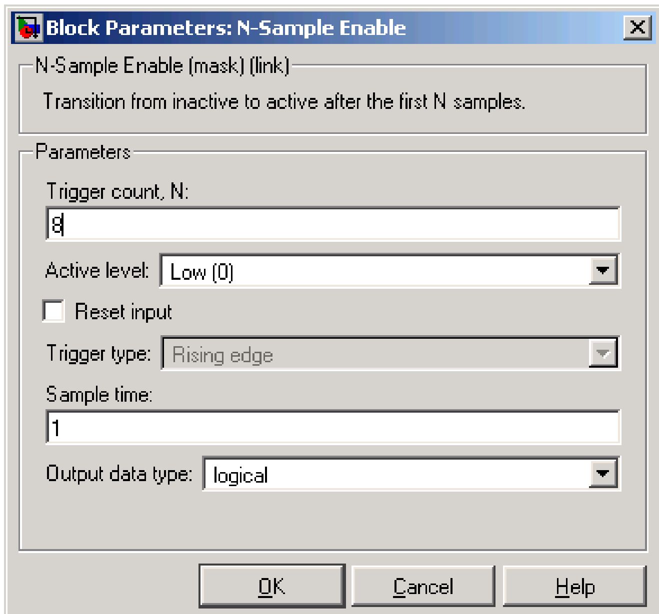
- `Falling edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)



- `Either edge` — Triggers a reset operation when the `Rst` input is a `Rising edge` or `Falling edge` (as described above).
- `Non-zero sample` — Triggers a reset operation at each sample time that the `Rst` input is not zero.

Note When running simulations in the Simulink `MultiTasking` mode, sample-based reset signals have a one-sample latency, and frame-based reset signals have one frame of latency. Thus, there is a one-sample or one-frame delay between the time the block detects a reset event, and when it applies the reset. For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

Dialog Box



Opening this dialog causes a running simulation to pause. See "Changing Source Block Parameters During Simulation" in the online Simulink documentation for details.

Trigger count
Specify the number of samples for which the block outputs the active value. Tunable.

Active level
Specify the value to output after the first *N* sample times, `0` or `1`. Tunable.

Reset input
Select to enable the Rst input port. The rate of the reset signal must be a positive integer multiple of the rate of the data signal input.

Trigger type
Select type of event that triggers a reset when the Rst port is enabled.

Sample time
Specify the sample period, T_s, for the block's counter. The block switches from the active value to the inactive value at $t=T_s*(N+1)$.

Output data type
Select the output data type.

Supported Data Types

- Double-precision floating point
- Boolean — The block accepts Boolean inputs to the Rst port, which is enabled when you set the **Reset input** parameter.

See Also

Counter	Signal Processing Blockset
N-Sample Switch	Signal Processing Blockset

Purpose

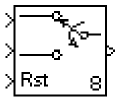
Switch between two inputs after specified number of sample periods

Library

Signal Management / Switches and Counters

`dspswit3`

Description



The N-Sample Switch block outputs the signal connected to the top input port during the first *N* sample times after the simulation begins or the block is reset, where you specify *N* in the **Switch count** parameter. Beginning with output sample *N*+1, the block outputs the signal connected to the bottom input until the next reset event or the end of the simulation.

You specify the sample period of the output in the **Sample time** parameter (that is, the output sample period is not inherited from the sample period of either input). The block applies a zero-order hold at the input ports, so the value the block reads from a given port between input sample times is the value of the most recent input to that port.

Both inputs must have the same dimension, except in the following two cases:

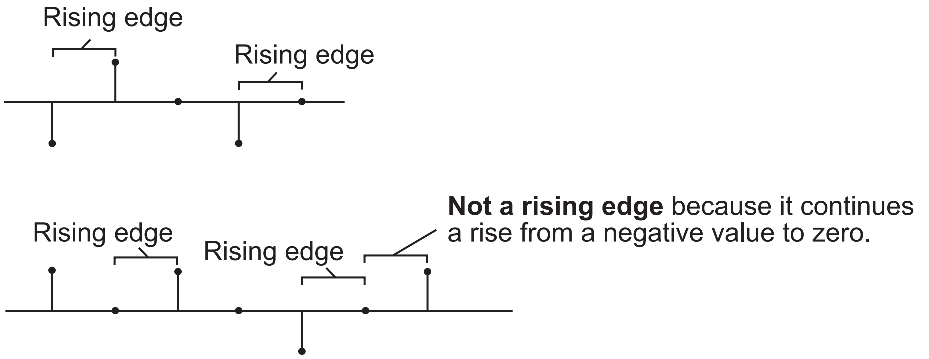
- When one input is a scalar, the block expands the scalar input to match the size of the other input.
- When one input is a 1-D vector and the other input is a row or column vector with the same number of elements, the block reshapes the 1-D vector to match the dimension of the other input.

The inputs must either both be frame based or both be sample based.

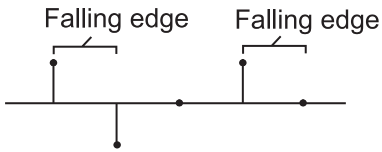
The **Reset input** check box enables the Rst input port. At any time during the count, a trigger event at the Rst port resets the counter to zero. The reset sample time must be a positive integer multiple of the input sample time. This block supports triggered subsystems when you select the **Reset input** check box.

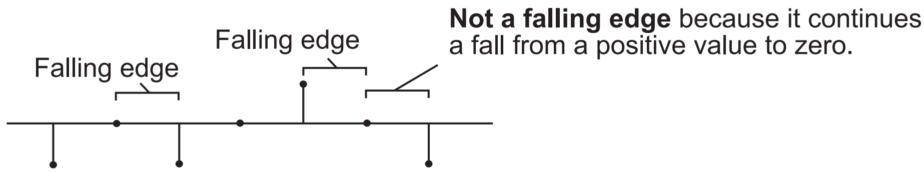
You specify the triggering event in the **Trigger type** pop-up menu, and can be one of the following:

- `Rising edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)



- `Falling edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)

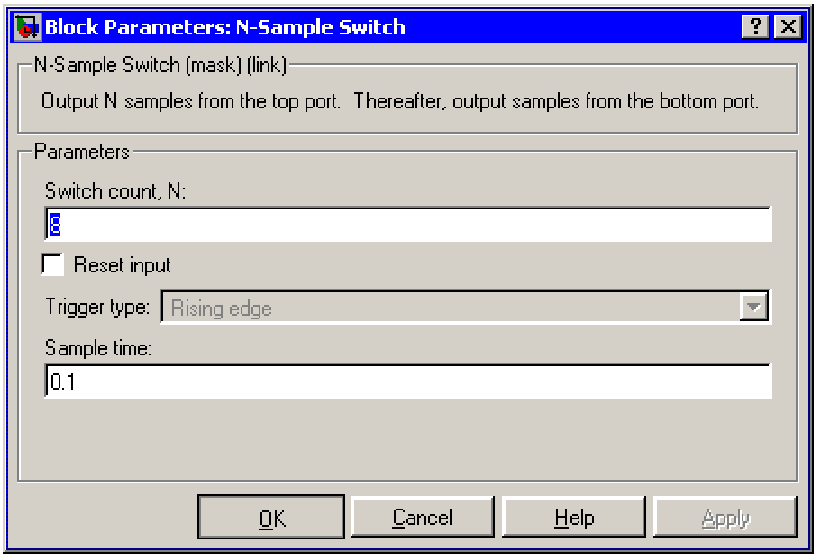




- `Either edge` — Triggers a reset operation when the `Rst` input is a `Rising edge` or `Falling edge` (as described above).
- `Non-zero sample` — Triggers a reset operation at each sample time that the `Rst` input is not zero.

Note When running simulations in the Simulink `MultiTasking` mode, sample-based reset signals have a one-sample latency, and frame-based reset signals have one frame of latency. Thus, there is a one-sample or one-frame delay between the time the block detects a reset event, and when it applies the reset. For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

Dialog Box



Switch count

The number of sample periods, N, for which the output is connected to the top input before switching to the bottom input. Tunable.

Reset input

Enables the Rst input port when selected. The rate of the reset signal must be a positive integer multiple of the rate of the data signal input.

Trigger type

The type of event at the Rst port that resets the block's counter. This parameter is enabled when you select **Reset input**. Tunable.

Sample time

The sample period, T_s, for the block's counter. The block switches inputs at $t=T_s*(N+1)$.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed and unsigned)
- Boolean — The block accepts Boolean inputs to the Rst port, which is enabled when you set the **Reset input** parameter.
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers

See Also

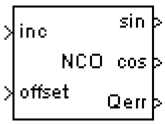
Counter	Signal Processing Blockset
N-Sample Enable	Signal Processing Blockset

Purpose Generate real or complex sinusoidal signals

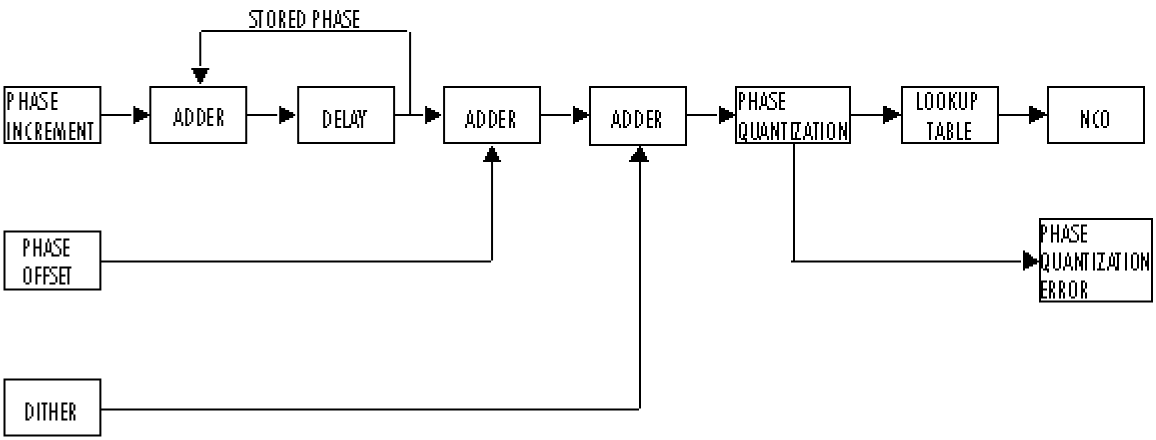
Library Signal Operations

`dspsigops`

Description



The NCO block generates a multichannel real or complex sinusoidal signal, with independent frequency and phase in each output channel. The amplitude of the created signal is always 1. The block implements the algorithm as shown in the following diagram:



The implementation of a numerically controlled oscillator (NCO) has two distinct parts. First, a phase accumulator accumulates the phase increment and adds in the phase offset. In this stage, an optional internal dither signal can also be added. The NCO output is then calculated by quantizing the results of the phase accumulator section and using them to select values from a lookup table.

Given a desired output frequency F_0, calculate the value of the **Phase increment** block parameter with

$$phase\ increment = \left(\frac{F_0 \cdot 2^N}{F_s}\right)$$

where N is the accumulator word length and

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

The frequency resolution of an NCO is defined by

$$\Delta f = \frac{1}{T_s \cdot 2^N}\ \mathrm{Hz}$$

Given a desired phase offset (in radians), calculate the **Phase offset** block parameter with

$$phase\ offset = \frac{2^N \cdot desired\ phase\ offset}{2\pi}$$

The spurious free dynamic range (SFDR) is estimated as follows for a lookup table with 2^P entries, where P is the number of quantized accumulator bits:

$$SFDR = (6P)\ \mathrm{dB} \qquad \text{without dither}$$
$$SFDR = (6P + 12)\ \mathrm{dB} \qquad \text{with dither}$$

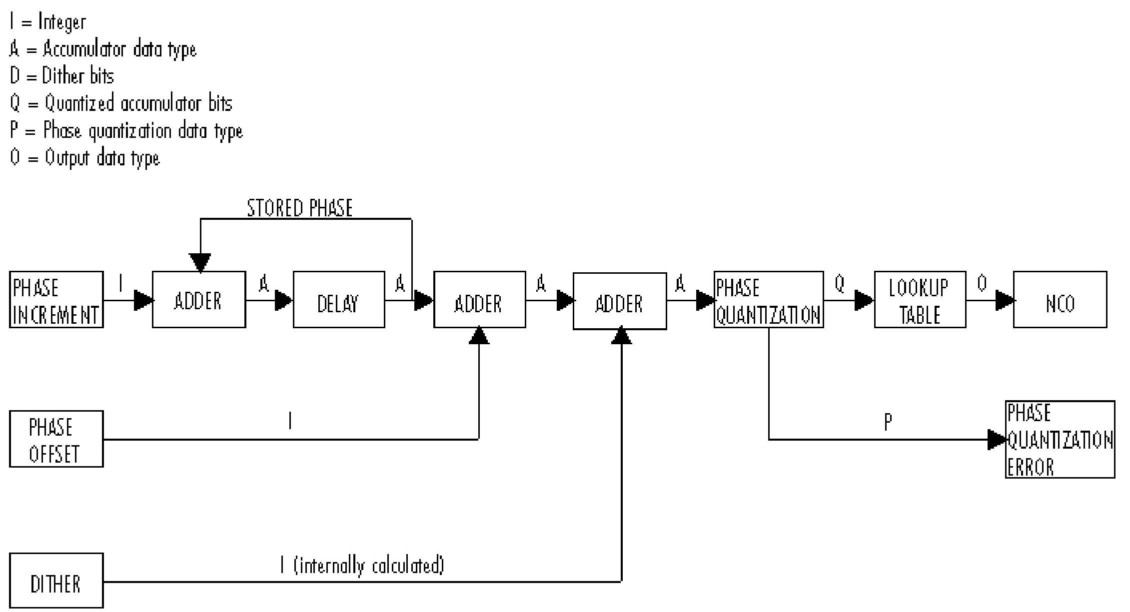
This block uses a quarter-wave lookup table technique that stores table values from 0 to π/2. The block calculates other values on demand using the accumulator data type, then casts them into the output data type. This can lead to quantization effects at the range limits of a given data type. For example, consider a case where you would expect the value of the sine wave to be –1 at π. Because the lookup table value at that point must be calculated, the block might not yield exactly –1, depending on the precision of the accumulator and output data types.

The NCO block supports real inputs only. All outputs are real except for the output signal in `Complex exponential` mode.

To produce a multichannel output, specify a vector quantity for the **Phase increment** and **Phase offset** parameters. Both parameters must have the same length, which defines the number of output channels. Each element of each vector is applied to a different output channel.

Fixed-Point Data Types

The following diagram shows the data types used within the NCO block.



- You can set the accumulator and output data types in the block dialog as discussed in “Dialog Box” on page 2-978 below.

Note The lookup table for this block is constructed from double-precision floating-point values. Thus, the maximum amount of precision you can achieve in your output is 53 bits. Setting the word length of the **Output** data type to values greater than 53 bits does not improve the precision of your output.

- The phase increment and phase offset inputs must be integers or fixed-point data types with zero fraction length.
- You specify the number of quantized accumulator bits in the **Number of quantized accumulator bits** parameter.
- The phase quantization error word length is equal to the accumulator word length minus the number of quantized accumulator bits, and the fraction length is zero.

Examples

The NCO block is used in the GSM Digital Down Converter product demo. Open this demo by typing `dspddc` at the MATLAB command line.

You can also try the following example. Design an NCO source with the following specifications:

- Desired output frequency $F_0 = 510$ Hz
- Frequency resolution $\Delta f = 0.05$ Hz
- Spurious free dynamic range $SFDR \geq 90$ dB
- Sample period $T_s = 1/8000$ s
- Desired phase offset $\pi/2$

1

Calculate the number of required accumulator bits from the equation for frequency resolution:

$$
\begin{aligned}
\Delta f &= \frac{1}{T_s \cdot 2^N}\,\text{Hz}\\
0.05 &= \frac{1}{\frac{1}{8000} \cdot 2^N}\,\text{Hz}\\
N &= 18
\end{aligned}
$$

Note that N must be an integer value. The value of N is rounded up to the nearest integer; 18 accumulator bits are needed to accommodate the value of the frequency resolution.

2

Using this best value of N, calculate the frequency resolution that will be achieved by the NCO block:

$$
\begin{aligned}
\Delta f &= \frac{1}{T_s \cdot 2^N}\,\text{Hz}\\
\Delta f &= \frac{1}{\frac{1}{8000} \cdot 2^{18}}\,\text{Hz}\\
\Delta f &= 0.0305
\end{aligned}
$$

3

Calculate the number of quantized accumulator bits from the equation for spurious free dynamic range and the fact that for a lookup table with 2^P entries, P is the number of quantized accumulator bits:

$$
\begin{aligned}
SFDR &= (6P + 12)\,\text{dB}\\
96\ \text{dB} &= (6P + 12)\ \text{dB}\\
P &= 14
\end{aligned}
$$

4

Select the number of dither bits. In general, a good choice for the number of dither bits is the accumulator word length minus the output word length; in this case 4.

5

Calculate the phase increment:

$$phase\ increment = \text{round}(\frac{F_0 \cdot 2^N}{F_s})$$
$$phase\ increment = \text{round}(\frac{501 \cdot 2^{18}}{8000})$$
$$phase\ increment = 16417$$

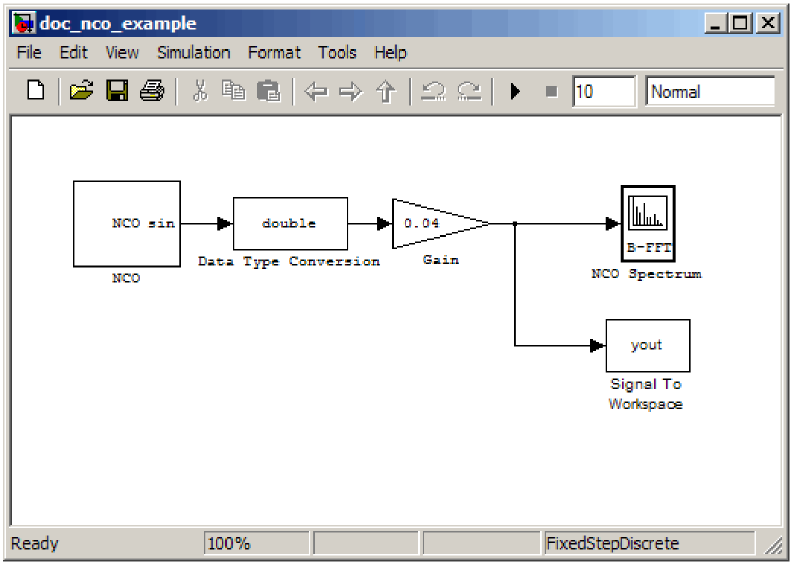
6

Calculate the phase offset:

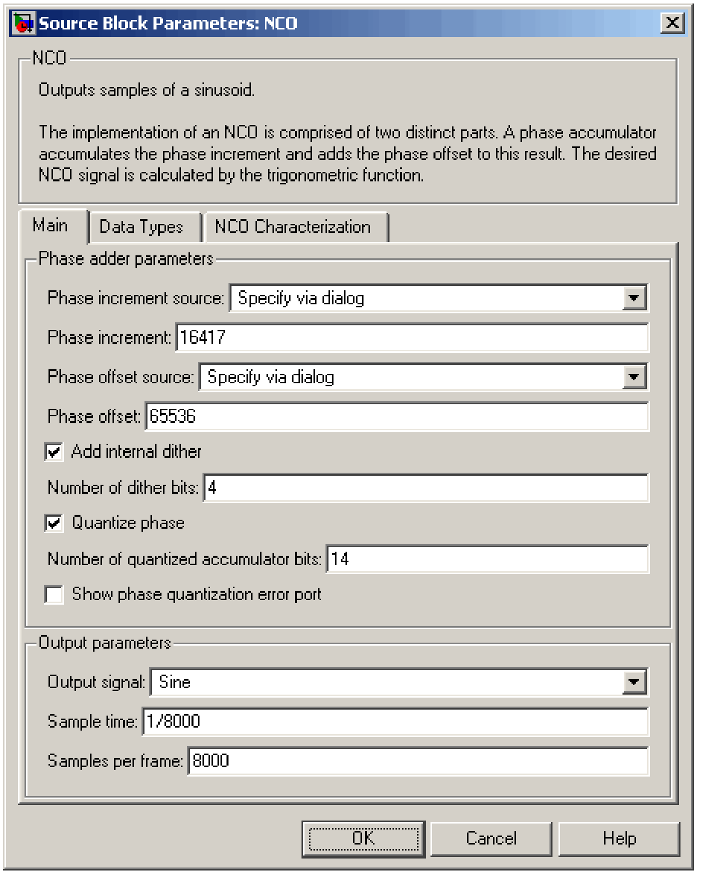
$$phase\ offset = \frac{2^{accumulator\ word\ length} \cdot desired\ phase\ offset}{2\pi}$$
$$phase\ offset = \frac{2^{18} \cdot \frac{\pi}{2}}{2\pi}$$
$$phase\ offset = 65536$$

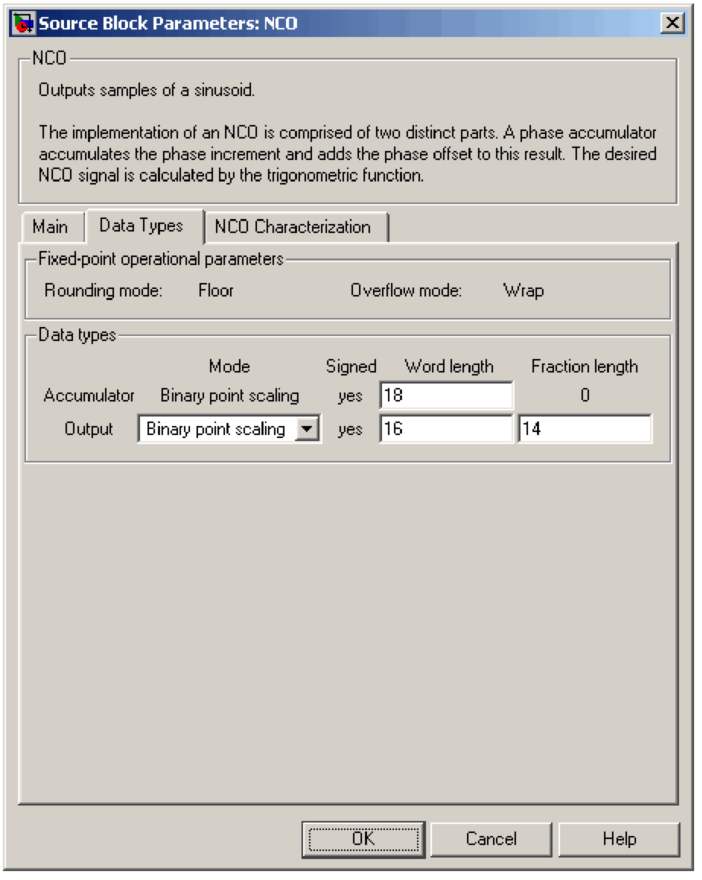
7

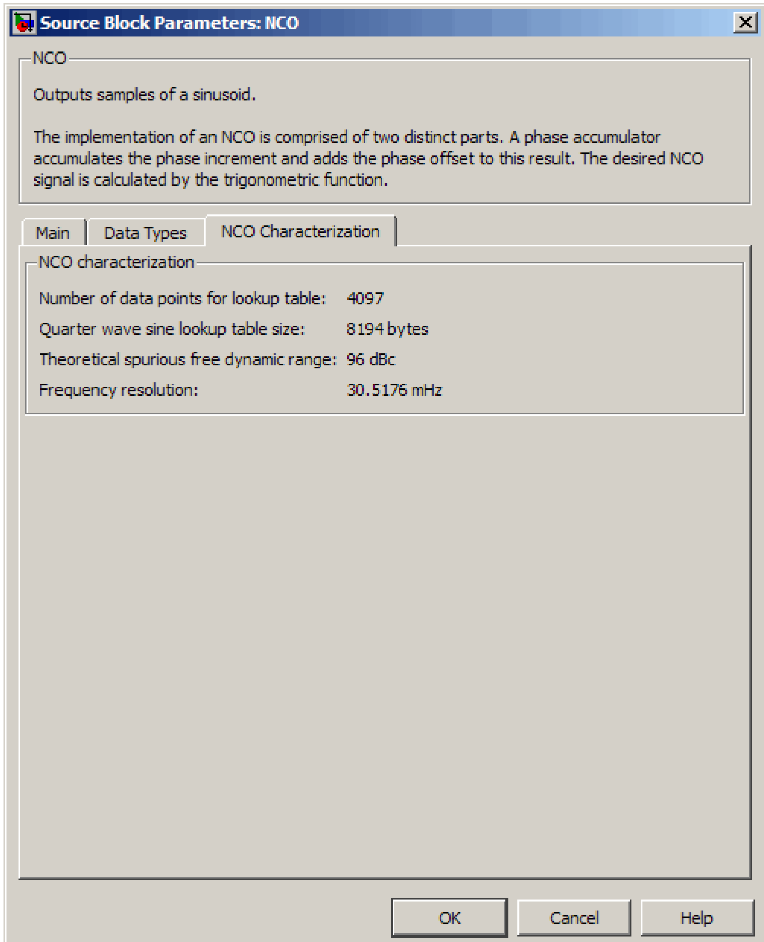
Type `doc_nco_example` at the MATLAB command line to open the following model:



The NCO block in the model is populated with the specifications and quantities you just calculated. The output word length and fraction length depend on the constraints of your hardware; this example uses a word length of 16 and a fraction length of 14. The three panes of the block mask appear as follows.


Source Block Parameters: NCO
NCO
Outputs samples of a sinusoid.
The implementation of an NCO is comprised of two distinct parts. A phase accumulator accumulates the phase increment and adds the phase offset to this result. The desired NCO signal is calculated by the trigonometric function.
Main
Data Types
NCO Characterization
Phase adder parameters
Phase increment source: Specify via dialog
Phase increment: 16417
Phase offset source: Specify via dialog
Phase offset: 65536
Add internal dither
Number of dither bits: 4
Quantize phase
Number of quantized accumulator bits: 14
Show phase quantization error port
Output parameters
Output signal: Sine
Sample time: 1/8000
Samples per frame: 8000
OK
Cancel
Help


Source Block Parameters: NCO
NCO
Outputs samples of a sinusoid.
The implementation of an NCO is comprised of two distinct parts. A phase accumulator accumulates the phase increment and adds the phase offset to this result. The desired NCO signal is calculated by the trigonometric function.
Main
Data Types
NCO Characterization
Fixed-point operational parameters
Rounding mode: Floor
Overflow mode: Wrap
Data types
Mode
Signed
Word length
Fraction length
Accumulator Binary point scaling yes 18 0
Output Binary point scaling yes 16 14
OK
Cancel
Help



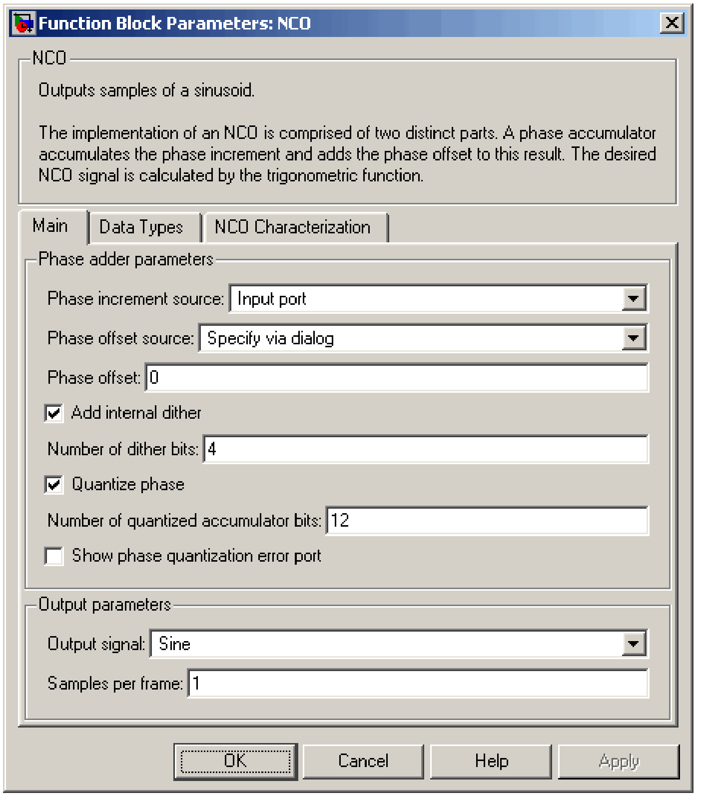
Looking at the **NCO Characterization** pane, you can verify that the specifications of this problem have been met.

8

Experiment with the model to observe the effects on the output shown on the Spectrum Scope. For example, try turning dithering on and off, and try changing the number of dither bits.

Dialog Box

The **Main** pane of the NCO dialog appears as follows.



Phase increment source

Choose how you specify the phase increment. The phase increment can come from an input port or from the dialog.

- If you select `Input port`, the inc port appears on the block icon.

- If you select `Specify via dialog`, the **Phase increment** parameter appears.

Phase increment

Specify the phase increment. Only integer data types, including fixed-point data types with zero fraction length, are allowed. The dimensions of the phase increment are dictated by those of the phase offset:

- When you specify the phase offset on the block dialog box, the phase increment must be a scalar or vector with the same length as the phase offset. Each element of the vector is applied to a different channel, and therefore the vector length defines the number of output channels.
- When you specify the phase offset via an input port, the phase increment length must be the same as the number of channels of a frame-based phase offset. If the phase offset is sample-based, the phase increment must have the same number of elements as the phase offset.

This parameter is visible only if `Specify via dialog` is selected for the **Phase increment source** parameter.

Phase offset source

Choose how you specify the phase offset. The phase offset can come from an input port or from the dialog.

- If you select `Input port`, the offset port appears on the block icon.
- If you select `Specify via dialog`, the **Phase offset** parameter appears.

When the phase offset comes in via an input port, it can be a scalar, a vector, or a full matrix. The frame status and dimensionality of the phase offset dictate that of the output. When a vector or matrix phase offset is sample based, the number of elements of the phase offset must match the number of channels of the data input. When a vector or matrix phase offset is frame based, different phase offsets are applied to each sample and channel per frame of the input. Only integer data types, including fixed-point data types with zero fraction length, are allowed.

Phase offset

Specify the phase offset. When you specify the phase offset using this parameter rather than via an input port, it must be a scalar or vector with the same length as the phase increment. Scalars are expanded to a vector with the same length as the phase increment. Each element of the phase offset vector is applied to a different channel of the input, and therefore the vector length defines the number of output channels. Only integer data types, including fixed-point data types with zero fraction length, are allowed.

This parameter is visible only if `Specify via dialog` is selected for the **Phase offset source** parameter.

Add internal dither

Select to add internal dithering to the NCO algorithm. Dithering is added using the PN Sequence Generator from the Communications Blockset™ product.

Number of dither bits

Specify the number of dither bits.

This parameter is visible only if **Add internal dither** is selected.

Quantize phase

Select to enable quantization of the accumulated phase.

Number of quantized accumulator bits

Specify the number of quantized accumulator bits. This determines the number of entries in the lookup table. The number of quantized accumulator bits must be less than the accumulator word length.

This parameter is visible only if **Quantize phase** is selected.

Show phase quantization error port

Select to output the phase quantization error. When you select this, the Qerr port appears on the block icon.

This parameter is visible only if **Quantize phase** is selected.

Output signal

Choose whether the block should output a `Sine`, `Cosine`, `Complex exponential`, or `Sine and cosine` signals. If you select `Sine and cosine`, the two signals output on different ports.

Sample time

Specify the sample time in seconds when the block is acting as a source. When either the phase increment or phase offset come in via block input ports, the sample time is inherited and this parameter is not visible.

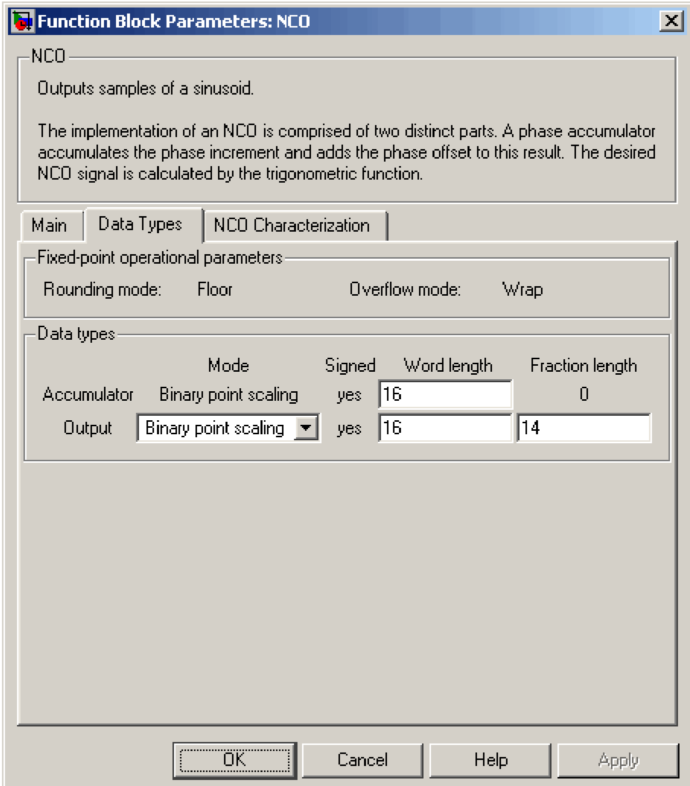
Samples per frame

Specify the number of samples per frame. When the value of this parameter is `1`, the block outputs a sample-based signal. When the value is greater than 1, the block outputs a frame-based signal of the specified size. In frame-based mode, the phase increment and phase offset can vary from channel to channel and from frame to frame, but they are constant along each channel in a given frame.

When the phase offset input port exists, it has the same frame status as any output port present. When the phase increment input port exists, it does not support frames.

This parameter is only visible if either **Phase increment source** or **Phase offset source** is set to `Specify via dialog`.

The **Data Types** pane of the NCO dialog appears as follows.



Rounding mode

The rounding mode used for this block when inputs are fixed point is always `Floor`.

Overflow mode

The overflow mode used for this block when inputs are fixed point is always `Wrap`.

Accumulator

Specify the word length of the accumulator data type. The fraction length is always zero; this is an integer data type.

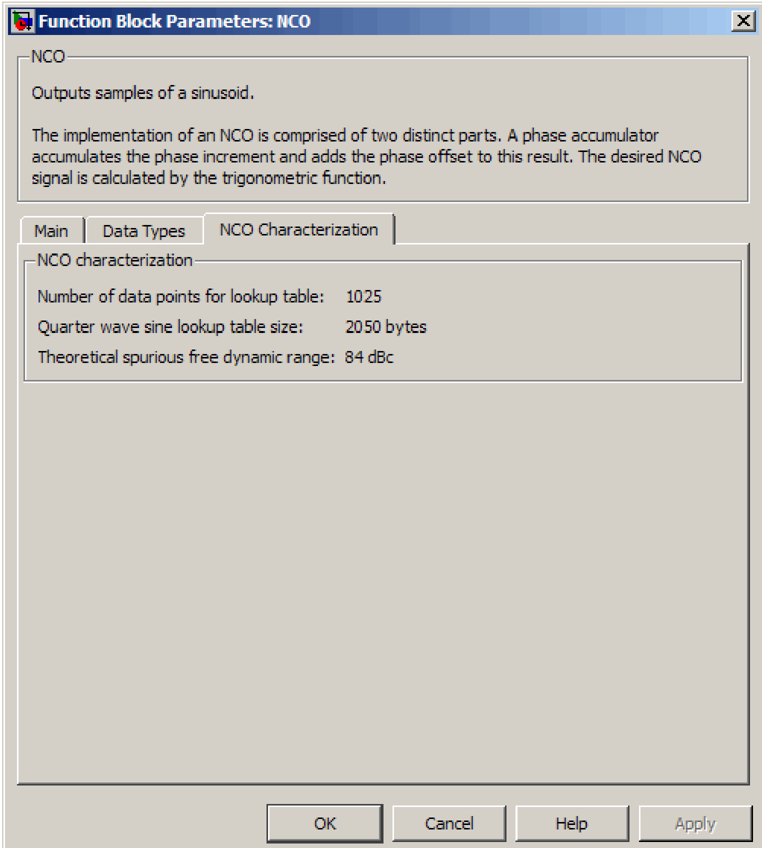
Output

Specify the output data type.

- Choose `double` or `single` for a floating-point implementation.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

Note The lookup table for this block is constructed from double-precision floating-point values. Thus, the maximum amount of precision you can achieve in your output is 53 bits. Setting the word length of the **Output** data type to values greater than 53 bits does not improve the precision of your output.

The **NCO Characterization** pane of the NCO dialog appears as follows.



The **NCO Characterization** pane does not have any parameters. Instead, it provides you with details on the NCO signal currently being implemented by the block:

- `Number of data points for lookup table` — The lookup table is implemented as a quarter-wave sine table. The number of lookup table data points is defined by

$$2^{\text{number of quantized accumulator bits}-2}+1$$

- `Quarter wave sine lookup table size` — The quarter wave sine lookup table size is defined by

$$\frac{(\text{number of data points for lookup table}) \cdot (\text{output word length})}{8} \text{ bytes}$$

- `Theoretical spurious free dynamic range` — The spurious free dynamic range (SFDR) is calculated as follows for a lookup table with 2^P entries:

$$SFDR = (6P) \text{ dB} \qquad \text{without dither}$$
$$SFDR = (6P + 12) \text{ dB} \qquad \text{with dither}$$

Supported Data Types

Port	Supported Data Types
inc	• Fixed point with zero fraction length • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
offset	• Fixed point with zero fraction length • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
sin	• Double-precision floating point • Single-precision floating point • Fixed point
Qerr	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

PN Sequence Generator	Communications Blockset
Sine Wave	Signal Processing Blockset

Purpose

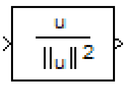
Perform vector normalization along rows, columns, or specified dimension

Library

Math Functions / Math Operations

`dspmathops`

Description



The Normalization block independently normalizes each row, column, or vector of the specified dimension of the input. The Normalization block accepts real and complex floating-point and fixed-point inputs except for complex unsigned fixed-point inputs. The block only accepts floating-point signals for the 2-norm mode, and both fixed-point and floating-point signals for the squared 2-norm mode. The output always has the same dimensions and frame status as the input.

This block treats an arbitrarily dimensioned input *U* as a collection of vectors oriented along the specified dimension. The block normalizes these vectors by either their norm or the square of their norm.

For example, consider a 3-dimensional input *U(i,j,k)* and assume that you want to normalize along the second dimension. First, define the 2-dimensional intermediate quantity *V(i,k)* such that each element of *V* is the norm of one of the vectors in *U*:

$$V(i,k) = \left(\sum_{j=1}^{J} U^2(i,j,k) \right)^{1/2}$$

Given *V*, the output of the block *Y(i, j,k)* in 2-norm mode is

$$Y(i,j,k) \frac{U(i,j,k)}{V(i,k)+b}$$

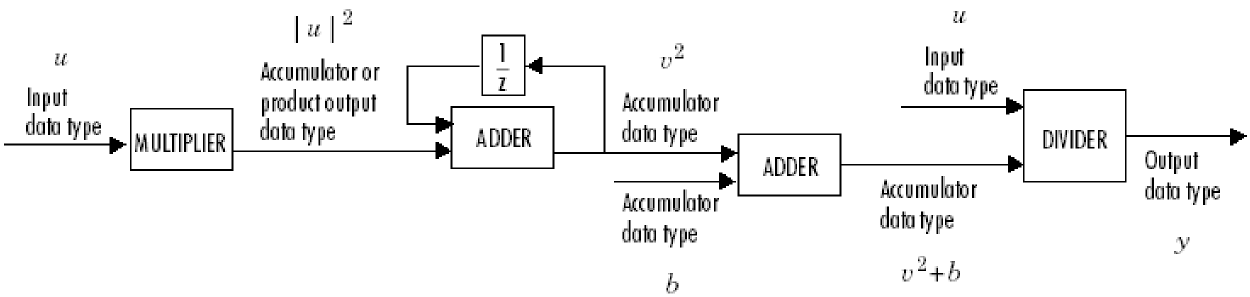
In squared 2-norm mode, the block output is

$$Y(i,j,k) = \frac{U(i,j,k)}{V(i,k)^2 + b}$$

The normalization bias, b, is typically chosen to be a small positive constant (for example, 1e-10) that prevents potential division by zero.

Fixed-Point Data Types

The following diagram shows the data types used within the Normalization block for fixed-point signals (squared 2-norm mode only).



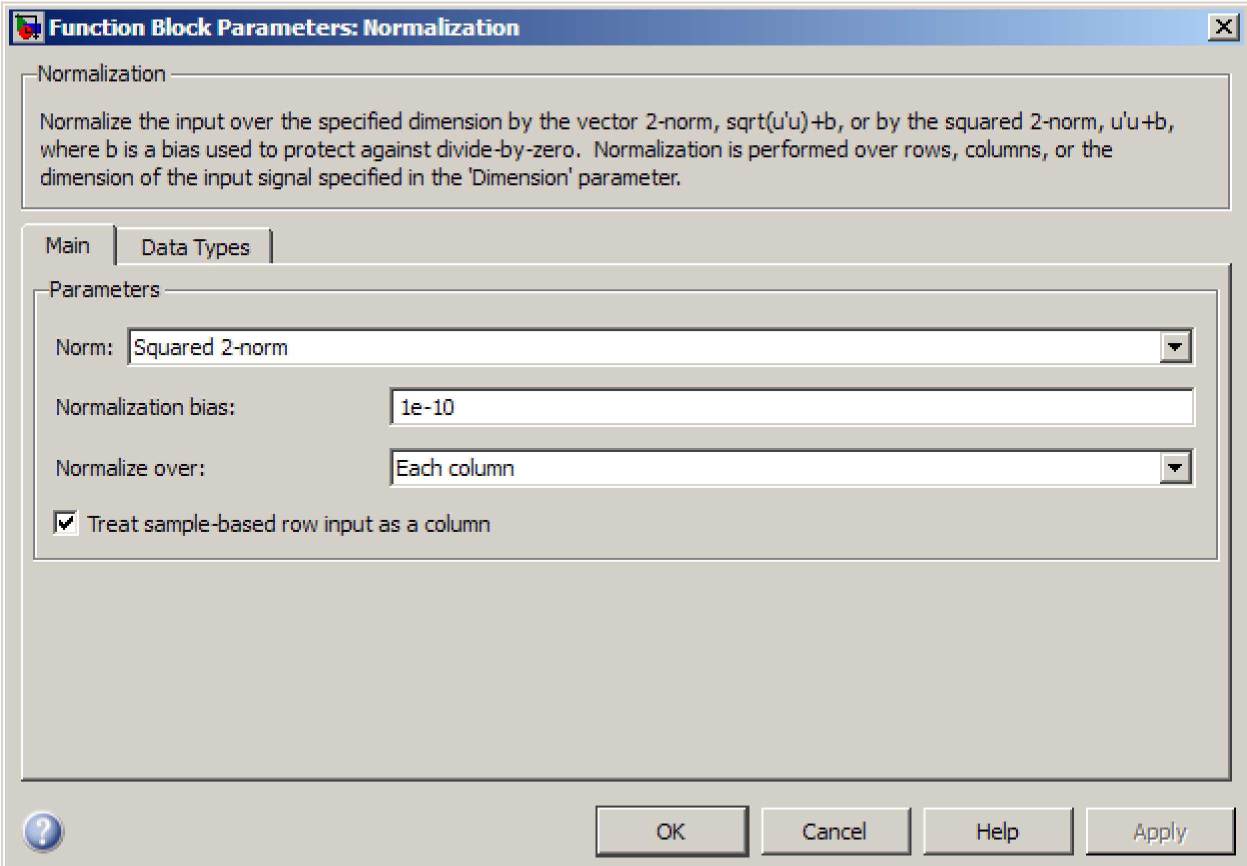
The output of the multiplier is in the product output data type when the input is real. When the input is complex, the result of the multiplication is in the accumulator data type. For details on the complex multiplication performed, see “Multiplication Data Types”. You can set the accumulator, product output, and output data types in the block dialog as discussed in “Dialog Box” on page 2-988.

Examples

See “Zero Algorithmic Delay” in the *Signal Processing Blockset User’s Guide* for an example.

Dialog Box

The **Main** pane of the Normalization dialog appears as follows.



Norm

Specify the type of normalization to perform, `2-norm` or `Squared 2-norm`. `2-norm` mode supports floating-point signals only. `Squared 2-norm` supports both fixed-point and floating-point signals.

Normalization bias

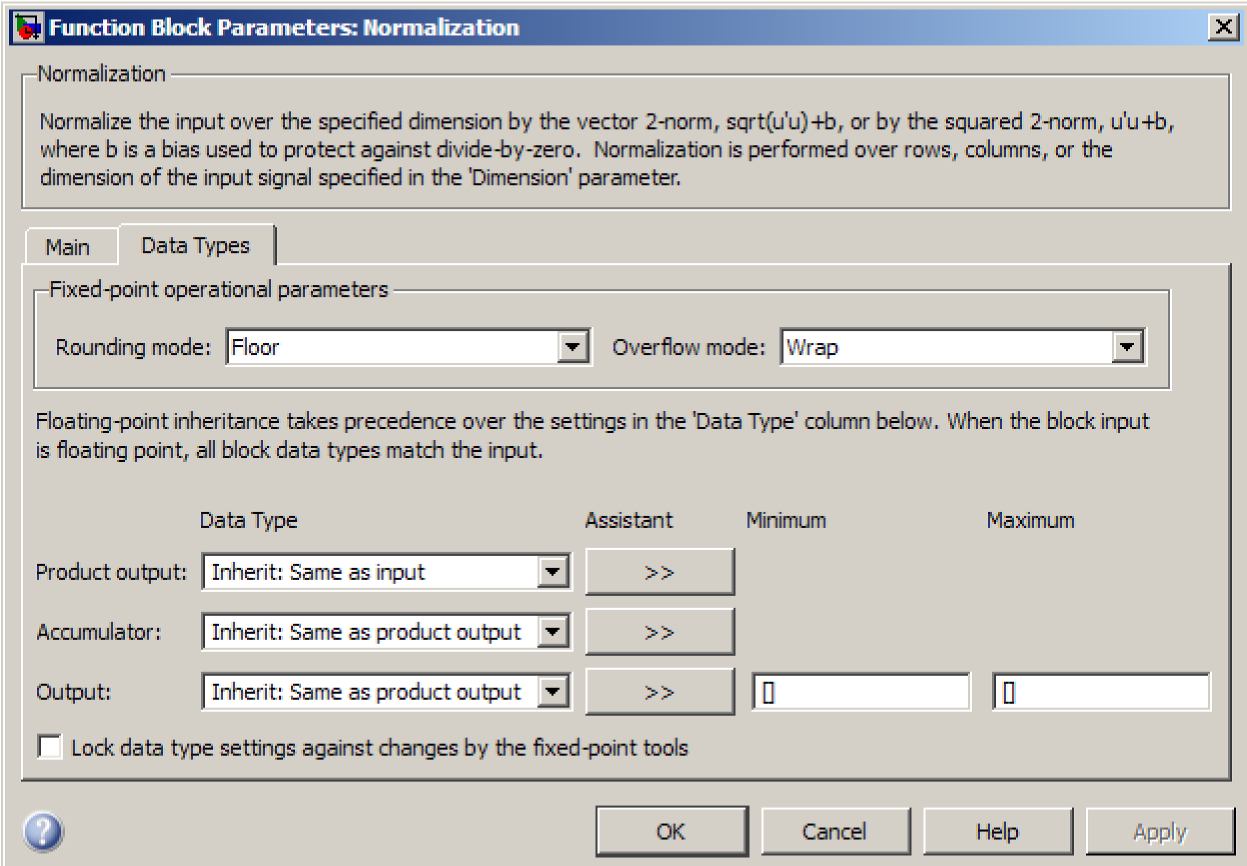
Specify the real value *b* to be added in the denominator to avoid division by zero. Tunable.

Normalize over
Specify whether to normalize along rows, columns, or the dimension specified in the **Dimension** parameter.

Dimension
Specify the one-based value of the dimension over which to normalize. The value of this parameter cannot exceed the number of dimensions in the input signal. This parameter is only visible if `Specified dimension` is selected for the **Normalize over** parameter.

Treat sample-based row input as column
Select to treat a sample-based row input as a column.

The **Data Types** pane of the Normalization dialog appears as follows.



Note The parameters on this pane are only applicable to fixed-point signals when the block is in squared 2-norm mode. See "Fixed-Point Data Types" on page 2-988 for a diagram of how the product output, accumulator, and output data types are used in this case.

Rounding mode
Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode for fixed-point operations.

Product output data type
Specify the product output data type. See “Fixed-Point Data Types” on page 2-988 and “Multiplication Data Types” for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Accumulator data type
Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-988 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same as product output`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Output data type

Specify the output data type. See "Fixed-Point Data Types" on page 2-988 for illustrations depicting the use of the output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as product output`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Array-Vector Multiply	Signal Processing Blockset
Reciprocal Condition	Signal Processing Blockset
`norm`	MATLAB

Purpose

Design Nyquist filter

Library

Filtering / Filter Designs

`dspfdesign`

Description

This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. You must have a Filter Design Toolbox license to design filters with this block. However, you can run models containing this block without a license. This allows you to run a model sent to you by a colleague who has designed a filter using this block, even if you do not have the Filter Design Toolbox product.

Dialog Box

See “Nyquist Filter Design Dialog Box — Main Pane” in the in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Purpose

Design octave filter

Library

Filtering / Filter Designs

`dspfdesign`

Description

This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. You must have a Filter Design Toolbox license to design filters with this block. However, you can run models containing this block without a license. This allows you to run a model sent to you by a colleague who has designed a filter using this block, even if you do not have the Filter Design Toolbox product.

Dialog Box

See "Octave Filter Design Dialog Box — Main Pane" in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Purpose

Truncate vectors by removing or keeping beginning or ending values

Library

Signal Operations

`dspsigops`

Description

The Offset block removes or keeps values from the beginning or end of a vector and outputs the result in a vector of user-specified length. The inputs to the In ports (In1, In2, ...) can be scalars or vectors, but they must be the same size and data type. The offset values are the inputs to the O ports (O1, O2, ...); they must be scalar values with the same data type. These offset values should be integer values because they determine the number of values the block discards or retains from each input vector. The block rounds any offset value that is a noninteger value to the nearest integer value. There is one output port for each pair of In and O ports. This block supports sample-based and frame-based signals.

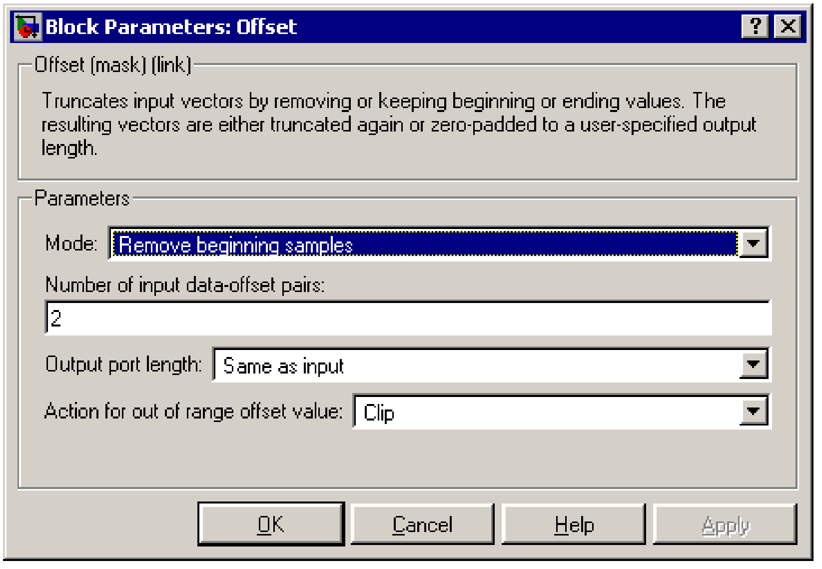
Use the **Mode** parameter to determine which values the block discards or retains from the input vector. To discard the initial values of the vector, select `Remove beginning samples`. To discard the final values of the vector, select `Remove ending samples`. To retain the initial values of the vector, select `Keep beginning samples`. To retain the final values of a vector, select `Keep ending` samples.

Use the **Number of input data-offset pairs** parameter to specify the number of inputs to the block. The number of input ports is twice the scalar value you enter. For example, if you enter `3`, ports In1, O1, In2, O2, In3, and O3 appear on the block.

The block uses the **Output port length** parameter to determine the length of the output vectors. If you select `Same as input`, the block outputs vectors that are the same length as the input to the In ports. If you select `User-defined`, the **Output length** parameter appears. Enter a scalar that represents the desired length of the output vectors. If your desired output length is greater than the number of values you extracted from your input vector, the block zero-pads the end of the vector to reach the length you specified.

Use the **Action for out of range offset value** parameter to determine how the block behaves when an offset value is not in the range 0 ≤ offset value ≤ N, where N is the input vector length. Select `Clip` if you want any offset values less than 0 to be set to 0 and any offset values greater than N to be set to N. Select `Clip and warn` if you want to be warned when any offset values less than 0 are set to 0 and any offset values greater than N are set to N. Select `Error` if you want the simulation to stop and display an error when the offset values are out of range.

Dialog Box



Mode

Use this parameter to determine which values the block discards or retains from the input vector. Your choices are `Remove beginning samples`, `Remove ending samples`, `Keep beginning samples`, and `Keep ending samples`.

Number of input data-offset pairs

Specify the number of inputs to the block. The number of input ports is twice the scalar value you enter.

Output port length
Use this parameter to specify the length of the output vectors. If you select `Same as input`, the output vectors are the same length as the input vectors. If you select `User-defined`, you can enter the desired length of the output vectors.

Output length
Enter a scalar that represents the desired length of the output vectors. This parameter is visible if, for the **Output port length** parameter, you select `User-defined`.

Action for out of range offset value
Use this parameter to determine how the block behaves when an offset value is not in the range such that $0 \leq$ offset value $\leq N$, where N is the input vector length. When you want any offset values less than 0 to be set to 0 and any offset values greater than N to be set to N, select `Clip`. When you want to be warned when any offset values less than 0 are set to 0 and any offset values greater than N are set to N, select `Clip and warn`. When you want the simulation to stop and display an error when the offset values are out of range, select `Error`.

Supported Data Types

Port	Supported Data Types
In	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
O	• Double-precision floating point • Single-precision floating point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Out	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

Purpose

Implement overlap-add method of frequency-domain filtering

Library

Filtering / Filter Implementations

`dsparch4`

Description

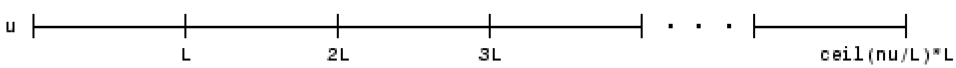
The Overlap-Add FFT Filter block uses an FFT to implement the *overlap-add method*, a technique that combines successive frequency-domain filtered sections of an input sequence.

Valid inputs to this block are 1-D vectors, sample-based vectors, frame-based vectors, and frame-based full matrices. All outputs are unbuffered into sample-based row vectors. The length of the output vector is equal to the number of channels in the input vector. An M-by-1 *sample-based* input has M channels, so it would result in a length-M sample-based output vector. An M-by-1 *frame-based* input has only one channel, so would result in a 1-by-1 (scalar) output.

The block's data output rate is M times faster than its data input rate, where M is the input frame-size. Thus, the block's data input and output rates are the same when the inputs are 1-D vectors, sample-based vectors, or frame-based row vectors. For frame-based column and frame-based full-matrix inputs, the block's data output rate is M times greater than the block's data input rate.

1-D vectors are treated as length-N sample-based vectors, and result in sample-based length-N row vectors.

The block breaks the scalar input sequence `u`, of length `nu`, into length-`L` nonoverlapping data sections,



which it linearly convolves with the filter's FIR coefficients,

$$H(z) = B(z) = b_1 + b_2 z^{-1} + \ldots + b_{n+1} z^{-n}$$

The numerator coefficients for H(z) are specified as a vector by the **FIR coefficients** parameter. The coefficient vector, `b = [b(1) b(2) ...`

`b(n+1)]`, can be generated by one of the filter design functions in the Signal Processing Toolbox product, such as `fir1`. All filter states are internally initialized to zero.

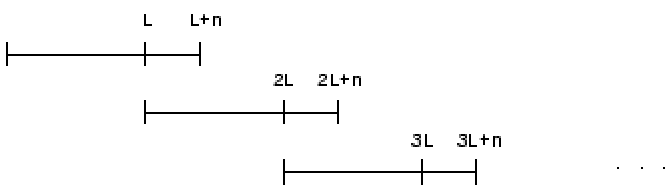
When either the filter coefficients or the inputs to the block are complex, the **Output** parameter should be set to `Complex`. Otherwise, the default **Output** setting, `Real`, instructs the block to take only the real part of the solution.

The block's overlap-add operation is equivalent to

```
y = ifft(fft(u(i:i+L-1),nfft) .* fft(b,nfft))
```

where you specify `nfft` in the **FFT size** parameter as a power-of-two value greater (typically *much* greater) than `n+1`. Values for **FFT size** that are not powers of two are rounded upwards to the nearest power-of-two value to obtain `nfft`.

The block overlaps successive output sections by `n` points and sums them.



The first `L` samples of each summation are output in sequence. The block chooses the parameter `L` based on the filter order and the FFT size.

```
L = nfft - n
```

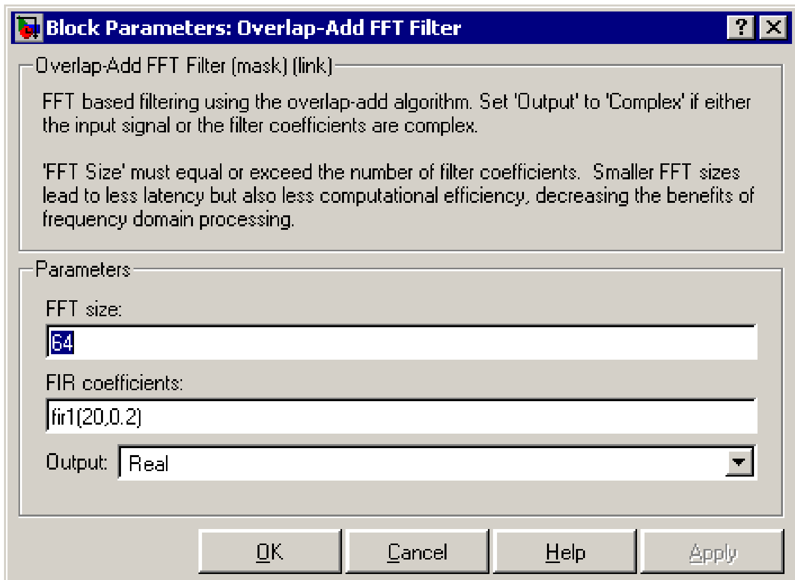
Latency

In *single-tasking* operation, the Overlap-Add FFT Filter block has a latency of `nfft-n+1` samples. The first `nfft-n+1` consecutive outputs from the block are zero; the first filtered input value appears at the output as sample `nfft-n+2`.

In *multitasking* operation, the Overlap-Add FFT Filter block has a latency of `2*(nfft-n+1)` samples. The first `2*(nfft-n+1)` consecutive outputs from the block are zero; the first filtered input value appears at the output as sample `2*(nfft-n)+3`.

Note For more information on latency and the Simulink software tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

Dialog Box



FFT size
: The size of the FFT, which should be a power-of-two value greater than the length of the specified FIR filter.

FIR coefficients
: The filter numerator coefficients.

Output

The complexity of the output; `Real` or `Complex`. When the input signal or the filter coefficients are complex, this should be set to `Complex`.

References

Oppenheim, A. V. and R. W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

Proakis, J. and D. Manolakis. *Digital Signal Processing*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Overlap-Save FFT Filter	Signal Processing Blockset product

Overlap-Save FFT Filter

Purpose

Implement overlap-save method of frequency-domain filtering

Library

Filtering / Filter Implementations

`dsparch4`

Description

The Overlap-Save FFT Filter block uses an FFT to implement the *overlap-save method*, a technique that combines successive frequency-domain filtered sections of an input sequence.

Valid inputs to this block are 1-D vectors, sample-based vectors, frame-based vectors, and frame-based full matrices. All outputs are unbuffered into sample-based row vectors. The length of the output vector is equal to the number of channels in the input vector. An M-by-1 sample-based input has M channels, so it would result in a length-M sample-based output vector. An M-by-1 frame-based input has only one channel, so would result in a 1-by-1 (scalar) output.

The block's data output rate is M times faster than its data input rate, where M is the input frame-size. Thus, the block's data input and output rates are the same when the inputs are 1-D vectors, sample-based vectors, or frame-based row vectors. For frame-based column and frame-based full-matrix inputs, the block's data output rate is M times greater than the block's data input rate.

1-D vectors are treated as length-N sample-based vectors, and result in sample-based length-N row vectors.

Overlapping sections of input `u` are circularly convolved with the FIR filter coefficients

$$H(z) = B(z) = b_1 + b_2 z^{-1} + \ldots + b_{n+1} z^{-n}$$

The numerator coefficients for $H(z)$ are specified as a vector by the **FIR coefficients** parameter. The coefficient vector, `b = [b(1) b(2) ... b(n+1)]`, can be generated by one of the filter design functions in the Signal Processing Toolbox product, such as `fir1`. All filter states are internally initialized to zero.

When either the filter coefficients or the inputs to the block are complex, the **Output** parameter should be set to `Complex`. Otherwise, the default **Output** setting, `Real`, instructs the block to take only the real part of the solution.

The circular convolution of each section is computed by multiplying the FFTs of the input section and filter coefficients, and computing the inverse FFT of the product.

```
y = ifft(fft(u(i:i+(L-1)),nfft) .* fft(b,nfft))
```

where you specify `nfft` in the **FFT size** parameter as a power of two value greater (typically *much* greater) than `n+1`. Values for **FFT size** that are not powers of two are rounded upwards to the nearest power-of-two value to obtain `nfft`.

The first `n` points of the circular convolution are invalid and are discarded. The Overlap-Save FFT Filter block outputs the remaining `nfft-n` points, which are equivalent to the linear convolution.

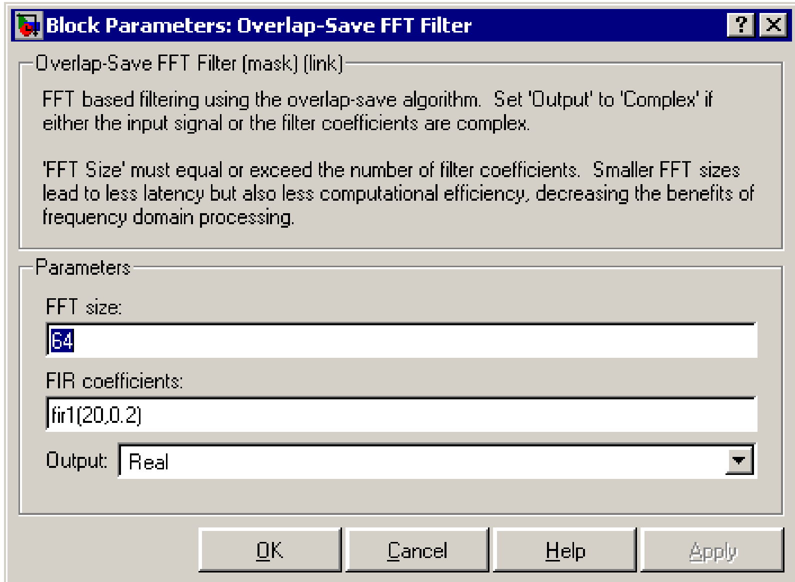
Latency

In *single-tasking* operation, the Overlap-Save FFT Filter block has a latency of `nfft-n+1` samples. The first `nfft-n+1` consecutive outputs from the block are zero; the first filtered input value appears at the output as sample `nfft-n+2`.

In *multitasking* operation, the Overlap-Save FFT Filter block has a latency of `2*(nfft-n+1)` samples. The first `2*(nfft-n+1)` consecutive outputs from the block are zero; the first filtered input value appears at the output as sample `2*(nfft-n)+3`.

Note For more information on latency and the Simulink environment tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

Dialog Box



FFT size
: The size of the FFT, which should be a power of two value greater than the length of the specified FIR filter.

FIR coefficients
: The filter numerator coefficients.

Output
: The complexity of the output; `Real` or `Complex`. When the input signal or the filter coefficients are complex, this should be set to `Complex`.

References

Oppenheim, A. V. and R. W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

Proakis, J. and D. Manolakis. *Digital Signal Processing*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Overlap-Add FFT Filter	Signal Processing Blockset

Purpose

Overwrite submatrix or subdiagonal of input

Library

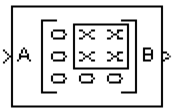
- Math Functions / Matrices and Linear Algebra / Matrix Operations

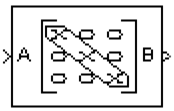
  `dspmtrx3`

- Signal Management / Indexing

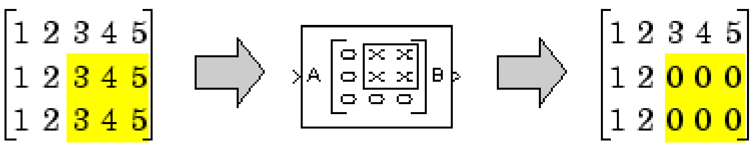
  `dspindex`

Description





The Overwrite Values block overwrites a contiguous submatrix or subdiagonal of an input matrix. You can provide the overwriting values by typing them in a block parameter, or through an additional input port, which is useful for providing overwriting values that change at each time step.



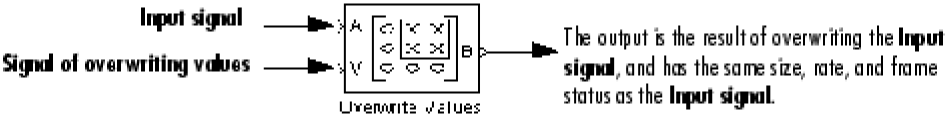
The block accepts both sample- and frame-based vectors and matrices. The output has the same size and frame status as the original input signal, not necessarily the same size and frame status as the signal containing the overwriting values. The input(s) and output of this block must have the same data type.

Specifying the Overwriting Values

The **Source of overwriting value(s)** parameter determines how you must provide the overwriting values, and has the following settings.

- `Specify via dialog` — You must provide the overwriting value(s) in the **Overwrite with** parameter. The block uses the same overwriting values to overwrite the specified portion of the input at each time step. To learn how to specify valid overwriting values, see "Valid Overwriting Values" on page 2-1012.
- `Second input port` — You must provide overwriting values through a second block input port, `V`. Use this setting to provide different

overwriting values at each time step. The output inherits its size, rate, and frame status from the input signal, *not* the overwriting values.



The rate at which you provide the overwriting values through input port V must match the rate at which the block receives each input matrix at input port A. The rate requirements depend on whether the input signal and overwriting values signal have the same frame status:

- When both signals are sample based, their sample rates must be the same.
- When both signals are frame based, their frame rates must be the same.
- When one signal is sample based and one signal is frame based, the sample rate of the sample-based signal must be the same as the frame rate of the frame-based signal.

Valid Overwriting Values

The overwriting values can be a single constant, vector, or matrix, depending on the portion of the input you are overwriting, regardless of whether you provide the overwriting values through an input port or by providing them in the **Overwrite with** parameter.

Valid Overwriting Values

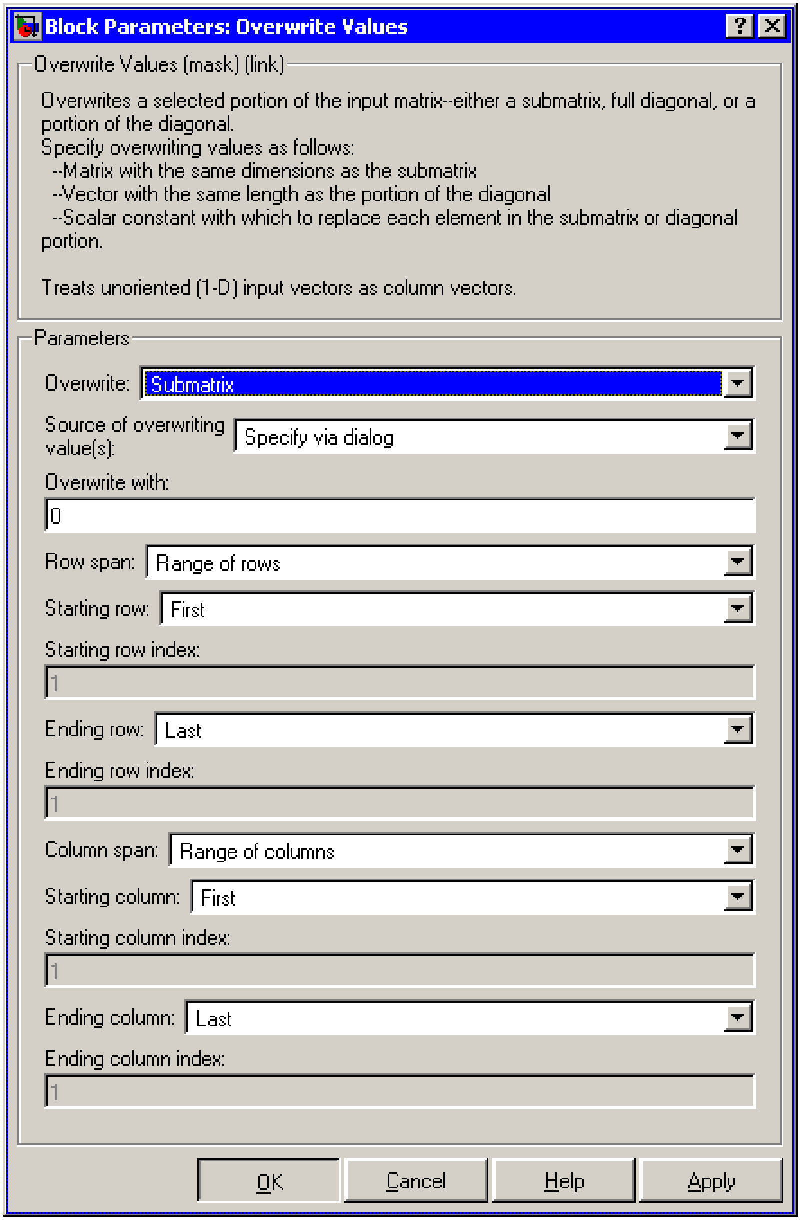
Portion of Input to Overwrite	Valid Overwriting Values	Example
A single element in the input $\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}$	Any constant value, v	$v = 9$ $\begin{bmatrix} x & x & x & x & x \\ x & x & x & 9 & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}$
A length-k portion of the diagonal $\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}$	Any length-k column or row vector, v	$k = 3 \quad v = [2 \quad 4 \quad 6]$ or $\begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$ $\begin{bmatrix} 2 & x & x & x & x \\ x & 4 & x & x & x \\ x & x & 6 & x & x \\ x & x & x & x & x \end{bmatrix}$
A length-k portion of a row $\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}$	Any length-k row vector, v	$k = 3 \quad v = [2 \quad 4 \quad 6]$ $\begin{bmatrix} x & x & x & x & x \\ x & 2 & 4 & 6 & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}$

Valid Overwriting Values (Continued)

Portion of Input to Overwrite	Valid Overwriting Values	Example
A length-k portion of a column $\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}$	Any length-k column vector, v	$k = 2 \quad v = \begin{bmatrix} 4 \\ 6 \end{bmatrix}$ $\begin{bmatrix} x & x & x & x & x \\ x & x & x & 4 & x \\ x & x & x & 6 & x \\ x & x & x & x & x \end{bmatrix}$
An m-by-n submatrix $\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}$	Any m-by-n matrix, v	$m = 2$ $n = 3 \quad v = \begin{bmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ $\begin{bmatrix} x & x & x & x & x \\ x & x & 4 & 5 & 6 \\ x & x & 7 & 8 & 9 \\ x & x & x & x & x \end{bmatrix}$

This block supports Simulink virtual buses.

Dialog Box


Block Parameters: Overwrite Values
Overwrite Values (mask) (link)
Overwrites a selected portion of the input matrix--either a submatrix, full diagonal, or a portion of the diagonal.
Specify overwriting values as follows:
--Matrix with the same dimensions as the submatrix
--Vector with the same length as the portion of the diagonal
--Scalar constant with which to replace each element in the submatrix or diagonal portion.
Treats unoriented (1-D) input vectors as column vectors.
Parameters
Overwrite:
Submatrix
Source of overwriting value(s):
Specify via dialog
Overwrite with:
0
Row span:
Range of rows
Starting row:
First
Starting row index:
1
Ending row:
Last
Ending row index:
1
Column span:
Range of columns
Starting column:
First
Starting column index:
1
Ending column:
Last
Ending column index:
1
OK
Cancel
Help
Apply

Note Only some of the following parameters are visible in the dialog box at any one time.

Overwrite
: Determines whether to overwrite a specified submatrix or a specified portion of the diagonal.

Source of overwriting value(s)
: Determines where you must provide the overwriting values: either through an input port, or by providing them in the **Overwrite with** parameter. For more information, see "Specifying the Overwriting Values" on page 2-1011.

Overwrite with
: The value(s) with which to overwrite the specified portion of the input matrix. Enabled only when **Source of overwriting value(s)** is set to `Specify via dialog`. To learn how to specify valid overwriting values, see "Valid Overwriting Values" on page 2-1012.

Row span
: The range of input rows to be overwritten. Options are `All rows`, `One row`, or `Range of rows`. For descriptions of these options, see "Dialog Box" on page 2-1015.

Row/Starting row
: The input row that is the first row of the submatrix that the block overwrites. For a description of the options for the **Row** and **Starting row** parameters, see Settings for Row, Column, Starting Row, and Starting Column Parameters on page 2-1021. **Row** is enabled when **Row span** is set to `One row`, and **Starting row** when **Row span** is set to `Range of rows`.

Row index/Starting row index
: Index of the input row that is the first row of the submatrix that the block overwrites. See how to use these parameters in Settings for Row, Column, Starting Row, and Starting Column Parameters

on page 2-1021. **Row index** is enabled when **Row** is set to `Index`, and **Starting row index** when **Starting row** is set to `Index`.

Row offset/Starting row offset
: The offset of the input row that is the first row of the submatrix that the block overwrites. See how to use these parameters in Settings for Row, Column, Starting Row, and Starting Column Parameters on page 2-1021. **Row offset** is enabled when **Row** is set to `Offset from middle` or `Offset from last`, and **Starting row offset** is enabled when **Starting row** is set to `Offset from middle` or `Offset from last`.

Ending row
: The input row that is the last row of the submatrix that the block overwrites. For a description of this parameter's options, see Settings for Ending Row and Ending Column Parameters on page 2-1022. This parameter is enabled when **Row span** is set to `Range of rows`, and **Starting row** is set to any option but `Last`.

Ending row index
: Index of the input row that is the last row of the submatrix that the block overwrites. See how to use this parameter in Settings for Ending Row and Ending Column Parameters on page 2-1022. Enabled when **Ending row** is set to `Index`.

Ending row offset
: The offset of the input row that is the last row of the submatrix that the block overwrites. See how to use this parameter in Settings for Ending Row and Ending Column Parameters on page 2-1022. Enabled when **Ending row** is set to `Offset from middle` or `Offset from last`.

Column span
: The range of input columns to be overwritten. Options are `All columns`, `One column`, or `Range of columns`. For descriptions of the analogous row options, see "Dialog Box" on page 2-1015.

Column/Starting column
: The input column that is the first column of the submatrix that the block overwrites. For a description of the options for the

Column and **Starting column** parameters, see Settings for Row, Column, Starting Row, and Starting Column Parameters on page 2-1021. **Column** is enabled when **Column span** is set to `One column`, and **Starting column** when **Column span** is set to `Range of columns`.

Column index/Starting column index
Index of the input column that is the first column of the submatrix that the block overwrites. See how to use these parameters in Settings for Row, Column, Starting Row, and Starting Column Parameters on page 2-1021. **Column index** is enabled when **Column** is set to `Index`, and **Starting column index** when **Starting column** is set to `Index`.

Column offset/Starting column offset
The offset of the input column that is the first column of the submatrix that the block overwrites. See how to use these parameters in Settings for Row, Column, Starting Row, and Starting Column Parameters on page 2-1021. **Column offset** is enabled when **Column** is set to `Offset from middle` or `Offset from last`, and **Starting column offset** is enabled when **Starting column** is set to `Offset from middle` or `Offset from last`.

Ending column
The input column that is the last column of the submatrix that the block overwrites. For a description of this parameter's options, see Settings for Ending Row and Ending Column Parameters on page 2-1022. This parameter is enabled when **Column span** is set to `Range of columns`, and **Starting column** is set to any option but `Last`.

Ending column index
Index of the input column that is the last column of the submatrix that the block overwrites. See how to use this parameter in Settings for Ending Row and Ending Column Parameters on page 2-1022. This parameter is enabled when **Ending column** is set to `Index`.

Ending column offset
The offset of the input column that is the last column of the submatrix that the block overwrites. See how to use this parameter in Settings for Ending Row and Ending Column Parameters on page 2-1022. This parameter is enabled when **Ending column** is set to `Offset from middle` or `Offset from last`.

Diagonal span
The range of diagonal elements to be overwritten. Options are `All elements`, `One element`, or `Range of elements`. For descriptions of these options, see "Overwriting a Subdiagonal" on page 2-1025.

Element/Starting element
The input diagonal element that is the first element in the subdiagonal that the block overwrites. For a description of the options for the **Element** and **Starting element** parameters, see Element and Starting Element Parameters on page 2-1025. **Element** is enabled when **Element span** is set to `One element`, and **Starting element** when **Element span** is set to `Range of elements`.

Element index/Starting element index
Index of the input diagonal element that is the first element of the subdiagonal that the block overwrites. See how to use these parameters in Element and Starting Element Parameters on page 2-1025. **Element index** is enabled when **Element** is set to `Index`, and **Starting element index** when **Starting element** is set to `Index`.

Element offset/Starting element offset
The offset of the input diagonal element that is the first element of the subdiagonal that the block overwrites. See how to use these parameters in Element and Starting Element Parameters on page 2-1025. **Element offset** is enabled when **Element** is set to `Offset from middle` or `Offset from last`, and **Starting element offset** is enabled when **Starting element** is set to `Offset from middle` or `Offset from last`.

Ending element

The input diagonal element that is the last element of the subdiagonal that the block overwrites. For a description of this parameter's options, see Ending Element Parameters on page 2-1026. This parameter is enabled when **Element span** is set to `Range of elements`, and **Starting element** is set to any option but `Last`.

Ending element index

Index of the input diagonal element that is the last element of the subdiagonal that the block overwrites. See how to use this parameter in Ending Element Parameters on page 2-1026. This parameter is enabled when **Ending element** is set to `Index`.

Ending element offset

The offset of the input diagonal element that is the last element of the subdiagonal that the block overwrites. See how to use this parameter in Ending Element Parameters on page 2-1026. This parameter is enabled when **Ending element** is set to `Offset from middle` or `Offset from last`.

Examples

Overwriting a Submatrix

To overwrite a submatrix, follow these steps:

1. Set the **Overwrite** parameter to `Submatrix`.

2. Specify the overwriting values as described in "Specifying the Overwriting Values" on page 2-1011.

3. Specify which rows and columns of the input matrix are contained in the submatrix that you want to overwrite by setting the **Row span** parameter to one of the following options and the **Column span** to the analogous column-related options:

   - `All rows` — The submatrix contains all rows of the input matrix.

   - `One row` — The submatrix contains only one row of the input matrix, which you must specify in the **Row** parameter, as described in the following table.

- `Range of rows` — The submatrix contains one or more rows of the input, which you must specify in the **Starting Row** and **Ending row** parameters, as described in the following tables.

4 When you set **Row span** to `One row` or `Range of rows`, you need to further specify the row(s) contained in the submatrix by setting the **Row** or **Starting row** and **Ending row** parameters. Likewise, when you set **Column span** to `One column` or `Range of columns`, you must further specify the column(s) contained in the submatrix by setting the **Column** or **Starting column** and **Ending column** parameters. For descriptions of the settings for these parameters, see the following tables.

Settings for Row, Column, Starting Row, and Starting Column Parameters

Settings for Specifying the Submatrix's First Row or Column	First Row of Submatrix (Only row for Row span = One row)	First Column of Submatrix (Only row for Row span = One row)
`First`	First row of the input	First column of the input
`Index`	Input row specified in the **Row index** parameter	Input column specified in the **Column index** parameter
`Offset from last`	Input row with the index `M - rowOffset` where `M` is the number of input rows, and `rowOffset` is the value of the **Row offset** or **Starting row offset** parameter	Input column with the index `N - colOffset` where `N` is the number of input columns, and `colOffset` is the value of the **Column offset** or **Starting column offset** parameter
`Last`	Last row of the input	Last column of the input

Settings for Row, Column, Starting Row, and Starting Column Parameters (Continued)

Settings for Specifying the Submatrix's First Row or Column	First Row of Submatrix (Only row for Row span = One row)	First Column of Submatrix (Only row for Row span = One row)
`Offset from middle`	Input row with the index `floor(M/2 + 1 - rowOffset)` where `M` is the number of input rows, and `rowOffset` is the value of the **Row offset** or **Starting row offset** parameter	Input column with the index `floor(N/2 + 1 - rowOffset)` where `N` is the number of input columns, and `colOffset` is the value of the or **Column offset** or **Starting column offset** parameter
`Middle`	Input row with the index `floor(M/2 + 1)` where `M` is the number of input rows	Input columns with the index `floor(N/2 + 1)` where `N` is the number of input columns

Settings for Ending Row and Ending Column Parameters

Settings for Specifying the Submatrix's Last Row or Column	Last Row of Submatrix	Last Column of Submatrix
`Index`	Input row specified in the **Ending row index** parameter	Input column specified in the **Ending column index** parameter
`Offset from last`	Input row with the index `M - rowOffset` where `M` is the number of input	Input column with the index `N - colOffset` where `N` is the number of input

Settings for Ending Row and Ending Column Parameters (Continued)

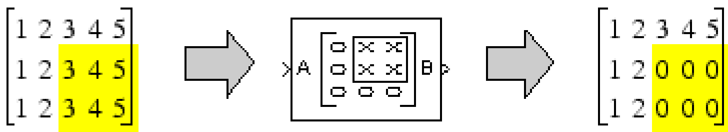
Settings for Specifying the Submatrix's Last Row or Column	Last Row of Submatrix	Last Column of Submatrix
	rows, and `rowOffset` is the value of the **Ending row offset** parameter	columns, and `colOffset` is the value of the **Ending column offset** parameter
`Last`	Last row of the input	Last column of the input
`Offset from middle`	Input row with the index `floor(M/2 + 1 - rowOffset)` where `M` is the number of input rows, and `rowOffset` is the value of the **Ending row offset** parameter	Input column with the index `floor(N/2 + 1 - rowOffset)` where `N` is the number of input columns, and `colOffset` is the value of the **Ending column offset** parameter
`Middle`	Input row with the index `floor(M/2 + 1)` where `M` is the number of input rows	Input columns with the index `floor(N/2 + 1)` where `N` is the number of input columns

For example, to overwrite the lower-right 2-by-3 submatrix of a 3-by-5 input matrix with all zeros, enter the following set of parameters:

- **Overwrite** = `Submatrix`
- **Source of overwriting value(s)** = `Specify via dialog`
- **Overwrite with** = `0`
- **Row span** = `Range of rows`
- **Starting row** = `Index`
- **Starting row index** = `2`
- **Ending row** = `Last`

- **Column span** = `Range of columns`
- **Starting column** = `Offset from last`
- **Starting column offset** = `2`
- **Ending column** = `Last`

The following figure shows the block with the above settings overwriting a portion of a 3-by-5 input matrix.



There are often several possible parameter combinations that select the *same* submatrix from the input. For example, instead of specifying `Last` for **Ending column**, you could select the same submatrix by specifying

- **Ending column** = `Index`
- **Ending column index** = `5`

Overwriting a Subdiagonal

To overwrite a subdiagonal, follow these steps:

1. Set the **Overwrite** parameter to `Diagonal`.

2. Specify the overwriting values as described in "Specifying the Overwriting Values" on page 2-1011.

3. Specify the subdiagonal that you want to overwrite by setting the **Diagonal span** parameter to one of the following options:

   - `All elements` — Overwrite the entire input diagonal.
   - `One element` — Overwrite one element in the diagonal, which you must specify in the **Element** parameter (described below).
   - `Range of elements` — Overwrite a portion of the input diagonal, which you must specify in the **Starting element** and **Ending element** parameters, as described in the following table.

4. When you set **Diagonal span** to `One element` or `Range of elements`, you need to further specify which diagonal element(s) to overwrite by setting the **Element** or **Starting element** and **Ending element** parameters. See the following tables.

Element and Starting Element Parameters

Settings for Element and Starting Element Parameters	First Element in Subdiagonal (Only element when Diagonal span = One element)
`First`	Diagonal element in first row of the input
`Index`	*k*th diagonal element, where *k* is the value of the **Element index** or **Starting element index** parameter

Element and Starting Element Parameters (Continued)

Settings for Element and Starting Element Parameters	First Element in Subdiagonal (Only element when Diagonal span = One element)
`Offset from last`	Diagonal element in the row with the index `M - offset` where `M` is the number of input rows, and `offset` is the value of the **Element offset** or **Starting element offset** parameter
`Last`	Diagonal element in the last row of the input
`Offset from middle`	Diagonal element in the input row with the index `floor(M/2 + 1 - offset)` where `M` is the number of input rows, and `offset` is the value of the **Element offset** or **Starting element offset** parameter
`Middle`	Diagonal element in the input row with the index `floor(M/2 + 1)` where `M` is the number of input rows

Ending Element Parameters

Settings for Ending Element Parameter	Last Element in Subdiagonal
`Index`	kth diagonal element, where k is the value of the **Ending element index** parameter
`Offset from last`	Diagonal element in the row with the index `M - offset` where `M` is the number of input rows, and `offset` is the value of the **Ending element offset** parameter
`Last`	Diagonal element in the last row of the input

Ending Element Parameters (Continued)

Settings for Ending Element Parameter	Last Element in Subdiagonal
`Offset from middle`	Diagonal element in the input row with the index `floor(M/2 + 1 - offset)` where `M` is the number of input rows, and `offset` is the value of the **Ending element offset** parameter
`Middle`	Diagonal element in the input row with the index `floor(M/2 + 1)` where `M` is the number of input rows

Supported Data Types

The input(s) and output of this block must have the same data type.

Port	Supported Data Types
A	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
V	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
B	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Reshape	Simulink
Selector	Simulink
Submatrix	Signal Processing Blockset
Variable Selector	Signal Processing Blockset
`reshape`	MATLAB

Purpose

Pad or truncate specified dimension(s)

Library

Signal Operations

`dspsigops`

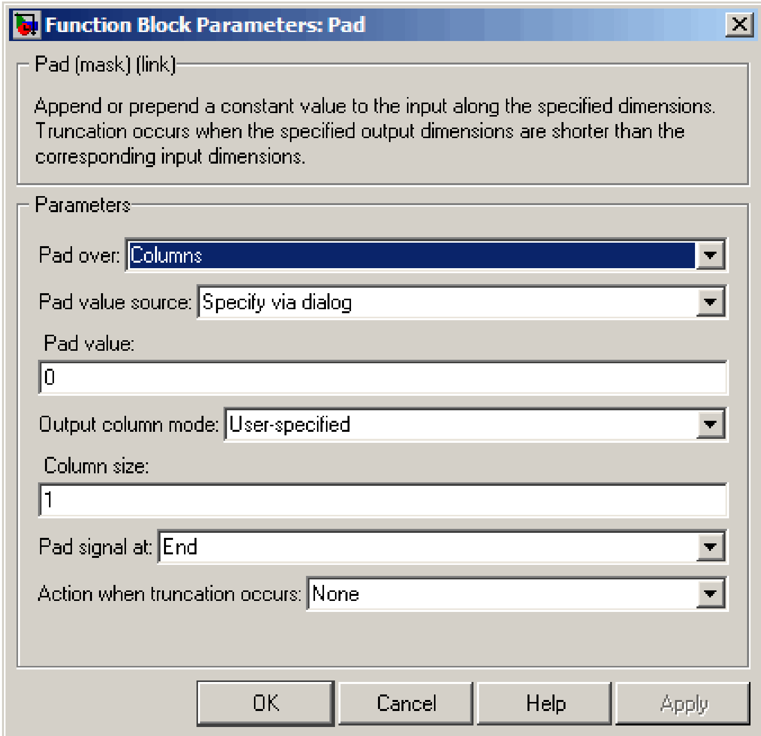
Description

The Pad block extends or crops the dimensions of the input by padding or truncating along its columns, rows, columns and rows, or any dimension(s) you specify. Truncation occurs when you specify output dimensions that are shorter than the corresponding input dimensions. If the input and output lengths are the same, the block is a pass-through.

You can enter the pad value in the block mask or via an input port. You can enter output sizes in the block mask, or have the block pad the specified dimensions until their length is the next highest power of two. The **Pad signal at** parameter controls whether the specified input dimensions are padded or truncated at their beginning, end, or both. For odd pad or truncation lengths, the extra pad value or truncation is applied to the end of the signal. When the block is in `Specified dimensions` mode, you can specify either the output size or the pad size.

You can have the block warn or error when an input signal is truncated using the **Action when truncation occurs** parameter.

Dialog Box



Pad over

Specify the dimensions over which to pad or truncate: `Columns`, `Rows`, `Columns and rows`, `None`, or `Specified dimensions`.

Dimensions to pad

Specify the one-based dimension(s) over which to pad or truncate. The value for this parameter can be a scalar or a vector. For example, specify `1` to pad columns. Specify `[1 2]` to pad columns and rows. Specify `[1 3 5]` to pad the first, third, and fifth dimensions.

This parameter is only visible when `Specified dimensions` is selected for the **Pad over** parameter.

Pad value source

Choose how you specify the pad value. The pad value can come from an input port or from the dialog:

- If you select `Input port`, the `PVal` port appears on the block icon.
- If you select `Specify via dialog`, the **Pad value** parameter appears.

Pad value

Specify the constant scalar value with which to pad the input. Tunable.

This parameter is only visible when `Specify via dialog` is selected for the **Pad value source** parameter.

Output column mode

Choose how you specify the column length of the output:

- If you select `User-specified`, the **Column size** parameter appears.
- If you select `Next power of two`, the block pads the output columns until their length is the next highest power of two. If the column length is already a power of two, the columns are not padded.

This parameter is only visible when `Columns` or `Columns and rows` is selected for the **Pad over** parameter.

Column size

Specify the column length of the output. If the specified column length is longer than the input column length, the columns are padded. If the specified column length is shorter than the input column length, the columns are truncated. This parameter is only visible when `User-specified` is selected for the **Output column mode** parameter.

Output row mode

Choose how you specify the output row length of the output:

- If you select `User-specified`, the **Row size** parameter appears.
- If you select `Next power of two`, the block pads the output rows until their length is the next highest power of two. If the row length is already a power of two, the rows are not padded.

This parameter is only visible when `Rows` or `Columns and rows` is selected for the **Pad over** parameter.

Row size
Specify the row length of the output. If the specified row length is longer than the input row length, the rows are padded. If the specified row length is shorter than the input row length, the rows are truncated. This parameter is only visible when `User-specified` is selected for the **Output row mode** parameter.

Specify
Choose whether you want to control the output length of the specified dimensions by specifying the pad size or the output size.

This parameter is only visible when `Specified dimensions` is selected for the **Pad over** parameter.

Pad size at beginning
Specify how many values to add to the beginning of the input signal along the specified dimension(s). This parameter must be a scalar or a vector with the same number of elements as the **Dimensions to pad** parameter. Each element in the **Pad size at beginning** parameter gives the pad length for the beginning of the corresponding dimension in the **Dimensions to pad** parameter. Values of this parameter must be zero or a positive integer.

This parameter is only visible if `Pad size` is selected for the **Specify** parameter.

Pad size at end

Specify how many values to add to the end of the input signal along the specified dimension(s). This parameter must be a scalar or a vector with the same number of elements as the **Dimensions to pad** parameter. Each element in the **Pad size at end** parameter gives the pad length for the end of the corresponding dimension in the **Dimensions to pad** parameter. Values of this parameter must be zero or a positive integer.

This parameter is only visible if `Pad size` is selected for the **Specify** parameter.

Output size mode

Choose how you specify the output length of the specified dimensions:

- If you select `User-specified`, the **Output size** parameter appears.
- If you select `Next power of two`, the block pads the specified dimensions until their length is the next highest power of two. If the dimension length is already a power of two, no padding occurs in that dimension.

This parameter is only visible if `Output size` is selected for the **Specify** parameter.

Output size

Specify the output length of the specified dimension(s). This parameter must be a scalar or a vector with the same number of elements as the **Dimensions to pad** parameter. Each element in the **Output size** vector gives the output length for the corresponding dimension in the **Dimensions to pad** vector. If the specified length is longer than the input length for a given dimensions, that dimension is padded. If the specified length is shorter than the input length for a given dimension, that dimension is truncated.

This parameter is only visible if `Output size` is selected for the **Specify** parameter.

Pad signal at

Specify whether to pad or truncate the signal at the `Beginning`, `End`, or `Beginning and end` of the specified dimension(s). When you select `Beginning and end`, half the pad length is added to the beginning of the signal, and half is added to the end of the signal. For an odd pad length, the extra value is added to the end of the signal. This also applies to truncation. In this mode, an equal number of values are truncated from the beginning and the end of the signal. In the case of an odd truncation length, the extra value is removed from the end of the signal.

Action when truncation occurs

Choose `None` when you do not want to be notified that the input is truncated. Select `Warning` to display a warning when the input is truncated. Choose `Error` when to display an error and terminate the simulation when the input is truncated.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating-point • Single-precision floating-point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating-point • Single-precision floating-point • Fixed point (signed and unsigned) • Boolean

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Concatenate	Simulink
Repeat	Signal Processing Blockset
Submatrix	Signal Processing Blockset
Upsample	Signal Processing Blockset
Variable Selector	Signal Processing Blockset

Parametric Equalizer

Purpose

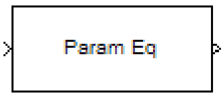
Design parametric equalizer

Library

Filtering / Filter Designs

`dspfdesign`

Description



This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. You must have a Filter Design Toolbox license to design filters with this block. However, you can run models containing this block without a license. This allows you to run a model sent to you by a colleague who has designed a filter using this block, even if you do not have the Filter Design Toolbox product.

Dialog Box

See "Parametric Equalizer Filter Design Dialog Box — Main Pane" in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Peak Finder

Purpose

Determine whether each value of input signal is local minimum or maximum

Library

Signal Operations

`dspsigops`

Description

The Peak Finder block outputs the number of local extrema in the input signal at the Cnt port. Optionally, it can also output the extrema indices, the extrema values, and a binary indicator of whether or not the extrema are maxima or minima.

The Peak Finder block compares the current signal value to the previous and next values to determine if the current value is an extremum. Use the **Peak type(s)** parameter to specify whether you are looking for maxima, minima, or both.

If you select the **Output peak indices** check box, the Idx port appears on the block. The block outputs the zero-based extrema indices at the Idx port. If you select the **Output peak values** check box, the Val port appears on the block. The block outputs the extrema values at the Val port. If you select either of these check boxes and `Maxima and Minima` is selected for the **Peak type(s)**, the Pol port also appears on the block. If the signal value is a maximum, the block outputs a 1 at the Pol ("Polarity") port. If the signal value is a minimum, the block outputs a 0 at the Pol port.

Note that nothing is output at the Idx, Val, and Pol ports for an input signal value that is not an extremum.

Use the **Maximum number of peaks to find** parameter to specify how many extrema to look for in each input signal. The block stops searching the input signal once this maximum number of extrema has been found.

If you select the **Ignore peaks within threshold of neighboring values** check box, the block no longer detects low-amplitude peaks. This feature allows the block to ignore noise within a threshold value that you define. Enter a threshold value for the **Threshold** parameter. Now, the current value is a maximum if (current – previous) > threshold and (current – next) > threshold. The current value is a minimum if (current – previous) < –threshold and (current – next) < –threshold.

This block supports single-channel, multichannel, sample-based, and frame-based inputs. These input signals must be real-valued fixed-point or floating-point scalars or vectors.

Examples

Example 1

Consider the input vector

```
[9 6 10 3 4 5 0 12]
```

The table below shows the analysis made by the Peak Finder block. Note that the first and last input signal values are not considered:

Previous, current, and next values	9 6 10	6 10 3	10 3 4	3 4 5	4 5 0	5 0 12
Current value if it is an extremum	6	10	3	—	5	0
Index of current value if it is an extremum	1	2	3	—	5	6
Polarity of current value if it is an extremum	0	1	0	—	1	0

Therefore, for this example the outputs at the block ports are

Cnt: `5`

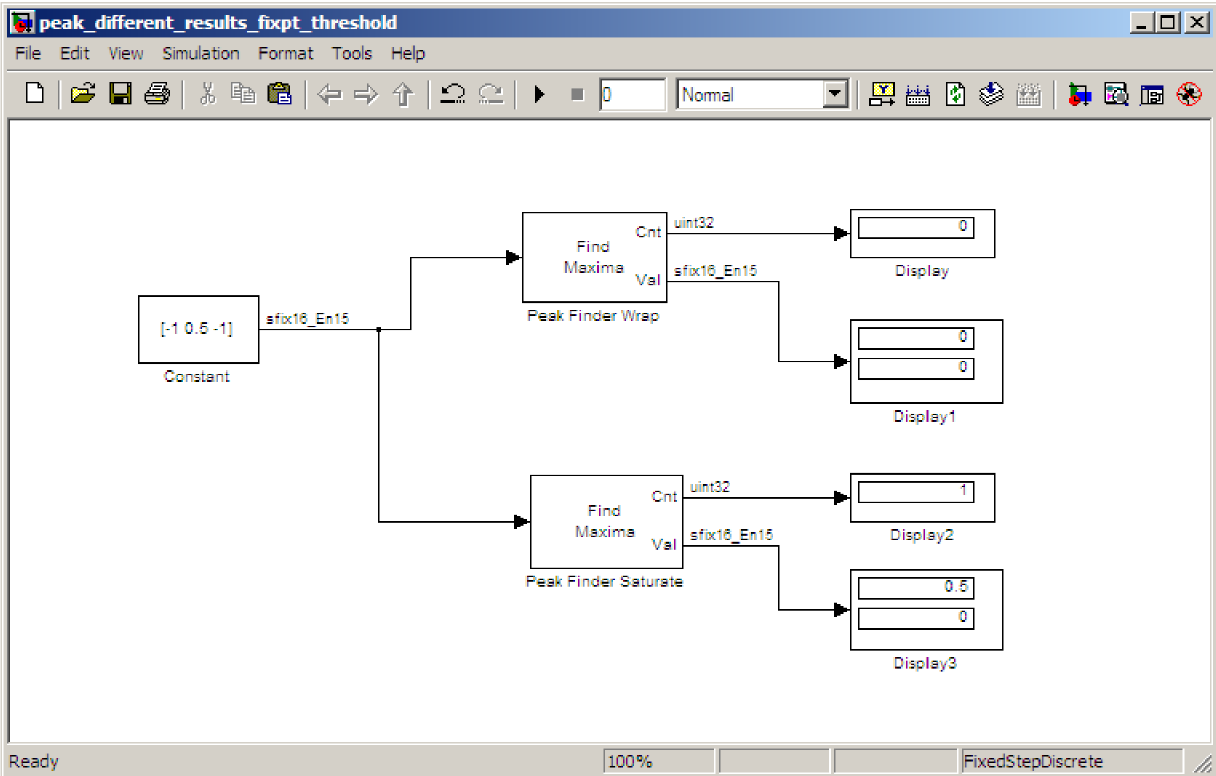
Idx: `[1 2 3 5 6]`

Val: `[6 10 3 5 0]`

Pol: `[0 1 0 1 0]`

Example 2

Note that the **Overflow mode** parameter can affect the output of the block when the input is fixed point. Consider the following model:



In this model, the settings in the Constant block are

- **Constant value** — `[-1 0.5 -1]`
- **Interpret vector parameters as 1–D** — not selected
- **Sampling mode** — `Sample based`
- **Sample time** — `1`
- **Output data type** — `<data type expression>`
- **Mode** — `Fixed point`
- **Sign** — `Signed`

- **Scaling** — `Binary point`
- **Word length** — `16`
- **Fraction length** — `15`

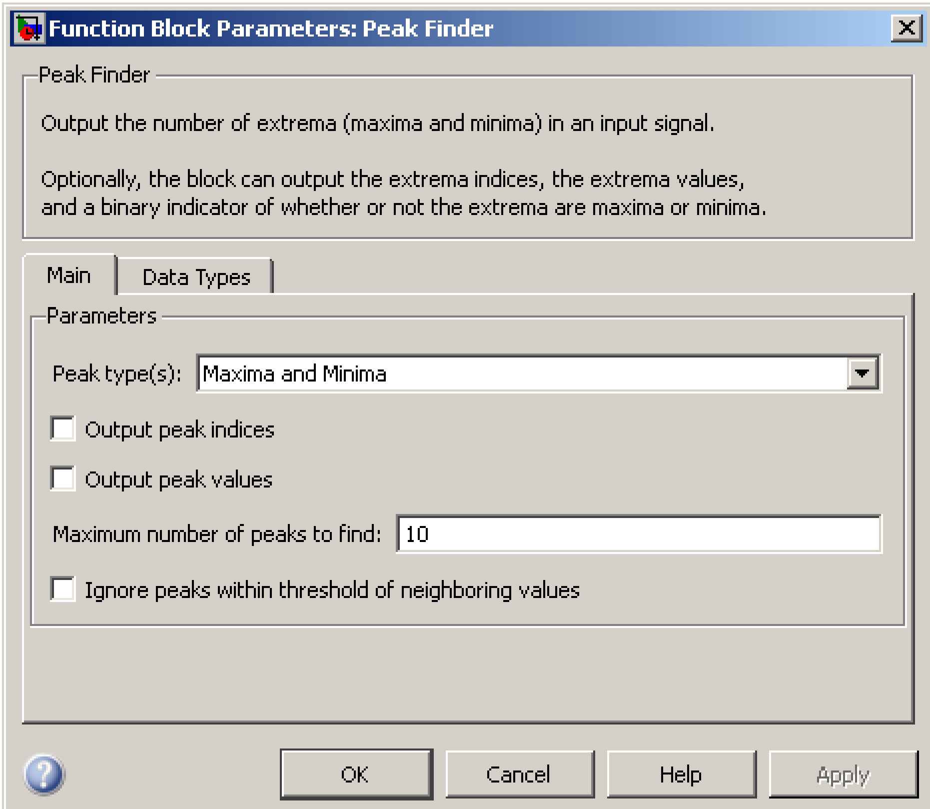
The settings in the Peak Finder blocks are

- **Peak type(s)** — `Maxima`
- **Output peak indices** — not selected
- **Output peak values** — selected
- **Maximum number of peaks to find** — `2`
- **Ignore peaks within threshold of neighboring values** — selected
- **Threshold** — `0.25`
- **Rounding mode** — `Floor`
- **Overflow mode** — `Wrap` for Peak Finder Wrap, `Saturate` for Peak Finder Saturate

Setting the **Overflow mode** parameter of the Peak Finder Wrap block to `Wrap` causes the calculations (current – previous) > threshold and (current – next) > threshold to wrap on overflow, thereby causing the maximum to be missed.

Dialog Box

The **Main** pane of the Peak Finder block dialog appears as follows.



Peak type(s)

Specify whether you are looking for maxima, minima, or both.

Output peak indices

Select this check box if you want the block to output the extrema indices at the Idx port.

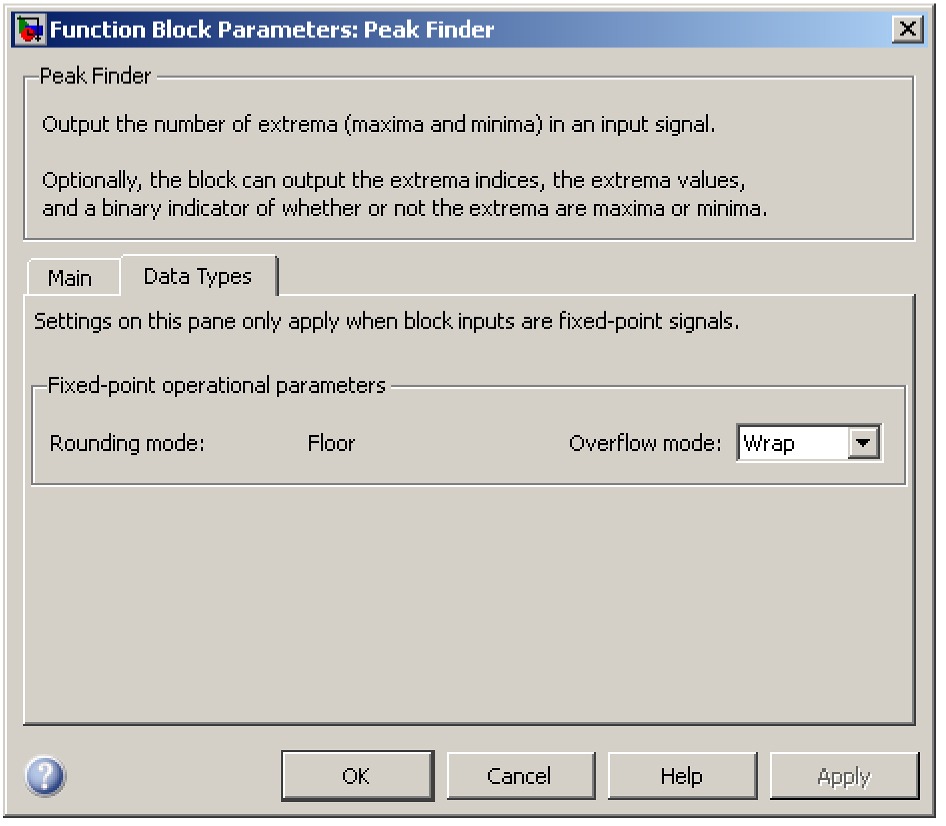
Output peak values
Select this check box if you want the block to output the extrema values at the Val port.

Maximum number of peaks to find
Enter the number of extrema to look for in each input signal. The block stops searching the input signal for extrema once the maximum number of extrema has been found. The value of this parameter must be an integer greater than or equal to one.

Ignore peaks within threshold of neighboring values
Select this check box if you want to eliminate the detection of peaks whose amplitudes are within a specified threshold of neighboring values.

Threshold
Enter your threshold value. This parameter appears if you select the **Ignore peaks within threshold of neighboring values** check box.

The **Data Types** pane of the Peak Finder block dialog appears as follows.



Rounding mode

The rounding mode of this block is always `Floor`.

Overflow mode

Select the overflow mode to be used when block inputs are fixed point.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Cnt	• 32-bit unsigned integers
Idx	• 32-bit unsigned integers
Val	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Pol	• Boolean

See Also

Maximum	Signal Processing Blockset
Minimum	Signal Processing Blockset

Peak-Notch Filter

Purpose

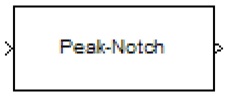
Design peak or notch filter

Library

Filtering / Filter Designs

`dspfdesign`

Description



This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. You must have a Filter Design Toolbox license to design filters with this block. However, you can run models containing this block without a license. This allows you to run a model sent to you by a colleague who has designed a filter using this block, even if you do not have the Filter Design Toolbox product.

Dialog Box

See "Peak/Notch Filter Design Dialog Box — Main Pane" in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Periodogram

Purpose

Power spectral density or mean-square spectrum estimate using periodogram method

Library

Estimation / Power Spectrum Estimation

`dspspect3`

Description

The Periodogram block estimates the power spectral density (PSD) or mean-square spectrum (MSS) of the input. It does so by using the periodogram method and Welch's averaged, modified periodogram method. When you set the **Number of spectral averages** parameter to 1, the block computes the periodogram of the input. When the **Number of spectral averages** is greater than 1, the block uses the Welch method to compute a modified periodogram of the input. The block averages the squared magnitude of the FFT computed over windowed sections of the input. It then normalizes the spectral average by the square of the sum of the window samples. See "Periodogram" and "Welch's Method" in the Signal Processing Toolbox documentation for more information.

The block treats M-by-N frame-based matrix input and M-by-N sample-based matrix input as M sequential time samples from N independent channels. The block computes a separate estimate for each of the N independent channels and generates an N_{fft}-by-N matrix output. When you select the **Inherit FFT length from input dimensions** check box, N_{fft} is the frame size of the input, which must be a power of 2. When you clear the **Inherit FFT length from input dimensions** check box, you can use the **FFT length** parameter to specify N_{fft} as a power of 2. The block either zero-pads or wraps the input to N_{fft} before computing the FFT.

Each column of the output matrix contains the estimate of the power spectral density of the corresponding input column at N_{fft} equally spaced frequency points. The frequency points are in the range $[0,F_s)$, where F_s is the sampling frequency of the signal. The output is always sample based.

When you select the **Inherit sample time from input** check box, the block computes the frequency data from the sample period of the input signal. For the block to produce valid output, the following conditions must hold:

- The input to the block is the original signal, with no samples added or deleted (by insertion of zeros, for example).
- The sample period of the time-domain signal in the simulation equals the sample period of the original time series.

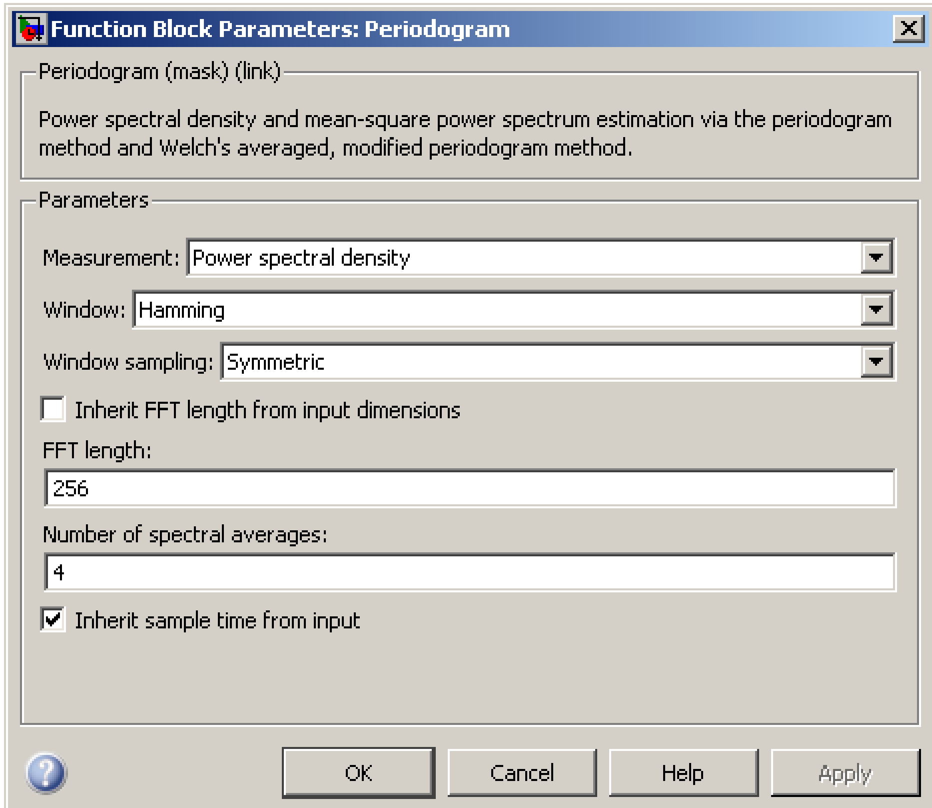
If these conditions do not hold, clear the **Inherit sample time from input** check box. You can then specify a sample time using the **Sample time of original time series** parameter.

The **Window**, **Stopband ripple**, **Beta**, and **Window sampling** parameters all apply to the specification of the window function. See the Window Function block reference page for more details on these four parameters.

Example

The `dspstfft` demo provides an illustration of using the Periodogram and Matrix Viewer blocks to create a spectrogram. The `dspsacomp` demo compares the Periodogram block with several other spectral estimation methods.

Dialog Box



Measurement

Specify the type of measurement for the block to perform: `Power spectral density` or `Mean-square spectrum`. Tunable.

Window

Select the type of window to apply. See the Window Function block reference page for more details. Tunable.

Stopband attenuation in dB

Enter the level, in decibels (dB), of stopband attenuation, R_s, for the Chebyshev window. This parameter becomes visible if, for the **Window** parameter, you choose `Chebyshev`. Tunable.

Beta

Enter the β parameter for the Kaiser window. This parameter becomes visible if, for the **Window** parameter, you chose `Kaiser`. Increasing **Beta** widens the mainlobe and decreases the amplitude of the sidelobes in the displayed frequency magnitude response. Tunable.

Window sampling

From the list, choose `Symmetric` or `Periodic`. Tunable.

Inherit FFT length from input dimensions

When you select this check box, the block uses the input frame size as the number of data points, N_{fft}, on which to perform the FFT. To specify the number of points on which to perform the FFT, clear the **Inherit FFT length from estimation order** check box. You can then specify a power of two FFT length using the **FFT length** parameter.

FFT length

Enter the number of data points on which to perform the FFT, N_{fft}. When N_{fft} is larger than the input frame size, the block zero-pads each frame as needed. When N_{fft} is smaller than the input frame size, the block wraps each frame as needed. This parameter becomes visible only when you clear the **Inherit FFT length from input dimensions** check box.

Number of spectral averages

Enter the number of spectra to average via "Welch's Method"; setting this parameter to `1` disables averaging.

Inherit sample time from input

If you select the **Inherit sample time from input** check box, the block computes the frequency data from the sample period of the input signal. For the block to produce valid output, the following conditions must hold:

- The input to the block is the original signal, with no samples added or deleted (by insertion of zeros, for example).
- The sample period of the time-domain signal in the simulation equals the sample period of the original time series.

If these conditions do not hold, clear the **Inherit sample time from input** check box. You can then specify a sample time using the **Sample time of original time series** parameter.

Sample time of original time series
Specify the sample time of the original time-domain signal. This parameter becomes visible only when you clear the **Inherit sample time from input** check box.

References

Oppenheim, A. V. and R. W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

Orfanidis, S. J. *Introduction to Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

Proakis, J. and D. Manolakis. *Digital Signal Processing*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point
Output	• Double-precision floating point • Single-precision floating point

See Also

Burg Method	Signal Processing Blockset
Inverse Short-Time FFT	Signal Processing Blockset
Magnitude FFT	Signal Processing Blockset

Short-Time FFT	Signal Processing Blockset
Spectrum Scope	Signal Processing Blockset
Window Function	Signal Processing Blockset
Yule-Walker Method	Signal Processing Blockset
`spectrum.periodogram`	Signal Processing Toolbox
`spectrum.welch`	Signal Processing Toolbox

See "Power Spectrum Estimation" for related information.

Permute Matrix

Purpose

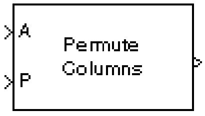
Reorder matrix rows or columns

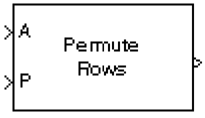
Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description





The Permute Matrix block reorders the rows or columns of M-by-N input matrix `A` as specified by indexing input `P`.

When the **Permute** parameter is set to `Rows`, the block uses the rows of `A` to create a new matrix with the same column dimension. Input `P` is a length-L vector whose elements determine where each row from `A` should be placed in the L-by-N output matrix.

```
% Equivalent MATLAB code
y = [A(P(1),:) ; A(P(2),:) ; A(P(3),:) ; ... ; A(P(end),:)]
```

For row permutation, a length-M 1-D vector input at the `A` port is treated as a M-by-1 matrix.

When the **Permute** parameter is set to `Columns`, the block uses the columns of `A` to create a new matrix with the same row dimension. Input `P` is a length-L vector whose elements determine where each column from `A` should be placed in the M-by-L output matrix.

```
% Equivalent MATLAB code
y = [A(:,P(1)) A(:,P(2)) A(:,P(3)) ... A(:,P(end))]
```

For column permutation, a length-N 1-D vector input at the `A` port is treated as a 1-by-N matrix.

When an index value in input `P` references a nonexistent row or column of matrix `A`, the block reacts with the behavior specified by the **Invalid permutation index** parameter. The following options are available:

- `Clip index` — Clip the index to the nearest valid value (1 or M for row permutation, and 1 or N for column permutation), and *do not* issue an alert. Example: For a 3-by-7 input matrix, a column index of 9 is clipped to 7, and a row index of -2 is clipped to 1.

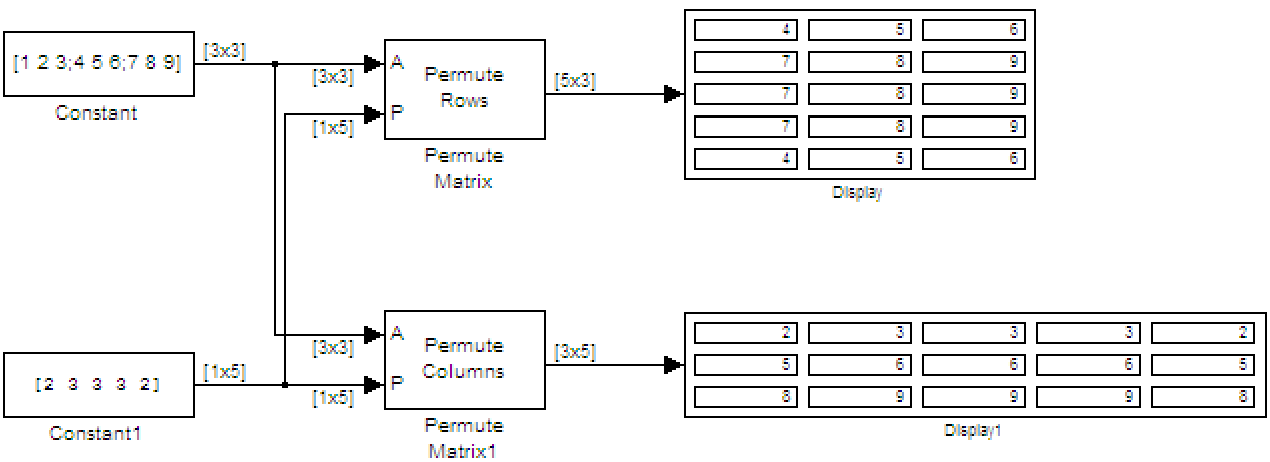
- `Clip and warn` — Display a warning message in the MATLAB command window, and clip the index as described above.
- `Generate error` — Display an error dialog box and terminate the simulation.

When length of the permutation vector `P` is not equal to the number of rows or columns of the input matrix `A`, you can choose to get an error dialog box and terminate the simulation by selecting **Error when length of P is not equal to Permute dimension size**.

When input `A` is frame based, the output is frame based; otherwise, the output is sample based.

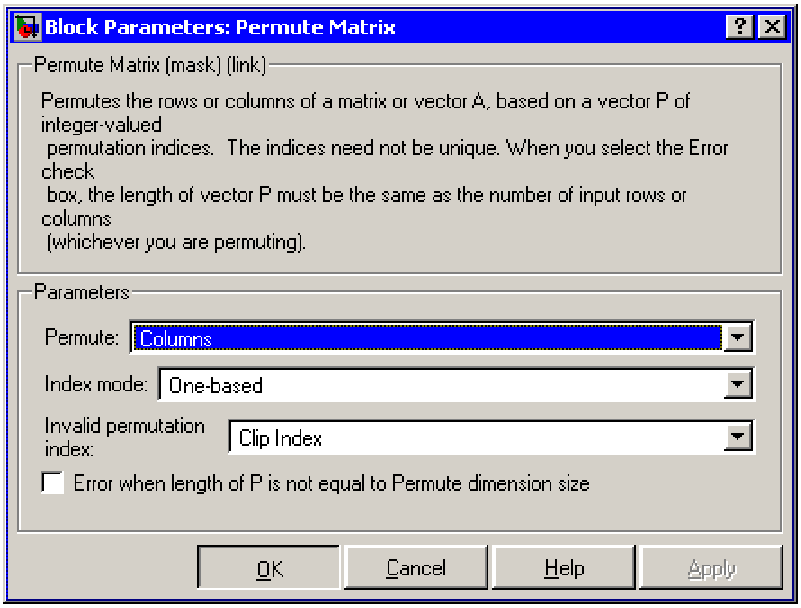
Examples

In the model below, the top Permute Matrix block places the second row of the input matrix in both the first and fifth rows of the output matrix, and places the third row of the input matrix in the three middle rows of the output matrix. The bottom Permute Matrix block places the second column of the input matrix in both the first and fifth columns of the output matrix, and places the third column of the input matrix in the three middle columns of the output matrix.



As shown in the example above, rows and columns of `A` can appear any number of times in the output, or not at all.

Dialog Box



Permute

Method of constructing the output matrix; by permuting rows or columns of the input.

Index mode

When set to `One-based`, a value of `1` in the permutation vector `P` refers to the first row or column of the input matrix `A`. When set to `Zero-based`, a value of `0` in `P` refers to the first row or column of `A`.

Invalid permutation index

Response to an invalid index value. Tunable.

Error when length of P is not equal to Permute dimension size

Option to display an error dialog box and terminate the simulation when the length of the permutation vector `P` is not equal to the number of rows or columns of the input matrix `A`.

Supported Data Types

Port	Supported Data Types
A	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated
P	• Double-precision floating point • Single-precision floating point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated

See Also

Submatrix	Signal Processing Blockset
Transpose	Signal Processing Blockset
Variable Selector	Signal Processing Blockset
`permute`	MATLAB

See "Reordering Channels in Multichannel Frame-Based Signals" for related information.

Polynomial Evaluation

Purpose

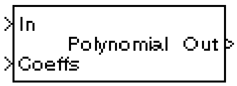
Evaluate polynomial expression

Library

Math Functions / Polynomial Functions

`dsppolyfun`

Description



The Polynomial Evaluation block applies a polynomial function to the real or complex input at the `In` port.

```
y = polyval(u)      % Equivalent MATLAB code
```

The Polynomial Evaluation block performs these types of operation more efficiently than the equivalent construction using Simulink Sum and Math Function blocks.

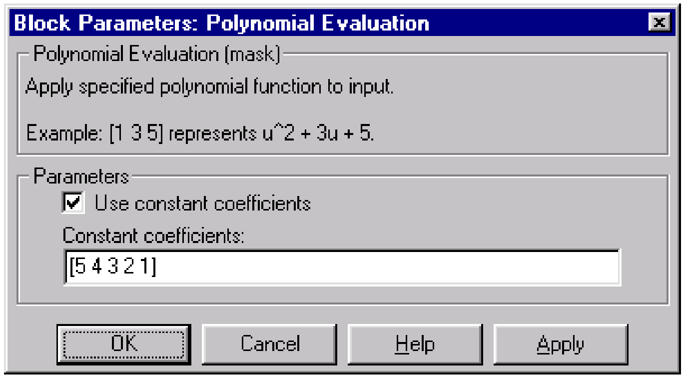
When you select the **Use constant coefficients** check box, you specify the polynomial expression in the **Constant coefficients** parameter. When you do not select **Use constant coefficients**, a variable polynomial expression is specified by the input to the `Coeffs` port. In both cases, the polynomial is specified as a vector of real or complex coefficients in order of descending exponents.

The table below shows some examples of the block's operation for various coefficient vectors.

Coefficient Vector	Equivalent Polynomial Expression
`[1 2 3 4 5]`	$y = u^4 + 2u^3 + 3u^2 + 4u + 5$
`[1 0 3 0 5]`	$y = u^4 + 3u^2 + 5$
`[1 2+i 3 4-3i 5i]`	$y = u^4 + (2+i)u^3 + 3u^2 + (4-3i)u + 5i$

Each element of a vector or matrix input to the `In` port is processed independently, and the output size and frame status are the same as the input.

Dialog Box



Use constant coefficients

Select to enable the **Constant coefficients** parameter and disable the `Coeffs` input port.

Constant coefficients

Specify the vector of polynomial coefficients to apply to the input, in order of descending exponents. This parameter is enabled when you select the **Use constant coefficients** check box.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Least Squares Polynomial Fit	Signal Processing Blockset
Math Function	Simulink
Sum	Simulink
`polyval`	MATLAB

Purpose

Use Schur-Cohn algorithm to determine whether all roots of input polynomial are inside unit circle

Library

Math Functions / Polynomial Functions

`dsppolyfun`

Description

The Polynomial Stability Test block uses the Schur-Cohn algorithm to determine whether all roots of a polynomial are within the unit circle.

```
y = all(abs(roots(u)) < 1)       % Equivalent MATLAB code
```

Each column of the M-by-N input matrix *u* contains M coefficients from a distinct polynomial,

$$f(x) = u_1 x^{M-1} + u_2 x^{M-2} + \ldots + u_M$$

arranged in order of descending exponents, u_1, u_2, ..., u_M. The polynomial has order M-1 and positive integer exponents.

Inputs can be frame based or sample based, and both represent the polynomial coefficients as shown above. For convenience, a length-M 1-D vector input is treated as an M-by-1 matrix.

The output is a 1-by-N matrix with each column containing the value `1` or `0`. The value `1` indicates that the polynomial in the corresponding column of the input is stable; that is, the magnitudes of all solutions to $f(x) = 0$ are less than 1. The value `0` indicates that the polynomial in the corresponding column of the input might be unstable; that is, the magnitude of at least one solution to $f(x) = 0$ is greater than or equal to 1.

The output is always sample based.

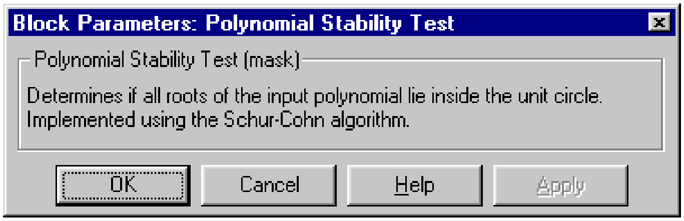
Applications

This block is most commonly used to check the pole locations of the denominator polynomial, $A(z)$, of a transfer function, $H(z)$.

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2 z^{-1} + \ldots + b_m z^{-(m-1)}}{a_1 + a_2 z^{-1} + \ldots + a_n z^{-(n-1)}}$$

The poles are the n-1 roots of the denominator polynomial, $A(z)$. When any poles are located outside the unit circle, the transfer function $H(z)$ is unstable. As is typical in DSP applications, the transfer function above is specified in descending powers of z^{-1} rather than z.

Dialog Box



Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Boolean — Block outputs are always Boolean.

See Also

Least Squares Polynomial Fit	Signal Processing Blockset
Polynomial Evaluation	Signal Processing Blockset
`polyfit`	MATLAB

Pseudoinverse

Purpose

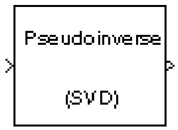
Compute Moore-Penrose pseudoinverse of matrix

Library

Math Functions / Matrices and Linear Algebra / Matrix Inverses

`dspinverses`

Description



The Pseudoinverse block computes the Moore-Penrose pseudoinverse of input matrix A.

```
[U,S,V] = svd(A,0)       % Equivalent MATLAB code
```

The pseudoinverse of A is the matrix $A^{\dagger}$ such that

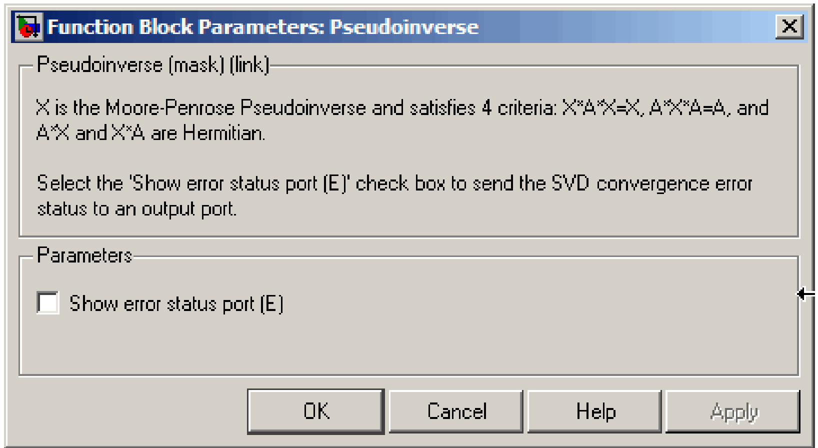
$$A^{\dagger} = VS^{\dagger}U^{*}$$

where U and V are orthogonal matrices, and S is a diagonal matrix. The pseudoinverse has the following properties:

- $AA^{\dagger} = (AA^{\dagger})^{*}$
- $A^{\dagger}A = (A^{\dagger}A)^{*}$
- $AA^{\dagger}A = A$
- $A^{\dagger}AA^{\dagger} = A^{\dagger}$

The output is always sample based.

Dialog Box



Show error status port

Select to enable the E output port, which reports a failure to converge. The possible values you can receive on the port are:

- 0 — The pseudoinverse calculation converges.
- 1 — The pseudoinverse calculation does not converge.

If the pseudoinverse calculation fails to converge, the output at port X is an undefined matrix of the correct size.

References

Golub, G. H., and C. F. Van Loan. *Matrix Computations*. 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

Supported Data Types

Port	Supported Data Types
A	• Double-precision floating point • Single-precision floating point
X	• Double-precision floating point • Single-precision floating point
E	• Boolean

See Also

Cholesky Inverse	Signal Processing Blockset
LDL Inverse	Signal Processing Blockset
LU Inverse	Signal Processing Blockset
Singular Value Decomposition	Signal Processing Blockset
`inv`	MATLAB

See “Matrix Inverses” for related information.

Pulse Shaping Filter

Purpose

Design pulse shaping filter

Library

Filtering / Filter Designs

`dspfdesign`

Description

This block brings the filter design capabilities of the filterbuilder function to the Simulink environment. Without a Filter Design Toolbox license, you can run models that contain this block, and can edit some, but not all, block parameters. To enable the full filter design functionality of this block, you must have a Filter Design Toolbox license.

Dialog Box

See “Pulse-shaping Filter Design Dialog Box—Main Pane” in the Signal Processing Toolbox documentation for more information about the parameters of this block. The **Data Types** and **Code Generation** panes are not available for blocks in the Signal Processing Blockset Filter Designs library.

Parameters of this block that do not change filter order or structure are tunable. Tunable parameters are enabled during simulation; nontunable parameters are not.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point

Port	Supported Data Types
	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

QR Factorization

Purpose Factor arbitrary matrix into unitary and upper triangular components

Library Math Functions / Matrices and Linear Algebra / Matrix Factorizations

`dspfactors`

Description The QR Factorization block uses a sequence of Householder transformations to triangularize the input matrix A. The block factors a column permutation of the M-by-N input matrix A as

$$A_e = QR$$

The column-pivoted matrix A_e contains the columns of A permuted as indicated by the contents of length-N permutation vector E.

```
Ae = A(:,E)     % Equivalent MATLAB code
```

The block selects a column permutation vector E, which ensures that the diagonal elements of matrix R are arranged in order of decreasing magnitude.

$$\left|r_{i+1,j+1}\right| < \left|r_{i,j}\right| \qquad i = j$$

The size of matrices Q and R depends on the setting of the **Output size** parameter:

- When you select `Economy` for the output size, Q is an M-by-min(M,N) unitary matrix, and R is a min(M,N)-by-N upper-triangular matrix.

  ```
  [Q R E] = qr(A,0)   % Equivalent MATLAB code
  ```

- When you select `Full` for the output size, Q is an M-by-M unitary matrix, and R is a M-by-N upper-triangular matrix.

  ```
  [Q R E] = qr(A)      % Equivalent MATLAB code
  ```

A length-M 1–D vector input is always treated as an M-by-1 matrix.

QR factorization is an important tool for solving linear systems of equations because of good error propagation properties and the invertability of unitary matrices:

$$Q^{-1} = Q'$$

where Q' is the complex conjugate transpose of Q.

Unlike LU and Cholesky factorizations, the matrix A does not need to be square for QR factorization. However, QR factorization requires twice as many operations as LU Factorization (Gaussian elimination).

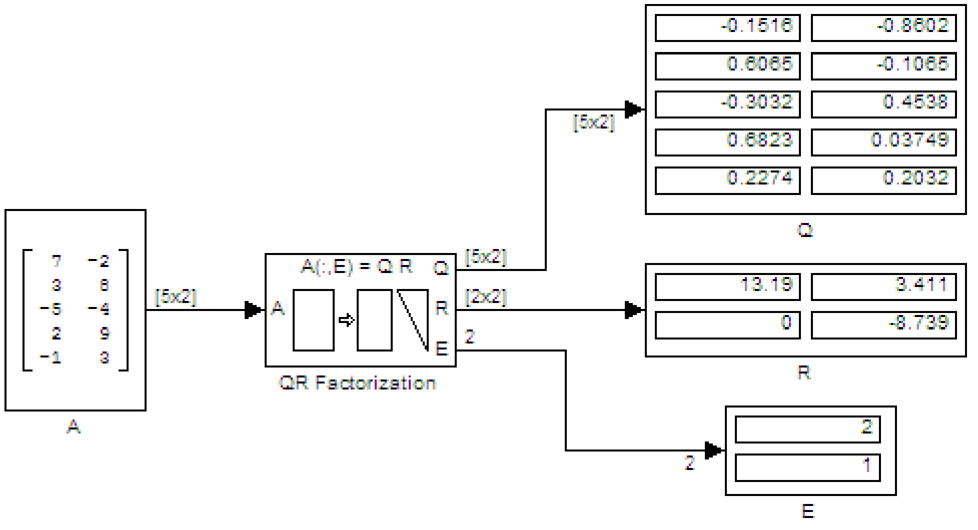
Examples

The **Output size** parameter of the QR factorization block has two settings: `Economy` and `Full`. When the M-by-N input matrix A has dimensions such that $M > N$, the dimensions of output matrices Q and R differ depending on the setting of the **Output size** parameter. If, however, the size of the input matrix A is such that $M \leq N$, output matrices Q and R have the same dimensions, regardless of whether the **Output size** is set to `Economy` or `Full`.

The input to the QR Factorization block in the following model is a `5`-by-`2` matrix A. When you change the setting of the **Output size** parameter from `Economy` to `Full`, the dimensions of the output given by the QR Factorization block also change.

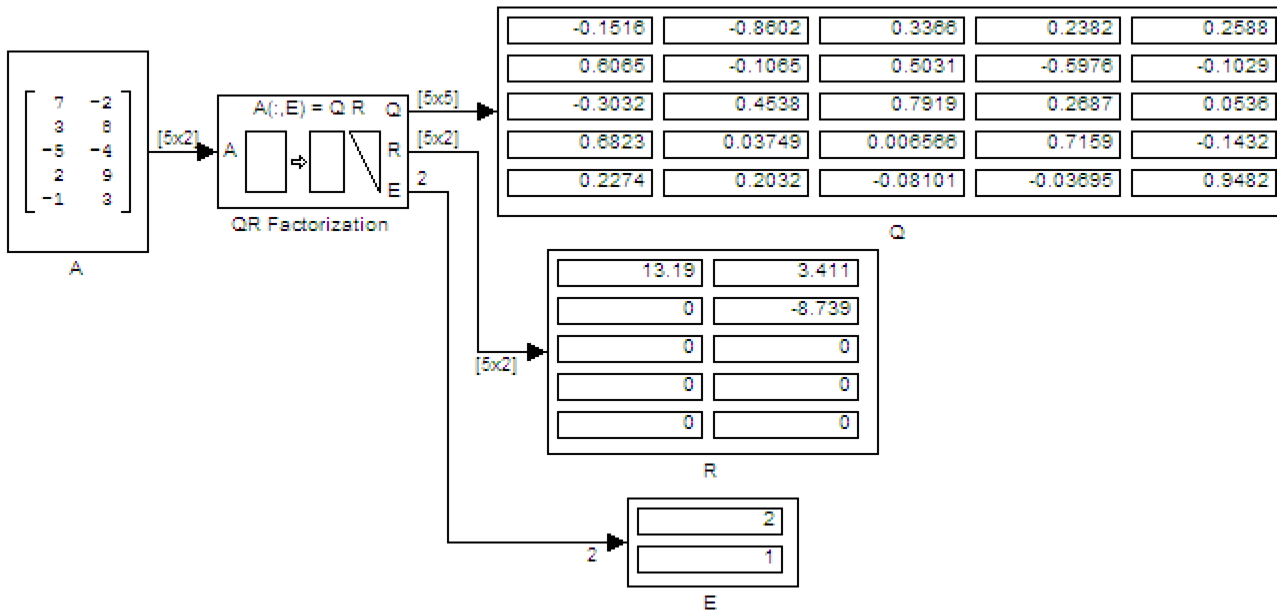
1. Open the model by typing `doc_qrfactorization_ref` at the MATLAB command line.

2. Double-click the QR Factorization block, set the **Output size** parameter to `Economy`, and run the model.



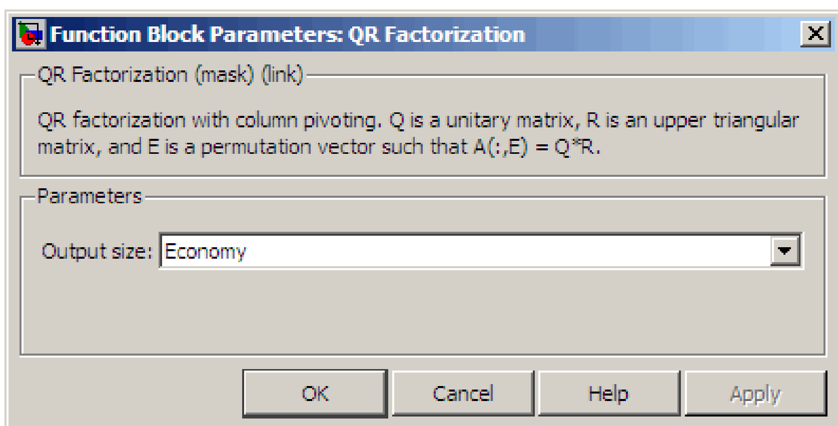
The QR Factorization block outputs a 5-by-2 matrix Q and a 2-by-2 matrix R.

3 Change the **Output size** parameter of the QR Factorization block to `Full` and rerun the model.



The QR Factorization block outputs a 5-by-5 matrix Q and a 5-by-2 matrix R.

Dialog Box



Output size

Specify the size of output matrices Q and R:

- `Economy` — When this output size is selected, the block outputs an M-by-min(M,N) unitary matrix Q and a min(M,N)-by-N upper-triangular matrix R.
- `Full` — When this output size is selected, the block outputs an M-by-M unitary matrix Q and a M-by-N upper-triangular matrix R.

References

Golub, G. H., and C. F. Van Loan. *Matrix Computations*. 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point
Output	• Double-precision floating point • Single-precision floating point

See Also

Cholesky Factorization	Signal Processing Blockset
LU Factorization	Signal Processing Blockset
QR Solver	Signal Processing Blockset
Singular Value Decomposition	Signal Processing Blockset
`qr`	MATLAB

See “Matrix Factorizations” for related information.

Purpose

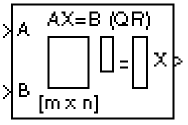
Find minimum-norm-residual solution to A*X*=B

Library

Math Functions / Matrices and Linear Algebra / Linear System Solvers

`dspsolvers`

Description



The QR Solver block solves the linear system A*X*=B, which can be overdetermined, underdetermined, or exactly determined. The system is solved by applying QR factorization to the M-by-N matrix, A, at the `A` port. The input to the `B` port is the right side M-by-L matrix, B. A length-M 1-D vector input at either port is treated as an M-by-1 matrix.

The output at the `x` port is the N-by-L matrix, X. X is always sample based, and is chosen to minimize the sum of the squares of the elements of B-AX. When B is a vector, this solution minimizes the vector 2-norm of the residual (B-AX is the residual). When B is a matrix, this solution minimizes the matrix Frobenius norm of the residual. In this case, the columns of X are the solutions to the L corresponding systems $AX_k=B_k$, where B_k is the kth column of B, and X_k is the kth column of *X*.

X is known as the minimum-norm-residual solution to AX=B. The minimum-norm-residual solution is unique for overdetermined and exactly determined linear systems, but it is not unique for underdetermined linear systems. Thus when the QR Solver is applied to an underdetermined system, the output *X* is chosen such that the number of nonzero entries in X is minimized.

Algorithm

QR factorization factors a column-permuted variant (A_e) of the M-by-N input matrix A as

$$A_e = QR$$

where Q is a M-by-min(M,N) unitary matrix, and R is a min(M,N)-by-N upper-triangular matrix.

The factored matrix is substituted for A_e in

$$A_e X = B_e$$

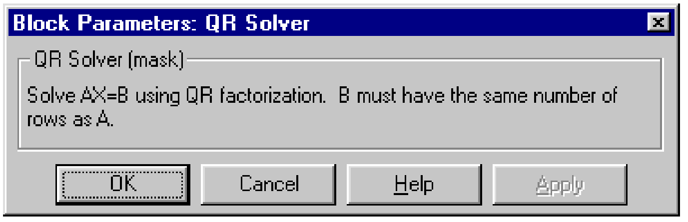
and

$$QRX = B_e$$

is solved for X by noting that $Q^{-1} = Q^*$ and substituting $Y = Q^* B_e$. This requires computing a matrix multiplication for Y and solving a triangular system for X.

$$RX = Y$$

Dialog Box



Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Levinson-Durbin	Signal Processing Blockset
LDL Solver	Signal Processing Blockset
LU Solver	Signal Processing Blockset
QR Factorization	Signal Processing Blockset
SVD Solver	Signal Processing Blockset

See “Linear System Solvers” for related information.

Purpose

Discretize input at specified interval

Library

Quantizers

`dspquant2`

Description

The Quantizer block is an implementation of the Simulink Quantizer block. See Quantizer for more information.

Queue

Purpose

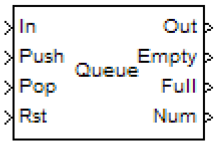
Store inputs in FIFO register

Library

Signal Management / Buffers

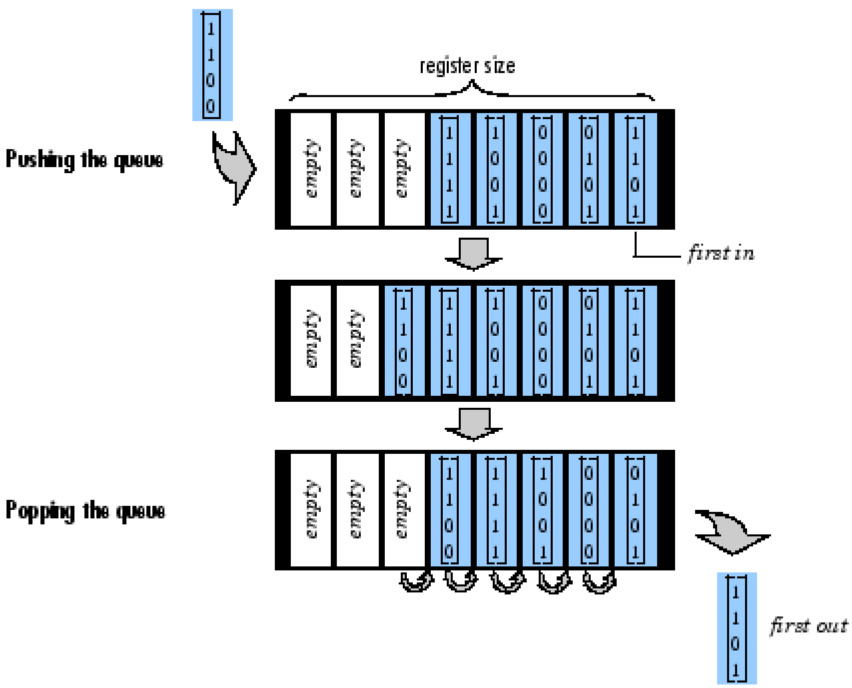
`dspbuff3`

Description



The Queue block stores a sequence of input samples in a first in, first out (FIFO) register. The register capacity is set by the **Register size** parameter, and inputs can be scalars, vectors, or matrices.

The block *pushes* the input at the `In` port onto the end of the queue when a trigger event is received at the `Push` port. When a trigger event is received at the `Pop` port, the block *pops* the first element off the queue and holds the `Out` port at that value. The first input to be pushed onto the queue is always the first to be popped off.



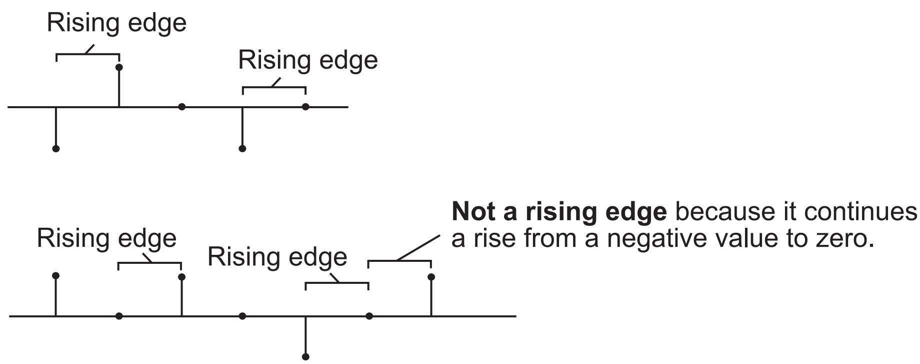
A trigger event at the optional `Rst` port empties the queue contents. When you select **Clear output port on reset**, then a trigger event at the `Rst` port empties the queue *and* sets the value at the `Out` port to zero. This setting also applies when a disabled subsystem containing the Queue block is reenabled; the `Out` port value is only reset to zero in this case when you select **Clear output port on reset**.

When two or more of the control input ports are triggered at the same time step, the operations are executed in the following order:

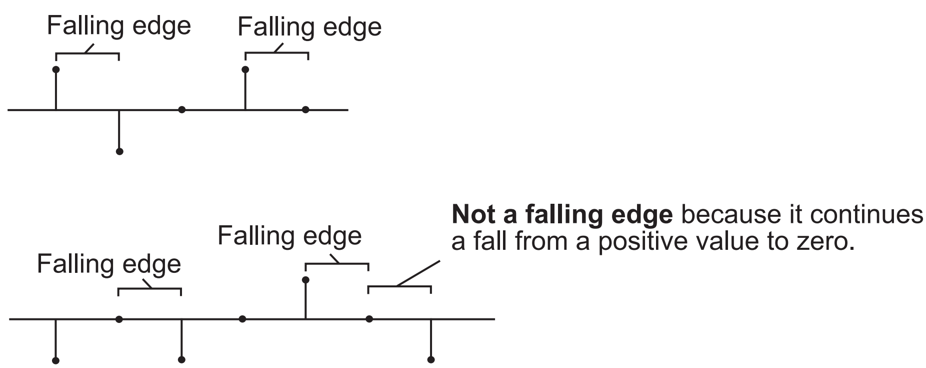
1. `Rst`
2. `Push`
3. `Pop`

The rate of the trigger signal must be the same as the rate of the data signal input. You specify the triggering event for the `Push`, `Pop`, `and` `Rst` ports by the **Trigger type** pop-up menu:

- `Rising edge` — Triggers execution of the block when the trigger input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero; see the following figure



- `Falling edge` — Triggers execution of the block when the trigger input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero; see the following figure



- `Either edge` — Triggers execution of the block when the trigger input is a `Rising edge` or `Falling edge` (as described above).
- `Non-zero sample` — Triggers execution of the block at each sample time that the trigger input is not zero.

Note When running simulations in the Simulink `MultiTasking` mode, sample-based trigger signals have a one-sample latency, and frame-based trigger signals have one frame of latency. Thus, there is a one-sample or one-frame delay between the time the block detects a trigger event, and when it applies the trigger. For more information on latency and the Simulink tasking modes, see “Excess Algorithmic Delay (Tasking Latency)” and “Scheduling Considerations” in the *Real-Time Workshop User’s Guide*.

Note If your model contains any referenced models that use the Queue block, you cannot simulate the top-level model in Simulink Accelerator mode.

The **Push onto full register** parameter specifies the block’s behavior when a trigger is received at the `Push` port but the register is full. The **Pop empty register** parameter specifies the block’s behavior when a trigger is received at the `Pop` port but the register is empty. The following options are available for both cases:

- `Ignore` — Ignore the trigger event, and continue the simulation.
- `Warning` — Ignore the trigger event, but display a warning message in the MATLAB Command Window.
- `Error` — Display an error dialog box and terminate the simulation.

Note The **Push onto full register** and **Pop empty register** parameters are diagnostic parameters. Like all diagnostic parameters on the Configuration Parameters dialog box, they are set to `Ignore` in the code generated for this block by Real-Time Workshop code generation software.

The **Push onto full register** parameter additionally offers the **Dynamic reallocation** option, which dynamically resizes the register to accept as many additional inputs as memory permits. To find out how many elements are on the queue at a given time, enable the `Num` output port by selecting the **Show number of register entries port** parameter.

Note When `Dynamic reallocation` is selected, the **System target file** parameter on the **Real-Time Workshop** pane of the Configuration Parameters dialog box must be set to `grt_malloc.tlc` `Generic Real-Time Target with dynamic memory allocation`.

Examples

Example 1

The table below illustrates the Queue block's operation for a **Register size** of `4`, **Trigger type** of `Either edge`, and **Clear output port on reset** enabled. Because the block triggers on both rising and falling edges in this example, each transition from `1` to `0` or `0` to `1` in the `Push`, `Pop`, and `Rst` columns below represents a distinct trigger event. A `1` in the `Empty` column indicates an empty queue, while a `1` in the `Full` column indicates a full queue.

In	Push	Pop	Rst	Queue	Out	Empty	Full	Num
1	0	0	0		0	1	0	0
2	1	0	0		0	0	0	1
3	0	0	0		0	0	0	2
4	1	0	0		0	0	0	3
5	0	0	0		0	0	1	4

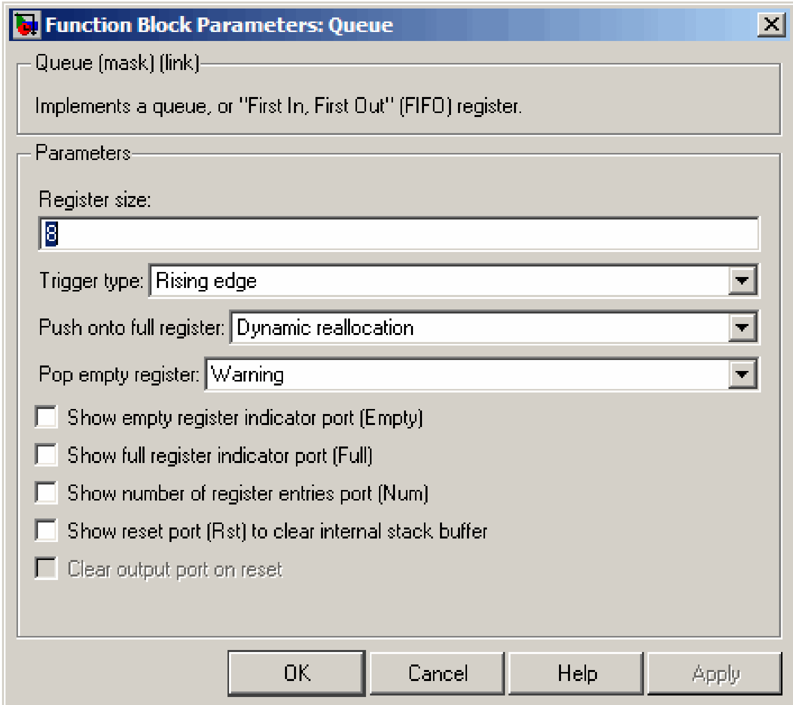
In	Push	Pop	Rst	Queue	Out	Empty	Full	Num
6	0	1	0	top [ ] [5] [4] [3] bottom	2	0	0	3
7	0	0	0	top [ ] [ ] [5] [4] bottom	3	0	0	2
8	0	1	0	top [ ] [ ] [ ] [5] bottom	4	0	0	1
9	0	0	0	top [ ] [ ] [ ] [ ] bottom	5	1	0	0
10	1	0	0	top [ ] [ ] [ ] [10] bottom	5	0	0	1
11	0	0	0	top [ ] [ ] [11] [10] bottom	5	0	0	2
12	1	0	1	top [ ] [ ] [ ] [12] bottom	0	0	0	1

Note that at the last step shown, the `Push` and `Rst` ports are triggered simultaneously. The `Rst` trigger takes precedence, and the queue is first cleared and then pushed.

Example 2

The `dspqdemo` demo provides another example of the operation of the Queue block.

Dialog Box



Register size
The number of entries that the FIFO register can hold.

Trigger type
The type of event that triggers the block's execution. The rate of the trigger signal must be the same as the rate of the data signal input.

Push onto full register
Response to a trigger received at the `Push` port when the register is full. Inputs to this port must have the same built-in data type as inputs to the `Pop` and `Rst` input ports.

When `Dynamic reallocation` is selected, the **System target file** parameter on the **Real-Time Workshop** pane of the Configuration Parameters dialog box must be set to `grt_malloc.tlc   Generic Real-Time Target with dynamic memory allocation`.

Pop empty register
Response to a trigger received at the `Pop` port when the register is empty. Inputs to this port must have the same built-in data type as inputs to the `Push` and `Rst` input ports.

Show empty register indicator port
Enable the `Empty` output port, which is high (`1`) when the queue is empty, and low (`0`) otherwise.

Show full register indicator port
Enable the `Full` output port, which is high (`1`) when the queue is full, and low (`0`) otherwise. The `Full` port remains low when you select `Dynamic reallocation` from the **Push onto full register** parameter.

Show number of register entries port
Enable the `Num` output port, which tracks the number of entries currently on the queue. When inputs to the `In` port are double-precision values, the outputs from the `Num` port are double-precision values. Otherwise, the outputs from the `Num` port are 32-bit unsigned integer values.

Show reset port to clear internal stack buffer
Enable the `Rst` input port, which empties the queue when the trigger specified by the **Trigger type** is received. Inputs to this port must have the same built-in data type as inputs to the `Push` and `Pop` input ports.

Clear output port on reset
Reset the `Out` port to zero, in addition to clearing the queue, when a trigger is received at the `Rst` input port.

Supported Data Types

<table>
<tr><th>Port</th><th>Supported Data Types</th></tr>
<tr><td>In</td><td>Double-precision floating point
Single-precision floating point
Fixed point (signed and unsigned)
Boolean
8-, 16-, and 32-bit signed integers
8-, 16-, and 32-bit unsigned integers</td></tr>
<tr><td>Push</td><td>Double-precision floating point
Single-precision floating point
Boolean
8-, 16-, and 32-bit signed integers
8-, 16-, and 32-bit unsigned integers
Inputs to this port must have the same built-in data type as inputs to the Pop and Rst input ports</td></tr>
<tr><td>Pop</td><td>Double-precision floating point
Single-precision floating point
Boolean
8-, 16-, and 32-bit signed integers
8-, 16-, and 32-bit unsigned integers
Inputs to this port must have the same built-in data type as inputs to the Push and Rst input ports.</td></tr>
</table>

Port	Supported Data Types
Rst	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers Inputs to this port must have the same built-in data type as inputs to the Push and Pop input ports.
Out	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Empty	• Double-precision floating point • Boolean
Full	• Double-precision floating point • Boolean
Num	• Double-precision floating point The block outputs a double-precision floating-point value at this port when the data type of the In port is double-precision floating-point. • 32-bit unsigned integers The block outputs a 32-bit unsigned integer value at this port when the data type of the In port is anything other than double-precision floating-point.

See Also

Buffer	Signal Processing Blockset
Delay Line	Signal Processing Blockset
Stack	Signal Processing Blockset

Purpose

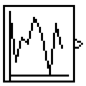
Generate randomly distributed values

Library

Signal Processing Sources

`dspsrcs4`

Description



The Random Source block generates a frame of M values drawn from a uniform or Gaussian pseudorandom distribution, where you specify M in the **Samples per frame** parameter.

This reference page contains a detailed discussion of the following Random Source block topics:

- "Distribution Type" on page 2-1087
- "Output Complexity" on page 2-1088
- "Output Repeatability" on page 2-1090
- "Specifying the Initial Seed" on page 2-1090
- "Sample Period" on page 2-1091
- "Dialog Box" on page 2-1092
- "Supported Data Types" on page 2-1095
- "See Also" on page 2-1095

Distribution Type

When the **Source type** parameter is set to `Uniform`, the output samples are drawn from a uniform distribution whose minimum and maximum values are specified by the **Minimum** and **Maximum** parameters, respectively. All values in this range are equally likely to be selected. A length-N vector specified for one or both of these parameters generates an N-channel output (M-by-N matrix) containing a unique random distribution in each channel.

For example, specify

- **Minimum** = `[ 0 0 -3 -3]`

- **Maximum** = [10 10 20 20]

to generate a four-channel output whose first and second columns contain random values in the range [0, 10], and whose third and fourth columns contain random values in the range [-3, 20]. When you specify only one of the **Minimum** and **Maximum** parameters as a vector, the block scalar expands the other parameter so it is the same length as the vector.

When the **Source type** parameter is set to `Gaussian`, you must also set the **Method** parameter, which determines the method by which the block computes the output, and has the following settings:

- `Ziggurat` — Produces Gaussian random values by using the Ziggurat method, which is the same method used by the MATLAB `randn` function.
- `Sum of uniform values` — Produces Gaussian random values by adding and scaling uniformly distributed random signals based on the central limit theorem. This theorem states that the probability distribution of the sum of a sufficiently high number of random variables approaches the Gaussian distribution. You must set the **Number of uniform values to sum** parameter, which determines the number of uniformly distributed random numbers to sum to produce a single Gaussian random value.

For both settings of the **Method** parameter, the output samples are drawn from the normal distribution defined by the **Mean** and **Variance** parameters. A length-N vector specified for one or both of the **Mean** and **Variance** parameters generates an N-channel output (M-by-N frame matrix) containing a distinct random distribution in each column. When you specify only one of these parameters as a vector, the block scalar expands the other parameter so it is the same length as the vector.

Output Complexity

The block's output can be either real or complex, as determined by the `Real` and `Complex` options in the **Complexity** parameter. These settings control all channels of the output, so real and complex data cannot be combined in the same output. For complex output with a `Uniform` distribution, the real and imaginary components in each channel are both drawn from the same uniform random distribution, defined by the **Minimum** and **Maximum** parameters for that channel.

For complex output with a `Gaussian` distribution, the real and imaginary components in each channel are drawn from normal distributions with different means. In this case, the **Mean** parameter for each channel should specify a complex value; the real component of the **Mean** parameter specifies the mean of the real components in the channel, while the imaginary component specifies the mean of the imaginary components in the channel. When either the real or imaginary component is omitted from the **Mean** parameter, a default value of 0 is used for the mean of that component.

For example, a **Mean** parameter setting of `[5+2i 0.5 3i]` generates a three-channel output with the following means.

Channel 1 mean	*real* = 5	*imaginary* = 2
Channel 2 mean	*real* = 0.5	*imaginary* = 0
Channel 3 mean	*real* = 0	*imaginary* = 3

For complex output, the **Variance** parameter, σ^2, specifies the *total variance* for each output channel. This is the sum of the variances of the real and imaginary components in that channel.

$$\sigma^2 = \sigma^2_{\mathrm{Re}} + \sigma^2_{\mathrm{Im}}$$

The specified variance is equally divided between the real and imaginary components, so that

$$\sigma^2_{\mathrm{Re}} = \frac{\sigma^2}{2}$$

$$\sigma^2_{\mathrm{Im}} = \frac{\sigma^2}{2}$$

Output Repeatability

The **Repeatability** parameter determines whether or not the block outputs the same signal each time you run the simulation. You can set the parameter to one of the following options:

- `Repeatable` — Outputs the same signal each time you run the simulation. The first time you run the simulation, the block randomly selects an initial seed. The block reuses these same initial seeds every time you rerun the simulation.
- `Specify seed` — Outputs the same signal each time you run the simulation. Every time you run the simulation, the block uses the initial seed(s) specified in the **Initial seed** parameter. Also see "Specifying the Initial Seed" on page 2-1090.
- `Not repeatable` — Does not output the same signal each time you run the simulation. Every time you run the simulation, the block randomly selects an initial seed.

Specifying the Initial Seed

When you set the **Repeatability** parameter to `Specify seed`, you must set the **Initial seed** parameter. The **Initial seed** parameter specifies the initial seed for the pseudorandom number generator. The generator produces an identical sequence of pseudorandom numbers each time it is executed with a particular initial seed.

Specifying Initial Seeds for Real Outputs

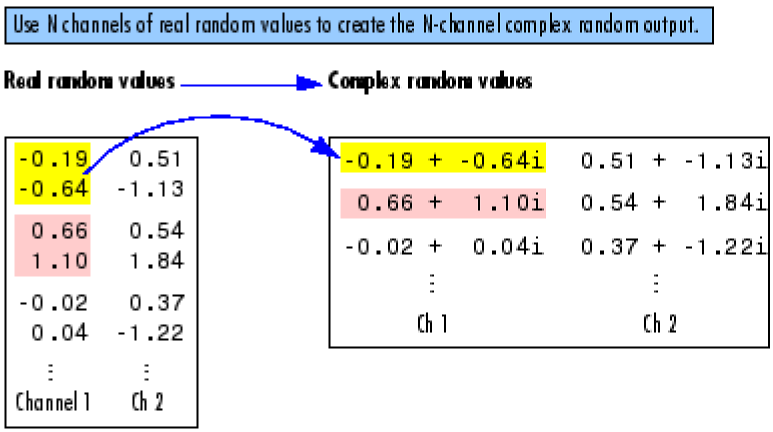
To specify the N initial seeds for an N-channel real-valued output, **Complexity** parameter set to `Real`, provide one of the following in the **Initial seed** parameter:

- Length-N vector of initial seeds — Uses each vector element as an initial seed for the corresponding channel in the N-channel output.
- Single scalar — Uses the scalar to generate N random values, which it uses as the seeds for the N-channel output.

Specifying Initial Seeds for Complex Outputs

To specify the initial seeds for an N-channel complex-valued output, **Complexity** parameter set to `Complex`, provide one of the following in the **Initial seed** parameter:

- Length-N vector of initial seeds — Uses each vector element as an initial seed for generating N channels of *real* random values. The block uses pairs of adjacent values in each of these channels as the real and imaginary components of the final output, as illustrated in the following figure.
- Single scalar — Uses the scalar to generate N random values, which it uses as the seeds for generating N channels of *real* random values. The block uses pairs of adjacent values in each of these channels as the real and imaginary components of the final output, as illustrated in the following figure.



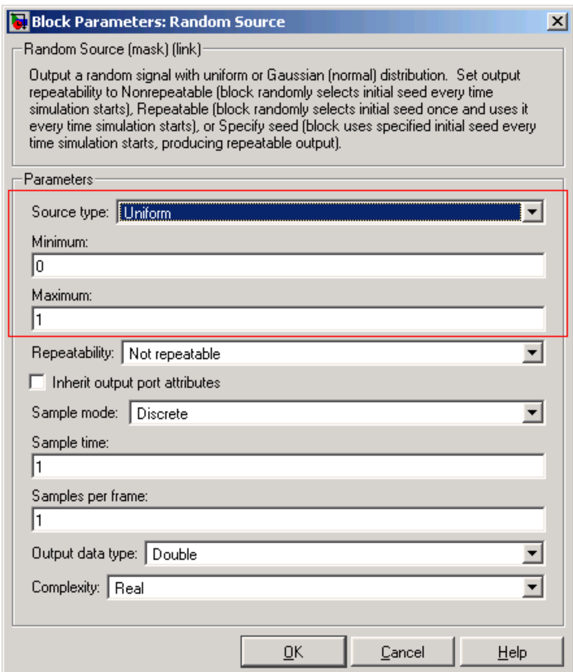
Sample Period

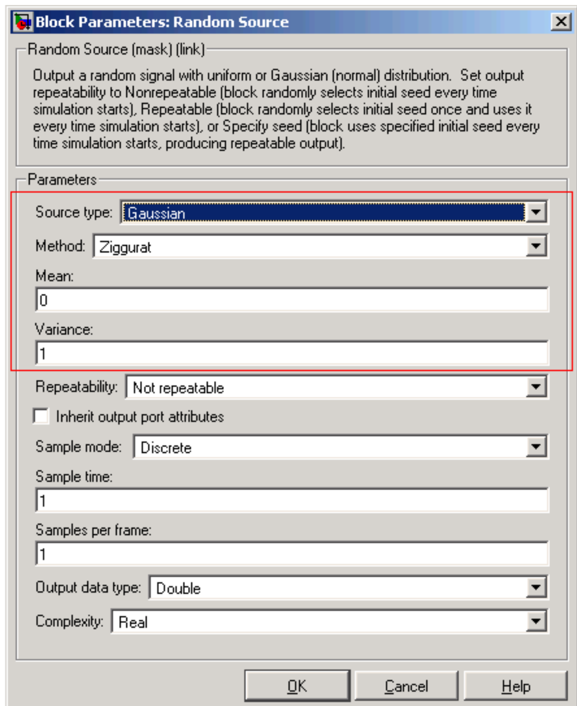
The **Sample time** parameter value, T_s, specifies the random sequence sample period when the **Sample mode** parameter is set to `Discrete`. In this mode, the block generates the number of samples specified by the **Samples per frame** parameter value, M, and outputs this frame with a period of M*T_s. For M=1, the output is sample based; otherwise, the output is frame based.

When **Sample mode** is set to `Continuous`, the block is configured for continuous-time operation, and the **Sample time** and **Samples per frame** parameters are disabled. Note that many Signal Processing Blockset blocks do not accept continuous-time inputs.

Dialog Box

Only some of the parameters described below are visible in the dialog box at any one time.





Opening this dialog box causes a running simulation to pause. See “Changing Source Block Parameters During Simulation” in the online Simulink documentation for details.

Source type
: The distribution from which to draw the random values, `Uniform` or `Gaussian`. For more information, see "Distribution Type" on page 2-1087.

Method
: The method by which the block computes the Gaussian random values, `Ziggurat` or `Sum of uniform values`. This parameter is enabled when **Source type** is set to `Gaussian`. For more information, see "Distribution Type" on page 2-1087.

Minimum
: The minimum value in the uniform distribution. This parameter is enabled when you select `Uniform` from the **Source type** parameter. Tunable.

Maximum
: The maximum value in the uniform distribution. This parameter is enabled when you select you select `Uniform` from the **Source type** parameter. Tunable.

Number of uniform values to sum
: The number of uniformly distributed random values to sum to compute a single number in a Gaussian random distribution. This parameter is enabled when the **Source type** parameter is set to `Gaussian`, and the **Method** parameter is set to `Sum of uniform values`. For more information, see "Distribution Type" on page 2-1087.

Mean
: The mean of the Gaussian (normal) distribution. This parameter is enabled when you select `Gaussian` from the **Source type** parameter. Tunable.

Variance
: The variance of the Gaussian (normal) distribution. This parameter is enabled when you select `Gaussian` from the **Source type** parameter. Tunable.

Repeatability

The repeatability of the block output: `Not repeatable`, `Repeatable`, or `Specify seed`. In the `Repeatable` and `Specify seed` settings, the block outputs the same signal every time you run the simulation. For details, see "Output Repeatability" on page 2-1090.

Initial seed

The initial seed(s) to use for the random number generator when you set the **Repeatability** parameter to `Specify seed`. For details, see "Specifying the Initial Seed" on page 2-1090. Tunable.

Inherit output port attributes

When you select this check box, block inherits the sample mode, sample time, output data type, complexity, and signal dimensions of a sample-based signal from a downstream block. When you select this check box, the **Sample mode**, **Sample time**, **Samples per frame**, **Output data type**, and **Complexity** parameters are disabled.

Suppose you want to back propagate a 1-D vector. The output of the Random Source block is a length M sample-based 1-D vector, where length M is inherited from the downstream block. When the **Minimum**, **Maximum**, **Mean**, or **Variance** parameter specifies N channels, the 1-D vector output contains M/N samples from each channel. An error occurs in this case when M is not an integer multiple of N.

Suppose you want to back propagate a M-by-N signal. When N>1, your signal has N channels. When N = 1, your signal has M channels. The value of the **Minimum**, **Maximum**, **Mean**, or **Variance** parameter can be a scalar or a vector of length equal to the number of channels. You can specify these parameters as either row or column vectors, except when the signal is a row vector. In this case, the **Minimum**, **Maximum**, **Mean**, or **Variance** parameter must also be specified as a row vector.

Sample mode
The sample mode, `Continuous` or `Discrete`. This parameter is enabled when the **Inherit output port attributes** check box is cleared.

Sample time
The sample period, T_s, of the random output sequence. The output frame period is $M*T_s$. This parameter is enabled when the **Inherit output port attributes** check box is cleared.

Samples per frame
The number of samples, M, in each output frame. When the value of this parameter is `1`, the block outputs a sample-based signal.

This parameter is enabled when the **Inherit output port attributes** check box is cleared.

Output data type
The data type of the output, single-precision or double-precision. This parameter is enabled when the **Inherit output port attributes** check box is cleared.

Complexity
The complexity of the output, `Real` or `Complex`. This parameter is enabled when the **Inherit output port attributes** check box is cleared.

Supported Data Types

- Double-precision floating-point
- Single-precision floating-point

See Also

Discrete Impulse	Signal Processing Blockset
Maximum	Signal Processing Blockset
Minimum	Signal Processing Blockset

Signal From Workspace	Signal Processing Blockset
Standard Deviation	Signal Processing Blockset
Variance	Signal Processing Blockset
Constant	Simulink
Random Number	Simulink
Signal Generator	Simulink
`rand`	MATLAB
`randn`	MATLAB

Purpose

Compute real cepstrum of input

Library

Transforms

`dspxfrm3`

Description

The Real Cepstrum block computes the real cepstrum of each channel in the real-valued M-by-N input matrix, `u`. For both sample-based and frame-based inputs, the block assumes that each input column is a frame containing M consecutive samples from an independent channel. The block does not accept complex-valued inputs.

The output is a real M_o-by-N matrix, where you specify M_o in the **FFT length** parameter. Each output column contains the length-M_o real cepstrum of the corresponding input column.

```
y = real(ifft(log(abs(fft(u,Mo)))))
```

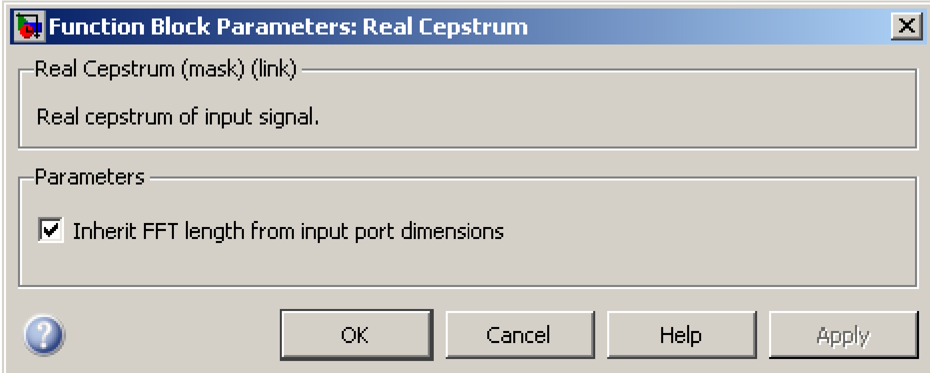
or, more compactly,

```
y = rceps(u,Mo)
```

When you select the **Inherit FFT length from input port dimensions** check box, the output frame size matches the input frame size (M_o=M). In this case, the block processes *sample-based* length-M row vector inputs as a single channel (that is, as an M-by-1 column vector), and returns the result as a length-M column vector. The block *always* processes 1-D vector inputs as a single channel, and returns the result as a length-M column vector.

The output is always sample based, and the output port rate is the same as the input port rate.

Dialog Box



Inherit FFT length from input port dimensions
: When you select this check box, the output frame size matches the input frame size.

FFT length
: The number of frequency points at which to compute the FFT, which is also the output frame size, M_o. This parameter is visible only when you clear the **Inherit FFT length from input port dimensions** check box.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Complex Cepstrum	Signal Processing Blockset
DCT	Signal Processing Blockset
FFT	Signal Processing Blockset
`rceps`	Signal Processing Toolbox

Purpose

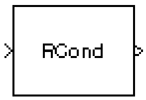
Compute reciprocal condition of square matrix in 1-norm

Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description



The Reciprocal Condition block computes the reciprocal of the condition number for a square input matrix A.

```
y = rcond(A)     % Equivalent MATLAB code
```

or

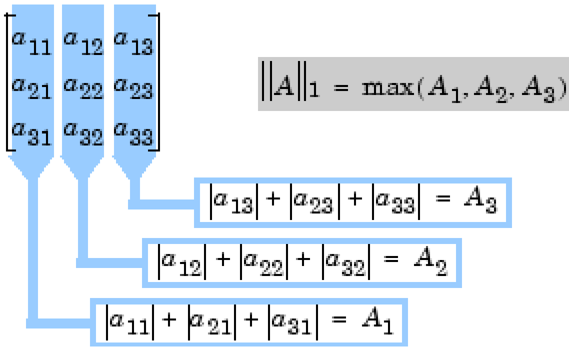
$$y = \frac{1}{\kappa} = \frac{1}{\|A^{-1}\|_1 \|A\|_1}$$

where κ is the condition number ($\kappa \geq 1$), and y is the scalar sample-based output ($0 \leq y < 1$).

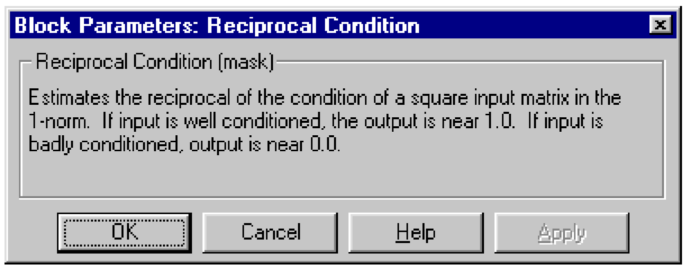
The matrix 1-norm, $\|A\|_1$, is the maximum column-sum in the M-by-M matrix A.

$$\|A\|_1 = \max_{1 \leq j \leq M} \sum_{i=1}^{M} |a_{ij}|$$

For a 3-by-3 matrix:



Dialog Box



References

Golub, G. H., and C. F. Van Loan. *Matrix Computations*. 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Matrix 1-Norm	Signal Processing Blockset
Normalization	Signal Processing Blockset
`rcond`	MATLAB

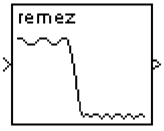
Purpose

Design and apply equiripple FIR filter

Library

`dspobslib`

Description



Note The Remez FIR Filter Design block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the Digital Filter block.

The Remez FIR Filter Design block implements the Parks-McClellan algorithm to design and apply a linear-phase filter with an arbitrary multiband magnitude response. The filter design, which uses the Signal Processing Toolbox `firpm` function, minimizes the maximum error between the desired frequency response and the actual frequency response. Such filters are called *equiripple* due to the equiripple behavior of their approximation error. The block applies the filter to a discrete-time input using the Direct-Form II Transpose Filter block.

An M-by-N sample-based matrix input is treated as M*N independent channels, and an M-by-N frame-based matrix input is treated as N independent channels. In both cases, the block filters each channel independently over time, and the output has the same size and frame status as the input.

The **Filter type** parameter allows you to specify one of the following filters:

- `Multiband`

  The multiband filter has an arbitrary magnitude response and linear phase.

- `Differentiator`

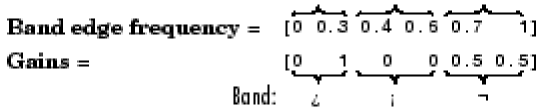
  The differentiator filter approximates the ideal differentiator. Differentiators are antisymmetric FIR filters with approximately linear magnitude responses. To obtain the correct derivative, scale the **Gains at these frequencies** vector by πF_s rad/s, where F_s is the sample frequency in Hertz.

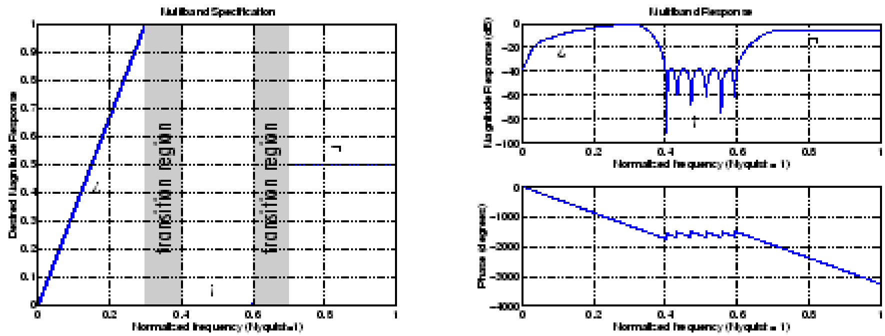
- `Hilbert Transformer`

  The Hilbert transformer filter approximates the ideal Hilbert transformer. Hilbert transformers are antisymmetric FIR filters with approximately constant magnitude.

The **Band-edge frequency vector** parameter is a vector of frequency points in the range `0` to `1`, where `1` corresponds to half the sample frequency. Each band is defined by the two bounding frequencies, so this vector must have even length. Frequency points must appear in ascending order. The **Gains at these frequencies** parameter is a vector of the same size containing the desired magnitude response at the corresponding points in the **Band-edge frequency vector**.

Each odd-indexed frequency-amplitude pair defines the left endpoint of a line segment representing the desired magnitude response in that frequency band. The corresponding even-indexed frequency-amplitude pair defines the right endpoint. Between the frequency bands specified by these end-points, there may be undefined sections of the specified frequency response. These are called “don’t care” or “transition” regions, and the magnitude response in these areas is a by-product of the optimization in the other specified frequency ranges.





The **Weights** parameter is a vector that specifies the emphasis to be placed on minimizing the error in certain frequency bands relative to others. This vector specifies one weight per band, so it is half the length of the **Band-edge frequency vector** and **Gains at these frequencies** vectors.

In most cases, differentiators and Hilbert transformers have only a single band, so the weight is a scalar value that does not affect the final filter. However, the **Weights** parameter is useful when using the block to design an antisymmetric multiband filter, such as a Hilbert transformer with stopbands.

Examples

Example 1: Multiband

Consider a lowpass filter with a transition band in the normalized frequency range 0.4 to 0.5, and 10 times greater error minimization in the stopband than in the passband.

In this case:

- **Filter type** = `Multiband`

- **Band-edge frequency vector** = `[0 0.4 0.5 1]`
- **Gains at these frequencies** = `[1 1 0 0]`
- **Weights** = `[1 10]`

Example 2: Differentiator

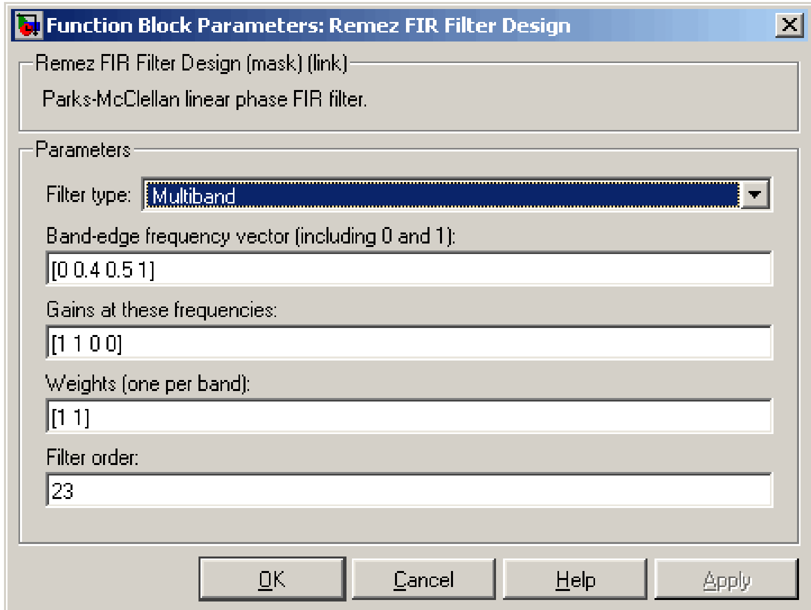
Assume the specifications for a differentiator filter require it to have order 21. The “ramp” response extends over the entire frequency range. In this case, specify:

- **Filter type** = `Differentiator`
- **Band-edge frequency vector** = `[0 1]`
- **Gains at these frequencies** = `[0 pi*Fs]`
- **Filter order** = `21`

For a type III even order filter, the differentiation band should stop short of half the sample frequency. For example, if the filter order is 20, you could specify the block parameters as follows:

- **Filter type** = `Differentiator`
- **Band-edge frequency vector** = `[0 0.9]`
- **Gains at these frequencies** = `[0 0.9*pi*Fs]`
- **Filter order** = `20`

Dialog Box



Filter type

The filter type. Tunable.

Band-edge frequency vector

A vector of frequency points, in ascending order, in the range 0 to 1. The value 1 corresponds to half the sample frequency. This vector must have even length. Tunable.

Gains at these frequencies

A vector of frequency-response magnitudes corresponding to the points in the **Band-edge frequency** vector. This vector must be the same length as the **Band-edge frequency** vector. Tunable.

Weights

A vector containing one weight for each frequency band. This vector must be half the length of the **Band-edge frequency** and **Gains at these frequencies** vectors. Tunable.

Filter order

The filter order.

References

Oppenheim, A. V. and R. W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

Proakis, J. and D. Manolakis. *Digital Signal Processing*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

Purpose

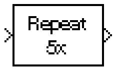
Resample input at higher rate by repeating values

Library

Signal Operations

`dspsigops`

Description



The Repeat block upsamples each channel of the M_i-by-N input to a rate L times higher than the input sample rate by repeating each consecutive input sample L times at the output. You specify the integer L in the **Repetition count** parameter.

This block supports triggered subsystems if, for **Frame-based mode,** you select `Maintain input frame rate`.

Sample-Based Operation

When the input is sample based, the block treats each of the M*N matrix elements as an independent channel, and upsamples each channel over time. The **Frame-based mode** parameter must be set to `Maintain input frame size`. The output sample rate is L times higher than the input sample rate ($T_{so} = T_{si}/L$), and the input and output sizes are identical.

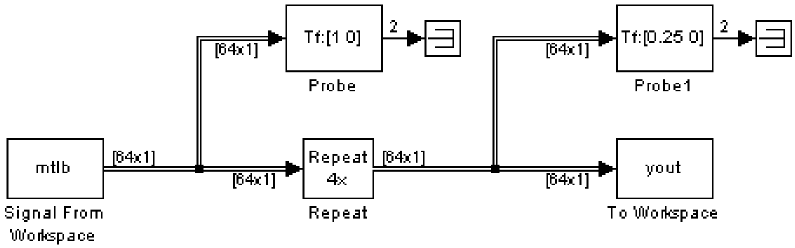
Frame-Based Operation

When the input is frame based, the block treats each of the N input columns as a frame containing M_i sequential time samples from an independent channel. The block upsamples each channel independently by repeating each row of the input matrix L times at the output. The **Frame-based mode** parameter determines how the block adjusts the rate at the output to accommodate the repeated rows. There are two available options:

- `Maintain input frame size`

  The block generates the output at the faster (upsampled) rate by using a proportionally shorter frame *period* at the output port than at the input port. For L repetitions of the input, the output frame period is L times shorter than the input frame period ($T_{fo} = T_{fi}/L$), but the input and output frame sizes are equal.

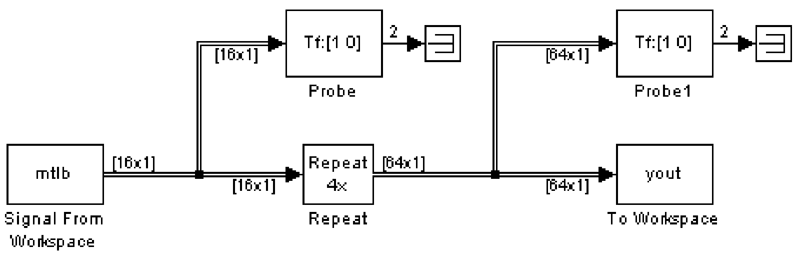
The model below shows a single-channel input with a frame period of 1 second being upsampled through 4-times repetition to a frame period of 0.25 second. The input and output frame sizes are identical.



- `Maintain input frame rate`

  The block generates the output at the faster (upsampled) rate by using a proportionally larger frame *size* than the input. For L repetitions of the input, the output frame size is L times larger than the input frame size ($M_o = M_i*L$), but the input and output frame rates are equal.

  The model below shows a single-channel input of frame size 16 being upsampled through 4-times repetition to a frame size of 64. The input and output frame rates are identical.



Zero Latency

The Repeat block has *zero-tasking latency* for all single-rate operations. The block is single-rate for the particular combinations of sampling mode and parameter settings shown in the table below.

Sampling Mode	Parameter Settings
Sample based	**Repetition count** parameter, L, is `1`.
Frame based	**Repetition count** parameter, L, is `1`, *or* **Frame-based mode** parameter is `Maintain input frame rate`.

The block also has zero latency for all multirate operations in the Simulink single-tasking mode.

Zero-tasking latency means that the block repeats the first input (received at *t*=0) for the first L output samples, the second input for the next L output samples, and so on. The **Initial condition** parameter value is not used.

Nonzero Latency

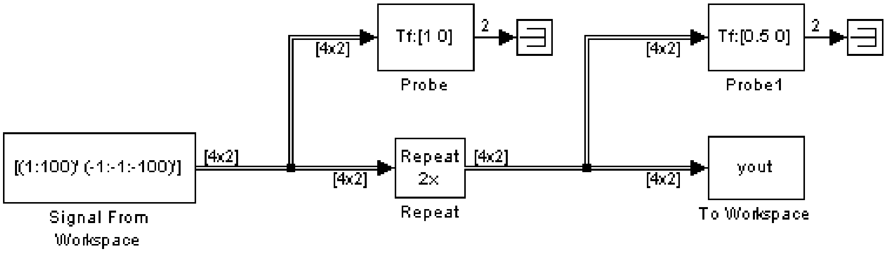
The Repeat block has tasking latency only for multirate operation in the Simulink multitasking mode:

- In sample-based mode, the initial condition for each channel is repeated for the first L output samples. The channel's first input appears as output sample L+1. The **Initial condition** value can be an M_i-by-N matrix containing one value for each channel, or a scalar to be applied to all signal channels.
- In frame-based mode, the first row of the initial condition matrix is repeated for the first L output samples, the second row of the initial condition matrix is repeated for the next L output samples, and so on. The first row of the first input matrix appears in the output as sample M_iL+1. The **Initial condition** value can be an M_i-by-N matrix, or a scalar to be repeated across all elements of the M_i-by-N matrix. See the example below for an illustration of this case.

Note For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

Examples

Construct the frame-based model shown below.



Adjust the block parameters as follows.

- Configure the Signal From Workspace block to generate a two-channel signal with frame size of 4 and sample period of 0.25. This represents an output frame period of 1 (0.25*4). The first channel should contain the positive ramp signal 1, 2, ..., 100, and the second channel should contain the negative ramp signal -1, -2, ..., -100.
  - **Signal** = `[(1:100)' (-1:-1:-100)']`
  - **Sample time** = `0.25`
  - **Samples per frame** = `4`
- Configure the Repeat block to upsample the two-channel input by increasing the output frame rate by a factor of 2 relative to the input frame rate. Set an initial condition matrix of

$$\begin{bmatrix} 11 & -11 \\ 12 & -12 \\ 13 & -13 \\ 14 & -14 \end{bmatrix}$$

- **Repetition count** = `2`
- **Initial condition** = `[11 -11;12 -12;13 -13;14 -14]`
- **Frame-based mode** = `Maintain input frame size`

- Configure the Probe blocks by clearing the **Probe width** and **Probe complex signal** check boxes (if desired).

This model is multirate because there are at least two distinct sample rates, as shown by the two Probe blocks. To run this model in the Simulink multitasking mode, in the **Solver** pane of the Configuration Parameters dialog box, set the **Type** list to `Fixed-step` and set the **Solver** list to `Discrete (no continuous states)`. For the **Tasking mode for periodic sample times** parameter, select `MultiTasking`. Also set the **Stop time** to `30`.

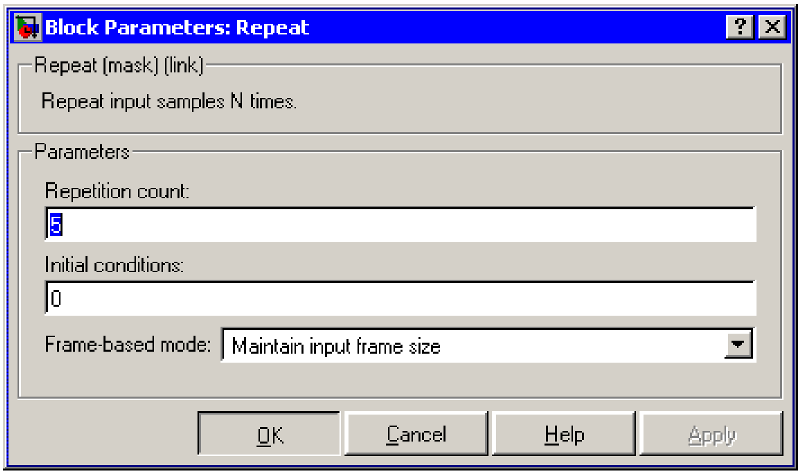
Run the model and look at the output, `yout`. The first few samples of each channel are shown below.

```
yout =
    11    -11
    11    -11
    12    -12
    12    -12
    13    -13
    13    -13
    14    -14
    14    -14
     1     -1
     1     -1
     2     -2
     2     -2
```

3	-3
3	-3
4	-4
4	-4
5	-5
5	-5

Since we ran this frame-based multirate model in multitasking mode, the block repeats each row of the initial condition matrix for L output samples, where L is the **Repetition count** of 2. The first row of the first input matrix appears in the output as sample 9 ,that is, sample M_iL+1, where M_i is the input frame size.

Dialog Box



Repetition count

The integer number of times, L, that the input value is repeated at the output. This is the factor by which the output frame size or sample rate is increased.

Initial conditions

The value with which the block is initialized for cases of nonzero latency; a scalar or matrix.

Frame-based mode

For frame-based operation, the method by which to implement the repetition (upsampling): `Maintain input frame size` that is, increase the frame rate, or `Maintain input frame rate`, that is, increase the frame size. The **Frame-based mode** parameter must be set to `Maintain input frame size` for sample-base inputs.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

FIR Interpolation	Signal Processing Blockset
Upsample	Signal Processing Blockset

RLS Adaptive Filter (Obsolete)

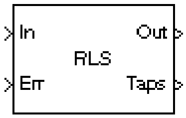
Purpose

Compute filter estimates for input using RLS adaptive filter algorithm

Library

`dspobslib`

Description



Note The RLS Adaptive Filter block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the RLS Filter block.

The RLS Adaptive Filter block recursively computes the recursive least squares (RLS) estimate of the FIR filter coefficients.

The corresponding RLS filter is expressed in matrix form as

$$
\begin{aligned}
k(n) &= \frac{\lambda^{-1}P(n-1)u(n)}{1+\lambda^{-1}u^{H}(n)P(n-1)u(n)} \\
y(n) &= \hat{w}^{H}(n-1)u(n) \\
e(n) &= d(n)-y(n) \\
\hat{w}(n) &= \hat{w}(n-1)+k(n)e^{*}(n) \\
P(n) &= \lambda^{-1}P(n-1)-\lambda^{-1}k(n)u^{H}(n)P(n-1)
\end{aligned}
$$

where λ^{-1} denotes the reciprocal of the exponential weighting factor. The variables are as follows

Variable	Description
n	The current algorithm iteration
$u(n)$	The buffered input samples at step n
$P(n)$	The inverse correlation matrix at step n
$k(n)$	The gain vector at step n
$\hat{w}(n)$	The vector of filter-tap estimates at step n

Variable	Description
$y(n)$	The filtered output at step n
$e(n)$	The estimation error at step n
$d(n)$	The desired response at step n
λ	The exponential memory weighting factor

The block icon has port labels corresponding to the inputs and outputs of the RLS algorithm. Note that inputs to the `In` and `Err` ports must be sample-based scalars. The signal at the `Out` port is a scalar, while the signal at the `Taps` port is a sample-based vector.

Block Ports	Corresponding Variables
`In`	u, the scalar input, which is internally buffered into the vector $u(n)$
`Out`	$y(n)$, the filtered scalar output
`Err`	$e(n)$, the scalar estimation error
`Taps`	$\hat{w}(0)$, the vector of filter-tap estimates

An optional `Adapt` input port is added when you select the **Adapt input** check box in the dialog box. When this port is enabled, the block continuously adapts the filter coefficients while the `Adapt` input is nonzero. A zero-valued input to the `Adapt` port causes the block to stop adapting, and to hold the filter coefficients at their current values until the next nonzero `Adapt` input.

The implementation of the algorithm in the block is optimized by exploiting the symmetry of the inverse correlation matrix $P(n)$. This decreases the total number of computations by a factor of two.

The **FIR filter length** parameter specifies the length of the filter that the RLS algorithm estimates. The **Memory weighting factor** corresponds to λ in the equations, and specifies how quickly the filter "forgets" past sample information. Setting λ=1 specifies an infinite memory; typically, 0.95≤λ≤1.

The **Initial value of filter taps** specifies the initial value $\hat{w}(0)$ as a vector, or as a scalar to be repeated for all vector elements. The initial value of $P(n)$ is

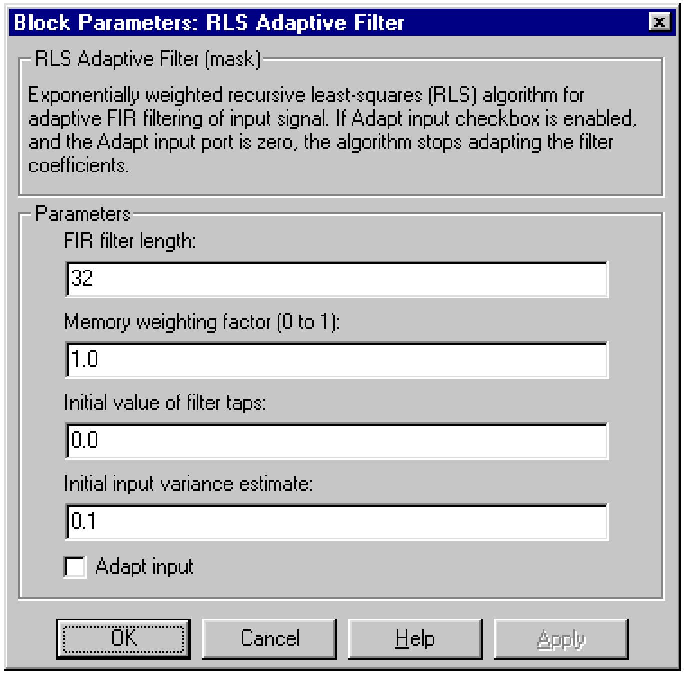
$$I\frac{1}{\hat{\sigma}^2}$$

where you specify $\hat{\sigma}^2$ in the **Initial input variance estimate** parameter.

Examples

The `rlsdemo` demo illustrates a noise cancellation system built around the RLS Adaptive Filter block.

Dialog Box



FIR filter length
: The length of the FIR filter.

Memory weighting factor
: The exponential weighting factor, in the range `[0,1]`. A value of `1` specifies an infinite memory. Tunable.

Initial value of filter taps
: The initial FIR filter coefficients.

Initial input variance estimate
: The initial value of 1/P(n).

Adapt input
: Enables the `Adapt` port.

References

Haykin, S. *Adaptive Filter Theory*. 3rd ed. Englewood Cliffs, NJ: Prentice Hall, 1996.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Kalman Adaptive Filter (Obsolete)	Signal Processing Blockset
LMS Adaptive Filter (Obsolete)	Signal Processing Blockset

See "Adaptive Filters" for related information.

Purpose

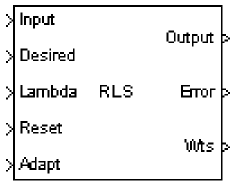
Compute filtered output, filter error, and filter weights for given input and desired signal using RLS adaptive filter algorithm

Library

Filtering / Adaptive Filters

`dspadpt3`

Description



RLS Filter

The RLS Filter block recursively computes the least squares estimate (RLS) of the FIR filter weights. The block estimates the filter weights, or coefficients, needed to convert the input signal into the desired signal. Connect the signal you want to filter to the Input port. This input signal can be a sample-based scalar or a single-channel frame-based signal. Connect the signal you want to model to the Desired port. The desired signal must have the same data type, frame status, complexity, and dimensions as the input signal. The Output port outputs the filtered input signal, which can be sample or frame based. The Error port outputs the result of subtracting the output signal from the desired signal.

The corresponding RLS filter is expressed in matrix form as

$$\mathbf{k}(n) = \frac{\lambda^{-1}\mathbf{P}(n-1)\mathbf{u}(n)}{1+\lambda^{-1}\mathbf{u}^{H}(n)\mathbf{P}(n-1)\mathbf{u}(n)}$$

$$y(n) = \mathbf{w}(n-1)\mathbf{u}(n)$$

$$e(n) = d(n) - y(n)$$

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mathbf{k}^{H}(n)e(n)$$

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \lambda^{-1}\mathbf{k}(n)\mathbf{u}^{H}(n)\mathbf{P}(n-1)$$

where λ^{-1} denotes the reciprocal of the exponential weighting factor. The variables are as follows

Variable	Description
n	The current time index
$\mathbf{u}(n)$	The vector of buffered input samples at step n

Variable	Description
$\mathbf{P}(n)$	The inverse correlation matrix at step n
$\mathbf{k}(n)$	The gain vector at step n
$\mathbf{w}(n)$	The vector of filter-tap estimates at step n
$y(n)$	The filtered output at step n
$e(n)$	The estimation error at step n
$d(n)$	The desired response at step n
λ	The forgetting factor

The implementation of the algorithm in the block is optimized by exploiting the symmetry of the inverse correlation matrix $P(n)$. This decreases the total number of computations by a factor of two.

Use the **Filter length** parameter to specify the length of the filter weights vector.

The **Forgetting factor (0 to 1)** parameter corresponds to λ in the equations. It specifies how quickly the filter "forgets" past sample information. Setting λ=1 specifies an infinite memory. Typically,

$1-\frac{1}{2L} < \lambda < 1$, where L is the filter length. You can specify a forgetting factor using the input port, Lambda, or enter a value in the **Forgetting factor (0 to 1)** parameter in the Block Parameters: RLS Filter dialog box.

Enter the initial filter weights, $\hat{w}(0)$, as a vector or a scalar for the **Initial value of filter weights** parameter. When you enter a scalar, the block uses the scalar value to create a vector of filter weights. This vector has length equal to the filter length and all of its values are equal to the scalar value.

The initial value of $P(n)$ is

$$\frac{1}{\sigma^2}I$$

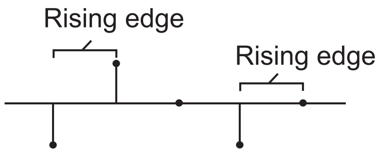
where you specify σ^2 in the **Initial input variance estimate** parameter.

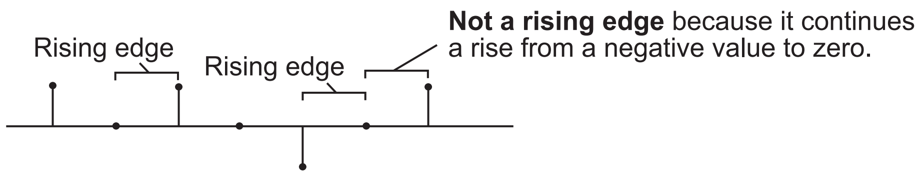
When you select the **Adapt port** check box, an Adapt port appears on the block. When the input to this port is nonzero, the block continuously updates the filter weights. When the input to this port is zero, the filter weights remain at their current values.

When you want to reset the value of the filter weights to their initial values, use the **Reset input** parameter. The block resets the filter weights whenever a reset event is detected at the Reset port. The reset signal rate must be the same rate as the data signal input.

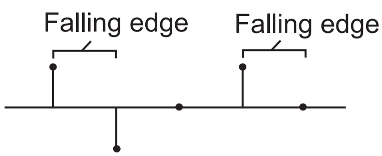
From the **Reset input** list, select `None` to disable the Reset port. To enable the Reset port, select one of the following from the **Reset input** list:

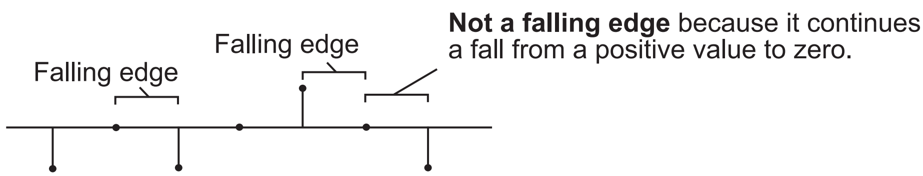
- `Rising edge` — Triggers a reset operation when the Reset input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero; see the following figure





- `Falling edge` — Triggers a reset operation when the Reset input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero; see the following figure





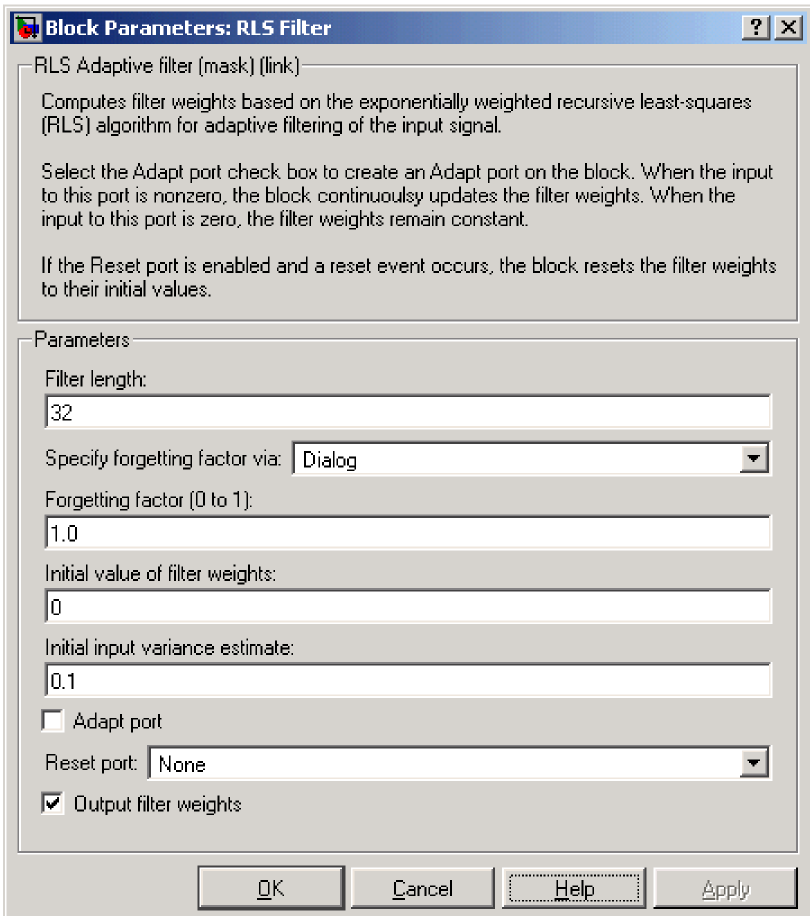
- `Either edge` — Triggers a reset operation when the Reset input is a `Rising edge` or `Falling edge`, as described above
- `Non-zero sample` — Triggers a reset operation at each sample time that the Reset input is not zero

Select the **Output filter weights** check box to create a Wts port on the block. For each iteration, the block outputs the current updated filter weights from this port.

Examples

The `rlsdemo` demo illustrates a noise cancellation system built around the RLS Filter block.

Dialog Box



Filter length
: Enter the length of the FIR filter weights vector.

Specify forgetting factor via
: Select `Dialog` to enter a value for the forgetting factor in the Block parameters: RLS Filter dialog box. Select `Input port` to specify the forgetting factor using the Lambda input port.

Forgetting factor (0 to 1)
: Enter the exponential weighting factor in the range $0 \leq \lambda \leq 1$. A value of `1` specifies an infinite memory. Tunable.

Initial value of filter weights
: Specify the initial values of the FIR filter weights.

Initial input variance estimate
: The initial value of 1/P(*n*).

Adapt port
: Select this check box to enable the Adapt input port.

Reset input
: Select this check box to enable the Reset input port.

Output filter weights
: Select this check box to export the filter weights from the Wts port.

References

Hayes, M.H. *Statistical Digital Signal Processing and Modeling.* New York: John Wiley & Sons, 1996.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Kalman Adaptive Filter (Obsolete)	Signal Processing Blockset
LMS Filter	Signal Processing Blockset

Block LMS Filter	Signal Processing Blockset
Fast Block LMS Filter	Signal Processing Blockset

See “Adaptive Filters” for related information.

Purpose

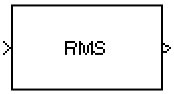
Compute root-mean-square value of input or sequence of inputs

Library

Statistics

`dspstat3`

Description



The RMS block computes the RMS value of each row or column of the input, along vectors of a specified dimension of the input, or of the entire input. The RMS block can also track the RMS value in a sequence of inputs over a period of time. The **Running RMS** parameter selects between basic operation and running operation.

Basic Operation

When you do not select the **Running RMS** check box, the block computes the RMS value of each row or column of the input, along vectors of a specified dimension of the input, or of the entire input at each individual sample time, and outputs the array y. Each element in y is the RMS value of the corresponding column, row, vector, or entire input. The output y depends on the setting of the **Find the RMS value over** parameter. For example, consider a 3-dimensional input signal of size M-by-N-by-P:

- `Entire input` — The output at each sample time is a scalar that contains the RMS value of the entire input. In this mode, the output is always sample based.
- `Each row` — The output at each sample time consists of an M-by-1-by-P array, where each element contains the RMS value of each vector over the second dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is an M-by-1 column vector. In this mode, the frame status of the output is the same as that of the input.
- `Each column` — The output at each sample time consists of a 1-by-N-by-P array, where each element contains the RMS value of each vector over the first dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is a 1-by-N

row vector. In this mode, the frame status of the output is the same as that of the input.

For convenience, length-M 1-D vector inputs are treated as M-by-1 column vectors when the block is in this mode. Sample-based length-M row vector inputs are also treated as M-by-1 column vectors when the **Treat sample-based row input as a column** check box is selected.

- `Specified dimension` — The output at each sample time depends on **Dimension**. If **Dimension** is set to 1, the output is the same as that when you select `Each column`. If **Dimension** is set to 2, the output is the same as when you select `Each row`. If **Dimension** is set to 3, the output at each sample time is an M-by-N matrix containing the RMS value of each vector over the third dimension of the input. In this mode, the frame status of the output is the same as that of the input.

The RMS value of the jth column of an M-by-N input matrix u is given by

$$y_j = \sqrt{\frac{\sum_{i=1}^{M} \left|u_{ij}\right|^2}{M}} \qquad 1 \le j \le N$$

```
y = sqrt(sum(u.*conj(u))/size(u,1))     % Equivalent MATLAB code
```

Running Operation

When you select the **Running RMS** check box, the block tracks the RMS value of successive inputs to the block. For sample-based M-by-N inputs, the output is a sample-based M-by-N matrix, with each element y_{ij} containing the RMS value of element u_{ij} over all inputs since the last reset. For frame-based M-by-N inputs, the output is a frame-based M-by-N matrix with each element y_{ij} containing the RMS value of the jth column over all inputs since the last reset, up to and including element u_{ij} of the current input.

N-D signals cannot be frame based. When the **Running RMS** check box is selected, each element of the N-D signal is treated as a separate channel.

There are $\prod d_i$ channels, where d_i is the size of the ith dimension.

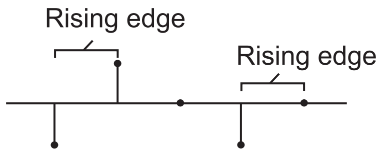
Resetting the Running RMS

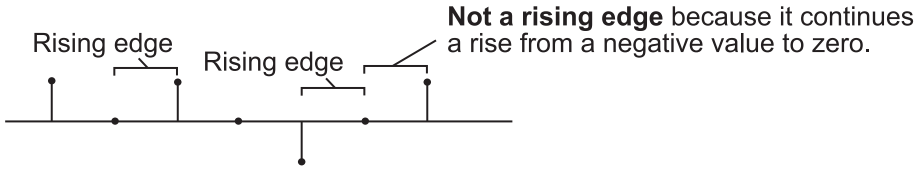
The block resets the running RMS whenever a reset event is detected at the optional `Rst` port. The reset sample time must be a positive integer multiple of the input sample time.

When the block is reset for sample-based inputs, the running RMS for each channel is initialized to the value in the corresponding channel of the current input. For frame-based inputs, the running RMS for each channel is initialized to the earliest value in each channel of the current input.

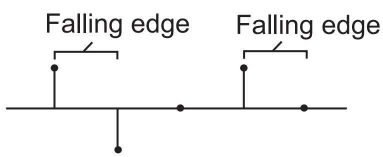
You specify the reset event in the **Reset port** parameter:

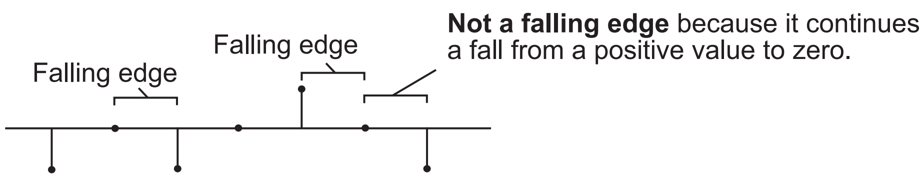
- `None` disables the `Rst` port.
- `Rising edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)





- `Falling edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)



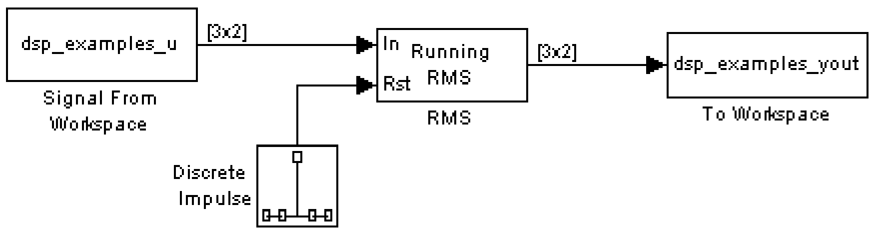


- `Either edge` — Triggers a reset operation when the `Rst` input is a `Rising edge` or `Falling edge`, as described earlier
- `Non-zero sample` — Triggers a reset operation at each sample time that the `Rst` input is not zero

Note When running simulations in the Simulink MultiTasking mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset. For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

Examples

The RMS block in the next model calculates the running RMS of a frame-based 3-by-2 (two-channel) matrix input, `u`. The running RMS is reset at t=2 by an impulse to the block's `Rst` port.



The RMS block has the following settings:

- **Running RMS** = Select this check box.
- **Reset port** = `Non-zero sample`

The Signal From Workspace block has the following settings:

- **Signal** = `dsp_examples_u`
- **Sample time** = `1/3`
- **Samples per frame** = `3`

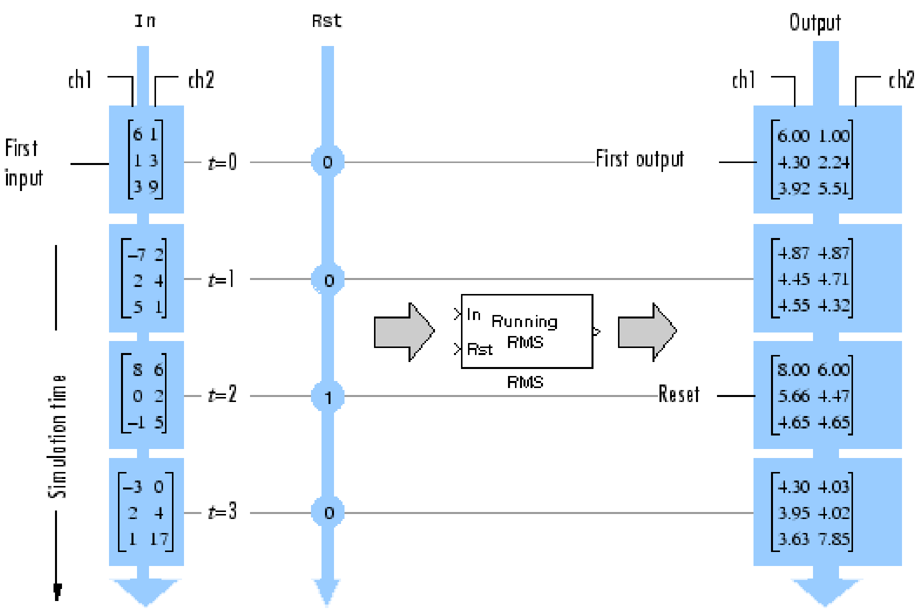
where

```
dsp_examples_u = [6 1 3 -7 2 5 8 0 -1 -3 2 1;1 3 9 2 4 1 6 2 5 0 4 17]'
```

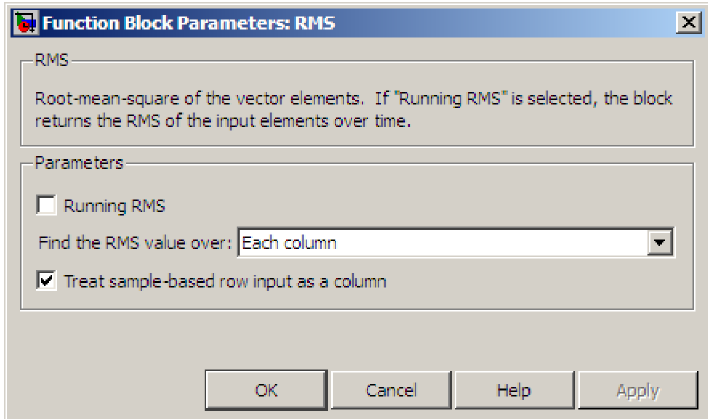
The Discrete Impulse block has the following settings:

- **Delay (samples)** = 2
- **Sample time** = 1
- **Samples per frame** = 1

The block's operation is shown in the next figure.



Dialog Box



Running RMS
: Enables running operation when selected.

Reset port
: Determines the reset event that causes the block to reset the running RMS. The reset signal rate must be a positive integer multiple of the rate of the data signal input. This parameter is enabled only when you set the **Running RMS** parameter. For more information, see "Resetting the Running RMS" on page 2-1128.

Find the RMS value over
: Specify whether to find the RMS value along rows, columns, entire input, or the dimension specified in the **Dimension** parameter. For more information, see "Basic Operation" on page 2-1126.

Treat sample-based row input as a column
: Select to treat sample-based length-M row vector inputs as M-by-1 column vectors. This parameter is only visible when the **Find the RMS value over** parameter is set to `Each column`.

Dimension
: Specify the dimension (one-based value) of the input signal, over which the RMS value is computed. The value of this parameter

cannot exceed the number of dimensions in the input signal. This parameter is only visible when the **Find the RMS value over** parameter is set to `Specified dimension`.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Boolean — The block accepts Boolean inputs to the `Rst` port.

See Also

Mean	Signal Processing Blockset
Variance	Signal Processing Blockset

Sample and Hold

Purpose

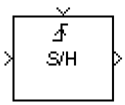
Sample and hold input signal

Library

Signal Operations

`dspsigops`

Description



The Sample and Hold block acquires the input at the signal port whenever it receives a trigger event at the trigger port (marked by). The block then holds the output at the acquired input value until the next triggering event occurs. When the acquired input is frame based, the output is frame based; otherwise, the output is sample based.

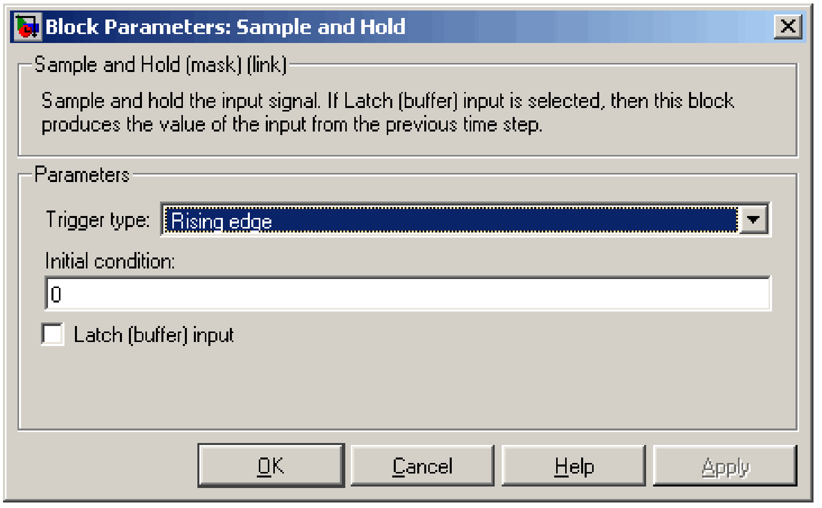
The trigger input must be a sample-based scalar with sample rate equal to the input frame rate at the signal port. You specify the trigger event in the **Trigger type** pop-up menu:

- `Rising edge` triggers the block to acquire the signal input when the trigger input rises from a negative value or zero to a positive value.
- `Falling edge` triggers the block to acquire the signal input when the trigger input falls from a positive value or zero to a negative value.
- `Either edge` triggers the block to acquire the signal input when the trigger input either rises from a negative value or zero to a positive value or falls from a positive value or zero to a negative value.

You specify the block's output prior to the first trigger event using the **Initial condition** parameter. When the acquired input is an M-by-N matrix, the **Initial condition** can be an M-by-N matrix, or a scalar to be repeated across all elements of the matrix. When the input is a length-M 1-D vector, the **Initial condition** can be a length-M row or column vector, or a scalar to be repeated across all elements of the vector.

If you select the **Latch (buffer) input** check box, the block outputs the value of the input from the previous time step until the next triggering event occurs. To use this block in a loop, select this check box.

Dialog Box



Trigger type
The type of event that triggers the block to acquire the input signal.

Initial condition
The block's output prior to the first trigger event.

Latch (buffer) input
If you select this check box, the block outputs the value of the input from the previous time step until the next triggering event occurs.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Trigger	• Any data type supported by the Trigger block
Outputs	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Downsample	Signal Processing Blockset
N-Sample Switch	Signal Processing Blockset

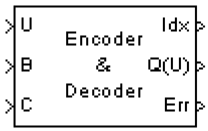
Purpose

Convert input signal into set of quantized output values or index values, or convert set of index values into quantized output signal

Library

`dspobslib`

Description



Scalar Quantizer

Note The Scalar Quantizer block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the Scalar Quantizer Encoder block or the Scalar Quantizer Decoder block.

The Scalar Quantizer block has three modes of operation. In `Encoder` mode, the block maps each input value to a quantization region by comparing the input value to the quantizer boundary points defined in the **Boundary points** parameter. The block outputs the index of the associated region. In `Decoder` mode, the block transforms the input index values into quantized output values, defined in the **Codebook** parameter. In the `Encoder and Decoder` mode, the block performs both the encoding and decoding operations. The block outputs the index values and the quantized output values.

You can select how you want to enter the **Boundary points** and/or **Codebook** values using the **Source of quantizer** parameters. When you select `Specify via dialog`, type the parameters into the block parameters dialog box. Select `Input ports`, and port B and/or C appears on the block. In `Encoder` and `Encoder and decoder` mode, the input to port B is used as the **Boundary points**. In `Decoder` and `Encoder and decoder` mode, the input to port C is used as the **Codebook**.

In `Encoder` and `Encoder and decoder` mode, the **Boundary points** are the values used to break up the input signal into regions. Each region is specified by an index number. When your first boundary point is `-inf` and your last boundary point is `inf`, your quantizer is unbounded. When your first and last boundary point is finite, your quantizer is bounded. When only your first or last boundary point is `-inf` or `inf`, your quantizer is semi-bounded.

For instance, when your input signal ranges from 0 to 11, you can create a bounded quantizer using the following boundary points:

```
[0 0.5 3.7 5.8 6.0 11]
```

The boundary points can have equal or varied spacing. Any input values between 0 and 0.5 would correspond to index 0. Input values between 0.5 and 3.7 would correspond to index 1, and so on.

Suppose you wanted to create an unbounded quantizer with the following boundary points:

```
[-inf 0 2 5.5 7.1 10 inf]
```

When your input signal has values less than 0, these values would be assigned to index 0. When your input signal has values greater than 10, these values would be assigned to index 6.

When an input value is the same as a boundary point, the **Tie-breaking rule** parameter defines the index to which the value is assigned. When you want the input value to be assigned to the lower index value, select `Choose the lower index`. To assign the input value with the higher index, select `Choose the higher index`.

In `Decoder` and `Encoder and decoder` mode, the **Codebook** is a vector of quantized output values that correspond to each index value.

In `Encoder` and `Encoder and decoder` mode, the **Searching method** determines how the appropriate quantizer index is found. Select `Linear` and the Scalar Quantizer block compares the input value to the first region defined by the first two boundary points. When the input value does not fall within this region, the block then compares the input value to the next region. This process continues until the input value is determined to be within a region and is associated with the appropriate index value. The computational cost of this process is of the order P, where P is the number of boundary points.

Select `Binary` for the **Searching method** and the block compares the input value to the middle value of the boundary points vector. When the input value is larger than this boundary point, the block discards the boundary points that are lower than this middle value. The block then compares the input value to the middle boundary point of the new range, defined by the remaining boundary points. This process continues until the input value is associated with the appropriate index value. The computational cost of this process is of the order $\log_2 P$, where P is the number of boundary points. In most cases, the `Binary` option is faster than the `Linear` option.

In `Decoder` mode, the input to this block is a vector of index values, where $0 \leq \text{index} < N$ and N is the length of the codebook vector. Use the **Action for out of range input** parameter to determine what happens when an input index value is out of this range. When you want any index values less than 0 to be set to 0 and any index values greater than or equal to N to be set to N -1, select `Clip`. When you want to be warned when any index values less than 0 are set to 0 and any index values greater than or equal to N are set to N -1, select `Clip and warn`. When you want the simulation to stop and display an error when the index values are out of range, select `Error`.

In `Encoder and decoder` mode, you can select the **Output the quantization error** check box. The quantization error is the difference between the input value and the quantized output value. Select this check box to output the quantization error for each input value from the Err port on this block.

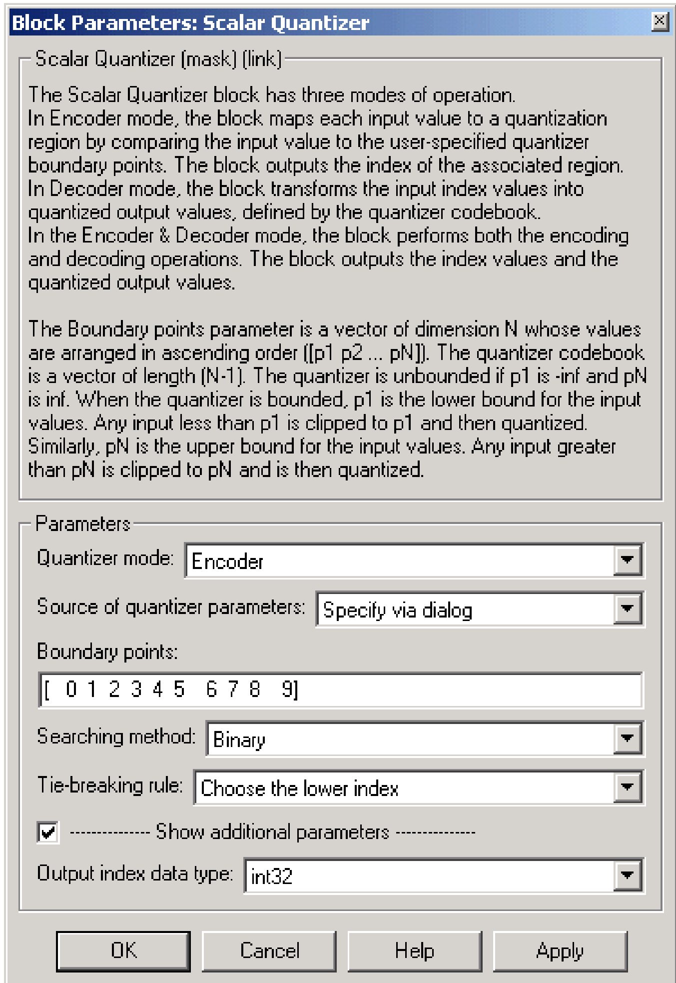
Data Type Support

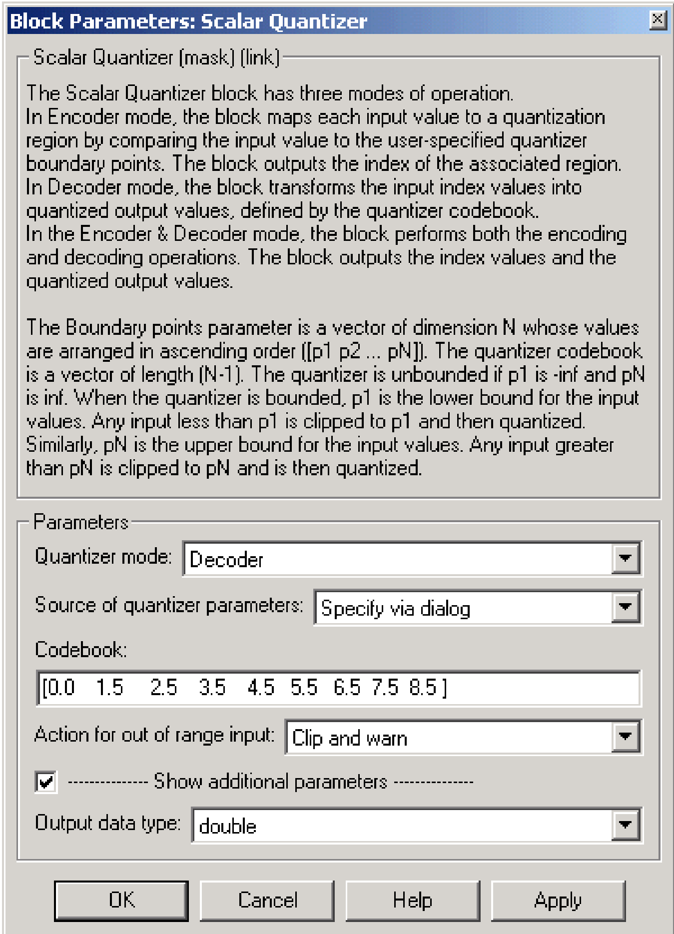
In `Encoder` mode, the input data values and the boundary points can be the input to the block at ports U and B. Similarly, in `Encoder and decoder` mode, the codebook values can also be the input to the block at port C. The data type of the input data values, boundary points, and codebook values can be `double`, `single`, `uint8`, `uint16`, `uint32`, `int8`, `int16`, or `int32`. In `Decoder` mode, the input to the block can be the index values and the codebook values. The data type of the index input to the block at port Idx can be `uint8`, `uint16`, `uint32`, `int8`, `int16`, or int32. The data type of the codebook values can be double, single, uint8, uint16, uint32, int8, int16, or int32.

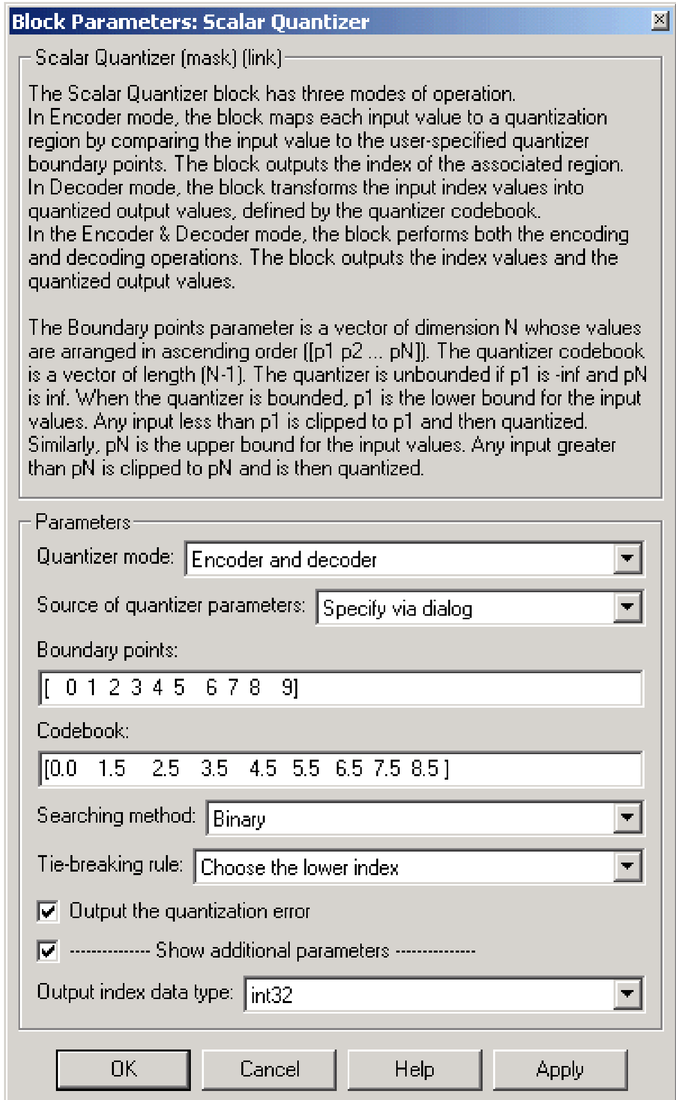
In Encoder mode, the output of the block is the index values. In Encoder and decoder mode, the output can also include the quantized output values and the quantization error. In Encoder and Encoder and decoder mode, use the **Output index data type** parameter to specify the data type of the index output from the block at port Idx. The data type of the index output can be uint8, uint16, uint32, int8, int16, or int32. The data type of the quantized output and the quantization error can be double, single, uint8, uint16, uint32, int8, int16, or int32. In Decoder mode, the output of the block is the quantized output values. Use the **Output data type** parameter to specify the data type of the quantized output values. The data type can be double, single, uint8, uint16, uint32, int8, int16, int32.

Note The input data, codebook values, boundary points, quantization error, and the quantized output values must have the same data type whenever they are present in any of the quantizer modes.

Dialog Box


Block Parameters: Scalar Quantizer
Scalar Quantizer (mask) (link)
The Scalar Quantizer block has three modes of operation.
In Encoder mode, the block maps each input value to a quantization region by comparing the input value to the user-specified quantizer boundary points. The block outputs the index of the associated region.
In Decoder mode, the block transforms the input index values into quantized output values, defined by the quantizer codebook.
In the Encoder & Decoder mode, the block performs both the encoding and decoding operations. The block outputs the index values and the quantized output values.
The Boundary points parameter is a vector of dimension N whose values are arranged in ascending order ([p1 p2 ... pN]). The quantizer codebook is a vector of length (N-1). The quantizer is unbounded if p1 is -inf and pN is inf. When the quantizer is bounded, p1 is the lower bound for the input values. Any input less than p1 is clipped to p1 and then quantized. Similarly, pN is the upper bound for the input values. Any input greater than pN is clipped to pN and is then quantized.
Parameters
Quantizer mode:
Encoder
Source of quantizer parameters:
Specify via dialog
Boundary points:
[0 1 2 3 4 5 6 7 8 9]
Searching method:
Binary
Tie-breaking rule:
Choose the lower index
-------------- Show additional parameters --------------
Output index data type:
int32
OK
Cancel
Help
Apply


Block Parameters: Scalar Quantizer
Scalar Quantizer (mask) (link)
The Scalar Quantizer block has three modes of operation.
In Encoder mode, the block maps each input value to a quantization region by comparing the input value to the user-specified quantizer boundary points. The block outputs the index of the associated region.
In Decoder mode, the block transforms the input index values into quantized output values, defined by the quantizer codebook.
In the Encoder & Decoder mode, the block performs both the encoding and decoding operations. The block outputs the index values and the quantized output values.
The Boundary points parameter is a vector of dimension N whose values are arranged in ascending order ([p1 p2 ... pN]). The quantizer codebook is a vector of length (N-1). The quantizer is unbounded if p1 is -inf and pN is inf. When the quantizer is bounded, p1 is the lower bound for the input values. Any input less than p1 is clipped to p1 and then quantized. Similarly, pN is the upper bound for the input values. Any input greater than pN is clipped to pN and is then quantized.
Parameters
Quantizer mode:
Decoder
Source of quantizer parameters:
Specify via dialog
Codebook:
[0.0 1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5]
Action for out of range input:
Clip and warn
------------- Show additional parameters -------------
Output data type:
double
OK
Cancel
Help
Apply


Block Parameters: Scalar Quantizer
Scalar Quantizer (mask) (link)
The Scalar Quantizer block has three modes of operation.
In Encoder mode, the block maps each input value to a quantization region by comparing the input value to the user-specified quantizer boundary points. The block outputs the index of the associated region.
In Decoder mode, the block transforms the input index values into quantized output values, defined by the quantizer codebook.
In the Encoder & Decoder mode, the block performs both the encoding and decoding operations. The block outputs the index values and the quantized output values.
The Boundary points parameter is a vector of dimension N whose values are arranged in ascending order ([p1 p2 ... pN]). The quantizer codebook is a vector of length (N-1). The quantizer is unbounded if p1 is -inf and pN is inf. When the quantizer is bounded, p1 is the lower bound for the input values. Any input less than p1 is clipped to p1 and then quantized. Similarly, pN is the upper bound for the input values. Any input greater than pN is clipped to pN and is then quantized.
Parameters
Quantizer mode:
Encoder and decoder
Source of quantizer parameters:
Specify via dialog
Boundary points:
[0 1 2 3 4 5 6 7 8 9]
Codebook:
[0.0 1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5]
Searching method:
Binary
Tie-breaking rule:
Choose the lower index
Output the quantization error
-------------- Show additional parameters --------------
Output index data type:
int32
OK
Cancel
Help
Apply

Quantizer mode
Specify `Encoder`, `Decoder`, or `Encoder and decoder` as a mode of operation.

Source of quantizer parameters
Choose `Specify via dialog` to type the parameters into the block parameters dialog box. Select `Input ports` to specify the parameters using the block's input ports. In `Encoder` and `Encoder and decoder` mode, input the **Boundary points** using port B. In `Decoder` and `Encoder and decoder` mode, input the **Codebook** values using port C.

Boundary points
Enter a vector of values that represent the boundary points of the quantizer regions. Tunable.

Codebook
Enter a vector of quantized output values that correspond to each index value. Tunable.

Searching method
Select `Linear` and the block finds the region in which the input value is located using a linear search. Select `Binary` and the block finds the region in which the input value is located using a binary search.

Tie-breaking rule
Set this parameter to determine the behavior of the block when the input value is the same as the boundary point. When you select `Choose the lower index`, the input value is assigned to lower index value. When you select `Choose the higher index`, the value is assigned to the higher index.

Action for out of range input
Choose the block's behavior when an input index value is out of range, where $0 \leq \text{index} < N$ and N is the length of the codebook vector. Select `Clip`, when you want any index values less than 0 to be set to 0 and any index values greater than or equal to N to be set to N-1. Select `Clip and warn`, when you want to be warned when any index values less than 0 are set to 0 and any

index values greater than or equal to N are set to N -1. Select `Error`, when you want the simulation to stop and display an error when the index values are out of range.

Output the quantization error
In `Encoder and decoder` mode, select this check box to output the quantization error from the Err port on this block.

Output index data type
In `Encoder` and `Encoder and decoder` mode, specify the data type of the index output from the block at port Idx. The data type can be `uint8`, `uint16`, `uint32`, `int8`, `int16`, or `int32`. This parameter becomes visible when you select the **Show additional parameters** check box.

Output data type
In `Decoder` mode, specify the data type of the quantized output. The data type can be `uint8`, `uint16`, `uint32`, `int8`, `int16`, `int32`, `single`, or `double`. This parameter becomes visible when you select `Specify via dialog` for the **Source of quantizer parameters** and you select the **Show additional parameters** check box.

References

Gersho, A. and R. Gray. *Vector Quantization and Signal Compression*. Boston: Kluwer Academic Publishers, 1992.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers

For more information on what data types are supported for each quantizer mode, see “Data Type Support” on page 2-1139.

See Also

Quantizer	Simulink
Scalar Quantizer Decoder	Signal Processing Blockset
Scalar Quantizer Design	Signal Processing Blockset
Scalar Quantizer Encoder	Signal Processing Blockset
Uniform Encoder	Signal Processing Blockset
Uniform Decoder	Signal Processing Blockset

Purpose

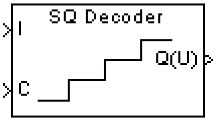
Convert each index value into quantized output value

Library

Quantizers

`dspquant2`

Description



Scalar Quantizer Decoder

The Scalar Quantizer Decoder block transforms the zero-based input index values into quantized output values. The set of all possible quantized output values is defined by the **Codebook values** parameter.

Use the **Codebook values** parameter to specify a matrix containing all possible quantized output values. You can select how you want to enter the codebook values using the **Source of codebook** parameter. When you select `Specify via dialog`, type the codebook values into the block parameters dialog box. When you select `Input port`, port C appears on the block. The block uses the input to port C as the **Codebook values** parameter.

The input to this block is a vector of integer index values, where 0 ≤ index <*N* and *N* is the number of distinct codeword vectors in the codebook matrix. Use the **Action for out of range index value** parameter to determine what happens when an input index value is outside this range. When you want any index value less than 0 to be set to 0 and any index value greater than or equal to *N* to be set to *N* -1, select `Clip`. When you want to be warned when clipping occurs, select `Clip and warn`. When you want the simulation to stop and the block to display an error when the index values are out of range, select `Error`.

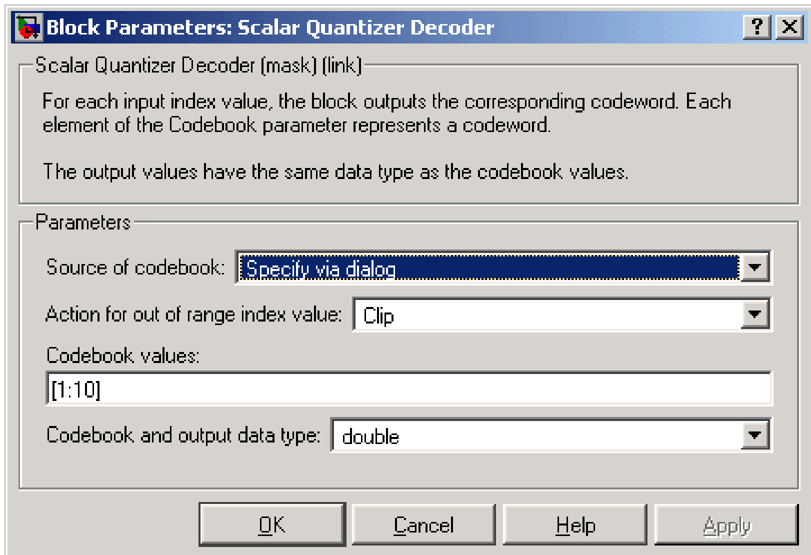
Data Type Support

The data type of the index values input at port I can be `uint8`, `uint16`, `uint32`, `int8`, `int16`, or `int32`. The data type of the codebook values input at port C can be `double`, `single`, or Fixed-point.

The output of the block is the quantized output values. If, for the **Source of codebook** parameter, you select `Specify via dialog`, the **Codebook and output data type** parameter appears. You can use this parameter to specify the data type of the codebook and quantized output values. In this case, the data type of the output values can be `Same as input`, `double`, `single`, `Fixed-point`, or `User-defined`.

If, for the **Source of codebook** parameter you select `Input port`, the quantized output values have the same data type as the codebook values input at port C.

Dialog Box



Source of codebook

Choose `Specify via dialog` to type the codebook values into the block parameters dialog box. Select `Input port` to specify the codebook using input port C.

Action for out of range index value

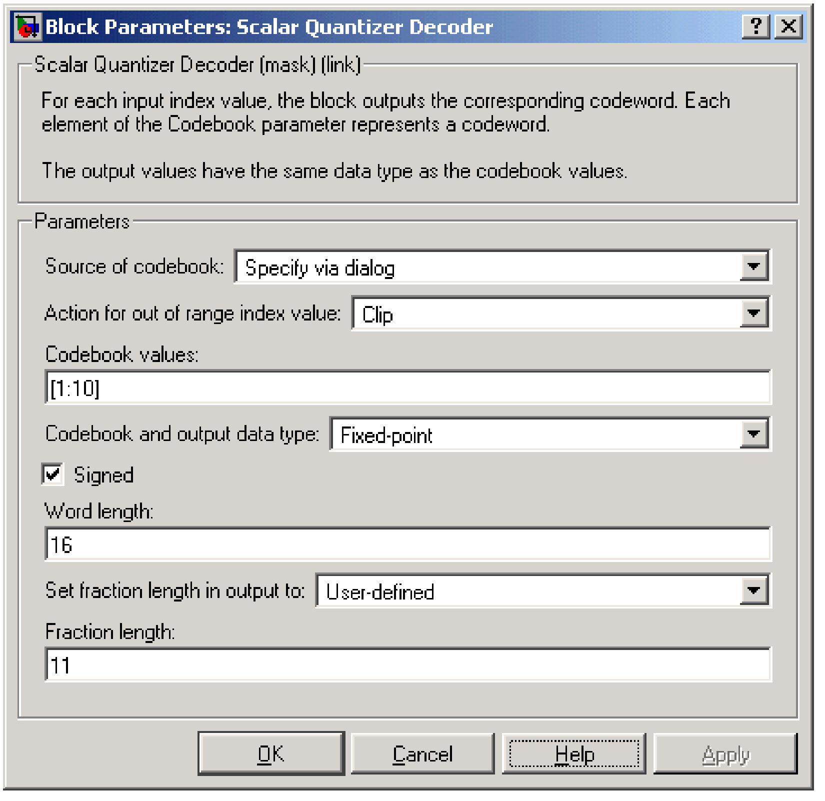
Use this parameter to determine the block's behavior when an input index value is out of range, where $0 \leq \text{index} < N$ and N is the length of the codebook vector. Select `Clip`, when you want any index values less than 0 to be set to 0 and any index values greater than or equal to N to be set to N -1. Select `Clip and warn`, when you want to be warned when clipping occurs. Select `Error`, when you want the simulation to stop and the block to display an error when the index values are outside the range.

Codebook values

Enter a vector of quantized output values that correspond to each index value. Tunable.

Codebook and output data type

Use this parameter to specify the data type of the codebook and quantized output values. The data type can be `Same as input`, `double`, `single`, `Fixed-point`, or `User-defined`. This parameter becomes visible when you select `Specify via dialog` for the **Source of codebook** parameter.



Signed

Select to output a signed fixed-point signal. Otherwise, the signal is unsigned. This parameter is only visible if, from the **Codebook and output data type** list, you select `Fixed-point`.

Word length

Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible if, from the **Codebook and output data type** list, you select `Fixed-point`.

Set fraction length in output to

Specify the scaling of the fixed-point output by either of the following two methods:

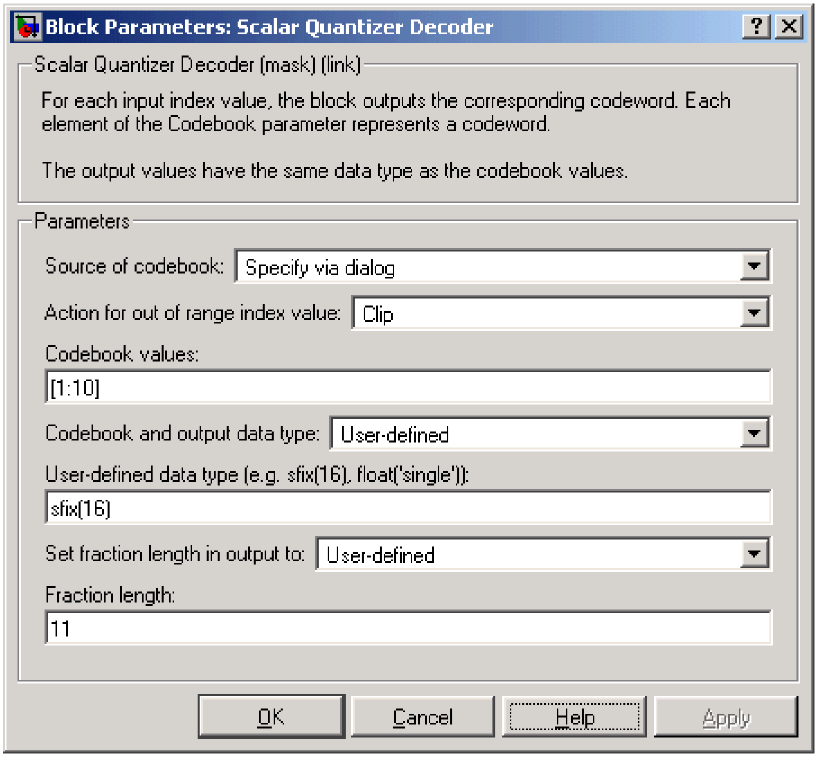
Choose `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.

Choose `User-defined` to specify the output scaling in the **Fraction length** parameter.

This parameter is only visible if, from the **Codebook and output data type** list, you select `Fixed-point` or when you select `User-defined` and the specified output data type is a fixed-point data type.

Fraction length

For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Codebook and output data type** parameter and `User-defined` for the **Set fraction length in output to** parameter.



User-defined data type

Specify any built-in or fixed-point data type. You can specify fixed-point data types using the following Simulink Fixed Point functions: `sfix`, `ufix`, `sint`, `uint`, `sfrac`, and `ufrac`. This parameter is only visible when you select `User-defined` for the **Codebook and output data type** parameter.

References

Gersho, A. and R. Gray. *Vector Quantization and Signal Compression*. Boston: Kluwer Academic Publishers, 1992.

Supported Data Types

Port	Supported Data Types
I	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
C	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers
Q(U)	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

For more information on what data types are supported for each quantizer mode, see "Data Type Support" on page 2-1147.

See Also

Quantizer	Simulink
Scalar Quantizer Design	Signal Processing Blockset
Scalar Quantizer Encoder	Signal Processing Blockset
Uniform Encoder	Signal Processing Blockset
Uniform Decoder	Signal Processing Blockset

Purpose

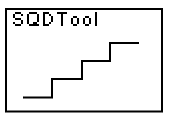
Start Scalar Quantizer Design Tool (SQDTool) to design scalar quantizer using Lloyd algorithm

Library

Quantizers

`dspquant2`

Description



Scalar Quantizer Design

Double-click on the Scalar Quantizer Design block to start SQDTool, a GUI that allows you to design and implement a scalar quantizer. Based on your input values, SQDTool iteratively calculates the codebook values that minimize the mean squared error until the stopping criteria for the design process is satisfied. The block uses the resulting quantizer codebook values and boundary points to implement your scalar quantizer encoder and/or decoder.

For the **Training Set** parameter, enter a set of observations, or samples, of the signal you want to quantize. This data can be any variable defined in the MATLAB workspace including a variable created using a MATLAB function, such as the default value `randn(10000,1)`.

You have two choices for the **Source of initial codebook** parameter. Select `Auto-generate` to have the block choose the values of the initial codebook vector. In this case, the minimum training set value becomes the first codeword, and the maximum training set value becomes the last codeword. Then, the remaining initial codewords are equally spaced between these two values to form a codebook vector of length N, where N is the **Number of levels** parameter. When you select `User defined`, enter the initial codebook values in the **Initial codebook** field. Then, set the **Source of initial boundary points** parameter. You can select `Mid-points` to locate the boundary points at the midpoint between the codewords. To calculate the mid-points, the block internally arranges the initial codebook values in ascending order. You can also choose `User defined` and enter your own boundary points in the **Initial boundary points (unbounded)** field. Only one boundary point can be located between two codewords. When you select `User defined` for the **Source of initial boundary points** parameter, the values you enter in the **Initial codebook** and **Initial boundary points (unbounded)** fields must be arranged in ascending order.

Note This block assumes that you are designing an unbounded quantizer. Therefore, the first and last boundary points are always `-inf` and `inf` regardless of any other boundary point values you might enter.

After you have specified the quantization parameters, the block performs an iterative process to design the optimal scalar quantizer. Each step of the design process involves using the Lloyd algorithm to calculate codebook values and quantizer boundary points. Then, the block calculates the squared quantization error and checks whether the stopping criteria has been satisfied.

The two possible options for the **Stopping criteria** parameter are `Relative threshold` and `Maximum iteration`. When you want the design process to stop when the fractional drop in the squared quantization error is below a certain value, select `Relative threshold`. Then, for **Relative threshold**, type the maximum acceptable fractional drop. When you want the design process to stop after a certain number of iterations, choose `Maximum iteration`. Then, enter the maximum number of iterations you want the block to perform in the **Maximum iteration** field. For **Stopping criteria**, you can also choose `Whichever comes first` and enter a **Relative threshold** and **Maximum iteration** value. The block stops iterating as soon as one of these conditions is satisfied.

With each iteration, the block quantizes the training set values based on the newly calculated codebook values and boundary points. When the training point lies on a boundary point, the algorithm uses the **Tie-breaking rules** parameter to determine which region the value is associated with. When you want the training point to be assigned to the lower indexed region, select `Lower indexed codeword`. To assign the training point with the higher indexed region, select `Higher indexed codeword`.

The **Searching methods** parameter determines how the block compares the training points to the boundary points. Select `Linear search` and SQDTool compares each training point to each quantization region sequentially. This process continues until all the training points are associated with the appropriate regions.

Select `Binary search` for the **Searching methods** parameter and the block compares the training point to the middle value of the boundary points vector. When the training point is larger than this boundary point, the block discards the lower boundary points. The block then compares the training point to the middle boundary point of the new range, defined by the remaining boundary points. This process continues until the training point is associated with the appropriate region.

Click **Design and Plot** to design the quantizer with the parameter values specified on the left side of the GUI. The performance curve and the staircase character of the quantizer are updated and displayed in the figures on the right side of the GUI.

Note You must click **Design and Plot** to apply any changes you make to the parameter values in the SQDTool dialog box.

SQDTool can export parameter values that correspond to the figures displayed in the GUI. Click the **Export Outputs** button, or press **Ctrl+E**, to export the **Final Codebook**, **Final Boundary Points**, and **Error** values to the workspace, a text file, or a MAT-file. The **Error** values represent the mean squared error for each iteration.

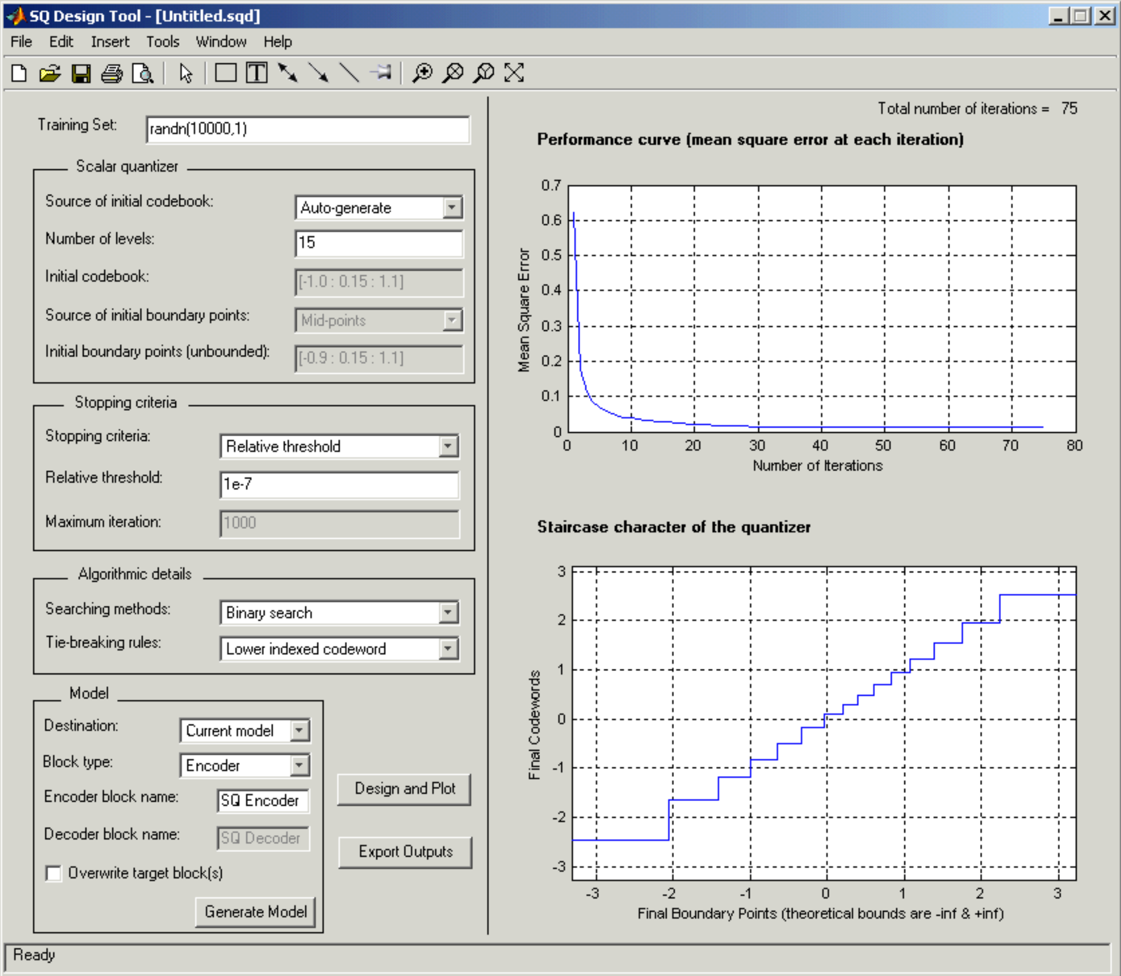
In the **Model** section of the GUI, specify the destination of the block that will contain the parameters of your quantizer. For **Destination**, select `Current model` to create a block with your parameters in the model you most recently selected. Type `gcs` in the MATLAB Command Window to display the name of your current model. Select `New model` to create a block in a new model file.

From the **Block type** list, select `Encoder` to design a Scalar Quantizer Encoder block. Select `Decoder` to design a Scalar Quantizer Decoder block. Select `Both` to design a Scalar Quantizer Encoder block and a Scalar Quantizer Decoder block.

In the **Encoder block name** field, enter a name for the Scalar Quantizer Encoder block. In the **Decoder block name** field, enter a name for the Scalar Quantizer Decoder block. When you have a Scalar Quantizer Encoder and/or Decoder block in your destination model with the same name, select the **Overwrite target block(s)** check box to replace the block's parameters with the current parameters. When you do not select this check box, a new Scalar Quantizer Encoder and/or Decoder block is created in your destination model.

Click **Generate Model**. SQDTool uses the parameters that correspond to the current plots to set the parameters of the Scalar Quantizer Encoder and/or Decoder blocks.

Dialog Box


SQ Design Tool - [Untitled.sqd]
File Edit Insert Tools Window Help
Training Set: randn(10000,1)
Scalar quantizer
Source of initial codebook: Auto-generate
Number of levels: 15
Initial codebook: [-1.0 : 0.15 : 1.1]
Source of initial boundary points: Mid-points
Initial boundary points (unbounded): [-0.9 : 0.15 : 1.1]
Stopping criteria
Stopping criteria: Relative threshold
Relative threshold: 1e-7
Maximum iteration: 1000
Algorithmic details
Searching methods: Binary search
Tie-breaking rules: Lower indexed codeword
Model
Destination: Current model
Block type: Encoder
Encoder block name: SQ Encoder
Decoder block name: SQ Decoder
Overwrite target block(s)
Generate Model
Design and Plot
Export Outputs
Total number of iterations = 75
Performance curve (mean square error at each iteration)
Mean Square Error
Number of Iterations
Staircase character of the quantizer
Final Codewords
Final Boundary Points (theoretical bounds are -inf & +inf)
Ready

Training Set

Enter the samples of the signal you would like to quantize. This data set can be a MATLAB function or a variable defined in the MATLAB workspace. The typical length of this data vector is `1e6`.

Source of initial codebook

Select `Auto-generate` to have the block choose the initial codebook values. Select `User defined` to enter your own initial codebook values.

Number of levels

Enter the length of the codebook vector. For a b-bit quantizer, the length should be $N = 2^b$.

Initial codebook

Enter your initial codebook values. From the **Source of initial codebook** list, select `User defined` in order to activate this parameter.

Source of initial boundary points

Select `Mid-points` to locate the boundary points at the midpoint between the codebook values. Choose `User defined` to enter your own boundary points. From the **Source of initial codebook** list, select `User defined` in order to activate this parameter.

Initial boundary points (unbounded)

Enter your initial boundary points. This block assumes that you are designing an unbounded quantizer. Therefore, the first and last boundary point are `-inf` and `inf`, regardless of any other boundary point values you might enter. From the **Source of initial boundary points** list, select `User defined` in order to activate this parameter.

Stopping criteria

Choose `Relative threshold` to enter the maximum acceptable fractional drop in the squared quantization error. Choose `Maximum iteration` to specify the number of iterations at which to stop. Choose `Whichever comes first` and the block stops the iteration process as soon as the relative threshold or maximum iteration value is attained.

Relative threshold
Type the value that is the maximum acceptable fractional drop in the squared quantization error.

Maximum iteration
Enter the maximum number of iterations you want the block to perform. From the **Stopping criteria** list, select `Maximum iteration` in order to activate this parameter.

Searching methods
Choose `Linear search` to use a linear search method when comparing the training points to the boundary points. Choose `Binary search` to use a binary search method when comparing the training points to the boundary points.

Tie-breaking rules
When a training point lies on a boundary point, choose `Lower indexed codeword` to assign the training point to the lower indexed quantization region. Choose `Higher indexed codeword` to assign the training point to the higher indexed region.

Design and Plot
Click this button to display the performance curve and the staircase character of the quantizer in the figures on the right side of the GUI. These plots are based on the current parameter settings.

You must click **Design and Plot** to apply any changes you make to the parameter values in the SQDTool GUI.

Export Outputs
Click this button, or press **Ctrl+E**, to export the **Final Codebook**, **Final Boundary Points**, and **Error** values to the workspace, a text file, or a MAT-file.

Destination
Choose `Current model` to create a Scalar Quantizer block in the model you most recently selected. Type `gcs` in the MATLAB Command Window to display the name of your current model. Choose `New model` to create a block in a new model file.

Block type
Select `Encoder` to design a Scalar Quantizer Encoder block. Select `Decoder` to design a Scalar Quantizer Decoder block. Select `Both` to design a Scalar Quantizer Encoder block and a Scalar Quantizer Decoder block.

Encoder block name
Enter a name for the Scalar Quantizer Encoder block.

Decoder block name
Enter a name for the Scalar Quantizer Decoder block.

Overwrite target block(s)
When you do not select this check box and a Scalar Quantizer Encoder and/or Decoder block with the same block name exists in the destination model, a new Scalar Quantizer Encoder and/or Decoder block is created in the destination model. When you select this check box and a Scalar Quantizer Encoder and/or Decoder block with the same block name exists in the destination model, the parameters of these blocks are overwritten by new parameters.

Generate Model
Click this button and SQDTool uses the parameters that correspond to the current plots to set the parameters of the Scalar Quantizer Encoder and/or Decoder blocks.

References

Gersho, A. and R. Gray. *Vector Quantization and Signal Compression*. Boston: Kluwer Academic Publishers, 1992.

Supported Data Types

- Double-precision floating point

See Also

Quantizer	Simulink
Scalar Quantizer Decoder	Signal Processing Blockset

Scalar Quantizer Design

Scalar Quantizer Encoder	Signal Processing Blockset
Uniform Encoder	Signal Processing Blockset
Uniform Decoder	Signal Processing Blockset

Scalar Quantizer Encoder

Purpose

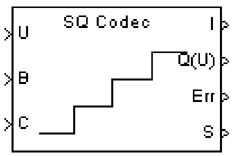
Encode each input value by associating it with index value of quantization region

Library

Quantizers

`dspquant2`

Description



Scalar Quantizer Encoder

The Scalar Quantizer Encoder block maps each input value to a quantization region by comparing the input value to the quantizer boundary points defined in the **Boundary points** parameter. The block outputs the zero-based index of the associated region.

You can select how you want to enter the **Boundary points** using the **Source of quantizer parameters**. When you select `Specify via dialog`, type the boundary points into the block parameters dialog box. When you select `Input port`, port B appears on the block. The block uses the input to port B as the **Boundary points** parameter.

Use the **Boundary points** parameter to specify the boundary points for your quantizer. These values are used to break up the set of input numbers into regions. Each region is specified by an index number.

Let `N` be the number of quantization regions. When the codebook is defined as `[c1 c2 c3 ...  cN]`, and the **Boundary points** parameter is defined as `[p0 p1 p2 p3 ...  pN]`, then `p0<c1<p1<c2 ...  p(N-1)<cN<pN` for a regular quantizer. When your quantizer is bounded, from the **Partitioning** list, select `Bounded`. You need to specify N+1 boundary points, or `[p0 p1 p2 p3 ...  pN]`. When your quantizer is unbounded, from the **Partitioning** list, select `Unbounded`. You need to specify N-1 boundary points, or`[p1 p2 p3 ...  p(N-1)]`; the block sets `p0` equal to `-inf` and `pN` equal to `inf`.

The block uses the **Partitioning** parameter to interpret the boundary points you enter. For instance, to create a bounded quantizer, from the **Partitioning** list, select `Bounded` and enter the following boundary points:

```
[0 0.5 3.7 5.8 6.0 11]
```

The block assigns any input values between 0 and 0.5 to index 0, input values between 0.5 and 3.7 to index 1, and so on. The block assigns any values that are less than 0 to index 0, the lowest index value. The block assigns any values that are greater than 11 to index 4, the highest index value.

To create an unbounded quantizer, from the **Partitioning** list, select `Unbounded` and enter the following boundary points:

```
[0 0.5 3.7 5.8 6.0 11]
```

The block assigns any input values between 0 and 0.5 to index 1, input values between 0.5 and 3.7 to index 2, and so on. The block assigns any input values less than 0 to index 0 and any values greater than 11 to index 6.

The **Searching method** parameter determines how the appropriate quantizer index is found. When you select `Linear`, the Scalar Quantizer Encoder block compares the input value to the first region defined by the first two boundary points. When the input value does not fall within this region, the block then compares the input value to the next region. This process continues until the input value is determined to be within a region and is associated with the appropriate index value. The computational cost of this process is of the order P, where P is the number of boundary points.

When you select `Binary` for the **Searching method**, the block compares the input value to the middle value of the boundary points vector. When the input value is larger than this boundary point, the block discards the boundary points that are lower than this middle value. The block then compares the input value to the middle boundary point of the new range, defined by the remaining boundary points. This process continues until the input value is associated with the appropriate index value. The computational cost of this process is of the order $\log_2 P$, where P is the number of boundary points. In most cases, the `Binary` option is faster than the `Linear` option.

When an input value is the same as a boundary point, the **Tie-breaking rule** parameter determines the region to which the value is assigned.

When you want the input value to be assigned to the lower indexed region, select `Choose the lower index`. To assign the input value with the higher indexed region, select `Choose the higher index`.

Select the **Output codeword** check box to output the codeword values that correspond to each index value at port Q(U).

Select the **Output the quantization error** check box to output the quantization error for each input value from the Err port on this block. The quantization error is the difference between the input value and the quantized output value.

When you select either the **Output codeword** check box or the **Output quantization error** check box, you must also enter your codebook values. If, from the **Source of quantizer parameters** list, you choose `Specify via dialog`, use the **Codebook** parameter to enter a vector of quantized output values that correspond to each region. If, from the **Source of quantizer parameters** list, you choose `Input port`, use input port C to specify your codebook values.

If, for the **Partitioning** parameter, you select `Bounded`, the **Output clipping status** check box and the **Action for out of range input** parameter appear. When you select the **Output clipping status** check box, port S appears on the block. Any time an input value is outside the range defined by the **Boundary points** parameter, the block outputs a 1 at the S port. When the value is inside the range, the blocks outputs a 0.

You can use the **Action for out of range input** parameter to determine the block's behavior when an input value is outside the range defined by the **Boundary points** parameter. Suppose the boundary points for a bounded quantizer are defined as `[p0 p1 p2 p3 ...  pN]` and the possible index values are defined as `[i0 i1 i2 ...  i(N-1)]`, where `i0=0` and `i0<i1<i2<...<i(N-1)`. When you want any input value less than `p0` to be assigned to index value `i0` and any input values greater than `pN` to be assigned to index value `i(N-1)`, select `Clip`. When you want to be warned when clipping occurs, select `Clip and warn`. When you want the simulation to stop and the block to display an error when the index values are out of range, select `Error`.

The Scalar Quantizer Encoder block accepts real floating-point and fixed-point inputs. For more information on the data types accepted by each port, see "Data Type Support" on page 2-1165 or "Supported Data Types" on page 2-1171.

Data Type Support

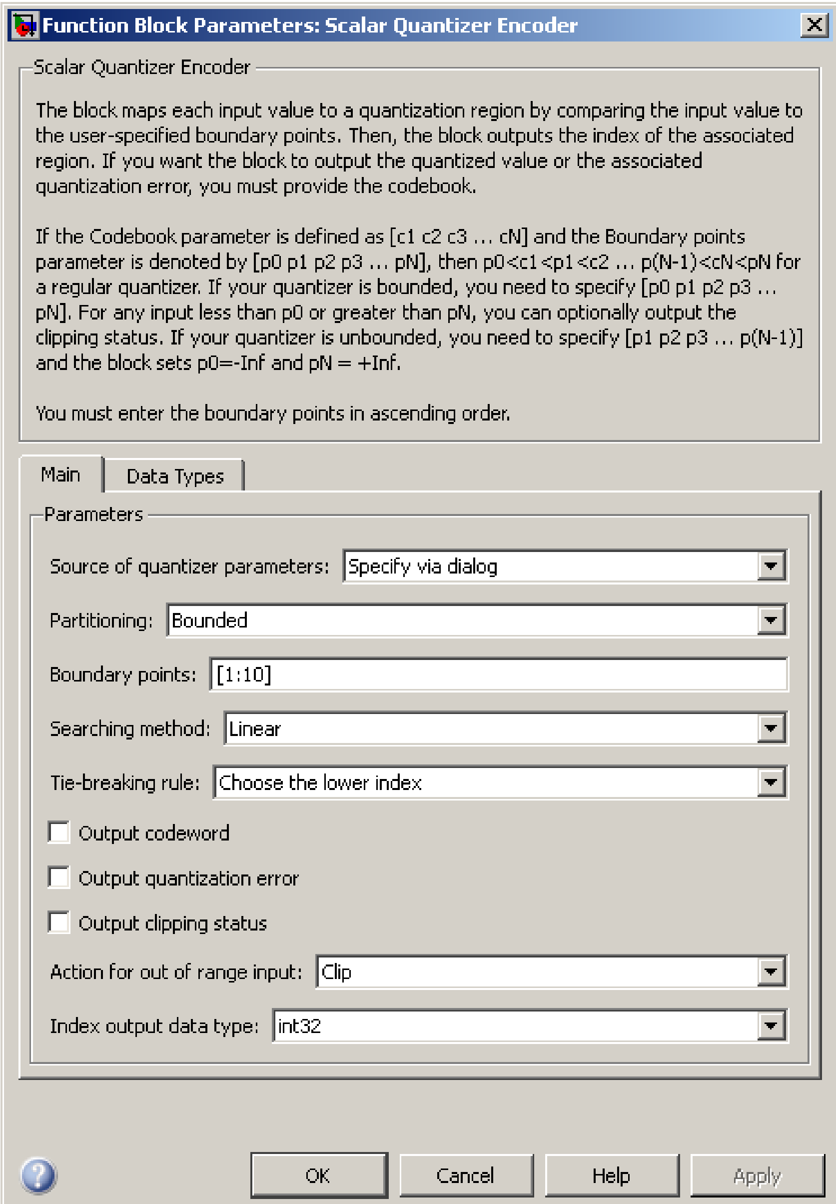
The input data values, boundary points, and codebook values can be input to the block at ports U, B, and C, respectively. The data type of the inputs can be `double`, `single`, or Fixed-point.

The outputs of the block can be the index values, the quantized output values, the quantization error, and the clipping status. Use the **Index output data type** parameter to specify the data type of the index output from the block at port I. You can choose `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`. The data type of the quantized output and the quantization error can be `double`, `single`, or Fixed-point. The clipping status values output at port S are Boolean values.

Note The input data, boundary points, codebook values, quantized output values, and the quantization error must have the same data type whenever they are present.

Dialog Box

The **Main** pane of the Scalar Quantizer Encoder block dialog appears as follows.


Function Block Parameters: Scalar Quantizer Encoder
Scalar Quantizer Encoder
The block maps each input value to a quantization region by comparing the input value to the user-specified boundary points. Then, the block outputs the index of the associated region. If you want the block to output the quantized value or the associated quantization error, you must provide the codebook.
If the Codebook parameter is defined as [c1 c2 c3 ... cN] and the Boundary points parameter is denoted by [p0 p1 p2 p3 ... pN], then p0<c1<p1<c2 ... p(N-1)<cN<pN for a regular quantizer. If your quantizer is bounded, you need to specify [p0 p1 p2 p3 ... pN]. For any input less than p0 or greater than pN, you can optionally output the clipping status. If your quantizer is unbounded, you need to specify [p1 p2 p3 ... p(N-1)] and the block sets p0=-Inf and pN = +Inf.
You must enter the boundary points in ascending order.
Main
Data Types
Parameters
Source of quantizer parameters:
Specify via dialog
Partitioning:
Bounded
Boundary points:
[1:10]
Searching method:
Linear
Tie-breaking rule:
Choose the lower index
Output codeword
Output quantization error
Output clipping status
Action for out of range input:
Clip
Index output data type:
int32
OK
Cancel
Help
Apply

Source of quantizer parameters
Choose `Specify via dialog` to enter the boundary points and codebook values using the block parameters dialog box. Select `Input port` to specify the parameters using the block's input ports. Input the boundary points and codebook values using ports B and C, respectively.

Partitioning
When your quantizer is bounded, select `Bounded`. When your quantizer is unbounded, select `Unbounded`.

Boundary points
Enter a vector of values that represent the boundary points of the quantizer regions. This parameter is visible when you select `Specify via dialog` from the **Source of quantizer parameters** list. Tunable.

Searching method
When you select `Linear`, the block finds the region in which the input value is located using a linear search. When you select `Binary`, the block finds the region in which the input value is located using a binary search.

Tie-breaking rule
Set this parameter to determine the behavior of the block when the input value is the same as the boundary point. When you select `Choose the lower index`, the input value is assigned to lower indexed region. When you select `Choose the higher index`, the value is assigned to the higher indexed region.

Output codeword
Select this check box to output the codeword values that correspond to each index value at port Q(U).

Output quantization error
Select this check box to output the quantization error for each input value at port Err.

Codebook

Enter a vector of quantized output values that correspond to each index value. If, for the **Partitioning** parameter, you select `Bounded` and your boundary points vector has length N, then you must specify a codebook of length N-1. If, for the **Partitioning** parameter, you select `Unbounded` and your boundary points vector has length N, then you must specify a codebook of length N+1.

This parameter is visible when you select `Specify via dialog` from the **Source of quantizer parameters** list and you select either the **Output codeword** or **Output quantization error** check box. Tunable.

Output clipping status

When you select this check box, port S appears on the block. Any time an input value is outside the range defined by the **Boundary points** parameter, the block outputs a 1 at this port. When the value is inside the range, the block outputs a 0. This parameter is visible when you select `Bounded` from the **Partitioning** list.

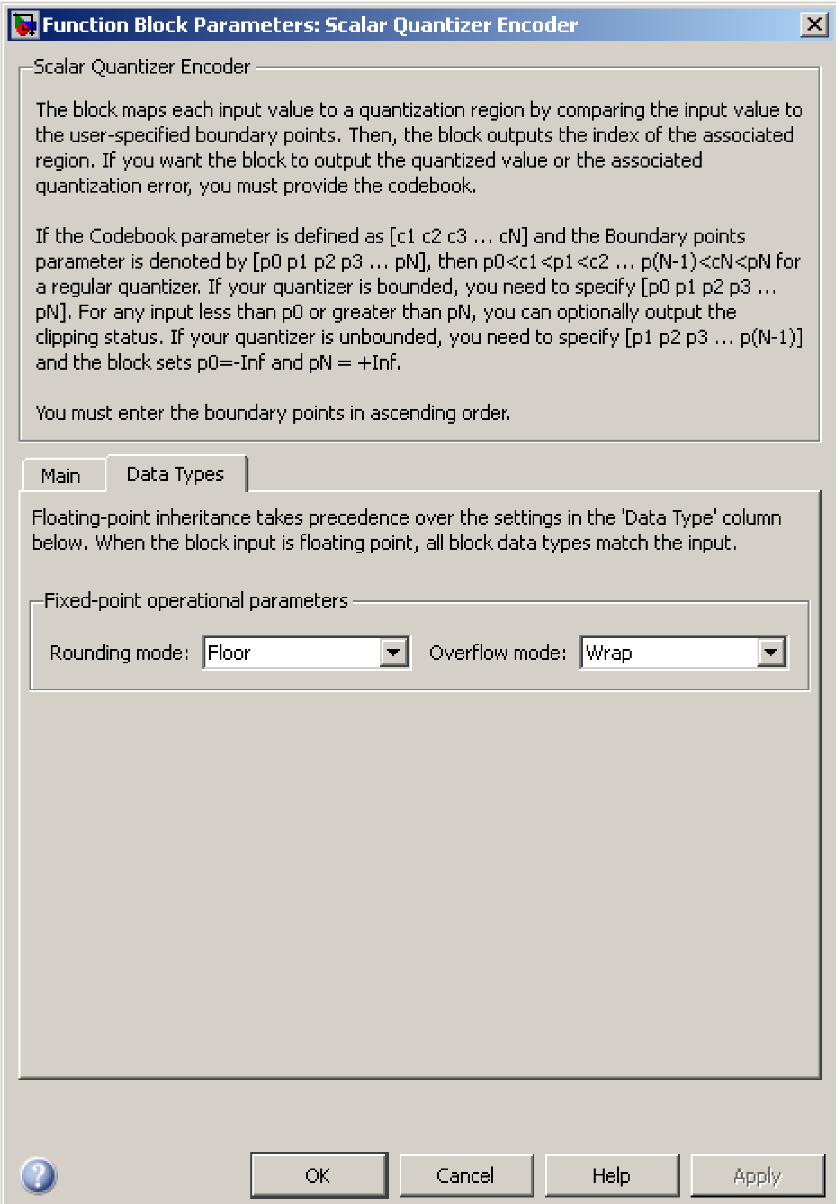
Action for out of range input

Use this parameter to determine the behavior of the block when an input value is outside the range defined by the **Boundary points** parameter. Suppose the boundary points are defined as `[p0 p1 p2 p3 ...  pN]` and the index values are defined as `[i0 i1 i2 ...  i(N-1)]`. When you want any input value less than `p0` to be assigned to index value `i0` and any input values greater than `pN` to be assigned to index value `i(N-1)`, select `Clip`. When you want to be warned when clipping occurs, select `Clip and warn`. When you want the simulation to stop and the block to display an error when the index values are out of range, select `Error`. This parameter is visible when you select `Bounded` from the **Partitioning** list.

Index output data type

Specify the data type of the index output from the block at port I. You can choose `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`.

The **Data Types** pane of the Scalar Quantizer Encoder block dialog appears as follows.


Function Block Parameters: Scalar Quantizer Encoder
Scalar Quantizer Encoder
The block maps each input value to a quantization region by comparing the input value to the user-specified boundary points. Then, the block outputs the index of the associated region. If you want the block to output the quantized value or the associated quantization error, you must provide the codebook.
If the Codebook parameter is defined as [c1 c2 c3 ... cN] and the Boundary points parameter is denoted by [p0 p1 p2 p3 ... pN], then p0<c1<p1<c2 ... p(N-1)<cN<pN for a regular quantizer. If your quantizer is bounded, you need to specify [p0 p1 p2 p3 ... pN]. For any input less than p0 or greater than pN, you can optionally output the clipping status. If your quantizer is unbounded, you need to specify [p1 p2 p3 ... p(N-1)] and the block sets p0=-Inf and pN = +Inf.
You must enter the boundary points in ascending order.
Main
Data Types
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input.
Fixed-point operational parameters
Rounding mode:
Floor
Overflow mode:
Wrap
OK
Cancel
Help
Apply

Rounding mode
Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode to be used when block inputs are fixed point.

References

Gersho, A. and R. Gray. *Vector Quantization and Signal Compression*. Boston: Kluwer Academic Publishers, 1992.

Supported Data Types

Port	Supported Data Types
U	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
B	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
C	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
I	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
Q(U)	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
Err	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
S	• Boolean

For more information on what data types are supported for each quantizer mode, see "Data Type Support" on page 2-1147.

See Also

Quantizer	Simulink
Scalar Quantizer Decoder	Signal Processing Blockset
Scalar Quantizer Design	Signal Processing Blockset
Uniform Encoder	Signal Processing Blockset
Uniform Decoder	Signal Processing Blockset

Purpose

Select input elements from vector, matrix, or multidimensional signal

Library

Signal Management / Indexing

`dspindex`

Description

The Selector block is an implementation of the Simulink Selector block. See Selector for more information.

Short-Time FFT

Purpose

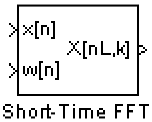
Compute nonparametric estimate of spectrum using short-time, fast Fourier transform (FFT) method

Library

Transforms

`dspxfrm3`

Description



The Short-Time FFT block computes a nonparametric estimate of the spectrum. The block buffers, applies a window, and zero pads the input signal. The block then takes the FFT of the signal, transforming it into the frequency domain.

Connect your sample-based or frame-based, single-channel analysis window to the w(n) port. For the **Analysis window length** parameter, enter the length of the analysis window, W. When your analysis window is a sample-based signal, the block buffers it into a frame-based signal with frame length W. When your analysis window is a frame-based signal and its frame length is not W, the block buffers the signal so that its frame length is W.

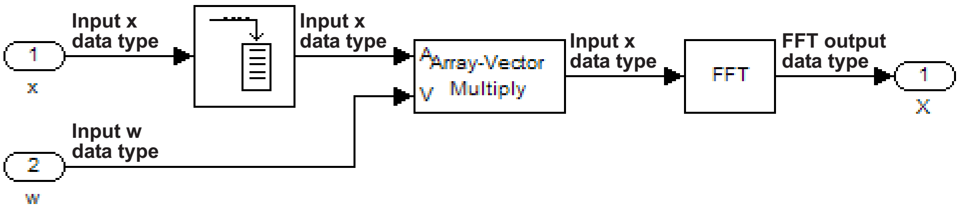
Connect your sample-based or frame-based, single-channel or multichannel input signal to the x(n) port. After the block buffers and windows this signal, it zero-pads the signal before computing the FFT. For the **FFT length** parameter, enter the length to which the block pads the input signal. For the **Overlap between consecutive windows (in samples)** parameter, enter the number of samples to overlap each frame of the input signal.

The complex-valued, sample-based, single-channel or multichannel short-time FFT is output at port X(n,k).

The Short-Time FFT block supports real and complex floating-point and fixed-point signals.

Fixed-Point Data Types

The following diagram shows the data types used within the Short-Time FFT subsystem block for fixed-point signals.



The settings for the fixed-point parameters of the Array-Vector Multiply block in the diagram above are as follows:

- **Rounding Mode** — `Floor`
- **Overflow Mode** — `Wrap`
- **Product output** — `Inherit via internal rule`
- **Accumulator** — `Inherit via internal rule`
- **Output** — `Same as first input`

The settings for the fixed-point parameters of the FFT block in the diagram above are as follows:

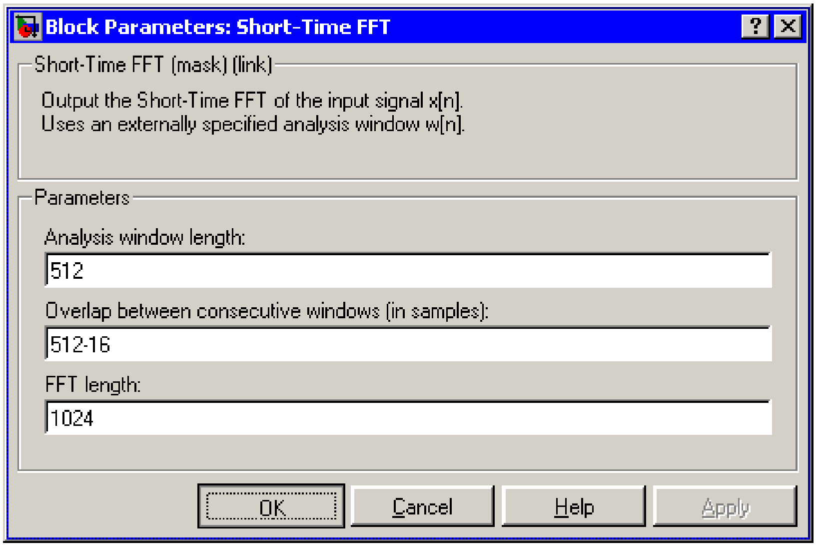
- **Rounding Mode** — `Floor`
- **Overflow Mode** — `Wrap`
- **Sine table** — `Same word length as input`
- **Product output** — `Inherit via internal rule`
- **Accumulator** — `Inherit via internal rule`
- **Output** — `Inherit via internal rule`

See the FFT and Array-Vector Multiply block reference pages for more information.

Examples

The dspstsa_win32 demo illustrates how to use the Short-Time FFT and Inverse Short-Time FFT blocks to remove the background noise from a speech signal.

Dialog Box



Analysis window length
Enter the frame length of the analysis window.

Overlap between consecutive windows (in samples)
Enter the number of samples of overlap for each frame of the input signal.

FFT length
Enter the length to which the block pads the input signal.

References

Quatieri, Thomas E. *Discrete-Time Speech Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 2001.

Supported Data Types

Port	Supported Data Types
x(n)	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
w(n)	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
X(n,k)	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

Burg Method	Signal Processing Blockset
Inverse Short-Time FFT	Signal Processing Blockset
Magnitude FFT	Signal Processing Blockset
Periodogram	Signal Processing Blockset
Spectrum Scope	Signal Processing Blockset
Window Function	Signal Processing Blockset
Yule-Walker Method	Signal Processing Blockset
`pwelch`	Signal Processing Toolbox

See "Power Spectrum Estimation" for related information.

Signal From Workspace

Purpose

Import signal from MATLAB workspace

Library

Signal Processing Sources

`dspsrcs4`

Description

The Signal From Workspace block imports a signal from the MATLAB workspace into the Simulink model. The **Signal** parameter specifies the name of a MATLAB workspace variable containing the signal to import, or any valid MATLAB expression defining a matrix or 3-D array.

When the **Signal** parameter specifies an M-by-N matrix (M≠1), each of the N columns is treated as a distinct channel. You specify the frame size in the **Samples per frame** parameter, M_o, and the output is an M_o-by-N matrix containing M_o consecutive samples from each signal channel. You specify the output sample period in the **Sample time** parameter, T_s, and the output frame period is M_o*T_s. For M_o=1, the output is sample based; otherwise the output is frame based. For convenience, an imported row vector (M=1) is treated as a single channel, so the output dimension is M_o-by-1.

When the **Signal** parameter specifies an M-by-N-by-P array, each of the P pages (an M-by-N matrix) is output in sequence with period T_s. The **Samples per frame** parameter must be set to `1`, and the output is always sample based.

Initial and Final Conditions

Unlike the Simulink From Workspace block, the Signal From Workspace block holds the output value constant between successive output frames (that is, no linear interpolation takes place). Additionally, the initial signal values are always produced immediately at *t*=0.

When the block has output all of the available signal samples, it can start again at the beginning of the signal, or simply repeat the final value or generate zeros until the end of the simulation. (The block does not extrapolate the imported signal beyond the last sample.) The **Form output after final data value by** parameter controls this behavior:

- When you specify `Setting To Zero`, the block generates zero-valued outputs for the duration of the simulation after generating the last frame of the signal.
- When you specify `Holding Final Value`, the block repeats the final sample for the duration of the simulation after generating the last frame of the signal.
- When you specify `Cyclic Repetition`, the block repeats the signal from the beginning after it reaches the last sample in the signal. If the frame size you specify in the **Samples per frame** parameter does not evenly divide the input length, a buffer block is inserted into the Signal From Workspace subsystem, and the model becomes multirate. If you do not want your model to become multirate, make sure the frame size evenly divides the input signal length.

  Select the **Warn when frame size does not evenly divide input length** parameter to be alerted when the input length is not an integer multiple of the frame size and your model will become multirate. Use the Model Explorer to turn these warnings on or off model-wide:

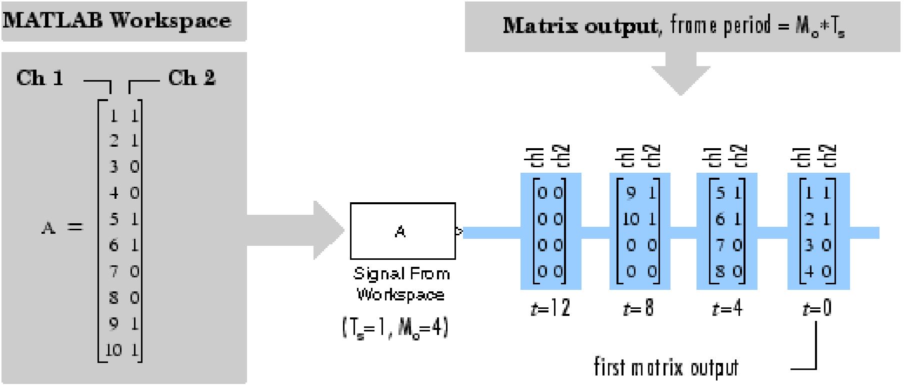
  1. Select **Model Explorer** from the **View** menu in your model.
  2. In the **Search** bar of the Model Explorer, search `by Property Name` for the `ignoreOrWarnInputAndFrameLengths` property. Each block with the **Warn when frame size does not evenly divide input length** check box appears in the list in the **Contents** pane.
  3. Select each of the blocks for which you wish to toggle the warning parameter, and select or deselect the check box in the `ignoreOrWarnInputAndFrameLengths` column.

Examples

Example 1

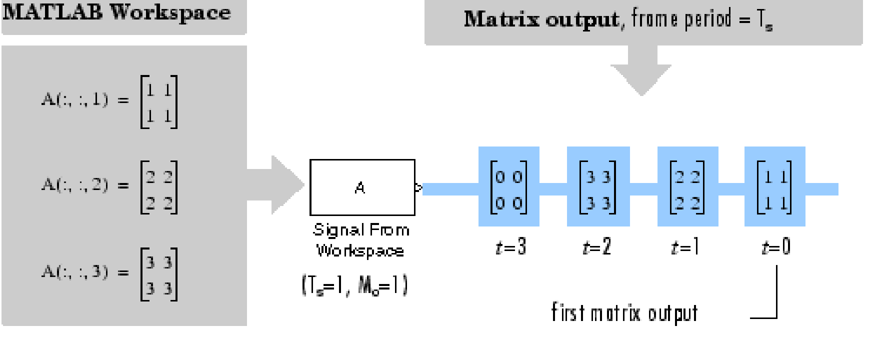
In the first model below, the Signal From Workspace imports a two-channel signal from the workspace matrix `A`. The **Sample time** is set to `1` and the **Samples per frame** is set to `4`, so the output is frame based with a frame size of 4 and a frame period of 4 seconds. The **Form**

output after final data value by parameter specifies `Setting To Zero`, so all outputs after the third frame (at t=8) are zero.



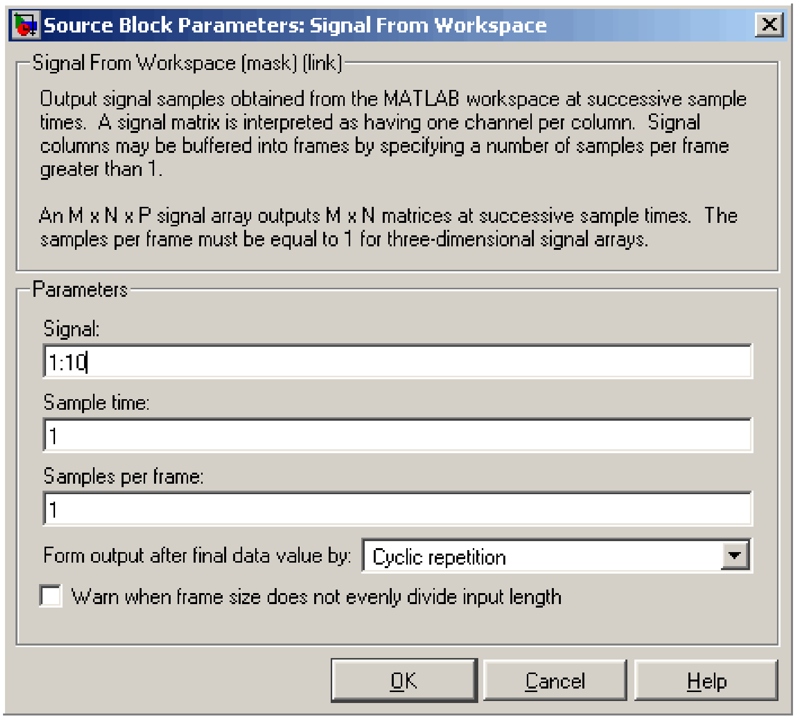
Example 2

In the second model below, the Signal From Workspace block imports a sample-based matrix signal from the 3-D workspace array `A`. Again, the **Form output after final data value by** parameter specifies `Setting To Zero`, so all outputs after the third (at t=2) are zero.



The **Samples per frame** parameter is set to `1` for 3-D input.

Dialog Box



Signal

The name of the MATLAB workspace variable from which to import the signal, or a valid MATLAB expression specifying the signal.

Sample time

The sample period, T_s, of the output. The output frame period is M_o*T_s.

Samples per frame

The number of samples, M_o, to buffer into each output frame. This value must be 1 when you specify a 3-D array in the **Signal** parameter.

Form output after final data value by

Specifies the output after all of the specified signal samples have been generated. The block can output zeros for the duration of the simulation (`Setting to zero`), repeat the final data sample (`Holding Final Value`) or repeat the entire signal from the beginning (`Cyclic Repetition`).

Warn when frame size does not evenly divide input length

Select this parameter to be alerted when the input length is not an integer multiple of the frame size and your model will become multirate. For more information, see "Initial and Final Conditions" on page 2-1179.

This parameter is only visible when `Cyclic Repetition` is selected for the **Form output after final data value by** parameter.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed and unsigned)
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers

See Also

From Audio Device	Signal Processing Blockset
From Wave File (Obsolete)	Signal Processing Blockset
Signal From Workspace	Signal Processing Blockset
From Workspace	Simulink
To Workspace	Simulink
Triggered Signal From Workspace	Signal Processing Blockset

See the sections below for related information:

- "Creating Sample-Based Signals"
- "Creating Frame-Based Signals"
- "Importing and Exporting Sample-Based Signals"
- "Importing and Exporting Frame-Based Signals"

Purpose

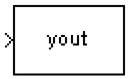
Write simulation data to array in MATLAB workspace

Library

Signal Processing Sinks

`dspsnks4`

Description



The Signal To Workspace block writes data from your simulation into an array in the MATLAB main workspace. The output array can be 2-D or 3-D, depending on whether the data is 1-D, sample based, or frame based. The Signal To Workspace block and the Simulink To Workspace block can output the same arrays when their parameters are set appropriately.

For more information on the Signal To Workspace block, see the following sections of this reference page:

- “Parameter Descriptions” on page 2-1185
- “Output Dimension Summary” on page 2-1187
- “Matching the Outputs of Signal To Workspace and To Workspace Blocks” on page 2-1187
- “Examples” on page 2-1188

Parameter Descriptions

The **Variable name** parameter is the name of the array in the MATLAB workspace into which the block logs the simulation data. The array is created in the workspace only after the simulation stops running. When you enter the name of an existing workspace variable, the block overwrites the variable with an array of simulation data after the simulation stops running.

When the block input is sample based or 1-D, the **Limit data points to last** parameter indicates how many *samples of data* to save. When the block input is frame based, this parameter indicates how many *frames of data* to save. When the simulation generates more than the specified maximum number of samples or frames, the simulation saves only the most recently generated data. To capture all data, set **Limit data points to last** to `inf`.

The **Decimation** parameter is the decimation factor. It can be set to any positive integer *d*, and allows you to write data at every *d*th sample. The default decimation, `1`, writes data at every time step.

The **Frames** parameter sets the dimension of the output array to 2-D or 3-D for frame-based inputs. The block ignores this parameter for 1-D and sample-based inputs. The **Frames** parameter has the following two settings:

- `Log frames separately (3-D array)`: Given an M-by-N frame-based input signal, the block outputs an M-by-N-by-K array, where K is the number of frames logged by the end of the simulation. (K is bounded above by the **Limit data points to last** parameter.) Each input frame is an element of the 3-D array. (See "Example 2: Frame-Based Inputs" on page 2-1189.)
- `Concatenate frames (2-D array)`: Given an M-by-N frame-based input signal with frame size *f*, the block outputs a (K**f*)-by-N matrix, where K**f* is the number of samples acquired by the end of the simulation. Each input frame is vertically concatenated to the previous frame to produce the 2-D array output. (See "Example 2: Frame-Based Inputs" on page 2-1189.)

Signal to Workspace always logs sample-based input data as 3-D arrays, regardless of the **Frame** parameter setting. Given an M-by-N sample-based signal, the block outputs an M-by-N-by-L array, where L is the number of samples logged by the end of the simulation (L is bounded above by the **Limit data points to last** parameter). Each sample-based matrix is an element of the 3-D array. (See "Example 1: Sample-Based Inputs" on page 2-1188.)

For 1-D vector inputs, the block outputs a 2-D matrix regardless of the setting of **Frame**. For a length-N 1-D vector input, the block outputs an L-by-N matrix. Each input vector is a row of the output matrix, vertically concatenated to the previous vector.

Output Dimension Summary

The following table summarizes the output array dimensions for various block inputs. In the table, *f* is the frame size of the input, K is the number of *frames* acquired by the end of the simulation, and L is the number of *samples* acquired by the end of the simulation (K and L are bounded above by the **Limit data points to last** parameter).

Input Signal Type	Signal To Workspace Output Dimension
Sample-based M-by-N matrix	M-by-N-by-L array
Length-N 1-D vector	L-by-N matrix
Frame-based M-by-N matrix; **Frame** set to `Log frames separately (3-D array)`	M-by-N-by-K array
Frame-based M-by-N matrix; **Frame** set to `Concatenate frames (2-D array)`	(K**f*)-by-N matrix K*f is the number of samples acquired by the end of the simulation.

Matching the Outputs of Signal To Workspace and To Workspace Blocks

The To Workspace block in the Simulink Sinks Library and the Signal To Workspace block can output the same array when they are given the same inputs. To match the blocks' outputs, set their parameters as follows.

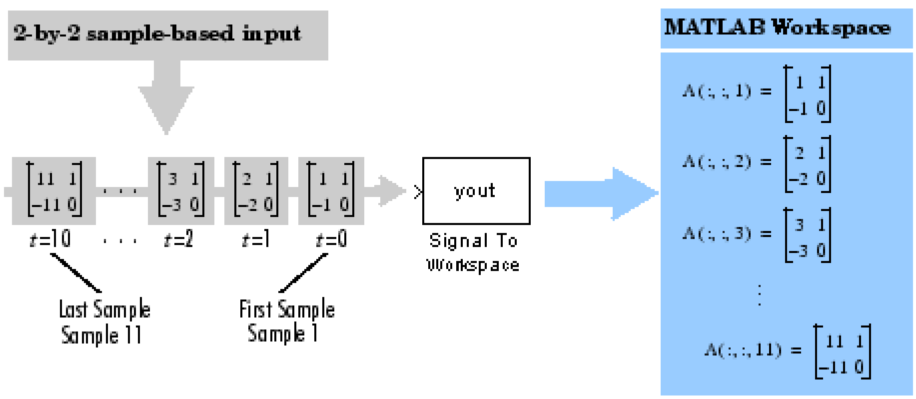
Block Parameters	Signal To Workspace	To Workspace
Limit data points to last	x (any positive integer or `inf`)	x
Decimation	y (any positive integer, not `inf`)	y

Block Parameters	Signal To Workspace	To Workspace
Sample Time	No such parameter	-1
Save format	No such parameter	`Array`
Frames	`Concatenate frames (2-D array)`	No such parameter

Examples

Example 1: Sample-Based Inputs

In the following model, the input to the Signal To Workspace block is a 2-by-2 sample-based matrix signal with a sample time of 1 (generated by a Signal From Workspace block). The Signal To Workspace block logs 11 samples by the end of the simulation, and creates a 2-by-2-by-11 array, A, in the MATLAB workspace.



The block settings are as follows.

Signal To Workspace Block Parameters	
Variable name	`yout`
Limit data points to last	`inf`
Decimation	`1`

Signal To Workspace Block Parameters	
Frames	Ignored since block input is not frame based
Configuration Dialog Box Parameters	
Start time	`0`
Stop time	`10`
Signal From Workspace Parameters (provides Signal To Workspace input)	
Signal	`input1` (defined below)
Sample time	`1`
Samples per frame	`1`
Form output after final data value by	`Setting to zero`

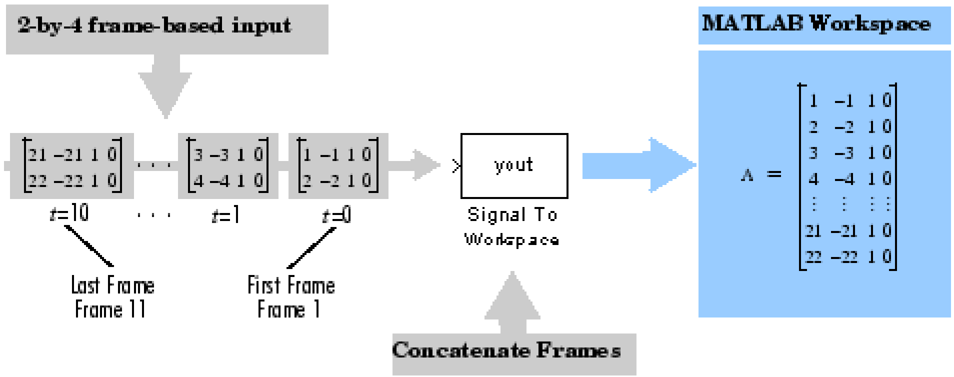
```
input1 = cat(3, [1 1; -1 0], [2 1; -2 0],...,[11 1; -11 0])
```

Example 2: Frame-Based Inputs

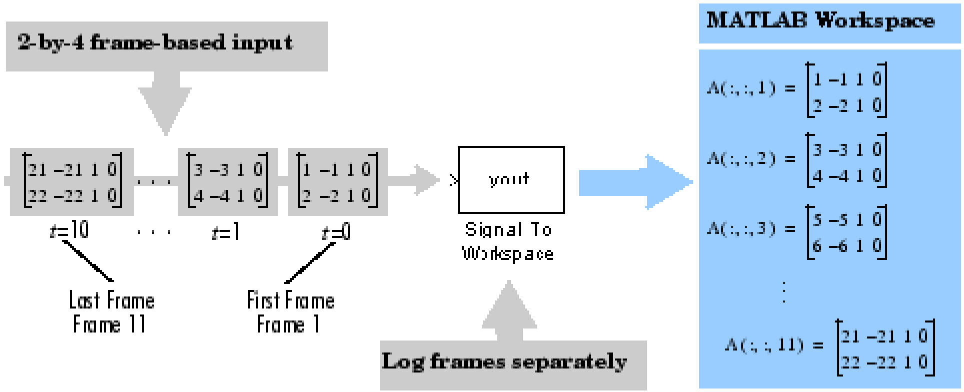
In the following model, the input to the Signal To Workspace block is a 2-by-4 frame-based matrix signal with a frame period of 1 (generated by a Signal From Workspace block). The block logs 11 frames (two samples per frame) by the end of the simulation. The frames are concatenated to create a 22-by-4 matrix, A, in the MATLAB workspace.

The block settings for the following model are similar to the settings used in Example 1, except **Frames** is set to `Concatenate frames (2-D array)` and the Signal From Workspace parameter, **Signal**, is set to `input2`, where

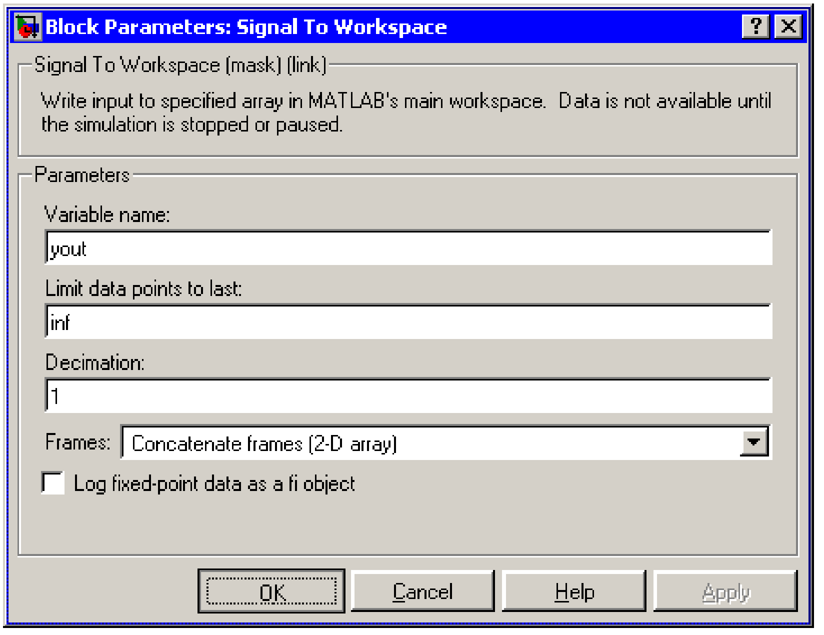
```
input2 = [1 -1 1 0; 2 -2 1 0; 3 -3 1 0;...; 22 -22 1 0]
```



In the 2-D output, there is no indication of where one frame ends and another begins. By setting **Frames** to `Log frames separately (3-D array)` in this model, you can easily see each frame in the MATLAB workspace, as illustrated in the following model. Each of the 11 frames is logged separately to create a 2-by-4-by-11 array, A, in the MATLAB workspace.



Dialog Box



Variable name

The name of the array that holds the input data.

Limit data points to last

The maximum number of input samples (for sample-based inputs) or input frames (for frame-based inputs) to be saved.

Decimation

The decimation factor, d. Data is written at every dth sample.

Frames

The output dimensionality for frame-based inputs. **Frames** can be set to `Concatenate frames (2-D array)` or `Log frames separately (3-D array)`. This parameter is ignored when inputs are not frame based.

Log fixed-point data as a fi object

Select to log fixed-point data to the MATLAB workspace as a Fixed-Point Toolbox `fi` object. Otherwise, fixed-point data is logged to the workspace as `double`.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed and unsigned)
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers

See Also

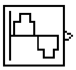
Triggered To Workspace	Signal Processing Blockset
To Workspace	Simulink

Purpose Generate continuous or discrete sine wave

Library Signal Processing Sources

`dspsrcs4`

Description



The Sine Wave block generates a multichannel real or complex sinusoidal signal, with independent amplitude, frequency, and phase in each output channel. A real sinusoidal signal is generated when the **Output complexity** parameter is set to `Real`, and is defined by an expression of the type

$$y = A\sin(2\pi ft + \phi)$$

where you specify A in the **Amplitude** parameter, f in hertz in the **Frequency** parameter, and φ in radians in the **Phase offset** parameter. A complex exponential signal is generated when the **Output complexity** parameter is set to `Complex`, and is defined by an expression of the type

$$y = Ae^{j(2\pi ft+\phi)} = A\left\{\cos(2\pi ft + \phi) + j\sin(2\pi ft + \phi)\right\}$$

Sections of This Reference Page

Generating Multichannel Outputs

For both real and complex sinusoids, the **Amplitude**, **Frequency**, and **Phase offset** parameter values (A, f, and φ) can be scalars or length-N vectors, where N is the desired number of channels in the output. When you specify at least one of these parameters as a length-N vector, scalar values specified for the other parameters are applied to every channel.

For example, to generate the three-channel output containing the real sinusoids below, set **Output complexity** to `Real` and the other parameters as follows:

- **Amplitude** = `[1 2 3]`
- **Frequency** = `[1000 500 250]`
- **Phase offset** = `[0 0 pi/2]`

$$y = \begin{cases} \sin(2000\pi t) & \text{(channel 1)} \\ 2\sin(1000\pi t) & \text{(channel 2)} \\ 3\sin\left(500\pi t + \dfrac{\pi}{2}\right) & \text{(channel 3)} \end{cases}$$

Output Sample Time and Samples Per Frame

In all discrete modes, the block buffers the sampled sinusoids into frames of size M, where you specify M in the **Samples per frame** parameter. The output is a frame-based M-by-N matrix with frame period $M*T_s$, where you specify T_s in the **Sample time** parameter. For M=1, the output is sample based.

Sample Mode

The **Sample mode** parameter specifies the block's sampling property, which can be `Continuous` or `Discrete`:

- `Continuous`

In continuous mode, the sinusoid in the *i*th channel, y_i, is computed as a continuous function,

$$y_i = A_i \sin(2\pi f_i t + \phi_i) \quad \text{(real)}$$

or

$$y_i = A_i e^{j(2\pi f_i t + \phi_i)} \qquad \text{(complex)}$$

and the block's output is continuous. In this mode, the block's operation is the same as that of a Simulink Sine Wave block with **Sample time** set to `0`. This mode offers high accuracy, but requires trigonometric function evaluations at each simulation step, which is computationally expensive. Additionally, because this method tracks absolute simulation time, a discontinuity will eventually occur when the time value reaches its maximum limit.

Note also that many Signal Processing Blockset blocks do not accept continuous-time inputs.

- `Discrete`

  In discrete mode, the block's discrete-time output can be generated by directly evaluating the trigonometric function, by table lookup, or by a differential method. The three options are explained below.

Discrete Computational Methods

When you select `Discrete` from the **Sample mode** parameter, the secondary **Computation method** parameter provides three options for generating the discrete sinusoid:

- `Trigonometric Fcn`

- `Table Lookup`
- `Differential`

Note To generate fixed-point sinusoids, you must select `Table Lookup`.

Trigonometric Fcn

The trigonometric function method computes the sinusoid in the *i*th channel, y_i, by sampling the continuous function

$$y_i = A_i \sin(2\pi f_i t + \phi_i) \quad \text{(real)}$$

or

$$y_i = A_i e^{j(2\pi f_i t + \phi_i)} \qquad \text{(complex)}$$

with a period of T_s, where you specify T_s in the **Sample time** parameter. This mode of operation shares the same benefits and liabilities as the `Continuous` sample mode described above.

At each sample time, the block evaluates the sine function at the appropriate time value *within the first cycle* of the sinusoid. By constraining trigonometric evaluations to the first cycle of each sinusoid, the block avoids the imprecision of computing the sine of very large numbers, and eliminates the possibility of discontinuity during extended operations (when an absolute time variable might overflow). This method therefore avoids the memory demands of the table lookup method at the expense of many more floating-point operations.

Table Lookup

The table lookup method precomputes the *unique* samples of every output sinusoid at the start of the simulation, and recalls the samples from memory as needed. Because a table of finite length can only be constructed when all output sequences repeat, the method requires that the period of every sinusoid in the output be evenly divisible by the sample period. That is, $1/(f_i T_s) = k_i$ must be an integer value for every channel $i = 1, 2, ..., N$.

When the **Optimize table for** parameter is set to `Speed`, the table constructed for each channel contains k_i elements. When the **Optimize table for** parameter is set to `Memory`, the table constructed for each channel contains $k_i/4$ elements.

For long output sequences, the table lookup method requires far fewer floating-point operations than any of the other methods, but can demand considerably more memory, especially for high sample rates (long tables). This is the recommended method for models that are intended to emulate or generate code for DSP hardware, and that therefore need to be optimized for execution speed.

Note The lookup table for this block is constructed from double-precision floating-point values. Thus, when you use the `Table lookup` computation mode, the maximum amount of precision you can achieve in your output is 53 bits. Setting the word length of the **Output** or **User-defined** data type to values greater than 53 bits does not improve the precision of your output.

Differential

The differential method uses an incremental algorithm. This algorithm computes the output samples based on the output values computed at the previous sample time (and precomputed update terms) by making use of the following identities.

$$\sin(t+T_s) = \sin(t)\cos(T_s) + \cos(t)\sin(T_s)$$
$$\cos(t+T_s) = \cos(t)\cos(T_s) - \sin(t)\sin(T_s)$$

The update equations for the sinusoid in the *i*th channel, y_i, can therefore be written in matrix form as

$$\begin{bmatrix} \sin\{2\pi f_i(t+T_s)+\phi_i\} \\ \cos\{2\pi f_i(t+T_s)+\phi_i\} \end{bmatrix} = \begin{bmatrix} \cos(2\pi f_i T_s) & \sin(2\pi f_i T_s) \\ -\sin(2\pi f_i T_s) & \cos(2\pi f_i T_s) \end{bmatrix} \begin{bmatrix} \sin(2\pi f_i t+\phi_i) \\ \cos(2\pi f_i t+\phi_i) \end{bmatrix}$$

where you specify T_s in the **Sample time** parameter. Since T_s is constant, the right-hand matrix is a constant and can be computed once at the start of the simulation. The value of $A_i\sin[2\pi f_i(t+T_s)+\varphi_i]$ is then computed from the values of $\sin(2\pi f_i t+\varphi_i)$ and $\cos(2\pi f_i t+\varphi_i)$ by a simple matrix multiplication at each time step.

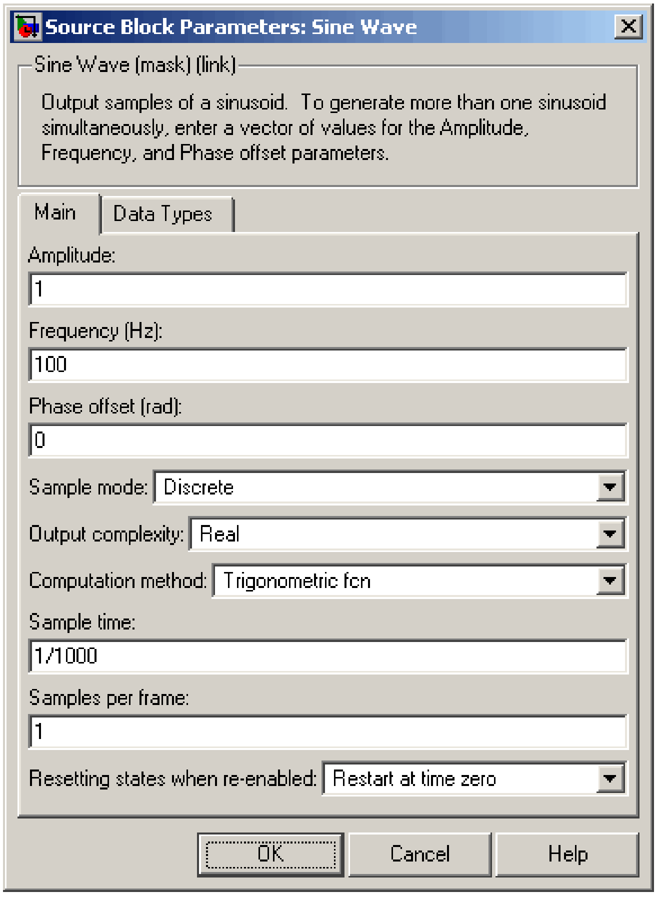
This mode offers reduced computational load, but is subject to drift over time due to cumulative quantization error. Because the method is not contingent on an absolute time value, there is no danger of discontinuity during extended operations (when an absolute time variable might overflow).

Examples

The `dspsinecomp` demo provides a comparison of all the available sine generation methods.

Dialog Box

The **Main** pane of the Sine Wave block dialog appears as follows.



Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters During Simulation" in the online Simulink documentation for details.

Amplitude

A length-N vector containing the amplitudes of the sine waves in each of N output channels, or a scalar to be applied to all N channels. The vector length must be the same as that specified for the **Frequency** and **Phase offset** parameters. Tunable when **Computation method** is to `Trigonometric fcn` or `Differential`.

Frequency

A length-N vector containing frequencies, in Hertz, of the sine waves in each of N output channels, or a scalar to be applied to all N channels. The vector length must be the same as that specified for the **Amplitude** and **Phase offset** parameters. You can specify positive, zero, or negative frequencies. Tunable when **Sample mode** is `Continuous` or **Computation method** is `Trigonometric fcn`.

Phase offset

A length-N vector containing the phase offsets, in radians, of the sine waves in each of N output channels, or a scalar to be applied to all N channels. The vector length must be the same as that specified for the **Amplitude** and **Frequency** parameters. Tunable when **Sample mode** is `Continuous` or **Computation method** is `Trigonometric fcn`.

Sample mode

The block's sampling behavior, `Continuous` or `Discrete`. This parameter is not tunable.

Output complexity

The type of waveform to generate: `Real` specifies a real sine wave, `Complex` specifies a complex exponential. This parameter is not tunable.

Computation method

The method by which discrete-time sinusoids are generated: `Trigonometric fcn`, `Table lookup`, or `Differential`. This parameter is not tunable. For more information on each of the available options, see "Discrete Computational Methods" on page 2-1195 in the Description section.

This parameter is only visible when you set the **Sample mode** to `Discrete`.

Note To generate fixed-point sinusoids, you must set the **Computation method** to `Table lookup`.

Optimize table for

Optimizes the table of sine values for `Speed` or `Memory` (this parameter is only visible when the **Computation method** parameter is set to `Table lookup`). When optimized for speed, the table contains *k* elements, and when optimized for memory, the table contains *k*/4 elements, where *k* is the number of input samples in one full period of the sine wave.

Sample time

The period with which the sine wave is sampled, T_s. The block's output frame period is $M*T_s$, where you specify *M* in the **Samples per frame** parameter. This parameter is disabled when you select `Continuous` from the **Sample mode** parameter. This parameter is not tunable.

Samples per frame

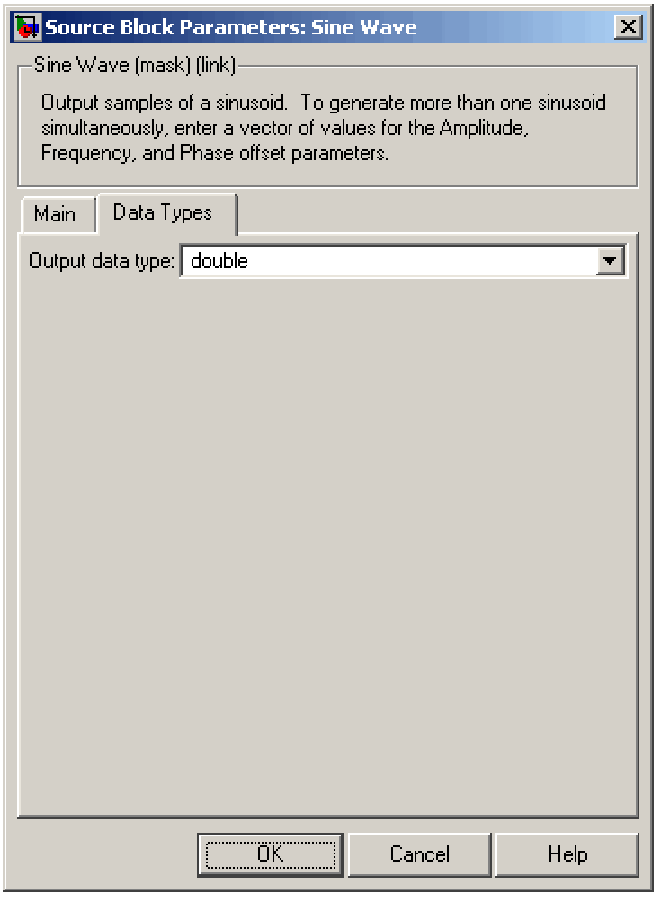
The number of consecutive samples from each sinusoid to buffer into the output frame, *M*. When the value of this parameter is `1`, the block outputs a sample-based signal.

This parameter is disabled when you select `Continuous` from the **Sample mode** parameter.

Resetting states when re-enabled

This parameter only applies when the Sine Wave block is located inside an enabled subsystem and the **States when enabling** parameter of the Enable block is set to `reset`. This parameter determines the behavior of the Sine Wave block when the subsystem is re-enabled. The block can either reset itself to its starting state (`Restart at time zero`), or resume generating the sinusoid based on the current simulation time (`Catch up to simulation time`). This parameter is disabled when you select `Continuous` from the **Sample mode** parameter.

The **Data Types** pane of the Sine Wave block dialog appears as follows.



Output data type

Specify the output data type in out of the following ways:

Choose one of the built-in data types from the list.

Choose `Fixed-point` to specify the output data type and scaling in the **Word length**, **Set fraction length in output to**, and **Fraction length** parameters.

Choose `User-defined` to specify the output data type and scaling in the **User-defined data type**, **Set fraction length in output to**, and **Fraction length** parameters.

Choose `Inherit via back propagation` to set the output data type and scaling to match the next block downstream.

Note The lookup table for this block is constructed from double-precision floating-point values. Thus, when you use the `Table lookup` computation mode, the maximum amount of precision you can achieve in your output is 53 bits. Setting the word length of the **Output** or **User-defined** data type to values greater than 53 bits does not improve the precision of your output.

Word length

Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter.

User-defined data type

Specify any built-in or fixed-point data type. You can specify fixed-point data types using the following Simulink Fixed Point functions: `sfix`, `ufix`, `sint`, `uint`, `sfrac`, and `ufrac`. This parameter is only visible when you select `User-defined` for the **Output data type** parameter.

Set fraction length in output to

Specify the scaling of the fixed-point output by either of the following two methods:

Choose `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.

Choose `User-defined` to specify the output scaling in the **Fraction length** parameter.

This parameter is only visible when you select `Fixed-point` for the **Output data type** parameter, or when you select `User-defined` and the specified output data type is a fixed-point data type.

Fraction length

For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Output data type** parameter and `User-defined` for the **Set fraction length in output to** parameter.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed only)
- 8-, 16-, and 32-bit signed integers

See Also

Chirp	Signal Processing Blockset
Complex Exponential	Signal Processing Blockset
Signal From Workspace	Signal Processing Blockset
Signal Generator	Simulink

Sine Wave	Simulink
`sin`	MATLAB

Singular Value Decomposition

Purpose

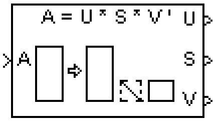
Factor matrix using singular value decomposition

Library

Math Functions / Matrices and Linear Algebra / Matrix Factorizations

`dspfactors`

Description



The Singular Value Decomposition block factors the M-by-N input matrix A such that

$$A = U \cdot diag(S) \cdot V^*$$

where

- U is an M-by-P matrix
- V is an N-by-P matrix
- S is a length-P vector
- P is defined as min(M,N)

When

- $M = N$, U and V are both M-by-M unitary matrices
- $M > N$, V is an N-by-N unitary matrix, and U is an M-by-N matrix whose columns are the first N columns of a unitary matrix
- $N > M$, U is an M-by-M unitary matrix, and V is an N-by-M matrix whose columns are the first M columns of a unitary matrix

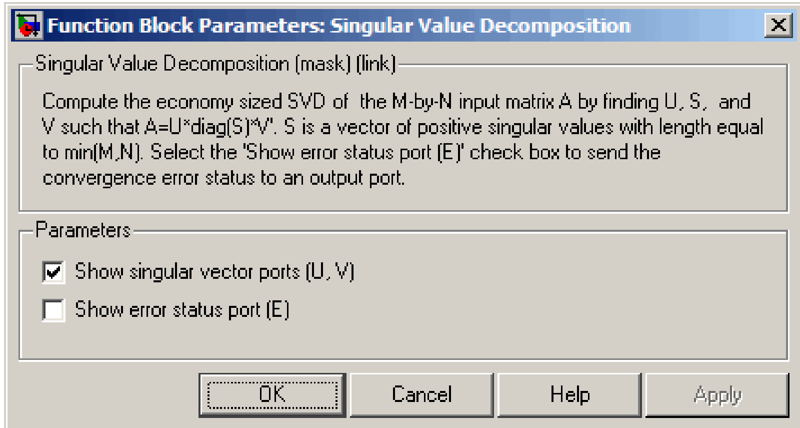
In all cases, S is a 1-D vector of positive singular values having length P.

Length-N row inputs are treated as length-N columns.

Note that the first (maximum) element of output S is equal to the 2-norm of the matrix A.

The output is always sample based.

Dialog Box



Show singular vector ports
Select to enable the `U` and `V` output ports.

Show error status port
Select to enable the E output port, which reports a failure to converge. The possible values you can receive on the port are:

- `0` — The singular value decomposition calculation converges.
- `1` — The singular value decomposition calculation does not converge.

If the singular value decomposition calculation fails to converge, the output at ports U, S, and V are undefined matrices of the correct size.

References

Golub, G. H., and C. F. Van Loan. *Matrix Computations*. 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

Supported Data Types

Port	Supported Data Types
A	• Double-precision floating point • Single-precision floating point
U	• Double-precision floating point • Single-precision floating point
S	• Double-precision floating point • Single-precision floating point
V	• Double-precision floating point • Single-precision floating point
E	• Boolean

See Also

Autocorrelation LPC	Signal Processing Blockset
Cholesky Factorization	Signal Processing Blockset
LDL Factorization	Signal Processing Blockset
LU Inverse	Signal Processing Blockset
Pseudoinverse	Signal Processing Blockset
QR Factorization	Signal Processing Blockset
SVD Solver	Signal Processing Blockset
`svd`	MATLAB

See "Matrix Factorizations" for related information.

Purpose

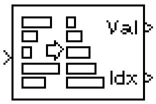
Sort input elements by value

Library

Statistics

`dspstat3`

Description



The Sort block ranks the values of the input elements using either a quick sort or an insertion sort algorithm. The quick sort algorithm uses a recursive sort method and is faster at sorting more than 32 elements. The insertion sort algorithm uses a non-recursive method and is faster at sorting less than 32 elements. You should also always use the insertion sort algorithm when you are generating code from the Sort block if you do not want recursive function calls in your code. To specify the sort method, use the **Sort algorithm** parameter.

The **Mode** parameter specifies the block's mode of operation, and can be set to `Value`, `Index`, or `Value and index`.

The Sort block supports real and complex floating-point and fixed-point inputs. Signed and unsigned fixed-point signals are supported. The block output has the same signedness as the input.

Value Mode

When **Mode** is set to `Value`, the block sorts the elements in each column of the M-by-N input matrix `u` in order of ascending or descending value, as specified by the **Sort order** parameter.

```
val = sort(u)
val = flipud(sort(u))
```

For convenience, length-M 1-D vector inputs and *sample-based* length-M row vector inputs are both treated as M-by-1 column vectors.

The output at each sample time, `val`, is an M-by-N matrix containing the sorted columns of `u`. The output has the same frame status as the input.

Complex inputs are sorted by *magnitude squared*. For complex value `u = a + bi`, the magnitude squared is $a^2 + b^2$.

Index Mode

When **Mode** is set to `Index`, the block sorts the elements in each column of the M-by-N input matrix `u`,

```
[val,idx] = sort(u)
[val,idx] = flipud(sort(u))
```

and outputs the sample-based M-by-N index matrix, `idx`. The `j`th column of `idx` is an index vector that permutes the `j`th column of `u` to the desired sorting order.

```
val(:,j) = u(idx(:,j),j)
```

The index value outputs are always 32-bit unsigned integer values.

As in `Value` mode, length-M 1-D vector inputs and *sample-based* length-M row vector inputs are both treated as M-by-1 column vectors.

Value and Index Mode

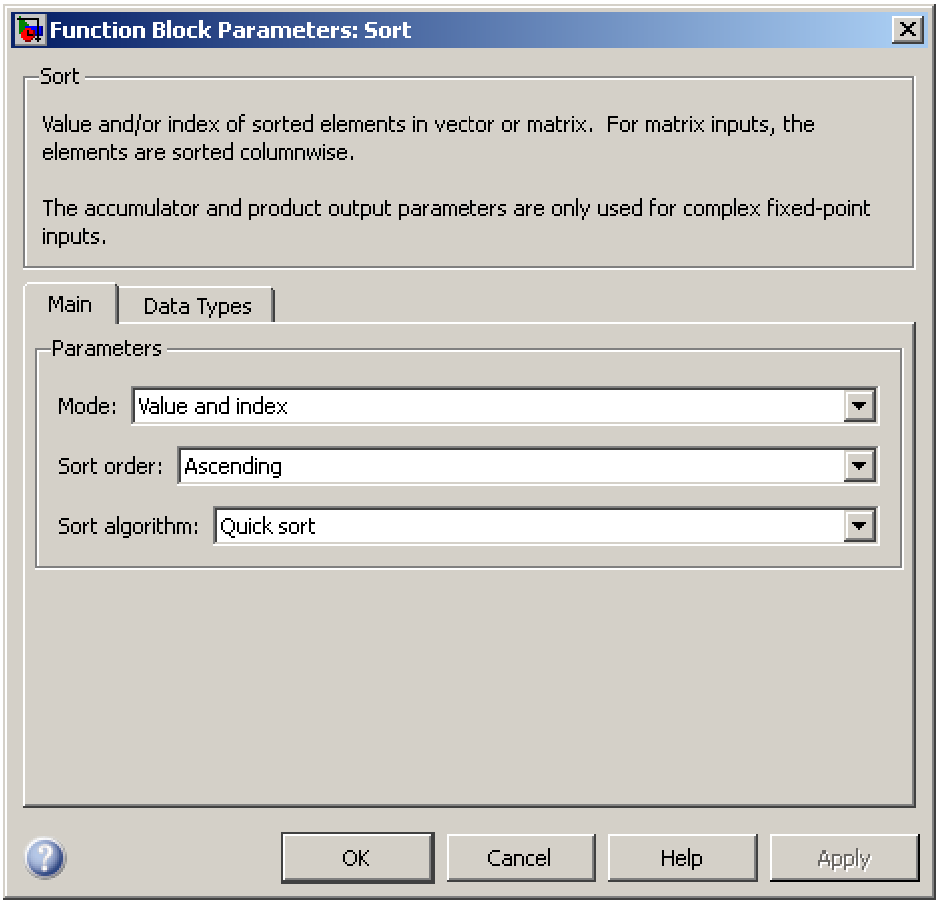
When **Mode** is set to `Value and index`, the block outputs both the sorted matrix, `val`, and the index matrix, `idx`.

Fixed-Point Data Types

The parameters on the **Data Types** pane are only used for complex fixed-point inputs. Complex fixed-point inputs are sorted by magnitude squared. The sum of the squares of the real and imaginary parts of such an input are formed before a comparison is made, as described in "Value Mode" on page 2-1209. The results of the squares of the real and imaginary parts are placed into the product output data type. The result of the sum of the squares is placed into the accumulator data type. These parameters are ignored for other types of inputs.

Dialog Box

The **Main** pane of the Sort block dialog appears as follows.



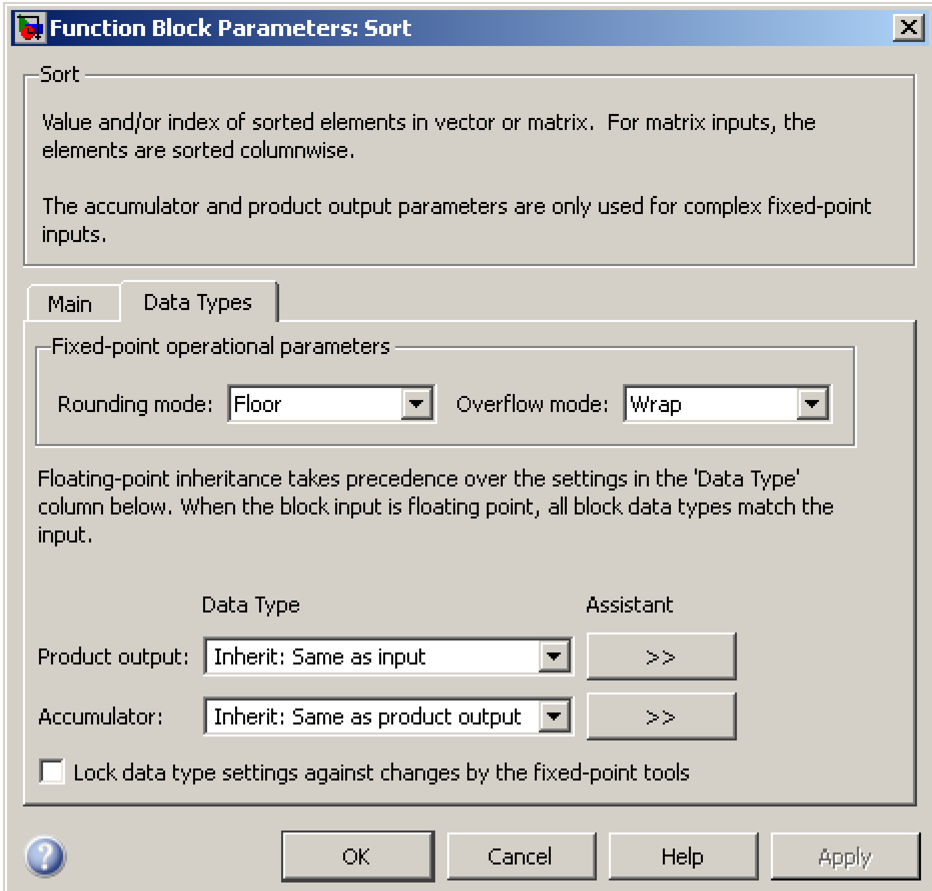
Mode

Specify the block's mode of operation: Output the sorted matrix (`Value`), the index matrix (`Index`), or both (`Value and index`).

Sort order
Specify the order in which to sort the training points, `Descending` or `Ascending`.

Sort algorithm
Specify whether the elements of the input are sorted using a `Quick sort` or an `Insertion sort` algorithm.

The **Data Types** pane of the Sort block dialog appears as follows.


Function Block Parameters: Sort
Sort
Value and/or index of sorted elements in vector or matrix. For matrix inputs, the elements are sorted columnwise.
The accumulator and product output parameters are only used for complex fixed-point inputs.
Main
Data Types
Fixed-point operational parameters
Rounding mode:
Floor
Overflow mode:
Wrap
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input.
Data Type
Assistant
Product output:
Inherit: Same as input
>>
Accumulator:
Inherit: Same as product output
>>
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Note The parameters on the **Data Types** pane are only used for complex fixed-point inputs. The sum of the squares of the real and imaginary parts of such an input are formed before a comparison is made, as described in “Value Mode” on page 2-1209. The results of the squares of the real and imaginary parts are placed into the product output data type. The result of the sum of the squares is placed into the accumulator data type. These parameters are ignored for other types of inputs.

Rounding mode
Select the rounding mode for fixed-point operations.

Overflow mode
Select the overflow mode for fixed-point operations.

Product output data type
Specify the product output data type. See “Fixed-Point Data Types” on page 2-1210 and “Multiplication Data Types” for illustrations depicting the use of the product output data type in this block. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Product output data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Accumulator data type

Specify the accumulator data type. See “Fixed-Point Data Types” on page 2-1210 for illustrations depicting the use of the accumulator data type in this block. You can set this parameter to:

- A rule that inherits a data type, for example, `Inherit: Same as product output`
- An expression that evaluates to a valid data type, for example, `fixdt([],16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Accumulator data type** parameter.

See “Using the Data Type Assistant” in *Simulink User’s Guide* for more information.

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, 32-, and 128-bit unsigned integers • 8-, 16-, 32-, and 128-bit signed integers
Val	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, 32-, and 128-bit unsigned integers

Port	Supported Data Types
Idx	• 8-, 16-, 32-, and 128-bit signed integers • 32-bit unsigned integers

See Also

Histogram	Signal Processing Blockset
Median	Signal Processing Blockset
`sort`	MATLAB

Purpose

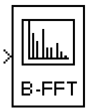
Compute and display periodogram of each input signal

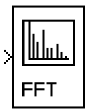
Library

Signal Processing Sinks

`dspsnks4`

Description





The Spectrum Scope block computes and displays the periodogram of the input. The input can be a sample-based or frame-based vector or a frame-based matrix.

Note When the **Buffer input** and **Specify FFT length** parameters are both cleared, the block input length must be a power of two.

Scope Properties Pane

The **Spectrum units** parameter allows you to specify the following information:

- The type of measurement for the block to compute (Power Spectral Density or Mean-Square Spectrum)
- The type of scaling for the block to use (linear or log)

You can set the **Spectrum units** parameter to one of the options shown in the following table.

Spectrum Units	Measurement Type	Scaling
`Watts`	Mean-Square Spectrum (MSS)	Linear
`dBW` `dBm`	Mean-Square Spectrum (MSS)	Logarithmic

Spectrum Units	Measurement Type	Scaling
`Watts/Hertz`	Power Spectral Density (PSD)	Linear
`dBW/Hertz` `dBm/Hertz`	Power Spectral Density (PSD)	Logarithmic

The *X*-axis units are always expressed in Hertz. The spacing between frequency points is $1/(N_{fft}T_s)$.

The **Spectrum type** parameter specifies the range of frequencies over which the block computes the spectrum. The available options are `One-sided ([0...Fs/2])` and `Two-sided ((-Fs/2...Fs/2])`, where F_s is the sampling frequency of the original time-domain signal. Both the one-sided and two-sided options compute the full power spectrum. The Spectrum Scope block only supports `One-sided ([0...Fs/2])` spectrums for real input signals.

Other Signal Processing Blockset FFT-based blocks, including the blocks in the Power Spectrum Estimation library, always compute the FFT at frequencies in the range $[0,F_s)$.

Select the **Buffer input** check box when the input to the block is sample based. You can also use buffering for frame-based inputs, but it is optional. When the block buffers the input, the **Buffer size** parameter specifies the number of input samples to buffer before computing and displaying the magnitude FFT. You also use the **Buffer overlap** parameter to specify the number of samples from the previous buffer to include in the current buffer. To compute the number of new input samples the block acquires before computing and displaying the magnitude FFT, subtract the buffer overlap from the buffer size.

The display update period is

$$(M_o - L) * T_s$$

where

- M_o = buffer size

- L = buffer overlap
- T_s = input sample period

For negative buffer overlap values, the block discards the appropriate number of input samples after the buffer fills. The block also updates the scope display at a slower rate than in the zero-overlap case.

The **Window** and **Window sampling** parameters apply to the specification of the window function. See the Window Function block reference page for more details on these parameters.

The block determines the FFT length, N_{fft}, in the following ways:

- If you clear the **Specify FFT length** check box and select **Buffer input**, the block uses the buffer size as the FFT size.
- If you clear both the **Specify FFT length** and **Buffer input** check boxes, the block uses the input size as the FFT size.
- If you select the **Specify FFT length** check box, the **FFT length** parameter appears in the dialog box. Enter the number of samples on which you want the block to perform the FFT. This value must be a power of two.

The block zero pads or wraps the buffer of each channel to the FFT length before computing the FFT.

The value of the **Number of spectral averages** parameter determines the number of spectra to average. Setting this parameter to 1 effectively disables averaging. See the Periodogram block reference page for more information.

Display Properties Pane

The **Display Properties** pane enables you to control how the block displays your data.

The **Show grid** parameter toggles the background grid on and off.

If you select the **Persistence** check box, the window maintains successive displays. That is, the scope does not erase the display after each frame (or collection of frames), but overlays successive input frames in the scope display.

If you select the **Frame number** check box, the block displays the number of the current frame in the input sequence on the scope window, and the block increments the count as each new input is received. Counting starts at 1 with the first input frame, and continues until the simulation stops.

If you select the **Channel legend** check box, a legend indicating the line color, style, and marker of each channel's data is added. When the input signal is labeled, that label appears in the channel legend. When the input signal is not labeled, but comes from a Concatenate block or a Mux block with labeled inputs, those labels appear in the channel legend. Otherwise, each channel in the legend is labeled with the channel number (`CH 1`, `CH 2`, etc.). Click and drag the legend to reposition it in the scope window; double-click the line label to edit the text. If you rerun the simulation, the labels revert to the defaults.

If you select the **Compact display** check box, the scope completely fills the figure window. The scope does not display menus and axis titles, but it does show the numerical axis labels within the axes. If you clear the **Compact display** check box, the scope displays the axis labels and titles in a gray border surrounding the scope axes, and the window's menus and toolbar become visible.

If you select the **Open scope at start of simulation** check box, the scope opens at the start of the simulation. If you clear this parameter, the scope does not open automatically during the simulation. You can use this feature when you have several scope blocks in a model, and you do not want to view all the associated scopes during the simulation.

To view a scope window that is not open during simulation, click **Open scope immediately** on the **Display Properties** pane of the desired Scope block.

The **Scope position** parameter specifies a four-element vector of the form

```
[left bottom width height]
```

specifying the position of the scope window on the screen, where (0,0) is the lower-left corner of the display. See the MATLAB `figure` function for more information.

Axis Properties Pane

If you select the **Inherit sample time from input** check box, the block computes the frequency data from the sample period of the input to the block. For the block to produce valid output, the following conditions must hold:

- The input to the block is the original signal, with no samples added or deleted (by insertion of zeros, for example).
- The sample period of the time-domain signal in the simulation equals the sample period of the original time series.

In cases where not all these conditions hold, specify the appropriate value for the **Sample time of original time series** parameter.

When you set the **Frequency display limits** to `Auto`, the block displays the full spectrum over the frequency range specified by the **Spectrum type** parameter.

The **Frequency display offset** parameter allows you to offset the range of values displayed on the frequency axis of the Spectrum Scope.

- When the **Frequency display offset** is `0`, the block displays the DC frequency (0 Hz) at 0 Hz.
- When **Frequency display offset** is a nonzero value, the block displays the DC frequency (0 Hz) at the value specified in the **Frequency display offset** parameter. If you set the **Frequency display limits** parameter to `User-defined`, the block does not automatically relabel the frequency axis. However, if you set the **Frequency display limits** to `Auto`, the values displayed on the frequency axis shift according to the **Frequency display offset** parameter.

For example, if the block has the following settings:

- **Spectrum Units** = `Watts/Hertz`
- **Spectrum Type** = `Two-sided ((-Fs/2...Fs/2])`
- **Frequency display offset (Hz)** = `0`
- **Frequency display limits** = `Auto`
- Sampling frequency (*Fs*) = 1000 Hz

Then, based on these settings:

- The values on the frequency axis of the spectrum scope range from -500 Hz to 500 Hz.
- The block centers the DC frequency (0 Hz) at 0 Hz.

If you change the **Frequency display offset (Hz)** parameter to `100`, the block:

- Relabels the frequency axis such that the values range from -400 Hz to 600 Hz.
- Centers the DC frequency (0 Hz) at 100 Hz.

When you set the **Frequency display limits** to `User-defined`, the **Minimum frequency (Hz)** and **Maximum frequency (Hz)** parameters set the range of the horizontal axis.

Minimum Y-limit and **Maximum Y-limit** parameters allow you to set the range of the vertical axis. Setting these parameters equates to setting the `ymin` and `ymax` values of the MATLAB `axis` function.

The **Y-axis label** is the text displayed to the left of the y-axis.

Line Properties Pane

Use the parameters on the **Line Properties** pane to help you distinguish between two or more independent channels of data on the scope.

The **Line visibilities** parameter specifies which channel's data is displayed on the scope, and which is hidden. The syntax specifies the visibilities in list form, where the term `on` or `off` as a list entry specifies the visibility of the corresponding channel's data. The list entries are separated by the pipe symbol, `|`.

For example, a five-channel signal would ordinarily generate five distinct plots on the scope. To disable plotting of the third and fifth lines, enter the following visibility specification in the **Line visibilities** parameter.

```
 on | on | off | on | off
ch 1  ch 2  ch 3  ch 4  ch 5
```

Note that the first (leftmost) list item corresponds to the first signal channel (leftmost column of the input matrix).

The **Line styles** parameter specifies the line style with which each channel's data is displayed on the scope. The syntax specifies the channel line styles in list form, with each list entry specifying a style for the corresponding channel's data. The list entries are separated by the pipe symbol, `|`.

For example, a five-channel signal would ordinarily generate all five plots with a solid line style. To plot each line with a different style, enter

```
  - | -- | : | -. | -
ch 1  ch 2  ch 3 ch 4  ch 5
```

These settings plot the signal channels with the following styles.

Line Style	Command to Type in Line Style Parameter	Appearance
Solid	-	
Dashed	--	
Dotted	:	

Line Style	Command to Type in Line Style Parameter	Appearance
Dash-dot	`-.`	
No line	`none`	No line appears

Note that the first (leftmost) list item, `'-'`, corresponds to the first signal channel (leftmost column of the input matrix). See the LineSpec property of the MATLAB `line` function for more information about the available markers.

The **Line markers** parameter specifies the marker style with which each channel's samples are represented on the scope. The syntax specifies the channels' marker styles in list form, with each list entry specifying a marker for the corresponding channel's data. The list entries are separated by the pipe symbol, `|`.

For example, a five-channel signal would ordinarily generate all five plots with no marker symbol (that is, the individual sample points are not marked on the scope). To instead plot each line with a different marker style, you could enter

```
  * | . | x | s | d
ch 1  ch 2  ch 3  ch 4  ch 5
```

These settings plot the signal channels with the following styles.

Marker Style	Command to Type in Marker Style Parameter	Appearance
Asterisk	`*`	
Point	`.`	
Cross	`x`	
Square	`s`	
Diamond	`d`	

Note that the leftmost list item, `'*'`, corresponds to the first signal channel or leftmost column of the input matrix. See the property of the MATLAB `line` function for more information about the available markers.

To produce a stem plot for the data in a particular channel, type the word `stem` instead of one of the basic marker shapes.

The **Line colors** parameter specifies the color in which each channel's data is displayed on the scope. The syntax specifies the channel colors in list form, with each list entry specifying a color (in one of the MATLAB `ColorSpec` formats) for the corresponding channel's data. The list entries are separated by the pipe symbol, `|`.

For example, a five-channel signal would ordinarily generate all five plots in the color black. To instead plot the lines with the color order below, enter

```
[0 0 0] | [0 0 1] | [1 0 0 ] | [0 1 0] | [.7529 0 .7529]
 ch 1      ch 2      ch 3       ch 4          ch 5
```

or

```
'k' | 'b' | 'r' | 'g' | [.7529 0 .7529]
ch 1  ch 2  ch 3  ch 4      ch 5
```

These settings plot the signal channels in the following colors (8-bit RGB equivalents shown in the center column).

Color	RGB Equivalent	Appearance
Black	(0,0,0)	
Blue	(0,0,255)	
Red	(255,0,0)	
Green	(0,255,0)	
Dark purple	(192,0,192)	

Note that the leftmost list item, `'k'`, corresponds to the first signal channel or leftmost column of the input matrix. See the MATLAB function `ColorSpec` for more information about the color syntax.

Spectrum Scope Window

The title that appears in the title bar of the scope window is the same as the block title. In addition to the standard MATLAB figure window menus such as **File**, **Window**, and **Help**, the Spectrum Scope window contains **View**, **Axes**, and **Channels** menus.

The options in the **View** menu allow you to zoom in and out of the scope window:

- To zoom in on the scope window, you must first select **View > Zoom In** or click the corresponding `Zoom In` toolbar button (). You can then zoom in by clicking in the center of your area of interest, or by clicking and dragging your cursor to draw a rectangular area of interest inside of the scope window.
- To zoom in on the *x*-axis of the scope window, you must first select **View > Zoom X** or click the corresponding `Zoom X-Axis` toolbar button () on the scope window. You can then zoom in on the *x*-axis with a single click inside the scope window, or by clicking and dragging the cursor along the *x*-axis over your area of interest.
- To zoom in on the *y*-axis of the scope window, you must first select **View > Zoom Y** or click the corresponding `Zoom Y-Axis` toolbar button (). You can then zoom in on the *y*-axis with a single click inside the scope window, or by clicking and dragging the cursor along the *y*-axis over your area of interest.
- To return to the original view of the scope window, you have the following options:
  - Select **Full View** from the **View** menu.
  - Click the `Restore default view` toolbar button () on the Spectrum Scope window.
  - Right-click inside the scope window and select `Reset to Original View`.

Note To zoom out in smaller increments, you can right-click inside of the scope window and select `Zoom Out`. You can also zoom out by holding down the **Shift** key and clicking the left mouse button inside the scope window.

The parameters that you set using the **Axes** menu apply to all channels. Many of the parameters in this menu are also accessible through the block parameters dialog box. For descriptions of these parameters, see

"Display Properties Pane" on page 2-1234. Below are descriptions of other parameters in the **Axes** menu:

- **Refresh** erases all data on the scope display, except for the most recent trace. This command is useful in conjunction with the **Persistence** setting.
- **Autoscale** resizes the *y*-axis to best fit the vertical range of the data.

Note The **Minimum Y-limit** and **Maximum Y-limit** parameters on the **Axis properties** pane of the block dialog are not updated to display the numerical limits selected by the autoscale feature. See "Zoom Capability for Spectrum Scope and Vector Scope Blocks" in the Signal Processing Blockset Release Notes for more information.

- **Save Axes Settings** allows you to save the current axes settings. When you select this option, the **Minimum Y-limit** and **Maximum Y-limit** parameters of the **Axes Properties** pane update with the current *y*-axes limits. The **Frequency display limits** parameter is set to `User-defined`, and the current *x*-axes limits are saved in the **Minimum Frequency (Hz)** and **Maximum Frequency (Hz)** parameters. To save these axes settings for your next MATLAB session, you need to resave your model.
- **Save Scope Position** automatically updates the **Scope position** parameter in the **Axis Properties** pane of the block dialog. When you select **Save Scope Position**, the block saves the current position and size of the scope window. To make the scope window open at a particular location on the screen when the simulation runs, drag the window to the desired location, resize it, and select **Save Scope Position** from the **Axes** menu.

The properties listed in the **Channels** menu apply to a particular channel. All of the parameters in this menu are also accessible through the block parameters dialog box. For descriptions of these parameters, see "Line Properties Pane" on page 2-1222.

Many of these options are also accessible by right-clicking the mouse anywhere on the scope display. The menu that is displayed contains a combination of the options available in the **View**, **Axes** and **Channels** menus.

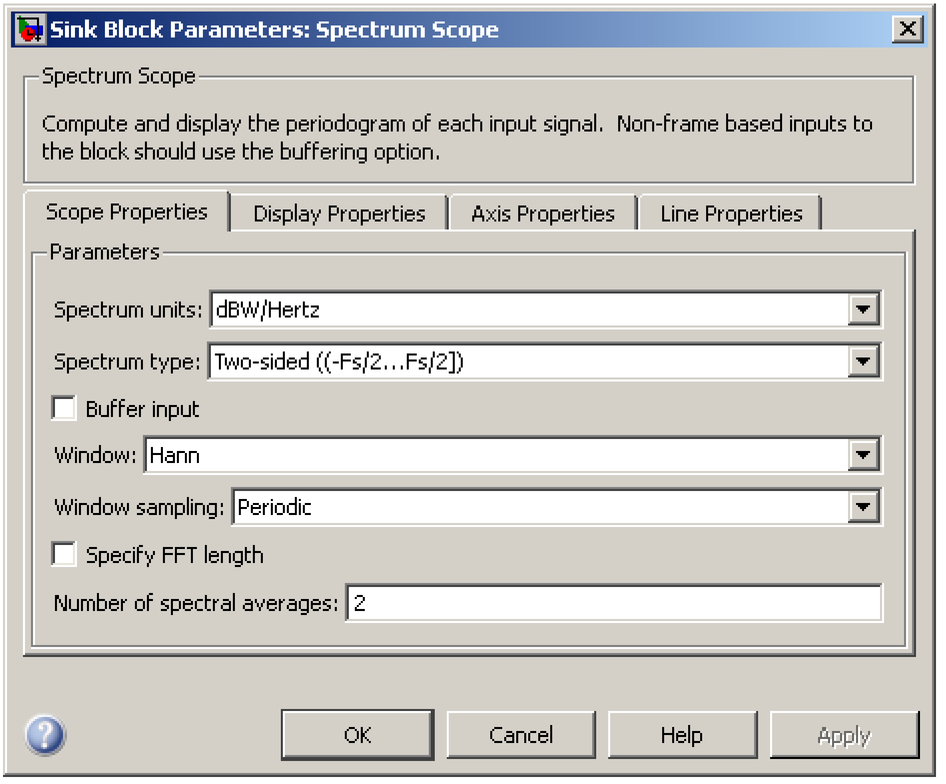
Note When you select **Compact Display** from the **Axes** menu, the scope window menus are no longer visible. Right-click in the Spectrum Scope window and click **Compact Display** to make the menus reappear.

Examples

See "Displaying Frequency-Domain Data" in the *Signal Processing Blockset User's Guide*.

Dialog Box

Scope Properties Pane



Spectrum units

Specify the spectrum units as described in the following table. The specified units determine the type of measurement to compute (Mean-Square Spectrum or Power Spectral Density). They also determine the type of Y-axis scaling (linear or logarithmic).

Spectrum Units	Measurement Type	Scaling
`Watts`	Mean-Square Spectrum (MSS)	Linear
`dBW` `dBm`	Mean-Square Spectrum (MSS)	Logarithmic
`Watts/Hertz`	Power Spectral Density (PSD)	Linear
`dBW/Hertz` `dBm/Hertz`	Power Spectral Density (PSD)	Logarithmic

You can only tune this parameter within the same **Measurement type**. The block cannot switch between computing the mean-square spectrum and the power spectral density while the simulation is running.

Spectrum type

Specify the range of frequencies over which to compute the magnitudes in the input. The available options are `One-sided ([0...Fs/2])` and `Two-sided ((-Fs/2...Fs/2])`, where F_s is the sampling frequency of the original time-domain signal. If you select `One-sided ([0...Fs/2])`, the input signal must be real-valued. Tunable.

Buffer input

Select this check box to rebuffer the input data. Sample-based inputs require that you select this check box. However, it is optional for frame-based inputs.

The toolbox does not support this functionality for use with external mode. Instead, clear this check box and use a Buffer block before the Spectrum Scope in your model.

Buffer size

Specify the number of input samples that the block buffers before computing and displaying the magnitude FFT. If you do not select the **Specify FFT length** check box, the **Buffer size** must be a power of two.

This parameter becomes visible only when you select the **Buffer input** check box.

Buffer overlap

Specify the number of samples from the previous buffer to include in the current buffer. To compute the number of new input samples the block acquires before computing and displaying the magnitude FFT, subtract the buffer overlap from the buffer size.

This parameter becomes visible only when you select the **Buffer input** check box.

Window

Specify the type of window to apply. See the Window Function block reference page for more details. Tunable.

Stopband attenuation in dB

Enter the level, in decibels (dB), of stopband attenuation, R_s, for the Chebyshev window. Tunable.

This parameter becomes visible only when you select `Chebyshev` for the **Window** parameter.

Beta

Enter the β parameter for the Kaiser window. Increasing **Beta** widens the mainlobe and decreases the amplitude of the sidelobes in the displayed frequency magnitude response. Tunable.

This parameter becomes visible only if you select `Kaiser` for the **Window** parameter.

Window sampling

Choose `Symmetric` or `Periodic`. Tunable.

This parameter becomes visible only if `Blackman`, `Hamming`, `Hann`, or `Hanning` is selected for the **Window** parameter.

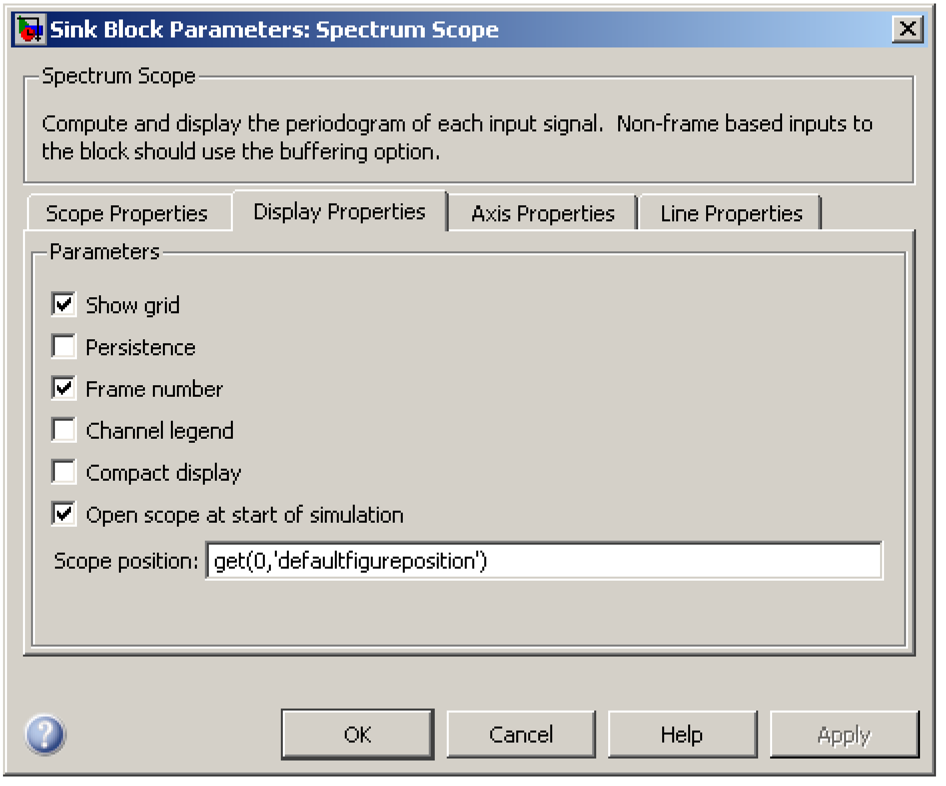
Specify FFT length
Select this check box to specify the FFT length yourself in the **FFT length** parameter.

FFT length
Enter the number of samples on which you want the block to perform the FFT. The value you specify must be a power of two.

This parameter becomes visible only when you select the **Specify FFT length** check box.

Number of spectral averages
The number of spectra to average. Setting this parameter to `1` effectively disables averaging. See the Periodogram block reference page for more information.

Display Properties Pane



Show grid
Toggle the scope grid on and off. Tunable.

Persistence
Select this check box to maintain successive displays. That is, the scope does not erase the display after each frame (or collection of frames), but overlays successive input frames in the scope display. Tunable.

Frame number
If you select this check box, the number of the current frame in the input sequence appears in the Vector Scope window. Tunable.

Channel legend
Toggles the legend on and off. Tunable.

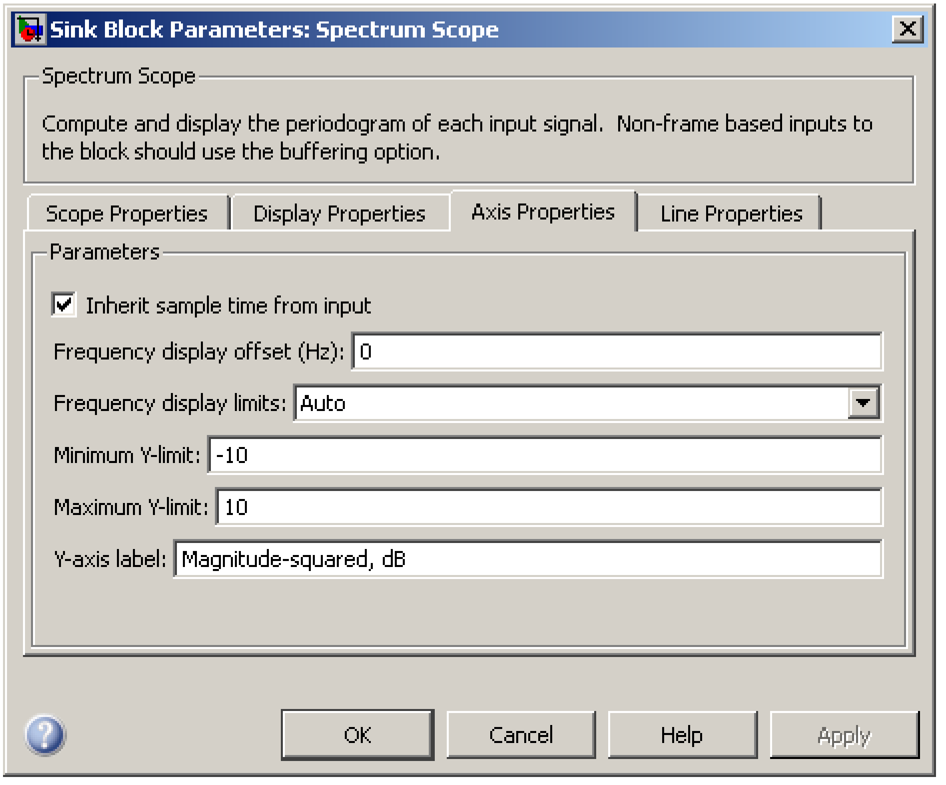
Compact display
Resizes the scope to fill the window. Tunable.

Open scope at start of simulation
Select this check box to open the scope at the start of the simulation. When you clear this parameter, the scope does not automatically open during the simulation. Tunable.

Open scope immediately
If the scope is not open during simulation, select this check box to open it. This parameter is visible only while the simulation is running.

Scope position
A four-element vector of the form `[left bottom width height]` specifying the position of the scope window. `(0,0)` is the lower-left corner of the display. Tunable.

Axis Properties Pane



Inherit sample time from input

If you select this check box, the block computes the time-domain sample period from the frame period and frame size of the frequency-domain input. Use this parameter only when the length of each frame of frequency-domain data equals the frame length of the time-domain data that generated it.

Sample time of original time series

Enter the sample period of the original time-domain signal.

Frequency display offset

The **Frequency display offset** parameter allows you to offset the range of values displayed on the frequency axis of the Spectrum Scope. The value specified in this field becomes the new label for the DC frequency (0 Hz). See the example in the "Axis Properties Pane" on page 2-1221 section for more information.

Frequency display limits

Select `Auto` to have the limits of the *x*-axis set for you automatically. Select `User-defined` to set the limits yourself in the **Minimum frequency** and **Maximum frequency** parameters.

Minimum frequency (Hz)

Specify the minimum frequency value of the *x*-axis in Hertz. This parameter is only visible if the **Frequency display limits** parameter is set to `User-defined`. Tunable.

Maximum frequency (Hz)

Specify the maximum frequency value of the *x*-axis in Hertz. This parameter is only visible if the **Frequency display limits** parameter is set to `User-defined`. Tunable.

Minimum Y-limit

Specify the minimum value of the *y*-axis. Setting this parameter is analogous to setting the `ymin` value of the MATLAB `axis` function.Tunable.

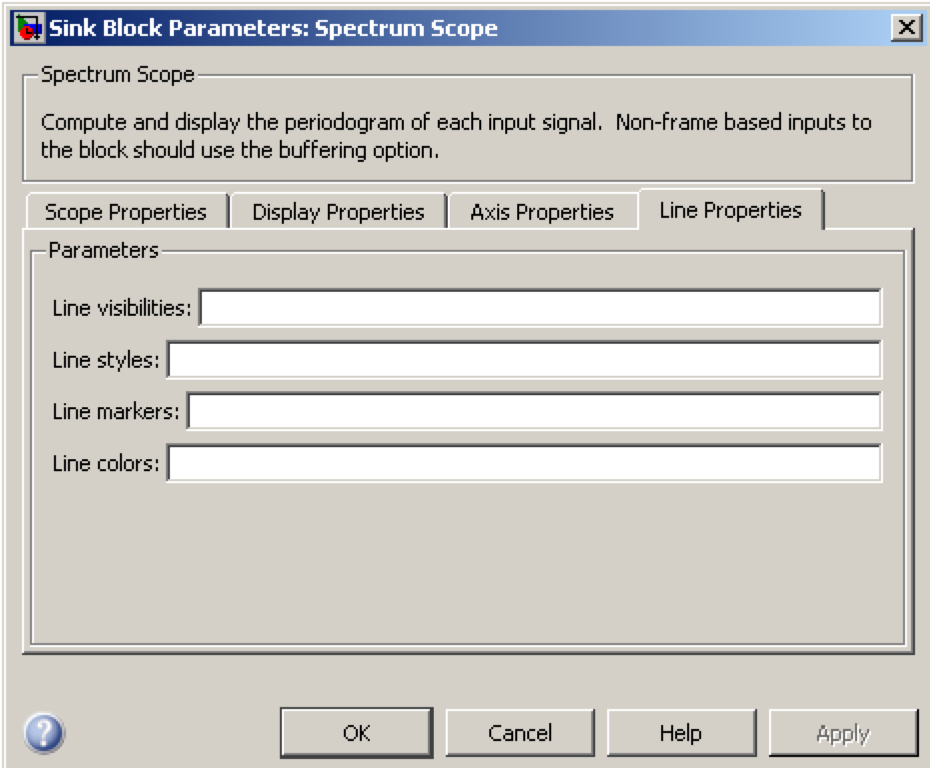
Maximum Y-limit

Specify the maximum value of the *y*-axis. Setting this parameter is analogous to setting the `ymax` value of the MATLAB `axis` function.Tunable.

Y-axis label

Specify text for the block to display to the left of the *y*-axis. Tunable.

Line Properties Pane



For more information about these parameters, see "Line Properties Pane" on page 2-1222 in the Vector Scope block reference page.

Line visibilities

Enter `on` or `off` to specify the visibility of the scope traces for various channels. Separate your choices for each channel with by a pipe (`|`) symbol. Tunable.

Line styles
: Enter the line styles of the scope traces for various channels using the MATLAB `line` function `LineStyle` formats. Separate your choices for each channel with by a pipe (`|`) symbol. Tunable.

Line markers
: Enter the line markers of the scope traces for various channels using the MATLAB `line` function `Marker` formats. Separate your choices for each channel with by a pipe (`|`) symbol. Tunable.

Line colors
: Enter the colors of the scope traces for various channels using the MATLAB `ColorSpec` formats. Separate your choices for each channel with by a pipe (`|`) symbol. Tunable.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

FFT	Signal Processing Blockset
Periodogram	Signal Processing Blockset
Short-Time FFT	Signal Processing Blockset
Vector Scope	Signal Processing Blockset
Window Function	Signal Processing Blockset

Stack

Purpose

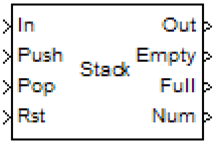
Store inputs into LIFO register

Library

Signal Management / Buffers

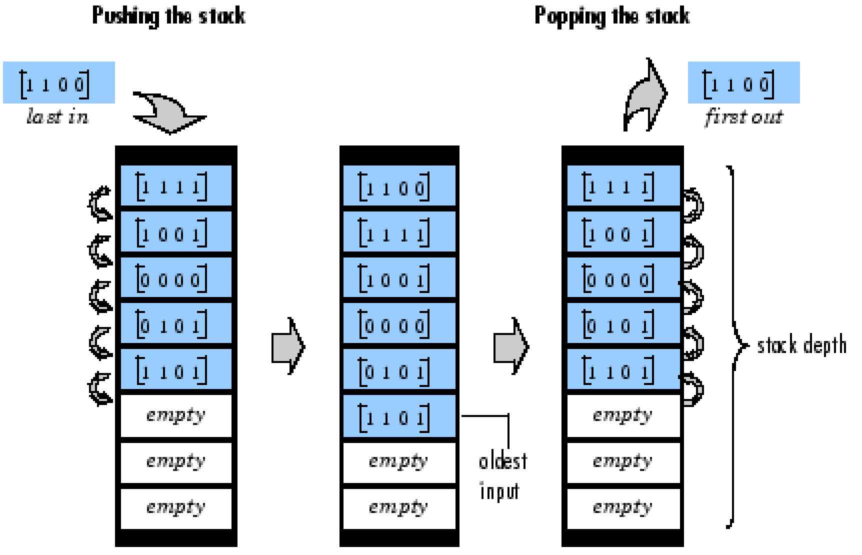
`dspbuff3`

Description



The Stack block stores a sequence of input samples in a last in, first out (LIFO) register. The register capacity is set by the **Stack depth** parameter, and inputs can be scalars, vectors, or matrices.

The block *pushes* the input at the `In` port onto the top of the stack when a trigger event is received at the `Push` port. When a trigger event is received at the `Pop` port, the block *pops* the top element off the stack and holds the `Out` port at that value. The last input to be pushed onto the stack is always the first to be popped off.



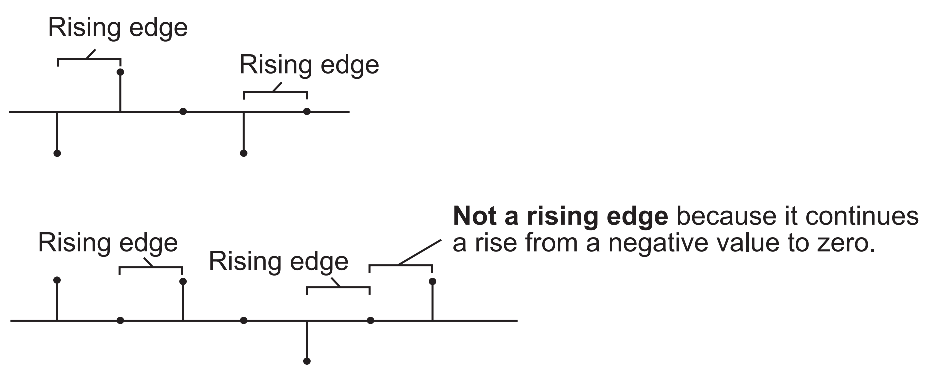
A trigger event at the optional `Rst` port empties the stack contents. When you select **Clear output port on reset**, then a trigger event at the `Rst` port empties the stack *and* sets the value at the `Out` port to zero. This setting also applies when a disabled subsystem containing the Stack block is reenabled; the `Out` port value is only reset to zero in this case when you select **Clear output port on reset**.

When two or more of the control input ports are triggered at the same time step, the operations are executed in the following order:

1. `Rst`
2. `Push`
3. `Pop`

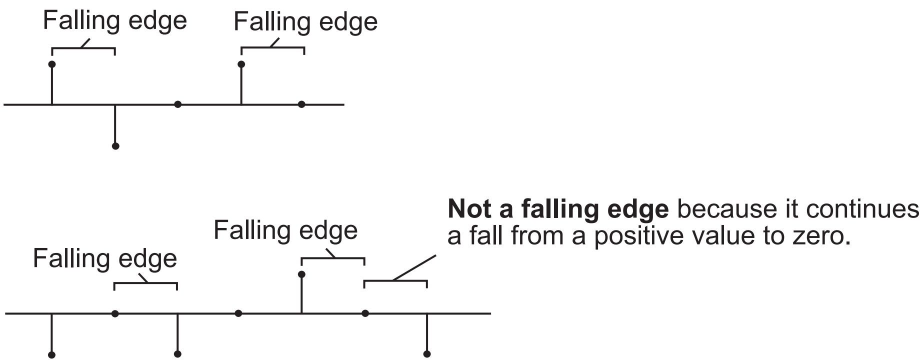
The rate of the trigger signal must be the same as the rate of the data signal input. You specify the triggering event for the `Push`, `Pop`, and `Rst` ports in the **Trigger type** pop-up menu:

- `Rising edge` — Triggers execution of the block when the trigger input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)



- `Falling edge` — Triggers execution of the block when the trigger input does one of the following:

- Falls from a positive value to a negative value or zero
- Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)



- `Either edge` — Triggers execution of the block when the trigger input is a `Rising edge` or `Falling edge` (as described above).
- `Non-zero sample` — Triggers execution of the block at each sample time that the trigger input is not zero.

Note When running simulations in the Simulink `MultiTasking` mode, sample-based trigger signals have a one-sample latency, and frame-based trigger signals have one frame of latency. Thus, there is a one-sample or one-frame delay between the time the block detects a trigger event, and when it applies the trigger. For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

Note If your model contains any referenced models that use the Stack block, you cannot simulate the top-level model in Simulink Accelerator mode.

The **Push full stack** parameter specifies the block's behavior when a trigger is received at the `Push` port but the register is full. The **Pop empty stack** parameter specifies the block's behavior when a trigger is received at the `Pop` port but the register is empty. The following options are available for both cases:

- `Ignore` — Ignore the trigger event, and continue the simulation.
- `Warning` — Ignore the trigger event, but display a warning message in the MATLAB command window.
- `Error` — Display an error dialog box and terminate the simulation.

Note The **Push full stack** and **Pop empty stack** parameters are diagnostic parameters. Like all diagnostic parameters on the Configuration Parameters dialog box, they are set to `Ignore` in the code generated for this block by Real-Time Workshop code generation software.

The **Push full stack** parameter additionally offers the **Dynamic reallocation** option, which dynamically resizes the register to accept as many additional inputs as memory permits. To find out how many elements are on the stack at a given time, enable the `Num` output port by selecting the **Show number of stack entries port** parameter.

Note When `Dynamic reallocation` is selected, the **System target file** parameter on the **Real-Time Workshop** pane of the Configuration Parameters dialog box must be set to `grt_malloc.tlc   Generic Real-Time Target with dynamic memory allocation`.

Examples

Example 1

The table below illustrates the Stack block's operation for a **Stack depth** of `4`, **Trigger type** of `Either edge`, and **Clear output port on reset** enabled. Because the block triggers on both rising and falling edges in this example, each transition from `1` to `0` or `0` to `1` in the `Push`, `Pop`, and `Rst` columns below represents a distinct trigger event. A `1` in the `Empty` column indicates an empty buffer, while a `1` in the `Full` column indicates a full buffer.

In	Push	Pop	Rst	Stack	Out	Empty	Full	Num
1	0	0	0	*top* *bottom*	0	1	0	0
2	1	0	0	*top* 2 *bottom*	0	0	0	1
3	0	0	0	*top* 3 2 *bottom*	0	0	0	2
4	1	0	0	*top* 4 3 2 *bottom*	0	0	0	3
5	0	0	0	*top* 5 4 3 2 *bottom*	0	0	1	4
6	0	1	0	*top* 4 3 2 *bottom*	5	0	0	3
7	0	0	0	*top* 3 2 *bottom*	4	0	0	2
8	0	1	0	*top* 2 *bottom*	3	0	0	1
9	0	0	0	*top* *bottom*	2	1	0	0

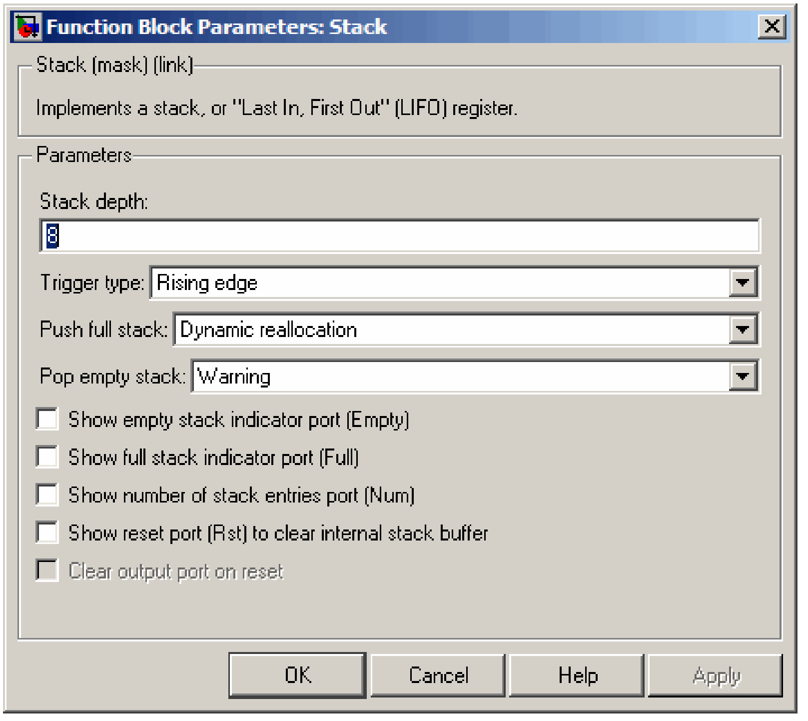
In	Push	Pop	Rst	Stack	Out	Empty	Full	Num
10	1	0	0		2	0	0	1
11	0	0	0		2	0	0	2
12	1	0	1		0	0	0	1

Note that at the last step shown, the `Push` and `Rst` ports are triggered simultaneously. The `Rst` trigger takes precedence, and the stack is first cleared and then pushed.

Example 2

The `dspqdemo` demo provides an example of the related Queue block.

Dialog Box



Stack depth
: The number of entries that the LIFO register can hold.

Trigger type
: The type of event that triggers the block's execution. The rate of the trigger signal must be the same as the rate of the data signal input.

Push full stack
: Response to a trigger received at the `Push` port when the register is full. Inputs to this port must have the same built-in data type as inputs to the `Pop` and `Rst` input ports.

When `Dynamic reallocation` is selected, the **System target file** parameter on the **Real-Time Workshop** pane of the Configuration Parameters dialog box must be set to `grt_malloc.tlc   Generic Real-Time Target with dynamic memory allocation`.

Pop empty stack
Response to a trigger received at the `Pop` port when the register is empty. Inputs to this port must have the same built-in data type as inputs to the `Push` and `Rst` input ports.

Show empty stack indicator port
Enable the `Empty` output port, which is high (1) when the stack is empty, and low (0) otherwise.

Show full stack indicator port
Enable the `Full` output port, which is high (1) when the stack is full, and low (0) otherwise. The `Full` port remains low when you select **Dynamic reallocation** from the **Push full stack** parameter.

Show number of stack entries port
Enable the `Num` output port, which tracks the number of entries currently on the stack. When inputs to the `In` port are double-precision values, the outputs from the `Num` port are double-precision values. Otherwise, the outputs from the `Num` port are 32-bit unsigned integer values.

Show reset port to clear internal stack buffer
Enable the `Rst` input port, which empties the stack when the trigger specified by the **Trigger type** is received. Inputs to this port must have the same built-in data type as inputs to the `Push` and `Pop` input ports.

Clear output port on reset
Reset the `Out` port to zero (in addition to clearing the stack) when a trigger is received at the `Rst` input port.

Supported Data Types

Port	Supported Data Types
In	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Push	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers Inputs to this port must have the same built-in data type as inputs to the Pop and Rst input ports
Pop	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers Inputs to this port must have the same built-in data type as inputs to the Push and Rst input ports.

Port	Supported Data Types
Rst	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers Inputs to this port must have the same built-in data type as inputs to the Push and Pop input ports.
Out	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Empty	• Double-precision floating point • Boolean
Full	• Double-precision floating point • Boolean
Num	• Double-precision floating point The block outputs a double-precision floating-point value at this port when the data type of the In port is double-precision floating-point. • 32-bit unsigned integers The block outputs a 32-bit unsigned integer value at this port when the data type of the In port is anything other than double-precision floating-point.

See Also

Buffer	Signal Processing Blockset
Delay Line	Signal Processing Blockset
Queue	Signal Processing Blockset

Purpose

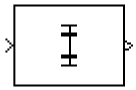
Find standard deviation of input or sequence of inputs

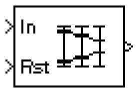
Library

Statistics

`dspstat3`

Description





The Standard Deviation block computes the standard deviation of each row or column of the input, along vectors of a specified dimension of the input, or of the entire input. The Standard Deviation block can also track the standard deviation of a sequence of inputs over a period of time. The **Running standard deviation** parameter selects between basic operation and running operation.

Basic Operation

When you do not select the **Running standard deviation** check box, the block computes the standard deviation of each row or column of the input, along vectors of a specified dimension of the input, or of the entire input at each individual sample time, and outputs the array y. Each element in y contains the standard deviation of the corresponding column, row, vector, or entire input. The output y depends on the setting of the **Find the standard deviation value over** parameter. For example, consider a 3-dimensional input signal of size M-by-N-by-P:

- `Entire input` — The output at each sample time is a scalar that contains the standard deviation of the entire input. In this mode, the output is always sample based.

  ```
  y = std(u(:))     % Equivalent MATLAB code
  ```

- `Each Row` — The output at each sample time consists of an M-by-1-by-P array, where each element contains the standard deviation of each vector over the second dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is an M-by-1 column vector. In this mode, the frame status of the output is the same as that of the input.

  ```
  y = std(u,0,2)     % Equivalent MATLAB code
  ```

- `Each Column` — The output at each sample time consists of a 1-by-N-by-P array, where each element contains the standard deviation of each vector over the first dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is a 1-by-N row vector. In this mode, the frame status of the output is the same as that of the input.

  ```
  y = std(u,0,1)     % Equivalent MATLAB code
  ```

  For convenience, length-M 1-D vector inputs are treated as M-by-1 column vectors when the block is in this mode. Sample-based length-M row vector inputs are also treated as M-by-1 column vectors when the **Treat sample-based row input as a column** check box is selected.

- `Specified Dimension` — The output at each sample time depends on **Dimension**. If **Dimension** is set to `1`, the output is the same as when you select `Each column`. If **Dimension** is set to `2`, the output is the same as when you select `Each row`. If **Dimension** is set to `3`, the output at each sample time is an M-by-N matrix containing the standard deviation of each vector over the third dimension of the input. In this mode, the frame status of the output is the same as that of the input.

  ```
  y = std(u,0,Dimension)     % Equivalent MATLAB code
  ```

For purely real or purely imaginary inputs, the standard deviation of the jth column of an M-by-N input matrix is the square root of its variance:

$$y_j = \sigma_j = \sqrt{\frac{\sum_{i=1}^{M} \left| u_{ij} - \mu_j \right|^2}{M-1}} \qquad 1 \le j \le N$$

For complex inputs, the output is the *total standard deviation*, which equals the square root of the *total variance*, or the square root of the sum of the variances of the real and imaginary parts. The standard deviation of each column in an M-by-N input matrix is given by:

$$\sigma_j = \sqrt{\sigma_{j,\mathrm{Re}}^2 + \sigma_{j,\mathrm{Im}}^2}$$

Note The total standard deviation does *not* equal the sum of the real and imaginary standard deviations.

Running Operation

When you select the **Running standard deviation** check box, the block tracks the standard deviation of successive inputs to the block. For sample-based M-by-N inputs, the output is a sample-based M-by-N matrix with each element y_{ij} containing the standard deviation of element u_{ij} over all inputs since the last reset. For frame-based M-by-N inputs, the output is a frame-based M-by-N matrix with each element y_{ij} containing the standard deviation of the jth column over all inputs since the last reset, up to and including element u_{ij} of the current input.

N-D signals cannot be frame based. When the block is set to `Running` mode, each element of the N-D signal is treated as a separate channel.

There are $\prod d_i$ channels, where d_i is the size of the ith dimension.

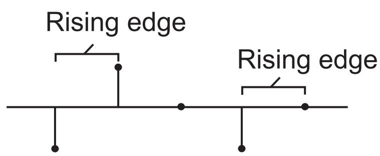
Resetting the Running Standard Deviation

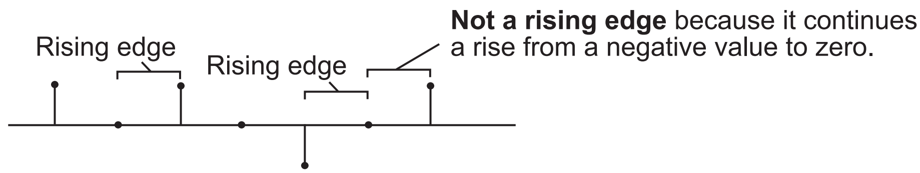
The block resets the running standard deviation whenever a reset event is detected at the optional `Rst` port. The reset sample time must be a positive integer multiple of the input sample time.

You specify the reset event in the **Reset port** parameter:

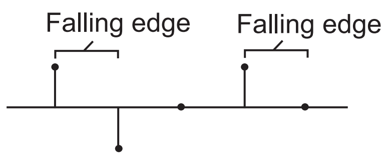
- `None` disables the `Rst` port.
- `Rising edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Rises from a negative value to a positive value or zero

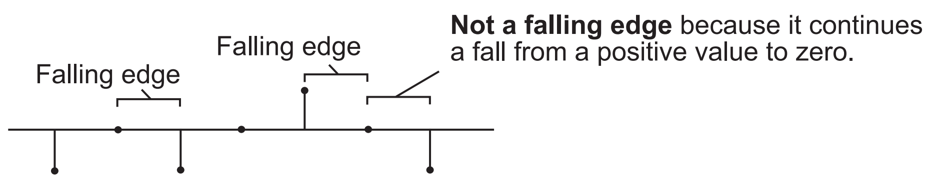
- Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)





- `Falling edge` — Triggers a reset operation when the `Rst` input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)





- `Either edge` — Triggers a reset operation when the `Rst` input is a `Rising edge` or `Falling edge` (as described earlier)
- `Non-zero sample` — Triggers a reset operation at each sample time that the `Rst` input is not zero

---

Note When running simulations in the Simulink MultiTasking mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset. For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

ROI Processing

To calculate the statistical value within a particular region of interest (ROI) of the input, select the **Enable ROI processing** check box. This option is only available when the **Find the standard deviation value over** parameter is set to `Entire input` and the **Running standard deviation** check box is not selected. ROI processing is only supported for 2-D inputs.

Note Full ROI processing is only available to users who have a Video and Image Processing Blockset license. If you only have a Signal Processing Blockset license, you can still use ROI processing, but are limited to the **ROI type** `Rectangles`.

Use the **ROI type** parameter to specify whether the ROI is a rectangle, line, label matrix, or binary mask. A binary mask is a binary image that enables you to specify which pixels to highlight, or select. In a label matrix, pixels equal to 0 represent the background, pixels equal to 1 represent the first object, pixels equal to 2 represent the second object, and so on. When the **ROI type** parameter is set to `Label matrix`, the Label and Label Numbers ports appear on the block. Use the Label

Numbers port to specify the objects in the label matrix for which the block calculates statistics. The input to this port must be a vector of scalar values that correspond to the labeled regions in the label matrix. For more information about the format of the input to the ROI port when the ROI is a rectangle or a line, see the Draw Shapes block reference page.

For rectangular ROIs, use the **ROI portion to process** parameter to specify whether to calculate the statistical value for the entire ROI or just the ROI perimeter.

Use the **Output** parameter to specify the block output. The block can output separate statistical values for each ROI or the statistical value for all specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select `Binary mask`.

If, for the **ROI type** parameter, you select `Rectangles` or `Lines`, the **Output flag indicating if ROI is within image bounds** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual statistics for each ROI

Flag Port Output	Description
0	ROI is completely outside the input image.
1	ROI is completely or partially inside the input image.

Output = Single statistic for all ROIs

Flag Port Output	Description
0	All ROIs are completely outside the input image.
1	At least one ROI is completely or partially inside the input image.

If the ROI is partially outside the image, the block only computes the statistical values for the portion of the ROI that is within the image.

If, for the **ROI type** parameter, you select `Label matrix`, the **Output flag indicating if input label numbers are valid** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual statistics for each ROI

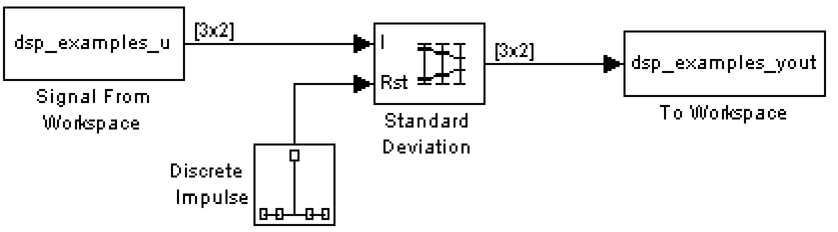
Flag Port Output	Description
0	Label number is not in the label matrix.
1	Label number is in the label matrix.

Output = Single statistic for all ROIs

Flag Port Output	Description
0	None of the label numbers are in the label matrix.
1	At least one of the label numbers is in the label matrix.

Examples

The Standard Deviation block in the next model calculates the running standard deviation of a frame-based 3-by-2 (two-channel) matrix input, `u`. The running standard deviation is reset at *t*=2 by an impulse to the block's `Rst` port.



The Standard Deviation block has the following settings:

- **Running standard deviation** = ☑
- **Reset port** = `Non-zero sample`

The Signal From Workspace block has the following settings:

- **Signal** = `dsp_examples_u`
- **Sample time** = `1/3`
- **Samples per frame** = `3`

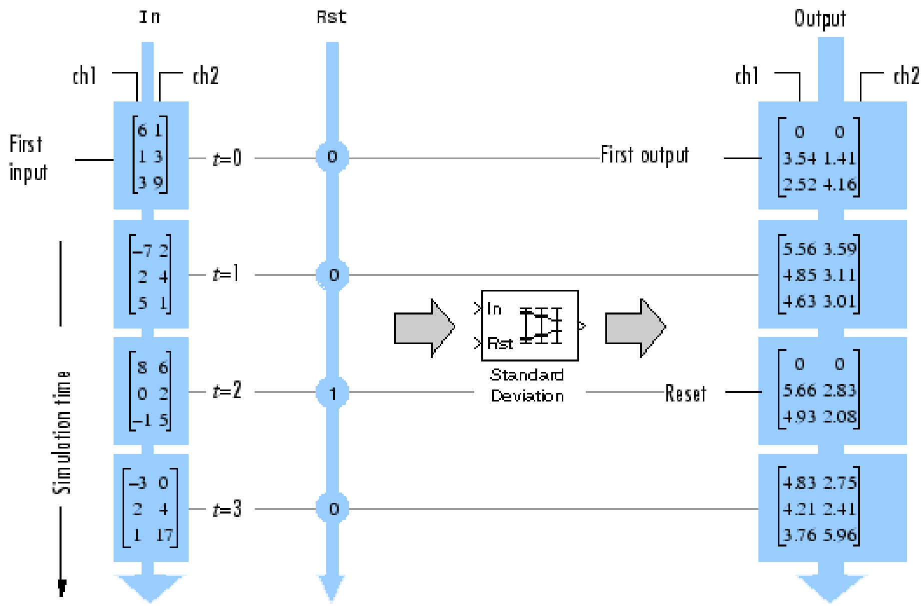
where

```
dsp_examples_u = [6 1 3 -7 2 5 8 0 -1 -3 2 1;1 3 9 2 4 1 6 2 5 0 4 17]'
```

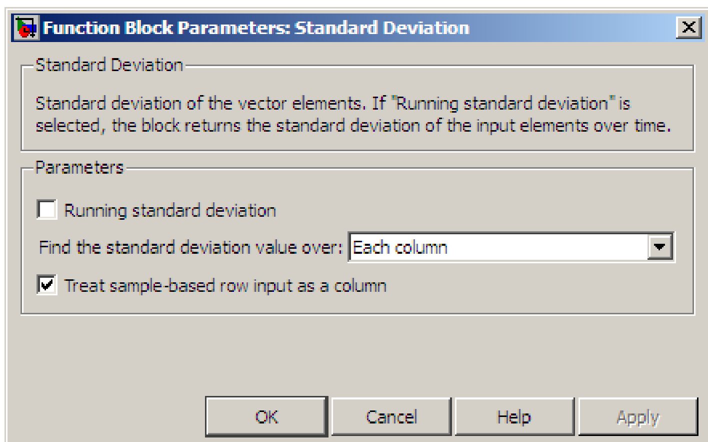
The Discrete Impulse block has the following settings:

- **Delay (samples)** = `2`
- **Sample time** = `1`
- **Samples per frame** = `1`

The block's operation is shown in the next figure.



Dialog Box



Running standard deviation
Enables running operation when selected.

Reset port

Determines the reset event that causes the block to reset the running standard deviation. The reset signal rate must be a positive integer multiple of the rate of the data signal input. This parameter is enabled only when you select **Running standard deviation**. For more information, see "Resetting the Running Standard Deviation" on page 2-1253.

Find the standard deviation value over

Specify whether to find the standard deviation value along rows, columns, entire input, or the dimension specified in the **Dimension** parameter. For more information, see "Basic Operation" on page 2-1251.

Treat sample-based row input as a column

Select to treat sample-based length-M row vector inputs as M-by-1 column vectors. This parameter is only visible when the **Find the standard deviation value over** parameter is set to `Each column`.

Dimension

Specify the dimension (one-based value) of the input signal, over which the standard deviation is computed. The value of this parameter cannot exceed the number of dimensions in the input signal. This parameter is only visible when the **Find the standard deviation value over** parameter is set to `Specified dimension`.

Enable ROI Processing

Select this check box to calculate the statistical value within a particular region of each image. This parameter is only available when the **Find the standard deviation value over** parameter is set to `Entire input`, and the block is not in running mode.

Note Full ROI processing is only available to users who have a Video and Image Processing Blockset license. If you only have a Signal Processing Blockset license, you can still use ROI processing, but are limited to the **ROI type** `Rectangles`.

ROI type
: Specify the type of ROI you want to use. Your choices are `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`.

ROI portion to process
: Specify whether you want to calculate the statistical value for the entire ROI or just the ROI perimeter. This parameter is only visible if, for the **ROI type** parameter, you specify `Rectangles`.

Output
: Specify the block output. The block can output a vector of separate statistical values for each ROI or a scalar value that represents the statistical value for all the specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select `Binary mask`.

Output flag

Output flag indicating if ROI is within image bounds

Output flag indicating if label numbers are valid

When you select either of these check boxes, the Flag port appears on the block. For a description of the Flag port output, see the tables in "ROI Processing" on page 2-1255.

The **Output flag indicating if ROI is within image bounds** check box is only visible when you select `Rectangles` or `Lines` as the **ROI type**.

The **Output flag indicating if label numbers are valid** check box is only visible when you select `Label matrix` for the **ROI type** parameter.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point
Reset	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
ROI	Rectangles and lines: • Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers Binary Mask: • Boolean
Label	• 8-, 16-, and 32-bit unsigned integers
Label Numbers	• 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
Output	• Double-precision floating point • Single-precision floating point
Flag	• Boolean

See Also

Mean	Signal Processing Blockset
RMS	Signal Processing Blockset
Variance	Signal Processing Blockset
`std`	MATLAB

Submatrix

Purpose

Select subset of elements (submatrix) from matrix input

Library

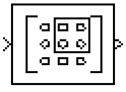
- Math Functions / Matrices and Linear Algebra / Matrix Operations

  `dspmtrx3`

- Signal Management / Indexing

  `dspindex`

Description



The Submatrix block extracts a contiguous submatrix from the M-by-N input matrix `u`. A length-M 1-D vector input is treated as an M-by-1 matrix. The **Row span** parameter provides three options for specifying the range of rows in `u` to be retained in submatrix output `y`:

- `All rows`

  Specifies that `y` contains all M rows of `u`.

- `One row`

  Specifies that `y` contains only one row from `u`. The **Starting row** parameter (described below) is enabled to allow selection of the desired row.

- `Range of rows`

  Specifies that `y` contains one or more rows from `u`. The **Row** and **Ending row** parameters (described below) are enabled to allow selection of the desired range of rows.

The **Column span** parameter contains a corresponding set of three options for specifying the range of columns in `u` to be retained in submatrix `y`: `All columns`, `One column`, or `Range of columns`. The `One column` option enables the **Column** parameter, and `Range of columns` options enable the **Starting column** and **Ending column** parameters.

The output has the same frame status as the input.

Range Specification Options

When you select `One row` or `Range of rows` from the **Row span** parameter, you specify the desired row or range of rows in the **Row** parameter, or the **Starting row** and **Ending row** parameters. Similarly, when you select `One column` or `Range of columns` from the **Column span** parameter, you specify the desired column or range of columns in the **Column** parameter, or the **Starting column** and **Ending column** parameters.

The **Row**, **Column**, **Starting row** or **Starting column** can be specified in six ways:

- `First`

  For rows, this specifies that the first row of `u` should be used as the first row of `y`. When all columns are to be included, this is equivalent to `y(1,:)  = u(1,:)`.

  For columns, this specifies that the first column of `u` should be used as the first column of `y`. When all rows are to be included, this is equivalent to `y(:,1) = u(:,1)`.

- `Index`

  For rows, this specifies that the row of `u`, `firstrow`, forward-indexed by the **Row index** parameter or the **Starting row index** parameter, should be used as the first row of `y`. When all columns are to be included, this is equivalent to `y(1,:)  = u(firstrow,:)`.

  For columns, this specifies that the column of `u`, forward-indexed by the **Column index** parameter or the **Starting column index** parameter, `firstcol`, should be used as the first column of `y`. When all rows are to be included, this is equivalent to `y(:,1) = u(:,firstcol)`.

- `Offset from last`

  For rows, this specifies that the row of `u` offset from row M by the **Row offset** or **Starting row offset** parameter, `firstrow`, should be used as the first row of `y`. When all columns are to be included, this is equivalent to `y(1,:)  = u(M-firstrow,:)`.

For columns, this specifies that the column of `u` offset from column N by the **Column offset** or **Starting column offset** parameter, `firstcol`, should be used as the first column of `y`. When all rows are to be included, this is equivalent to `y(:,1) = u(:,N-firstcol)`.

- `Last`

  For rows, this specifies that the last row of `u` should be used as the only row of `y`. When all columns are to be included, this is equivalent to `y = u(M,:)`.

  For columns, this specifies that the last column of `u` should be used as the only column of `y`. When all rows are to be included, this is equivalent to `y = u(:,N)`.

- `Offset from middle`

  For rows, this specifies that the row of `u` offset from row M/2 by the **Starting row offset** parameter, `firstrow`, should be used as the first row of `y`. When all columns are to be included, this is equivalent to `y(1,:)  = u(M/2-firstrow,:)`.

  For columns, this specifies that the column of `u` offset from column N/2 by the **Starting column offset** parameter, `firstcol`, should be used as the first column of `y`. When all rows are to be included, this is equivalent to `y(:,1) = u(:,N/2-firstcol)`.

- `Middle`

  For rows, this specifies that the middle row of `u` should be used as the only row of `y`. When all columns are to be included, this is equivalent to `y = u(M/2,:)`.

  For columns, this specifies that the middle column of `u` should be used as the only column of `y`. When all rows are to be included, this is equivalent to `y = u(:,N/2)`.

The **Ending row** or **Ending column** can similarly be specified in five ways:

- `Index`

  For rows, this specifies that the row of `u` forward-indexed by the **Ending row index** parameter, `lastrow`, should be used as the last row of `y`. When all columns are to be included, this is equivalent to `y(end,:)  = u(lastrow,:)`.

  For columns, this specifies that the column of `u` forward-indexed by the **Ending column index** parameter, `lastcol`, should be used as the last column of `y`. When all rows are to be included, this is equivalent to `y(:,end) = u(:,lastcol)`.

- `Offset from last`

  For rows, this specifies that the row of `u` offset from row M by the **Ending row offset** parameter, `lastrow`, should be used as the last row of `y`. When all columns are to be included, this is equivalent to `y(end,:)  = u(M-lastrow,:)`.

  For columns, this specifies that the column of `u` offset from column N by the **Ending column offset** parameter, `lastcol`, should be used as the last column of `y`. When all rows are to be included, this is equivalent to `y(:,end) = u(:,N-lastcol)`.

- `Last`

  For rows, this specifies that the last row of `u` should be used as the last row of `y`. When all columns are to be included, this is equivalent to `y(end,:)  = u(M,:)`.

  For columns, this specifies that the last column of `u` should be used as the last column of `y`. When all rows are to be included, this is equivalent to `y(:,end) = u(:,N)`.

- `Offset from middle`

  For rows, this specifies that the row of `u` offset from row M/2 by the **Ending row offset** parameter, `lastrow`, should be used as the last row of `y`. When all columns are to be included, this is equivalent to `y(end,:)  = u(M/2-lastrow,:)`.

For columns, this specifies that the column of `u` offset from column N/2 by the **Ending column offset** parameter, `lastcol`, should be used as the last column of `y`. When all rows are to be included, this is equivalent to `y(:,end) = u(:,N/2-lastcol)`.

- `Middle`

  For rows, this specifies that the middle row of `u` should be used as the last row of `y`. When all columns are to be included, this is equivalent to `y(end,:)  = u(M/2,:)`.

  For columns, this specifies that the middle column of `u` should be used as the last column of `y`. When all rows are to be included, this is equivalent to `y(:,end) = u(:,N/2)`.

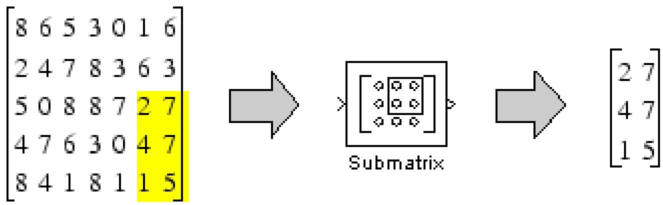
This block supports Simulink virtual buses.

Examples

To extract the lower-right 3-by-2 submatrix from a 5-by-7 input matrix, enter the following set of parameters:

- **Row span** = `Range of rows`
- **Starting row** = `Index`
- **Starting row index** = `3`
- **Ending row** = `Last`
- **Column span** = `Range of columns`
- **Starting column** = `Offset from last`
- **Starting column offset** = `1`
- **Ending column** = `Last`

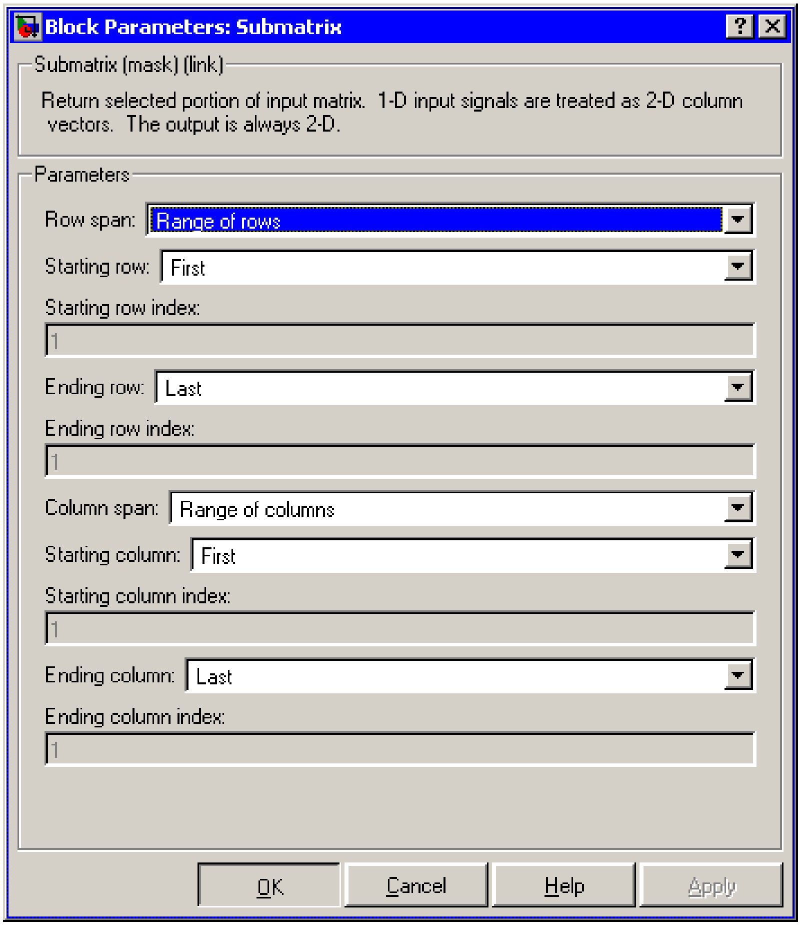
The figure below shows the operation for a 5-by-7 matrix with random integer elements, `randint(5,7,10)`.



There are often several possible parameter combinations that select the *same* submatrix from the input. For example, instead of specifying `Last` for **Ending column**, you could select the same submatrix by specifying

- **Ending column** = `Index`
- **Ending column index** = `7`

Dialog Box



The parameters displayed in the dialog box vary for different menu combinations. Only some of the parameters listed below are visible in the dialog box at any one time.

Row span

The range of input rows to be retained in the output. Options are `All rows`, `One row`, or `Range of rows`.

Row/Starting row
The input row to be used as the first row of the output. **Row** is enabled when you select `One row` from **Row span**, and **Starting row** when you select `Range of rows` from **Row span**.

Row index/Starting row index
The index of the input row to be used as the first row of the output. **Row index** is enabled when you select `Index` from Row, and **Starting row index** when you select `Index` from **Starting row**.

Row offset/Starting row offset
The offset of the input row to be used as the first row of the output. **Row offset** is enabled when you select `Offset from middle` or `Offset from last` from **Row**, and Starting row offset is enabled when you select `Offset from middle` or `Offset from last` from **Starting row**.

Ending row
The input row to be used as the last row of the output. This parameter is enabled when you select `Range of rows` from **Row span** and you select any option but `Last` from **Starting row**.

Ending row index
The index of the input row to be used as the last row of the output. This parameter is enabled when you select `Index` from **Ending row**.

Ending row offset
The offset of the input row to be used as the last row of the output. This parameter is enabled when you select `Offset from middle` or `Offset from last` from **Ending row**.

Column span
The range of input columns to be retained in the output. Options are `All columns`, `One column`, or `Range of columns`.

Column/Starting column
The input column to be used as the first column of the output. **Column** is enabled when you select `One column` from **Column**

span, and **Starting column** is enabled when you select `Range of columns` from **Column span**.

Column index/Starting column index
The index of the input column to be used as the first column of the output. **Column index** is enabled when you select `Index` from Column, and **Starting column index** is enabled when you select `Index` from **Starting column**.

Column offset/Starting column offset
The offset of the input column to be used as the first column of the output. **Column offset** is enabled when you select `Offset from middle` or `Offset from last` from Column. **Starting column offset** is enabled when you select `Offset from middle` or `Offset from last` from **Starting column**.

Ending column
The input column to be used as the last column of the output. This parameter is enabled when you select `Range of columns` from **Column span** and you select any option but `Last` from **Starting column**.

Ending column index
The index of the input column to be used as the last column of the output. This parameter is enabled when you select `Index` from **Ending column**.

Ending column offset
The offset of the input column to be used as the last column of the output. This parameter is enabled when you select `Offset from middle` or `Offset from last` from **Ending column**.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated

See Also

Reshape	Simulink
Selector	Simulink
Variable Selector	Signal Processing Blockset
`reshape`	MATLAB

See “Splitting Multichannel Sample-Based Signals into Several Multichannel Signals” for related information.

Purpose

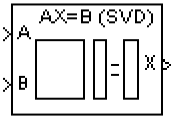
Solve AX=B using singular value decomposition

Library

Math Functions / Matrices and Linear Algebra / Linear System Solvers

`dspsolvers`

Description

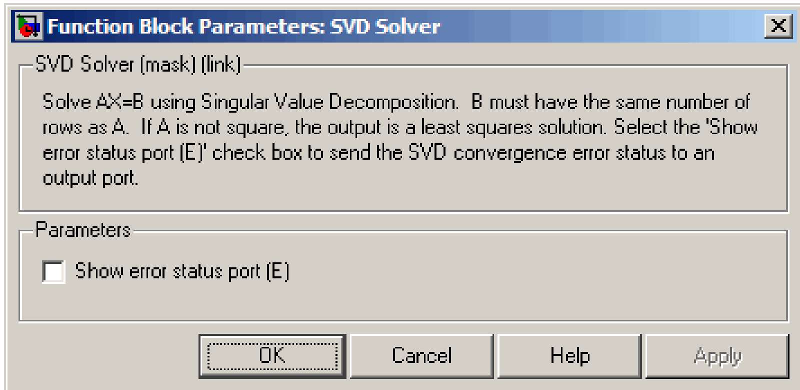


The SVD Solver block solves the linear system AX=B, which can be overdetermined, underdetermined, or exactly determined. The system is solved by applying singular value decomposition (SVD) factorization to the M-by-N matrix A, at the A port. The input to the B port is the right side M-by-L matrix, B. A length-M 1-D vector input at either port is treated as an M-by-1 matrix.

The output at the X port is the N-by-L matrix, X. X is always sample based, and is chosen to minimize the sum of the squares of the elements of B-AX (the residual). When B is a vector, this solution minimizes the vector 2-norm of the residual. When B is a matrix, this solution minimizes the matrix Frobenius norm of the residual. In this case, the columns of X are the solutions to the L corresponding systems AX_k=B_k, where B_k is the kth column of B, and X_k is the kth column of X.

X is known as the minimum-norm-residual solution to AX=B. The minimum-norm-residual solution is unique for overdetermined and exactly determined linear systems, but it is not unique for underdetermined linear systems. Thus when the SVD Solver block is applied to an underdetermined system, the output X is chosen such that the number of nonzero entries in X is minimized.

Dialog Box



Show error status port

Select to enable the E output port, which reports a failure to converge. The possible values you can receive on the port are:

- `0` — The singular value decomposition calculation converges.
- `1` — The singular value decomposition calculation does not converge.

If the singular value decomposition calculation fails to converge, the output at port X is an undefined matrix of the correct size.

Supported Data Types

Port	Supported Data Types
A	• Double-precision floating point • Single-precision floating point
B	• Double-precision floating point • Single-precision floating point
X	• Double-precision floating point • Single-precision floating point
E	• Boolean

See Also

Autocorrelation LPC	Signal Processing Blockset
Cholesky Solver	Signal Processing Blockset
LDL Solver	Signal Processing Blockset
Levinson-Durbin	Signal Processing Blockset
LU Inverse	Signal Processing Blockset
Pseudoinverse	Signal Processing Blockset
QR Solver	Signal Processing Blockset
Singular Value Decomposition	Signal Processing Blockset

See “Linear System Solvers” for related information.

Purpose

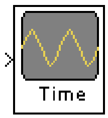
Display time-domain signals

Library

Signal Processing Sinks

`dspsnks4`

Description



Note Prior to R2010a, the Signal Processing Blockset Time Scope block was an implementation of the Simulink Scope block. For information about working with the Simulink Scope block, see the Simulink Scope reference page.

The Time Scope block displays signals in the time domain. The block accepts input signals with the following characteristics:

- Discrete sample time
- Real-valued
- Fixed or variable size
- Floating- or fixed-point data type
- Frame or sample based
- N-dimensional

Inputs to the scope can be frame based or sample based. However, you must set the **Input processing** parameter to specify how the block should interpret the input signal. The scope always treats 1-D signals as column vectors.

You can use the Time Scope block in models running in Normal or Accelerator simulation modes. The software does not support this block in models running in Rapid Accelerator or External mode. For more information about these modes, see “How the Acceleration Modes Work” in the *Simulink User’s Guide*.

You can use the Time Scope block inside of subsystems, but not in conditionally executed subsystems. See “About Conditional Subsystems” in the *Simulink User’s Guide*.

For an example that uses the Time Scope block, see the “Displaying Time-Domain Data in the Time Scope” section in the *Signal Processing Blockset User’s Guide*.

For more information on the graphical user interface of the Time Scope block, see the following sections:

- “Toolbar” on page 2-1280
- “Playback Toolbar” on page 2-1282
- “Core Pane” on page 2-1283
- “Visuals Pane” on page 2-1285
- “Tools Pane” on page 2-1288

Displaying Multiple Signals

You can configure the Time Scope block to display multiple signals on the same axes. You can set the number of input ports on the Time Scope block in the following ways:

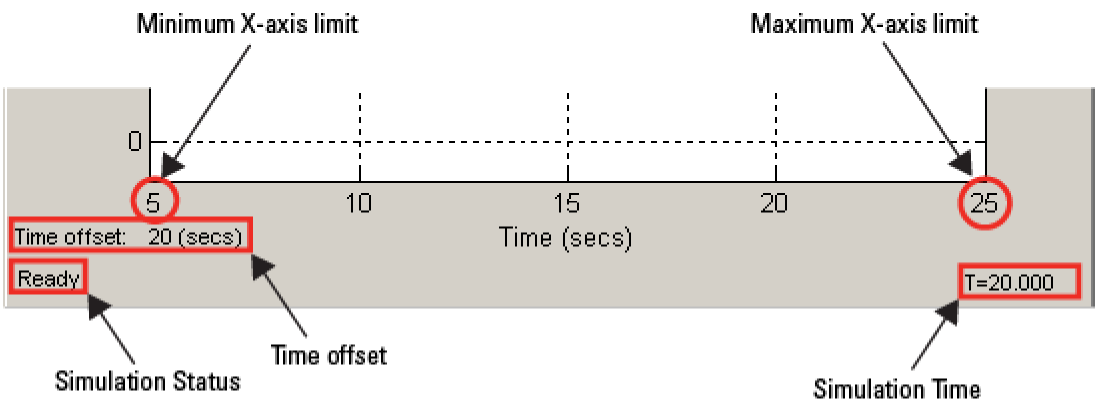
- Right-click the Time Scope block in your model and point your cursor to the **Number of Input Ports** item on the context menu. You can then select the number of input ports for the Time Scope block. To configure a Time Scope block to have more than three input ports, select **More**, and enter a number for the **Number of input ports** parameter.
- Open the Time Scope window by double-clicking the Time Scope block in your model. To specify how many input ports the Time Scope block should have, select **File > Number of Input Ports** from the scope menu.

Note When you provide multiple inputs to the Time Scope, all input signals must have the same data type, sample rate, and number of rows.

Signal Display

The value you enter for the **Time range** parameter on the **Axis Properties** tab of the **Time Scope — Visuals:Time Domain Options** dialog box defines the length of simulation time for which the Time Scope displays data. For example, if you set the **Time range** to 20 seconds, the scope displays 20 seconds worth of simulation data at a time. The values on the X-axis of the scope remain the same throughout simulation.

To communicate the simulation time that corresponds to the current display, the scope uses the **Time offset** and **Simulation time** indicators on the scope window. The following figure highlights these and other important aspects of the Time Scope window.



- **Minimum X-axis limit** — The scope sets the minimum X-axis limit using the value of the **Time display offset** parameter on the **Axis Properties** tab of the **Time Scope — Visuals:Time Domain Options** dialog box.

- **Maximum X-axis limit** — The scope sets the maximum X-axis limit by summing the value of **Time display offset** parameter with the value of the **Time range** parameter. You can set the values of these parameters on the **Axis Properties** tab of the **Time Scope — Visuals:Time Domain Options** dialog box.
- **Time offset** — The `Time offset` is always a non-negative integer multiple of the **Time range** parameter, where `0` ≤ `Time offset` ≤ **Simulation time**. For example, if you set the **Time range** to `20` seconds, and you see a `Time offset` of `0 (secs)` on the scope window, the scope is displaying data for the first `0` to `20` seconds of simulation time. When you see the `Time offset` change to `20 (secs)`, the scope is displaying data for simulation times is greater than `20` seconds and less than or equal to `40` seconds. The scope continues to update the `Time offset` at `20` second intervals until the simulation is complete.
- **Simulation Status** — Provides the current status of the model simulation. Can be one of the following:
  - `Initializing`
  - `Ready`
  - `Running`
  - `Paused`

  The **Simulation Status** is part of the **Status Bar** in the Time Scope window. You can choose to hide or display the entire **Status Bar** by selecting **View > Status Bar** from the Time Scope menu.
- **Simulation Time** — When the model is running or simulation has been paused, the scope displays the current simulation time. If the model simulation completes or is stopped, the scope displays the time at which the simulation stopped. The **Simulation time** is part of the **Status Bar** in the Time Scope window. You can choose to hide or display the entire **Status Bar** by selecting **View > Status Bar** from the Time Scope menu.

Toolbar

The Time Scope toolbar contains the following buttons.

Icon	Menu Location	Shortcut Keys	Description
	**Tools > Zoom In**	N/A	When this tool is active, you can zoom in on the scope window. To do so, click in the center of your area of interest, or click and drag your cursor to draw a rectangular area of interest inside the scope window.
	**Tools > Zoom X**	N/A	When this tool is active, you can zoom in on the X-axis. To do so, click inside the scope window, or click and drag your cursor along the X-axis over your area of interest.
	**Tools > Zoom Y**	N/A	When this tool is active, you can zoom in on the Y-axis. To do so, click inside the scope window, or click and drag your cursor along the Y-axis over your area of interest.
	**Tools > Scale Axes Limits**	**Ctrl+A**	Click this button to scale the axes in the active scope window. Alternatively, you can enable automatic axes scaling by selecting one of the following options from the **Tools** menu: • **Automatically Scale Axes Limits** — When you select this option, the scope scales the axes as needed during simulation. • **Scale Axes Limits at Stop** — When you select this option, the scope scales the axes each time the simulation is stopped.

You can control whether this toolbar appears in the scope window by selecting **View > Toolbar** from the scope menu.

Playback Toolbar

The Playback Toolbar contains the following buttons.

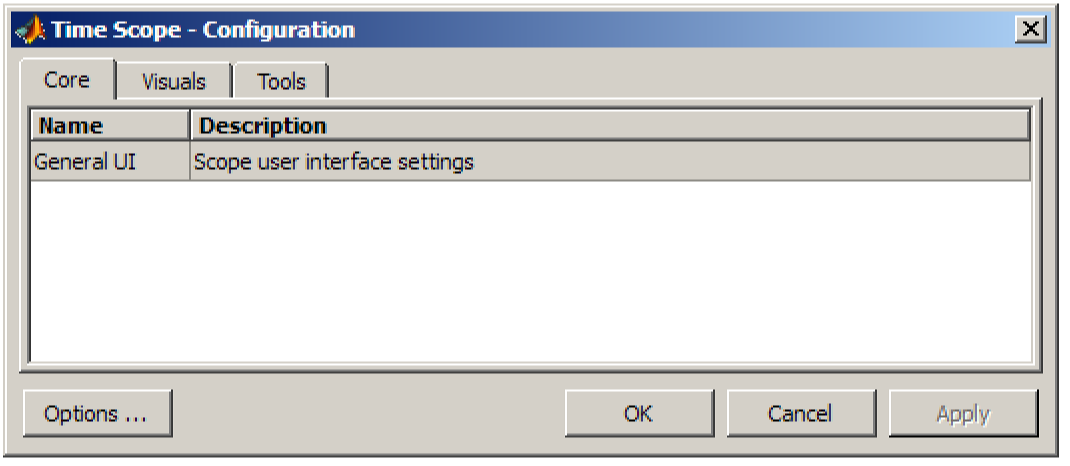
Icon	Menu Location	Shortcut Keys	Description
	**Playback > Stop**	**Ctrl+T**, **s**	Stop the model simulation. This button appears only when the model simulation is running or paused.
	**Playback > Start**	**Ctrl+T**, **p**, **Space**	Start the model simulation. This button appears only when the model simulation is stopped.
	**Playback > Continue**	**p**, **Space**	Continue the model simulation. This button appears only when the model simulation is paused.
	**Playback > Pause**	**p**, **Space**	Pause the model simulation. This button appears only when the model simulation is running.
	**Playback > Step Forward**	**Right arrow**, **Page Down**	Advance the model simulation forward by one time step. This button starts the model simulation, allows it to run for one time step, and then pauses it again. The scope window then updates with the latest data.
	**View > Highlight Simulink Block**	**Ctrl+L**	Bring the model window forward and highlight the block whose display you are currently viewing. The Time Scope block that corresponds to the active Time Scope window will flash three times in the model.

You can control whether this toolbar appears in the scope window by selecting **View > Playback Toolbar** from the scope menu.

To see a full listing of the shortcut keys for these playback controls, select **Help > Keyboard Command Help** from the scope menu.

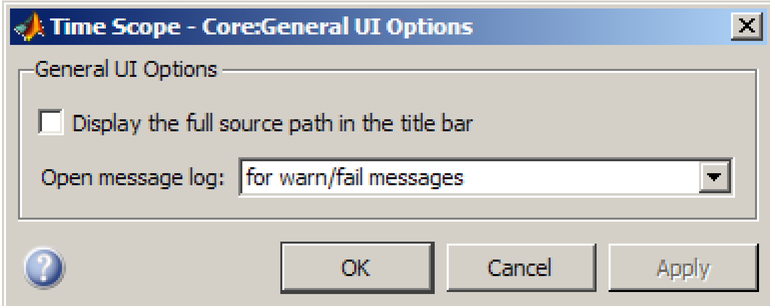
Core Pane

The Core pane in the Time Scope Configuration dialog box controls the general settings for the GUI.



General UI Options

To open the General UI Options dialog box, select **General UI** on the **Core** pane of the **Time Scope – Configuration** dialog box, and click **Options**. The following dialog box appears.



Display the full source path in the title bar

If you select this check box, the GUI displays the model name and full Simulink path to the data source in the title bar. Otherwise, it displays a shortened name.

Open message log

Use this parameter to control when the Message Log window opens. The Message Log window helps you debug any issues with the block. Using the drop-down menu, you can choose to open the Message Log window under any of the following conditions:

- `for any new messages`
- `for warn/fail messages`
- `only for fail messages`
- `manually`

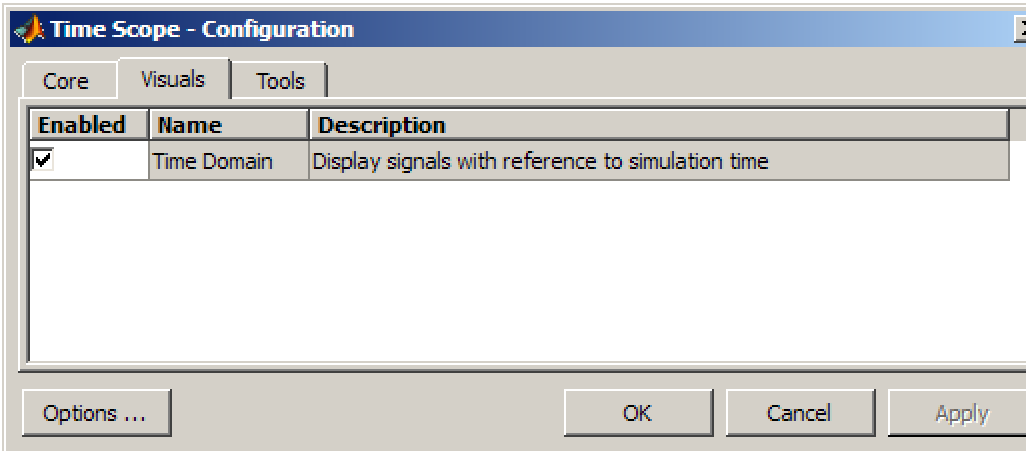
You can open the Message Log at any time by selecting **Help > Message Log**. The Message Log dialog box provides a system level record of loaded configuration settings and registered extensions. The Message Log displays summaries and details of each message, and you can filter the display of messages by **Type** and **Category**.

- The **Type** parameter allows you to select which types of messages to display in the Message Log. You can select `All`, `Info`, `Warn`, or `Fail`.

- The **Category** parameter allows you to select the category of messages to display in the Message Log. You can select `All`, `Configuration` or `Extension`.
  - The scope uses `Configuration` messages to indicate when new configuration files are loaded.
  - The scope uses `Extension` messages to indicate when components are registered. For example, you may see a `Simulink registered` message, indicating that a component has been successfully registered with Simulink and is available for configuration.

Visuals Pane

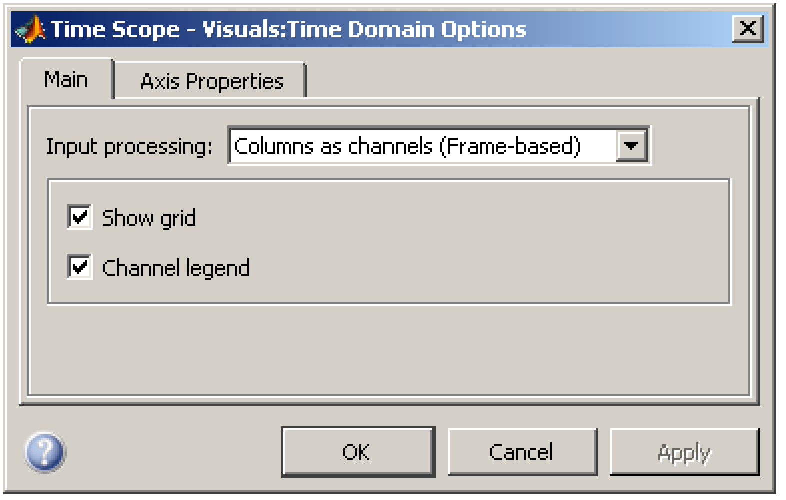
The Visuals pane in the Time Scope Configuration dialog box controls the general settings of the GUI.



Time Domain
Click the **Options** button to access the **Time Domain Options** dialog box.

Main Pane

The **Main** pane of the Time Domain Options dialog box appears as follows.



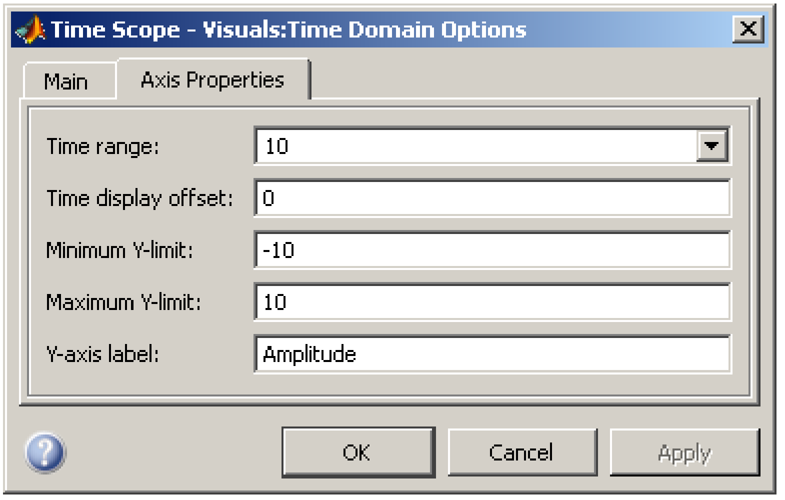
Input processing
Specify whether the block should treat the input signal as `Columns as channels (Frame-based)` or `Elements as channels (Sample-based)`).

Show grid
When you select this check box, a grid appears in the scope window. To hide the grid, clear this check box. Tunable.

Channel legend
Select this check box to show the legend in the scope window. When the legend appears, you can place it anywhere inside of the scope window. To turn the legend off, clear the **Channel legend** check box. Tunable.

Axis Properties Pane

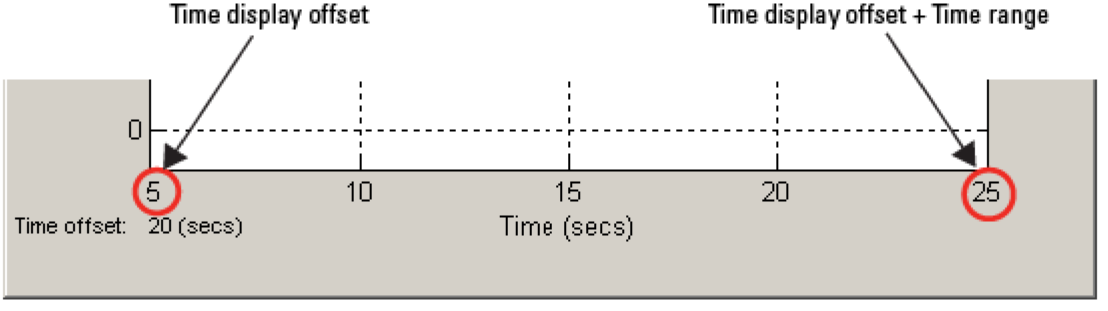
The **Axis Properties** pane of the Time Domain Options dialog box appears as follows.



Time range

Set the time range in seconds. If you set the **Input processing** parameter to `Columns as channels (Frame-based)`, you can use the sample time of the input to the block by selecting `Input sample time`. This option is not available when you set the **Input processing** parameter to `Elements as channels (Sample-based)`. Tunable.

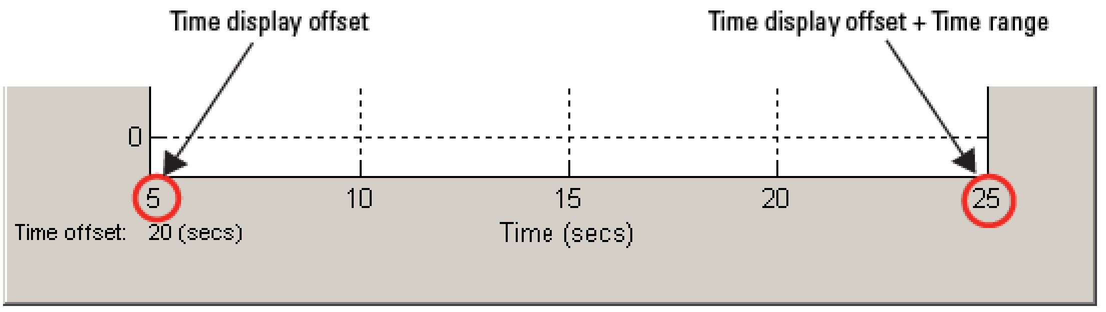
The scope sets the X-axis limits using the value of this parameter and the value of the **Time display offset** parameter, as shown in the following figure.



Time display offset

This parameter allows you to offset the values displayed on the X-axis by a specified number of seconds. You can shift the values displayed on the X-axis by specifying a nonzero value. Tunable.

The scope computes the X-axis range using the values of the **Time display offset** and **Time range** parameters, as shown in the following figure.



Minimum Y-limit

Specify the minimum value of the Y-axis. Tunable.

Maximum Y-limit

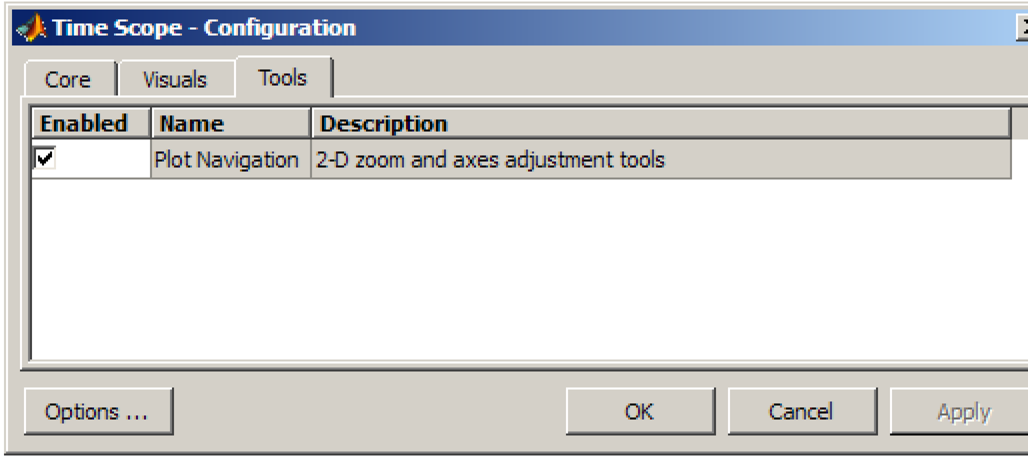
Specify the maximum value of the Y-axis. Tunable.

Y-axis label

Specify the text for the scope to display to the left of the Y-axis. Tunable.

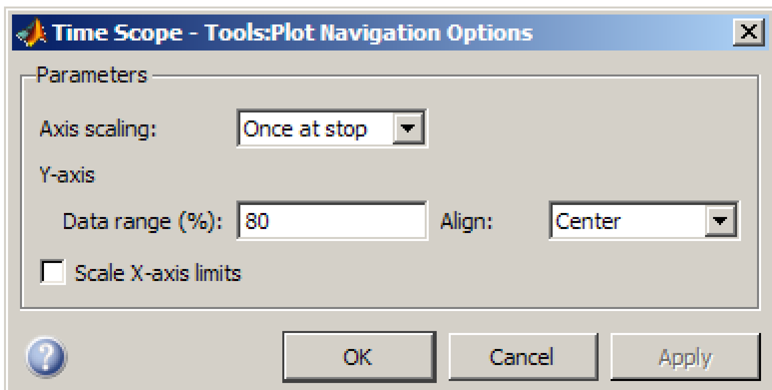
Tools Pane

The Tools pane in the **Time Scope – Configuration** dialog box contains the tools that appear on the Time Scope GUI.



Plot Navigation

Click **Plot Navigation**, and then click the **Options** button to open the **Plot Navigation Options** dialog box.



Axis Scaling

This parameter is Tunable. You can select one of the following options:

- `Manual` — When you select this option, the block does not automatically scale the axes. You can manually scale the axes in any of the following ways:
  - Select **Tools > Scale axes limits**.
  - Press the **Scale Axes Limits** toolbar button.
  - When the scope GUI is the active window, press **Ctrl** and **A** simultaneously.
- `Auto` — When you select this option, the block scales the axes as needed, both during and after simulation. Selecting this option enables the **Do not allow Y-axis limits to shrink** check box. By default, this parameter is selected, and the block does not shrink the Y-axis limits when scaling the axes.
- `Once at stop` — Selecting this option causes the block to scale the axes each time simulation is stopped. The block does not scale the axes during simulation.

Do not allow Y-axis limits to shrink

When you select this parameter, the Y-axis limits are only allowed to grow during axes scaling operations. If you clear this check box, the Y-axis limits may shrink during axes scaling operations.

This parameter appears only when you select `Auto` for the **Axis Scaling** parameter. When you set the **Axis Scaling** parameter to `Manual` or `Once at stop`, the Y-axis limits are allowed to shrink. Tunable.

Y-axis Data range (%)

Set the percentage of the Y-axis the block should use to display the data when scaling the axes (valid values are between 1 and 100). For example, if you set this parameter to 100, the block scales the Y-axis limits such that your data uses the entire Y-axis range. If you then set this parameter to 30, the block increases the Y-axis range such that your data uses only 30% of the Y-axis range. Tunable.

Y-axis Align

Specify where the block should align your data with respect to the Y-axis when it scales the axes. You can select `Top`, `Center`, or `Bottom`. Tunable.

Scale X-axis limits

Check this box to allow the block to scale the X-axis limits when it scales the axes. Tunable.

X-axis Data range (%)

Set the percentage of the X-axis the block should use to display the data when scaling the axes (valid values are between 1 and 100). For example, if you set this parameter to 100, the block scales the X-axis limits such that your data uses the entire X-axis range. If you then set this parameter to 30, the block increases the X-axis range such that your data uses only 30% of the X-axis range. Use the X-axis **Align** parameter to specify data placement with respect to the X-axis.

This parameter appears only when you select the **Scale X-axis limits** check box. Tunable.

X-axis Align

Specify how the block should align your data with respect to the X-axis: `Left`, `Center` or `Right`. This parameter appears only when you select the **Scale X-axis limits** check box. Tunable.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Spectrum Scope | Vector Scope | Scope

Tutorials

- Demo: Autoscaling and Curve Fitting
- Demo: Audio Sample Rate Conversion

How To

- "Displaying Time-Domain Data in the Time Scope"

Time-Varying Direct-Form II Transpose Filter (Obsolete)

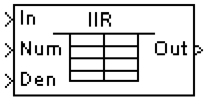
Purpose

Apply variable IIR filter to input

Library

`dspobslib`

Description



Note This block is now just an implementation of the Digital Filter block.

Time-Varying Lattice Filter (Obsolete)

Purpose Apply variable lattice filter to input

Library `dspobslib`

Description

Note This block is now just an implementation of the Digital Filter block.

Purpose

Generate matrix with Toeplitz symmetry

Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

dspmtrx3

Description

The Toeplitz block generates a Toeplitz matrix from inputs defining the first column and first row. The top input (Col) is a vector containing the values to be placed in the first *column* of the matrix, and the bottom input (Row) is a vector containing the values to be placed in the first *row* of the matrix.

```
y = toeplitz(Col,Row)       % Equivalent MATLAB code
```

The other elements of the matrix obey the relationship

```
y(i,j) = y(i-1,j-1)
```

and the output has dimension [length(Col) length(Row)]. The y(1,1) element is inherited from the Col input. For example, the following inputs

```
Col = [1 2 3 4 5]
Row = [7 7 3 3 2 1 3]
```

produce the Toeplitz matrix

$$\begin{bmatrix} 1 & 7 & 3 & 3 & 2 & 1 & 3 \\ 2 & 1 & 7 & 3 & 3 & 2 & 1 \\ 3 & 2 & 1 & 7 & 3 & 3 & 2 \\ 4 & 3 & 2 & 1 & 7 & 3 & 3 \\ 5 & 4 & 3 & 2 & 1 & 7 & 3 \end{bmatrix}$$

When both of the inputs are sample based, the output is sample based. Otherwise, the output is frame based.

When you select the **Symmetric** check box, the block generates a symmetric (Hermitian) Toeplitz matrix from a single input, u, defining both the first row and first column of the matrix.

```
y = toeplitz(u)      % Equivalent MATLAB code
```

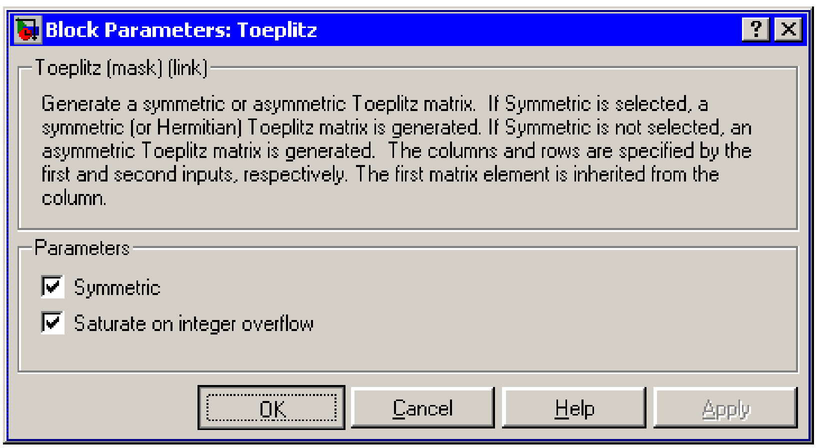
The output has dimension [length(u) length(u)]. For example, the Toeplitz matrix generated from the input vector [1 2 3 4] is

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 2 & 3 \\ 3 & 2 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix}$$

The output has the same frame status as the input.

The Toeplitz block supports real and complex floating-point and fixed-point inputs.

Dialog Box



Symmetric

When selected, enables the single-input configuration for symmetric Toeplitz matrix output.

Saturate on integer overflow

When you generate a symmetric Toeplitz matrix with this block, if the input vector is complex, the output is a symmetric Hermitian matrix whose elements satisfy the relationship

$$y(i, j) = \mathrm{conj}(y(j, i))$$

For fixed-point signals the conjugate operation could result in an overflow. When you select this parameter, overflows saturate. This parameter is only visible with the **Symmetric** parameter is selected. This parameter is ignored for floating-point signals.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers (real signals only)
Toep Col	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
Toep Row	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Constant Diagonal Matrix	Signal Processing Blockset
`toeplitz`	MATLAB

Purpose

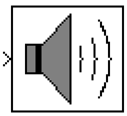
Write audio data to computer's audio device

Library

Signal Processing Sinks

`dspsnks4`

Description



The To Audio Device block sends audio data to your computer's audio device. This block has the following limitations:

- Not supported for use with the Simulink Model block.
- Not currently supported on Solaris platforms.

Use the **Device** parameter to specify the device to which you want to send the audio data. This parameter is automatically populated based on the audio devices installed on your system. If you plug or unplug an audio device from your system, type `clear mex` at the MATLAB command prompt to update the list.

Select the **Inherit sample rate from input** check box if you want the block to inherit the sample rate of the audio signal from the input to the block. If you clear this check box, the **Sample rate (Hz)** parameter appears on the block. Use this parameter to specify the number of samples per second in the signal.

Use the **Device data type** to specify the data type of the audio data that is sent to the device. You can choose:

- `8-bit integer`
- `16-bit integer`
- `24-bit integer`
- `32-bit float`
- `Determine from input data type`

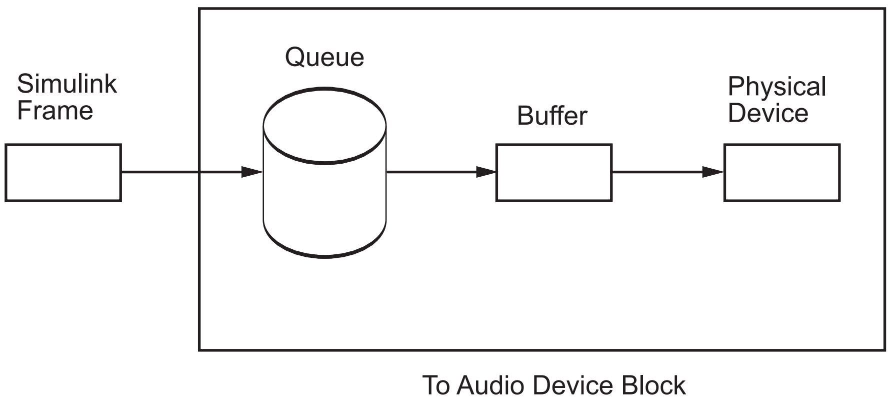
If you choose `Determine from input data type`, the following table summarizes the block's behavior.

Input Data Type	Device Data Type
Double-precision floating point or single-precision floating point	32-bit floating point
32-bit integer	24-bit integer
16-bit integer	16-bit integer
8-bit integer	8-bit integer

If you choose `Determine from input data type` and the device does not support the input data type, the block uses the next lowest-precision data type supported by the device.

Buffering

The To Audio Device block buffers the data from a Simulink signal using the process illustrated by the following figure.



1. At the start of the simulation, the queue is filled with silence. Specify the size of this queue using the **Queue duration (seconds)** parameter. As Simulink runs, the block appends Simulink frames to the bottom of the queue.

2. At each time step, the blocks sends a buffer of samples from the top of the queue to the audio device. Select the **Automatically determine**

buffer size check box to allow the block to use a conservative buffer size. See the From Audio Device block reference page for the equation the block uses to calculate this buffer size. If you clear this check box, the **Buffer size (samples)** parameter appears on the block. Use this parameter to specify the size of the buffer in samples.

3. The block writes the buffer of audio data to the device. If the queue did not contain enough data to completely fill the buffer, the block fills the remaining portion of the buffer with zeros. This data has a the data type specified by the **Device data type** parameter.

When the simulation throughput rate is lower than the hardware throughput rate, the queue, which is initially full, becomes empty. If the queue is empty, the block sends zeros (silence) to the audio device. When the simulation throughput rate is higher than the hardware throughput rate, the To Audio Device block waits to write data to the queue.

To minimize the chance of dropouts, the block checks to make sure the queue duration is at least as large as the maximum of the buffer size and the frame size. If it is not, the queue duration is automatically set to this maximum value.

Troubleshooting

When Simulink cannot keep up with an audio device that is operating in real time, the queue becomes empty and gaps occur in the audio data that the block sends to the device. To receive a warning, which will indicate the number of samples in the gap, type the following command on the MATLAB command line:

```
warning('on', ...
     'spblks:block:ToAudioDevice:toAudioDeviceDroppedSamples');
```

Here are several ways to deal with this situation:

- *Increase the queue duration.*

  The **Queue duration (seconds)** parameter specifies the duration of the signal, in seconds, that can be buffered during the simulation.

This is the maximum length of time that the block's data supply can lag the hardware's data demand.

- *Increase the buffer size.*

  The size of the buffer processed in each interrupt from the audio device affects the performance of your model. If the buffer is too small, a large portion of hardware resources are used to write data to the device. If the buffer is too big, Simulink must wait for the device to empty the buffer before it can write the data to the queue, which introduces latency.

- *Increase the simulation throughput rate.*

  Two useful methods for improving simulation throughput rates are increasing the signal frame size and compiling the simulation into native code:

  - Increase frame sizes and convert sample-based signals to frame-based signals throughout the model to reduce the amount of block-to-block communication overhead. This can increase throughput rates in many cases. However, larger frame sizes generally result in greater model latency due to initial buffering operations.
  - Generate executable code with Real-Time Workshop code generation software. Native code runs much faster than Simulink and should provide rates adequate for real-time audio processing.

Other ways to improve throughput rates include simplifying the model and running the simulation on a faster PC processor. For other ideas on improving simulation performance, see "Delay and Latency" and "Improving Simulation Performance and Accuracy" in the Simulink documentation.

Channel-to-Speaker Mapping on Windows Operating Systems

The To Audio Device and From Audio Device blocks can support multiple channels. On Windows operating systems, the channel-to-speaker mapping is defined as listed below. This mapping only applies when your sound card is properly configured and capable of receiving the audio data you send. If the number of channels on the card does not match the number of channels on the block, or if you specify a data type for the **Device data type** parameter that is not supported by your device, the Windows mixer intervenes to translate from one format to another. If the Windows mixer does intervene, the channel-to-speaker mapping might differ from what is specified here.

- Single channel input — Front center speaker

  On systems with two speakers, the front center channel is split between the right and left speakers.

- Multichannel input — Channels are assigned to speakers as follows:
  - One channel — Front center
  - Two channels — Front left, front right
  - Four channels — Front left, front right, rear left, rear right
  - Six channels — Front left, front right, front center, low frequency, rear left, rear right
  - Eight channels — Front left, front right, front center, low frequency, rear left, rear right, front left center, front right center
  - For all other channel combinations, the channel assignment is dictated by the audio card.

Audio Hardware API

The To Audio Device and From Audio Device blocks use the open-source PortAudio library in order to communicate with the audio hardware on a given computer. The PortAudio library supports a range of API's designed to communicate with the audio hardware on a given platform. The following API choices were made when building the PortAudio library for the Signal Processing Blockset product:

- Windows: DirectSound, WDM-KS
- Linux: OSS, ALSA

- Mac: CoreAudio

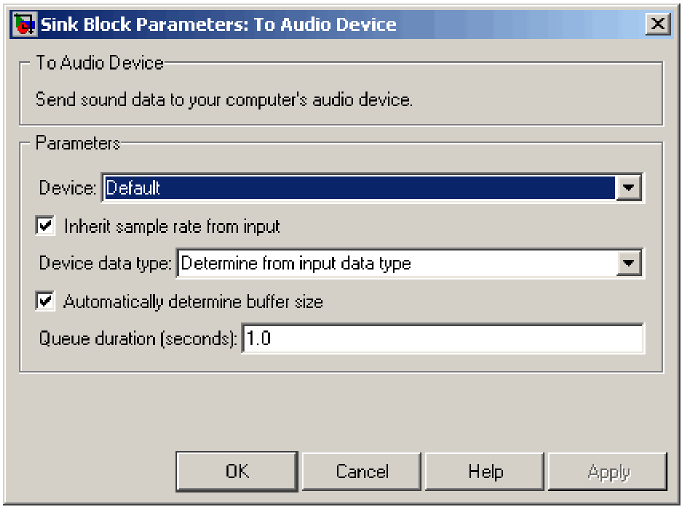
To select or change the Audio Hardware API, select **Preferences** from the MATLAB **File** menu. Then select Signal Processing Blockset from the tree menu.

If you are interested in using a different audio API, please search for PortAudio on the Matlab Central website.

Example

See the Positional Audio demo for an example of how to use this block. You can open this demo by typing `dspAudioPos` at the MATLAB command line.

Dialog Box



Device
: Specify which device to send the audio data to.

Inherit sample rate from input
: Select this check box if you want the block to inherit the sample rate of the audio signal from the input to the block.

Sample rate (Hz)
Specify the number of samples per second in the signal. This parameter is visible when the **Inherit sample rate from input** check box is cleared.

Device data type
Specify the data type of the audio data sent to the device.

Automatically determine buffer size
Select this check box to allow the block to calculate a conservative buffer size.

Buffer size (samples)
Specify the size of the buffer. This parameter is visible when the **Automatically determine buffer size** check box is cleared.

Queue duration (seconds)
Specify the size of the queue in seconds.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • 32-bit signed integers • 16-bit signed integers • 8-bit unsigned integers

See Also

From Audio Device	Signal Processing Blockset
To Multimedia File	Signal Processing Blockset
`audioplayer`	MATLAB
`sound`	MATLAB

To Multimedia File

Purpose

Write video frames and audio samples to multimedia file

Library

Signal Processing Sinks

`dspsnks4`

`vipsnks`

Description

To Multimedia File

The To Multimedia File block writes video frames, audio samples, or both to a multimedia (`.avi, .wav, .wma, or .wmv`) file. Video processing requires the Video and Image Processing Blockset product.

You can compress the video frames or audio samples by selecting a compression algorithm. You can connect as many of the input ports as you want. Therefore, you can control the type of video and/or audio the multimedia file receives.

Note This block supports code generation for platforms that have file I/O available. You cannot use this block with Real-Time Windows Target software, because that product does not support file I/O.

This block performs best on platforms with Version 11 or later of Windows Media® Player software. On Linux® and Mac® platforms, this block supports only uncompressed RGB24 AVI files whose size is less than 4 GB.

Ports

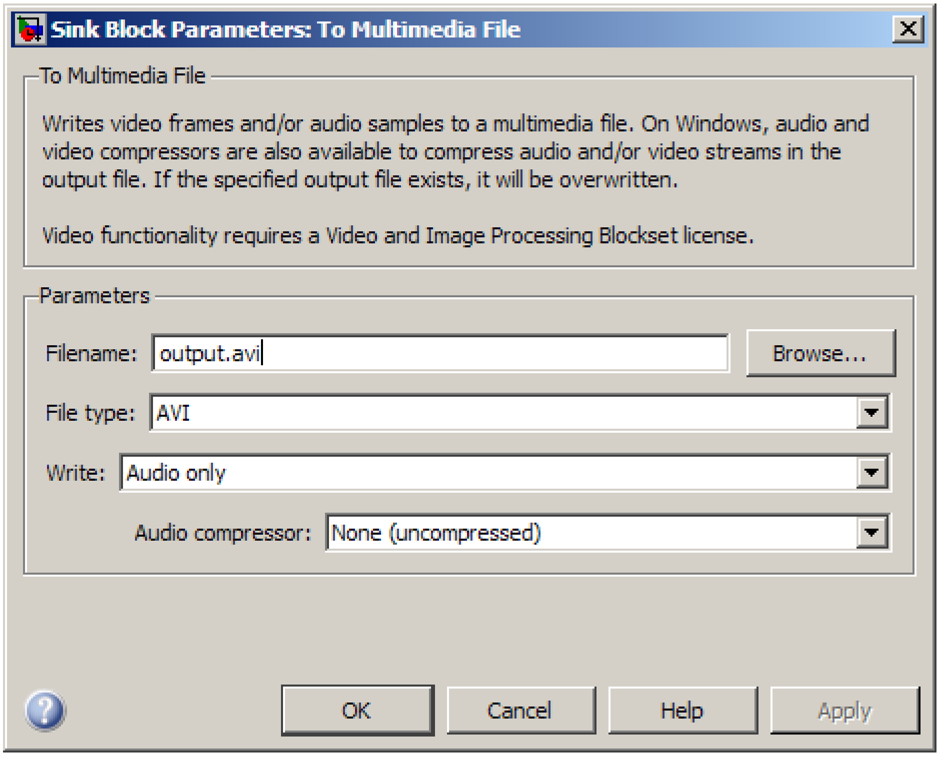
Port	Description
Image	M-by-N-by-3 matrix RGB, Intensity, or YCbCr 4:2:2 signal.
R, G, B	Matrix that represents one plane of the RGB video stream. Inputs to the R, G, or B port must have the same dimensions and data type.
Audio	Vector of audio data
Y, Cb, Cr	Matrix that represents one frame of the YCbCr video stream. The Y, Cb, and Cr ports use the following dimensions: Y: M x N Cb: $M \text{ x } \frac{N}{2}$ Cr: $M \text{ x } \frac{N}{2}$

Frame Size Matching

The video and audio input signals for the To Multimedia File block must have the same frame period. You might need to adjust the frame size of the audio signal so that the frame period matches the video signal frame period. To calculate the frame size, divide the frequency of the audio signal (samples per second) by the frame rate of the video signal (frames per second).

Dialog Box

The **Main** pane of the To Multimedia File block dialog appears as follows.



File name

Specify the name of the multimedia file. The block saves the file in your current folder. To specify a different file, click the **Browse** button, and then navigate to the new file.

File type

Specify the file type of the multimedia file. You can select `avi`, `wav`, `wma`, or `wmv`.

Write

Specify whether the block writes video frames, audio samples, or both to the multimedia file. You can select `Video and audio`, `Video only`, or `Audio only`.

Audio compressor

Select the type of compression algorithm to use to compress the audio data. This compression reduces the size of the multimedia file. Choose `None (uncompressed)` to save uncompressed audio data to the multimedia file.

Note The other items available in this parameter list are the audio compression algorithms installed on your system. For information about a specific audio compressor, see the documentation for that compressor.

Audio data type

Select the audio data type. You can use the **Audio data type** parameter only for uncompressed wave files.

Video compressor

Select the type of compression algorithm to use to compress the video data. This compression reduces the size of the multimedia file. Choose `None (uncompressed)` to save uncompressed video data to the multimedia file.

Note The other items available in this parameter list are the video compression algorithms installed on your system. For information about a specific video compressor, see the documentation for that compressor.

File color format

Select the color format of the data stored in the file. You can select either `RGB` or `YCbCr 4:2:2`.

Image signal

Specify how the block accepts a color video signal. If you select `One multidimensional signal`, the block accepts an M-by-N-by-P color video signal, where P is the number of color planes, at one port. If you select `Separate color signals`, additional ports appear on the block. Each port accepts one M-by-N plane of an RGB video stream.

Supported Data Types

For the block to display video data properly, double- and single-precision floating-point pixel values must be between `0` and `1`. Any other data type requires the pixel values between the minimum and maximum values supported by their data type.

Port	Supported Data Types	Supports Complex Values?
Image	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16- 32-bit signed integers • 8-, 16- 32-bit unsigned integers	No
R, G, B	Same as Image port	No
Audio	• Double-precision floating point • Single-precision floating point • 16-bit signed integers • 32-bit signed integers • 8-bit unsigned integers	No
Y, Cb, Cr	Same as Image port	No

See Also

From Multimedia File	Signal Processing Blockset
Frame Rate Display	Video and Image Processing Blockset
To Video Display	Video and Image Processing Blockset
Video To Workspace	Video and Image Processing Blockset
Video Viewer	Video and Image Processing Blockset

To Wave Device (Obsolete)

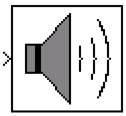
Purpose

Send audio data to standard Windows audio device in real time

Library

`dspwin32`

Description



Note The To Wave Device block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the To Audio Device block.

The To Wave Device block sends audio data to a standard Windows audio device in real time. It is compatible with most popular Windows hardware, including Sound Blaster cards. The data is sent to the hardware in uncompressed pulse code modulation (PCM) format, and should typically be sampled at one of the standard Windows audio device rates: 8000, 11025, 22050, or 44100 Hz. Some hardware might support other rates in addition to these.

Note Models that contain both the To Wave Device block and the From Wave Device block require a duplex-capable sound card.

The **Use default audio device** check box allows the To Wave Device block to detect and use the system's default audio hardware. You should select this option for systems that have a single sound device installed, or when the default sound device on a multiple-device system is your desired target. When the default sound device is *not* your desired output device, clear **Use default audio device**, and set the desired hardware in the **Audio device** parameter. This parameter lists the names of the installed audio devices.

The block input can contain audio data from a mono or stereo signal. A mono signal is represented as either a sample-based scalar or a frame-based length-M vector, where M is frame size. A stereo signal is represented as a sample-based length-2 vector or a frame-based M-by-2 matrix.

When the input data type is uint8, the block conveys the signal samples to the audio device using 8 bits. When the input data type is double, single, int16, or fixed point with a word length of 16 and a fraction length of 15, the block conveys the signal samples to the audio device using 16 bits by default. For inputs of data type double and single, you can also set the block to convey the signal samples using 24 bits by selecting the **Enable 24-bit output for double- and single-precision input signals** check box. The 24-bit sample width requires more memory but in general yields better fidelity.

The amplitude of the input must be in a valid range that depends on the input data type, as shown in the following table. Amplitudes outside the valid range are clipped to the nearest allowable value.

Input Data Type	Valid Input Amplitude Range
double	$-1 \le amplitude < 1$
single	$-1 \le amplitude < 1$
int16	$-32768 \le amplitude \le 32767$
uint8	$0 \le amplitude \le 255$
Fixed point with a word length of 16 and a fraction length of 15	$-1 \le amplitude \le 1 - 2^{-15}$

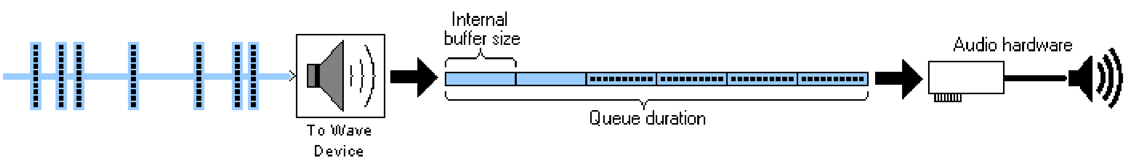
Buffering

Because audio devices generate real-time audio output, the Simulink environment must maintain a continuous flow of data to a device throughout simulation. Delays in passing data to the audio hardware can result in hardware errors or distortion of the output. This means that the To Wave Device block must in principle supply data to the audio hardware as quickly as the hardware reads the data. However, the To Wave Device block often *cannot* match the throughput rate of the audio hardware, especially when the simulation is running within Simulink rather than as generated code. Simulink execution speed can vary during the simulation as the host operating system services other processes. The block must therefore rely on a buffering strategy to ensure that signal data is available to the hardware on demand.

Note This block requires real-time execution of the parent model for best performance.

The following block parameters control the memory management for this block:

- **Queue duration**
- **Automatically determine internal buffer size** or **User-defined internal buffer size**
- **Initial output delay**



The **Queue duration** parameter defines the overall size of the block's buffer. The block reads in chunks of data in the size of the input dimensions and stores them in the buffer. The internal buffer size defines the dimensions of the block output to the hardware. You can define the internal buffer size yourself in the **User-defined internal buffer size parameter**. If you select **Automatically determine internal buffer size** instead, the internal buffer size is calculated for you according to the following rules:

- If the input to the block has a frame size of 32 samples or larger, the internal buffer size be the same as the input frame size.
- If the input to the block has a frame size smaller than 32 samples, the internal buffer size is based on the input sample rate according to the following table, where

$$F_s = sampling\ frequency = \frac{1}{sample\ time}$$

F_s (Hz)	Internal Buffer Size (samples)
$F_s < 8000$	$\min(64, 2 * F_s)$
$8000 \le F_s < 22{,}050$	128
$22{,}050 \le F_s < 44{,}100$	256
$44{,}100 \le F_s < 96{,}000$	512
$F_s \ge 96{,}000$	1024

To minimize the chance of dropouts, the block checks to make sure that the queue duration is at least as big as twice the internal buffer size. If it is not, the queue duration is automatically set to twice the internal buffer size.

The **Initial output delay** parameter enables you to preload the buffer before the block starts to output data to the audio device, which can be helpful for models that do not run in real time. However, for real-time applications, it is best to set the initial output delay to zero (one frame of delay), or as close to zero as possible.

Troubleshooting

If you are getting undesirable audio output using the To Wave Device block, first determine whether your model can run in real time. Replace the To Wave Device block with a To Wave File block, run the model, and compare the model's simulation stop time to the elapsed time on your watch. If the model simulation stop time is less than the elapsed time on your watch, your model can probably run in real time. Then,

- If your model can run in real time,
  1. Select **Automatically determine internal buffer size**. This alone might solve the problem. If not,
  2. Try increasing the **Queue duration** parameter to a relatively large value, such as 0.5 s.

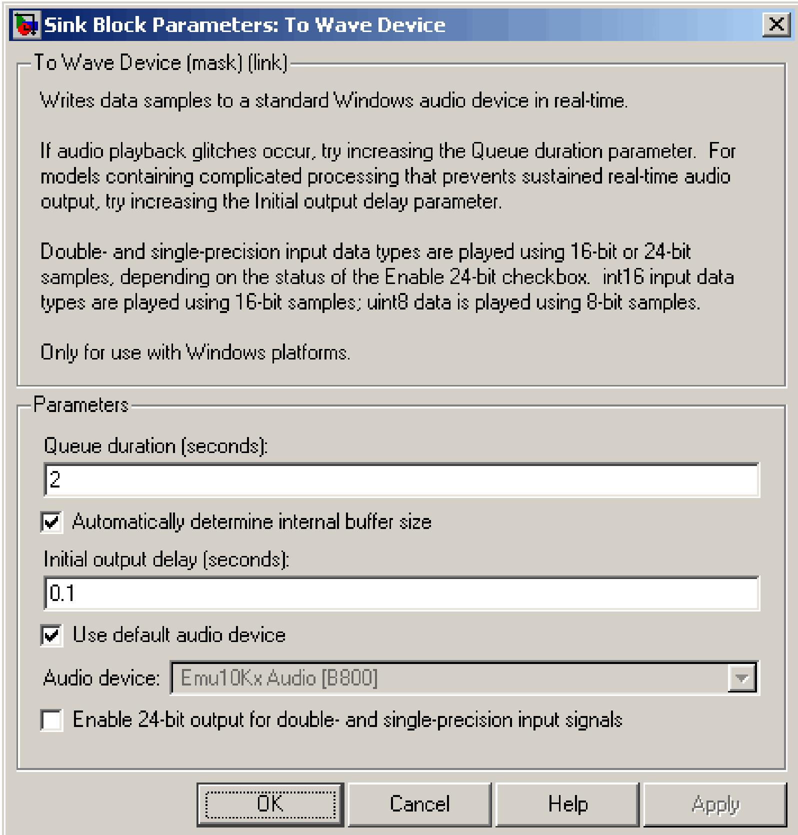
If one or both of these options restores desirable audio output, you can try reducing the internal buffer size and/or queue duration until the quality of the audio output again degrades.

- If your model is not running in real time, try to make it run in real time by

  1. Optimizing the model (using a more efficient implementation), or
  2. Using the Simulink Accelerator mode, or
  3. Generating stand-alone code

  If none of these are possible, but the model only runs for a short period of time, set the **Queue duration** parameter to a size equal to a significant fraction of the model stop time and use a similarly large initial delay. This is not an optimal solution, but might work in some cases.

Dialog Box



Queue duration (seconds)
: Specify the overall buffer size. To minimize the chance of dropouts, the block checks to make sure that the queue duration is as least as large as twice the internal buffer size. If it is not, the queue duration is automatically set to twice the internal buffer size.

Automatically determine internal buffer size
: Select to have the block automatically select the internal buffer size for you. For details, see "Buffering" on page 2-1313.

User-defined internal buffer size (samples)

Define the internal buffer size, or the size of the chunks of data sent by the block to the audio hardware device.

This parameter is only visible when **Automatically determine internal buffer size** is not selected.

Initial output delay (seconds)

Specify the amount of time by which to delay the initial output to the audio device. During this time data accumulates in the block's buffer. Any value less than or equal to the queue duration specifies the smallest possible initial delay, which is a single frame.

Use default audio device

Select to direct audio output to the system's default audio device.

Audio device

This parameter lists the names of the installed audio devices. Specify the name of the audio device to receive the audio output. Select **Use default audio device** when the system has only a single audio card installed.

This parameter is only enabled when the **Use default audio device** check box is not selected.

Enable 24-bit output for double and single precision input signals

Select to output 24-bit data when inputs are double- or single-precision. Otherwise, the block outputs 16-bit data for double- and single-precision inputs.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Signed fixed point with a word length of 16 and a fraction length of 15 • 16-bit signed integers • 8-bit unsigned integers

See Also

From Wave Device (Obsolete)	Signal Processing Blockset
To Wave File (Obsolete)	Signal Processing Blockset
`audioplayer`	MATLAB
`sound`	MATLAB

To Wave File (Obsolete)

Purpose

Write audio data to file in Microsoft Wave (`.wav`) format

Library

`dspwin32`

Description

To Wave

Note The To Wave File block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the To Multimedia File block.

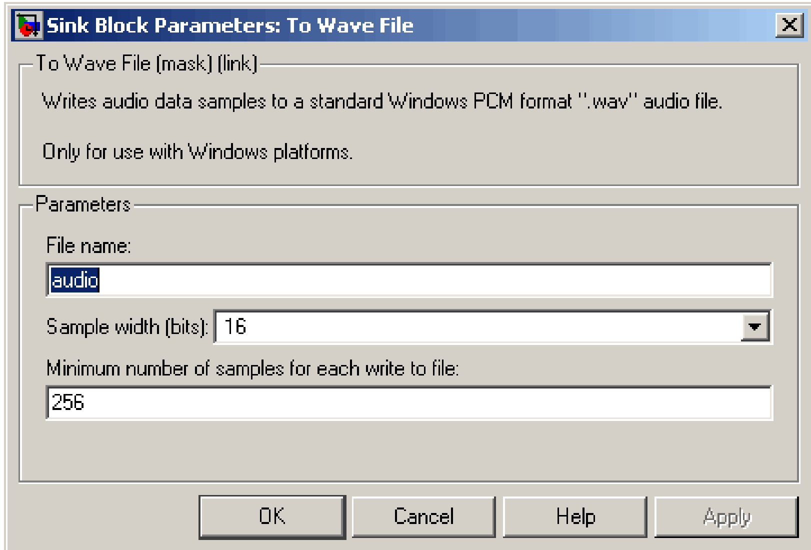
The To Wave File block streams audio data to a Microsoft Wave (`.wav`) file in the uncompressed pulse code modulation (PCM) format. For compatibility reasons, the sample rate of the discrete-time input signal should typically be one of the standard Windows audio device rates (8000, 11025, 22050, or 44100 Hz), although the block supports arbitrary rates.

The input to the block, u, can contain audio data with one or more channels. A signal with C channels is represented as a sample-based length-C vector or a frame-based M-by-C matrix. The amplitude of the input should be in the range ±1. Values outside this range are clipped to the nearest allowable value.

```
wavwrite(u,Fs,bits,'filename')    % Equivalent MATLAB code
```

Note AVI files are the only supported file type for non-Windows platforms.

Dialog Box



File name

Specify the path and name of the file to write. Paths can be relative or absolute. You do not need to specify the `.wav` extension.

Sample width (bits)

Specify the number of bits used to represent the signal samples in the file. The higher sample width settings require more memory but yield better fidelity for double- and single-precision inputs:

- `8` — Allocates 8 bits to each sample, allowing a resolution of 256 levels
- `16` — Allocates 16 bits to each sample, allowing a resolution of 65536 levels
- `24` — Allocates 24 bits to each sample, allowing a resolution of 16777216 levels
- `32` — Allocates 32 bits to each sample, allowing a resolution of 2^{32} levels ranging from -1 to 1

The 8-, 16-, and 24-bit modes output integer data, while the 32-bit mode outputs single-precision floating-point data.

Minimum number of samples for each write to file

Specify the number of consecutive samples, L, to write with each file access. To reduce the required number of file accesses, the block writes L consecutive samples to the file during each access for $L \geq M$. For $L < M$, the block instead writes M consecutive samples during each access. Larger values of L result in fewer file accesses, which reduces run-time overhead.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Signed fixed point with a word length of 16 and a fraction length of 15 • 16-bit signed integers • 8-bit unsigned integers

See Also

From Multimedia File	Signal Processing Blockset
To Audio Device	Signal Processing Blockset
Signal To Workspace	Signal Processing Blockset
To Workspace	Simulink
`wavwrite`	MATLAB

Transpose

Purpose

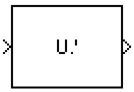
Compute matrix transpose

Library

Math Functions / Matrices and Linear Algebra / Matrix Operations

`dspmtrx3`

Description



The Transpose block transposes the M-by-N input matrix to size N-by-M. When you select the **Hermitian** check box, the block performs the Hermitian (complex conjugate) transpose.

```
y = u'    % Equivalent MATLAB code
```

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \end{bmatrix} \xrightarrow{\text{u'}} \begin{bmatrix} u^*_{11} & u^*_{21} \\ u^*_{12} & u^*_{22} \\ u^*_{13} & u^*_{23} \end{bmatrix}$$

When you do not select the **Hermitian** check box, the block performs the nonconjugate transpose.

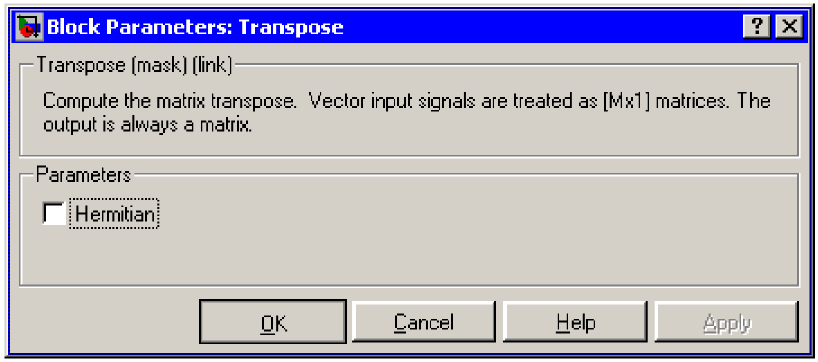
```
y = u.'    % Equivalent MATLAB code
```

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \end{bmatrix} \xrightarrow{\text{u.'}} \begin{bmatrix} u_{11} & u_{21} \\ u_{12} & u_{22} \\ u_{13} & u_{23} \end{bmatrix}$$

A length-M 1-D vector input is treated as an M-by-1 matrix. The output is always sample based.

The Transpose block supports real and complex floating-point and fixed-point data types. When **Hermitian** is selected, the block input must be a signed data type.

Dialog Box



Hermitian

When selected, specifies the complex conjugate transpose.

Saturate on integer overflow

This parameter is only visible when the **Hermitian** parameter is selected because overflows can occur when computing the complex conjugate of complex fixed-point signals. When you select this parameter, such overflows saturate. This parameter is ignored for floating-point signals and for real-valued fixed-point signals.

Supported Data Types

When **Hermitian** is selected, the block input must be a signed data type.

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Math Function	Simulink
Permute Matrix	Signal Processing Blockset
Reshape	Simulink
Submatrix	Signal Processing Blockset

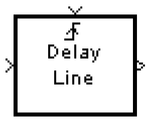
Purpose

Buffer sequence of inputs into frame-based output

Library

`dspobslib`

Description



Note The Triggered Delay Line block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the Delay Line block.

The Triggered Delay Line block acquires a collection of M_o input samples into a frame, where you specify M_o in the **Delay line size** parameter. The block buffers a single sample from input 1 whenever it is triggered by the control signal at input 2 (). When the next triggering event occurs, the newly acquired input sample is appended to the output frame so that the new output overlaps the previous output by M_o-1 samples. Between triggering events the block ignores input 1 and holds the output at its last value.

You specify the triggering event at input 2 in the **Trigger type** pop-up menu:

- `Rising edge` triggers execution of the block when the trigger input rises from a negative value to zero or a positive value, or from zero to a positive value.
- `Falling edge` triggers execution of the block when the trigger input falls from a positive value to zero or a negative value, or from zero to a negative value.
- `Either edge` triggers execution of the block when either a rising or falling edge (as described above) occurs.

The Triggered Delay Line block has zero latency, so the new input appears at the output in the same simulation time step. The output frame period is the same as the input sample period, $T_{fo}=T_{si}$.

Sample-Based Operation

In sample-based operation, the Triggered Delay Line block buffers a sequence of sample-based length-N vector inputs (1-D, row, or column) into a sequence of overlapping sample-based M_o-by-N matrix outputs, where you specify M_o in the **Delay line size** parameter (M_o>1). That is, each input vector becomes a *row* in the sample-based output matrix. When M_o=1, the input is simply passed through to the output, and retains the same dimension. Sample-based full-dimension matrix inputs are not accepted.

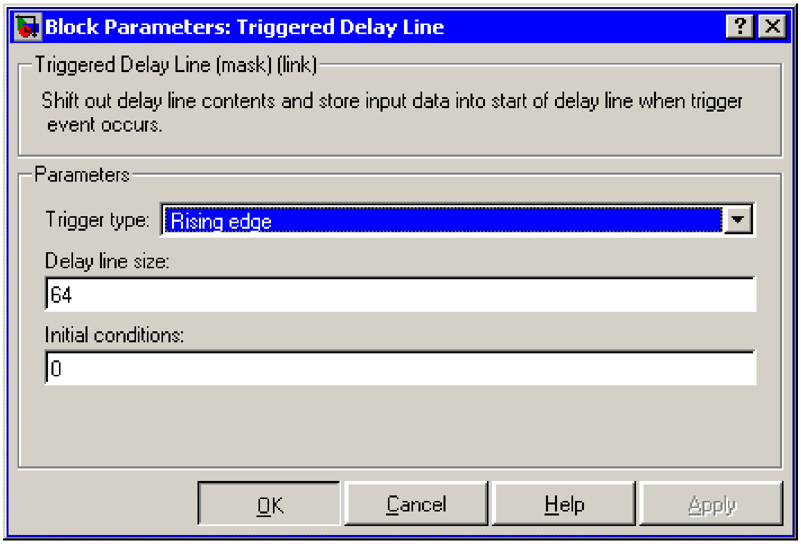
Frame-Based Operation

In frame-based operation, the Triggered Delay Line block rebuffers a sequence of frame-based M_i-by-N matrix inputs into an sequence of overlapping frame-based M_o-by-N matrix outputs, where M_o is the output frame size specified by the **Delay line size** parameter (that is, the number of consecutive samples from the input frame to rebuffer into the output frame). M_o can be greater or less than the input frame size, M_i. Each of the N input channels is rebuffered independently.

Initial Conditions

The Triggered Delay Line block's buffer is initialized to the value specified by the **Initial condition** parameter. The block always outputs this buffer at the first simulation step (*t*=0). When the block's output is a vector, the **Initial condition** can be a vector of the same size or a scalar value to be repeated across all elements of the initial output. When the block's output is a matrix, the **Initial condition** can be a matrix of the same size or a scalar to be repeated across all elements of the initial output.

Dialog Box



Trigger type
: The type of event that triggers the block's execution.

Delay line size
: The length of the output frame (number of rows in output matrix), M_o.

Initial condition
: The value of the block's initial output, a scalar, vector, or matrix.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Trigger	• Any data type supported by the Trigger block
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Buffer	Signal Processing Blockset
Delay Line	Signal Processing Blockset
Unbuffer	Signal Processing Blockset

Purpose

Import signal samples from MATLAB workspace when triggered

Library

Signal Operations

`dspsigops`

Description

The Triggered Signal From Workspace block imports signal samples from the MATLAB workspace into the Simulink model when triggered by the control signal at the input port ( ). The **Signal** parameter specifies the name of a MATLAB workspace variable containing the signal to import, or any valid MATLAB expression defining a matrix or 3-D array.

When the **Signal** parameter specifies an M-by-N matrix ($M \neq 1$), each of the N columns is treated as a distinct channel. You specify the frame size in the **Samples per frame** parameter, M_o, and the output when triggered is an M_o-by-N matrix containing M_o consecutive samples from each signal channel. For $M_o=1$, the output is sample based; otherwise the output is frame based. For convenience, an imported row vector ($M=1$) is treated as a single channel, so the output dimension is M_o-by-1.

When the **Signal** parameter specifies an M-by-N-by-P array, the block generates a single page of the array (an M-by-N matrix) at each trigger time. The **Samples per frame** parameter must be set to `1`, and the output is always sample based.

Trigger Event

You specify the triggering event at the input port in the **Trigger type** pop-up menu:

- `Rising edge` triggers execution of the block when the trigger input rises from a negative value to zero or a positive value, or from zero to a positive value.
- `Falling edge` triggers execution of the block when the trigger input falls from a positive value to zero or a negative value, or from zero to a negative value.

- `Either edge` triggers execution of the block when either a rising or falling edge (as described above) occurs.

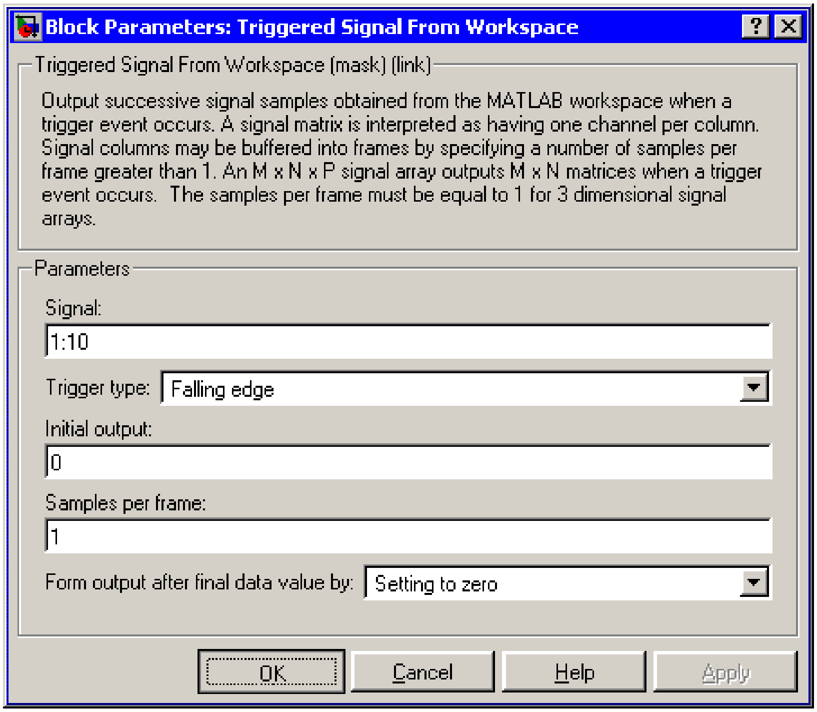
Initial and Final Conditions

The **Initial output** parameter specifies the output of the block from the start of the simulation until the first trigger event arrives. Between trigger events, the block holds the output value constant at its most recent value (that is, no linear interpolation takes place). For single-channel signals, the **Initial output** parameter value can be a vector of length M_o or a scalar to repeat across the M_o elements of the initial output frames. For matrix outputs (M_o-by-N or M-by-N), the **Initial output** parameter value can be a matrix of the same size or a scalar to be repeated across all elements of the initial output.

When the block has output all of the available signal samples, it can start again at the beginning of the signal, or simply repeat the final value or generate zeros until the end of the simulation. (The block does not extrapolate the imported signal beyond the last sample.) The **Form output after final data value by** parameter controls this behavior:

- When you specify `Setting To Zero`, the block generates zero-valued outputs for the duration of the simulation after generating the last frame of the signal.
- When you specify `Holding Final Value`, the block repeats the final sample for the duration of the simulation after generating the last frame of the signal.
- When you specify `Cyclic Repetition`, the block repeats the signal from the beginning after generating the last frame. When there are not enough samples at the end of the signal to fill the final frame, the block zero-pads the final frame as necessary to ensure that the output for each cycle is identical (for example, the *i*th frame of one cycle contains the same samples as the *i*th frame of any other cycle).

Dialog Box



Signal

The name of the MATLAB workspace variable from which to import the signal, or a valid MATLAB expression specifying the signal.

Trigger type

The type of event that triggers the block's execution.

Initial output

The value to output until the first trigger event is received.

Samples per frame

The number of samples, M_o, to buffer into each output frame. This value must be 1 when you specify a 3-D array in the **Signal** parameter.

Form output after final data value by

Specifies the output after all of the specified signal samples have been generated. The block can output zeros for the duration of the simulation (`Setting to zero`), repeat the final data sample (`Holding Final Value`) or repeat the entire signal from the beginning (`Cyclic Repetition`).

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed and unsigned)
- 8-, 16-, and 32-bit signed integers
- 8-, 16-, and 32-bit unsigned integers

See Also

Signal From Workspace	Signal Processing Blockset
Signal To Workspace	Signal Processing Blockset
Triggered To Workspace	Signal Processing Blockset

Purpose

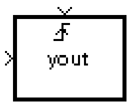
Write input sample to MATLAB workspace when triggered

Library

Signal Processing Sinks

`dspsnks4`

Description



The Triggered To Workspace block creates a matrix or array variable in the MATLAB workspace, where it stores the acquired inputs at the end of a simulation. The block overwrites an existing variable with the same name.

For an M-by-N frame-based input, the block creates an N-column workspace matrix in which each group of M rows represents a single input frame from each of N channels (the most recent frame occupying the last M rows). The maximum size of this workspace variable is limited to P-by-N, where P is the **Maximum number of rows** parameter. (When the simulation progresses long enough for the block to acquire more than P samples, it stores only the most recent P samples.) The **Decimation factor**, D, allows you to store only every Dth input frame.

For an M-by-N sample-based input, the block creates a three-dimensional array in which each M-by-N page represents a single sample from each of M*N channels (the most recent input matrix occupying the last page). The maximum size of this variable is limited to M-by-N-by-P, where P is the **Maximum number of rows** parameter. (When the simulation progresses long enough for the block to acquire more than P inputs, it stores only the last P inputs.) The **Decimation factor**, D, allows you to store only every Dth input matrix.

The block acquires and buffers a single frame from input 1 whenever it is triggered by the control signal at input 2 ( ). At all other times, the block ignores input 1. You specify the triggering event at input 2 in the **Trigger type** pop-up menu:

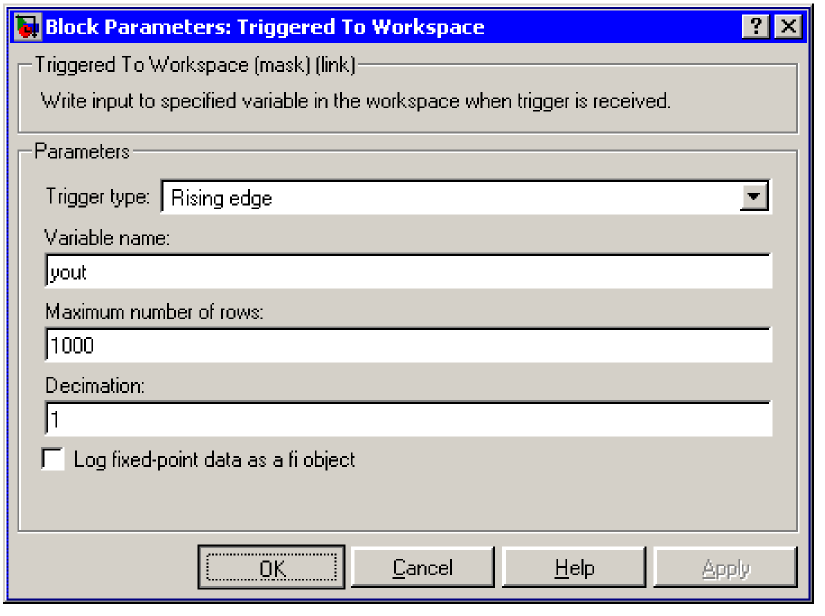
- `Rising edge` triggers execution of the block when the trigger input rises from a negative value to zero or a positive value, or from zero to a positive value.

- `Falling edge` triggers execution of the block when the trigger input falls from a positive value to zero or a negative value, or from zero to a negative value.
- `Either edge` triggers execution of the block when either a rising or falling edge (as described above) occurs.

To save a record of the sample time corresponding to each sample value, open the Configuration Parameters dialog box. In the **Select** pane, click **Data Import/Export**. In the **Save to workspace** section, select the **Time** check box.

The nontriggered version of this block is the Simulink To Workspace block.

Dialog Box



Trigger type

The type of event that triggers the block's execution.

Variable name
The name of the workspace matrix in which to store the data.

Maximum number of rows
The maximum number of rows (one row per time step) to be saved, P.

Decimation
The decimation factor, D.

Log fixed-point data as a fi object
Select to log fixed-point data to the MATLAB workspace as a Fixed-Point Toolbox `fi` object. Otherwise, fixed-point data is logged to the workspace as `double`.

Supported Data Types

Port	Supported Data Types
Input	• Any data type supported by the To Workspace block
Trigger	• Any data type supported by the Trigger block

See Also

Signal From Workspace	Signal Processing Blockset
Signal To Workspace	Signal Processing Blockset
Triggered Signal From Workspace	Signal Processing Blockset

Two-Channel Analysis Subband Filter

Purpose

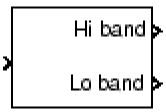
Decompose signal into high-frequency subband and low-frequency subband

Library

Filtering / Multirate Filters

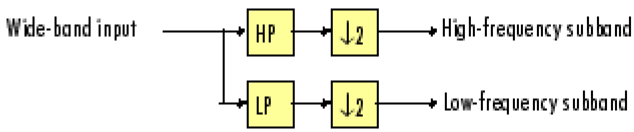
`dspmlti4`

Description



The Two-Channel Analysis Subband Filter block decomposes the input into a high-frequency subband and a low-frequency subband, each with half the bandwidth and half the sample rate of the input.

The block filters the input with a pair of highpass and lowpass FIR filters, and then downsamples the results by 2, as illustrated in the following figure.



Note that the block implements the FIR filtering and downsampling steps together using a polyphase filter structure, which is more efficient than the straightforward filter-then-decimate algorithm illustrated above. Each subband is the first phase of the respective polyphase filter.

You must provide the vector of filter coefficients for the two filters. Each filter should be a half-band filter that passes the frequency band that the other filter stops. For frame-based inputs, you also need to specify whether the change in the sample rate of the output gets reflected by a change in the frame size, or the frame rate.

Note By connecting many copies of this block, you can implement a multilevel dyadic analysis filter bank. In some cases, it is more efficient to use the Dyadic Analysis Filter Bank block instead. For more information, see "Creating Multilevel Dyadic Analysis Filter Banks" on page 2-1342.

Sections of This Reference Page

Specifying the FIR Filters

You must provide the vector of numerator coefficients for the lowpass and highpass filters in the **Lowpass FIR filter coefficients** and **Highpass FIR filter coefficients** parameters.

For example, to specify a filter with the following transfer function, enter the vector `[b(1) b(2) ...  b(m)]`.

$$H(z) = B(z) = b_1 + b_2 z^{-1} + \ldots + b_m z^{-(m-1)}$$

Each filter should be a half-band filter that passes the frequency band that the other filter stops. When you plan to use the Two-Channel Synthesis Subband Filter block to reconstruct the input to this block, you need to design perfect reconstruction filters to use in the synthesis subband filter.

The best way to design perfect reconstruction filters is to use the Wavelet Toolbox `wfilters` function in to design both the filters both in this block and in the Two-Channel Synthesis Subband Filter block. You can also use Filter Design Toolbox functions and Signal

Processing Toolbox functions. To learn how to design your own perfect reconstruction filters, see “References” on page 2-1353.

The block initializes all filter states to zero.

Sample-Based Operation

- “Valid Sample-Based Inputs” on page 2-1340
- “Sample-Based Outputs” on page 2-1340

Valid Sample-Based Inputs

The block accepts all M-by-N sample-based matrix inputs. The block treats such inputs as $M \cdot N$ independent channels, and decomposes each channel over time.

Sample-Based Outputs

Given a sample-based M-by-N input, the block outputs two M-by-N sample-based matrices whose sample rates are half the input sample rate. Each output matrix element is the high- or low-frequency subband output of the corresponding input matrix element. Depending on the Simulink configuration parameters, some sample-based outputs can have one sample of latency, as described in “Latency” on page 2-1341.

Frame-Based Operation

- “Valid Frame-Based Inputs” on page 2-1340
- “Frame-Based Outputs” on page 2-1340

Valid Frame-Based Inputs

The block accepts M-by-N frame-based matrix inputs where M is a multiple of two. The block treats such inputs as N independent channels, and decomposes each channel over time.

Frame-Based Outputs

Given a valid frame-based input, the block outputs two frame-based matrices. Each output column is the high- or low-frequency subband of the corresponding input column.

The sample rate of the outputs are half that of the input. The **Framing** parameter sets whether the block halves the sample rate by halving the output frame size, or halving the output frame rate:

- `Maintain input frame size` — The input and output frame *sizes* are the same, but the frame *rate* of the outputs are half that of the input. So, the overall sample rate of the output is half that of the input. This setting causes the block to have one frame of latency, as described in "Latency" on page 2-1341.
- `Maintain input frame rate` — The input and output frame *rates* are the same, but the frame *size* of the outputs are half that of the input (the input frame size must be a multiple of two). So, the overall sample rate of the output is half that of the input.

Latency

In some cases, the block has nonzero tasking latency, which means that there is a constant delay between the time that the block receives an input, and produces the corresponding output, as summarized below and in the following table:

- For sample-based inputs, there are cases where the block exhibits *one-sample latency*. In such cases, when the block receives the *n*th input sample, it produces the outputs corresponding to the *n-1*th input sample. When the block receives the first input sample, the block outputs an initial value of zero in each output channel.
- For frame-based inputs, there are cases where the block exhibits *one-frame latency*. In such cases, when the block receives the *n*th input frame, it produces the outputs corresponding to the *n-1*th input frame. When the block receives the first input frame, the block outputs a frame of zeros.

Note For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

Amount of Block Latency for All Possible Block Settings

Input	Latency	No Latency
Sample based	One sample of latency when the **Tasking mode for periodic sample times** parameter is set to `MultiTasking` or `Auto` in the **Solver** pane of the Configuration Parameters dialog box. The first output sample of each channel is always `0`.	The **Tasking mode for periodic sample times** parameter is set to `SingleTasking` in the **Solver** pane of the Configuration Parameters dialog box.
Frame based	One frame of latency when the **Framing** parameter is set to `Maintain input frame size`. The first output frame is always all zeros.	The **Framing** parameter is set to `Maintain input frame rate`.

Creating Multilevel Dyadic Analysis Filter Banks

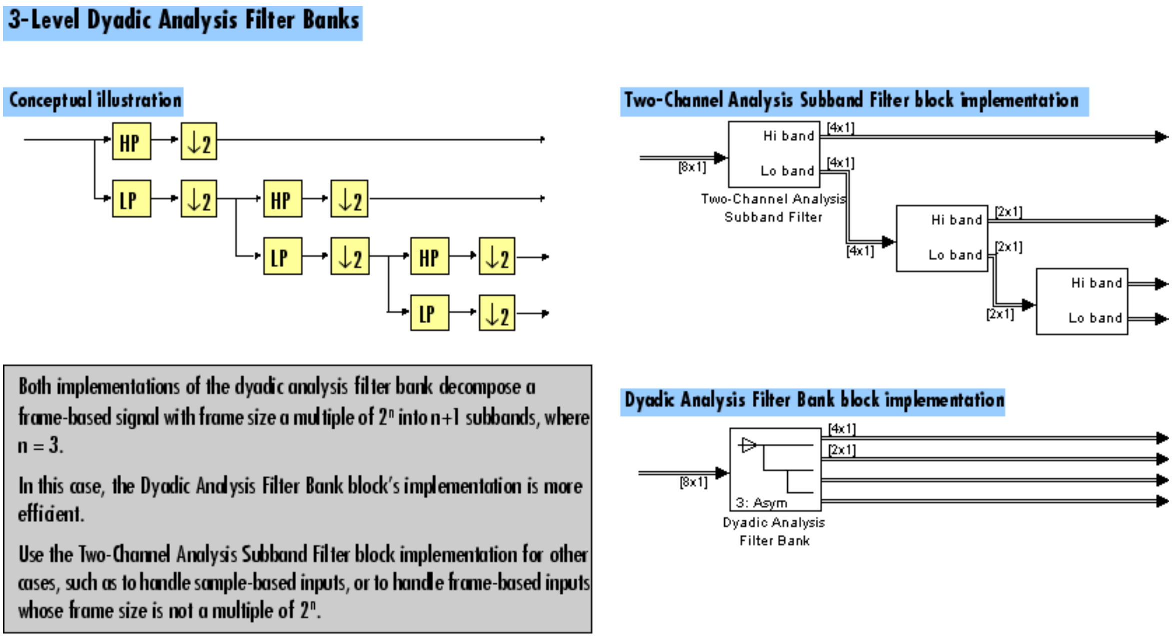
The Two-Channel Analysis Subband Filter block is the basic unit of a dyadic analysis filter bank. You can connect several of these blocks to implement an *n*-level filter bank, as illustrated in the following figure. For a review of dyadic analysis filter banks, see the Dyadic Analysis Filter Bank block reference page.

When you create a filter bank by connecting multiple copies of this block, the output values of the filter bank differ depending on whether there is latency. See the previous table,

For instance, for frame-based inputs, the filter bank output values differ depending on whether you set the **Framing** parameter to `Maintain input frame rate` (no latency), or `Maintain input frame size` (one frame of latency for every block). Though the output values differ, both sets of values are valid; the difference arises from changes in latency.

In some cases, rather than connecting several Two-Channel Analysis Subband Filter blocks, it is faster and requires less memory to use the Dyadic Analysis Filter Bank block. In particular, use the Dyadic

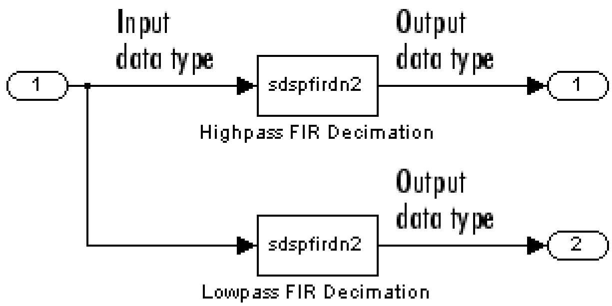
Analysis Filter Bank block when you want to decompose a frame-based signal with frame size a multiple of 2^n into n+1 or 2^n subbands. In all other cases, use Two-Channel Analysis Subband Filter blocks to implement your filter banks.



The Dyadic Analysis Filter Bank block allows you to specify the filter bank filters by providing vectors of filter coefficients, just as this block does. The Dyadic Analysis Filter Bank block provides an additional option of using wavelet-based filters that the block designs by using a wavelet you specify.

Fixed-Point Data Types

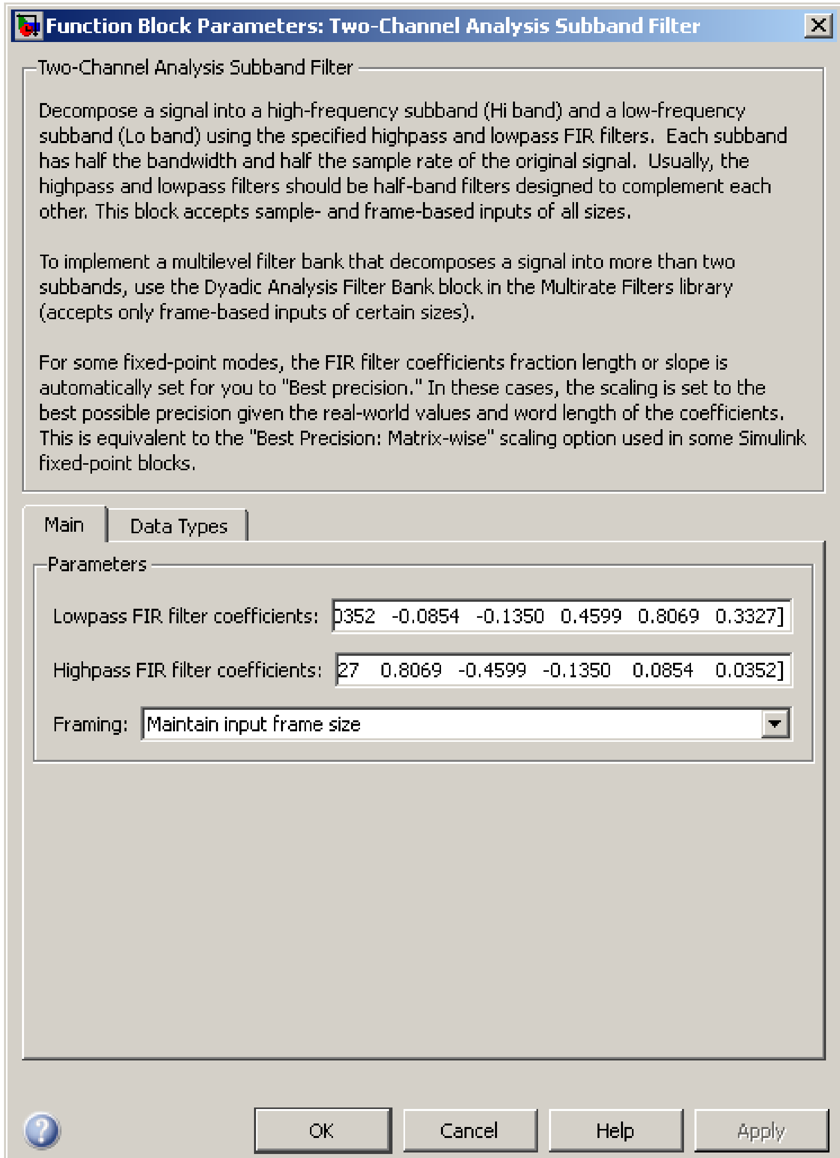
The Two-Channel Analysis Subband Filter block is comprised of two FIR Decimation blocks as shown in the following diagram.



For fixed-point signals, you can set the coefficient, product output, accumulator, and output data types of the FIR Decimation blocks as discussed in "Dialog Box" on page 2-1345. For a diagram showing the usage of these data types, see the FIR Decimation block reference page.

Dialog Box

The **Main** pane of the Two-Channel Analysis Subband Filter block dialog appears as follows.



Lowpass FIR filter coefficients

Specify a vector of lowpass FIR filter coefficients, in descending powers of z. The lowpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the **Highpass FIR filter coefficients** parameter. The default values of this parameter specify a filter based on a 3rd-order Daubechies wavelet. When you use the Two-Channel Synthesis Subband Filter block to reconstruct the input to this block, you need to design perfect reconstruction filters to use in the synthesis subband filter. For more information, see "Specifying the FIR Filters" on page 2-1339.

Highpass FIR filter coefficients

Specify a vector of highpass FIR filter coefficients, in descending powers of z. The highpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the **Lowpass FIR filter coefficients** parameter. The default values of this parameter specify a filter based on a 3rd-order Daubechies wavelet. When you use the Two-Channel Synthesis Subband Filter block to reconstruct the input to this block, you need to design perfect reconstruction filters to use in the synthesis subband filter. For more information, see "Specifying the FIR Filters" on page 2-1339.

Framing

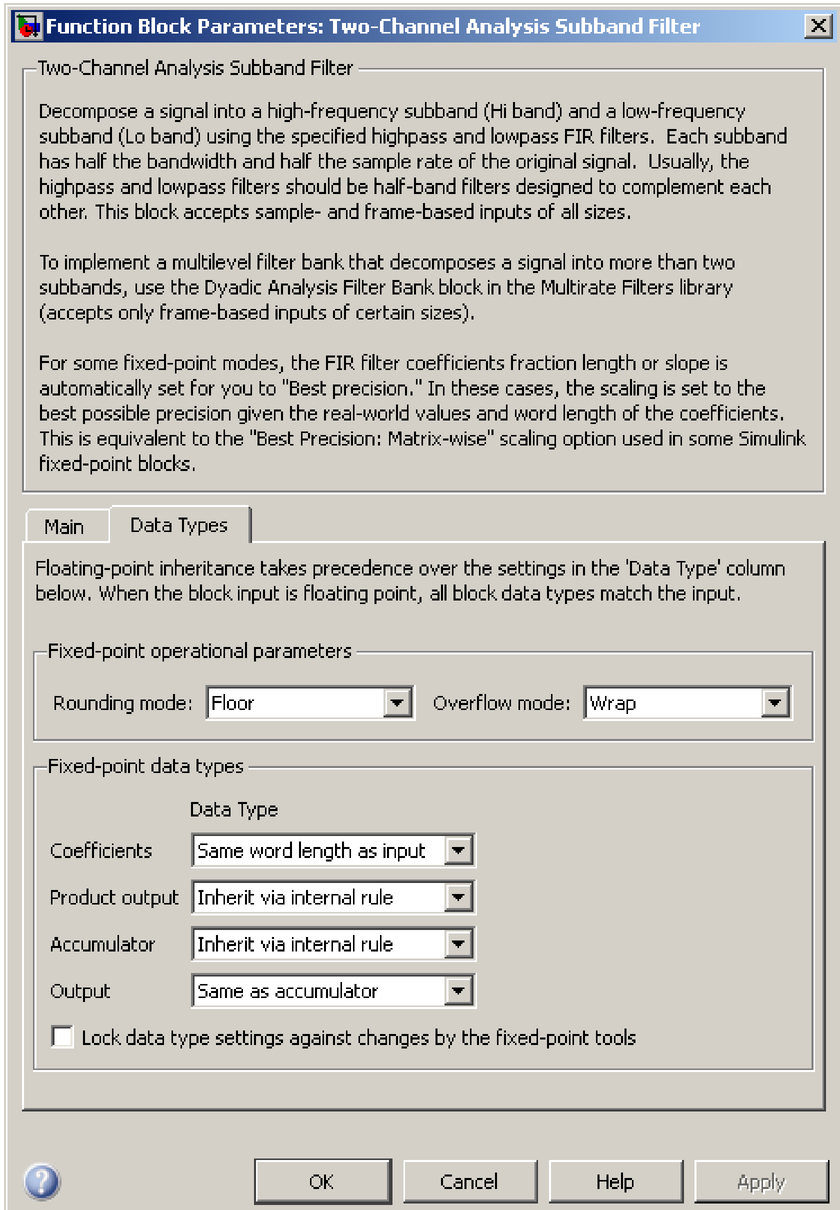
Specify the method by which to implement the decimation for frame-based inputs:

Select `Maintain input frame size` to halve the output frame rate

Select `Maintain input frame rate` to halve the output frame size

For more information, see "Frame-Based Operation" on page 2-1340. Some settings of this parameter causes the block to have nonzero latency, as described in "Latency" on page 2-1341.

The **Data Types** pane of the Two-Channel Analysis Subband Filter block dialog appears as follows.


Function Block Parameters: Two-Channel Analysis Subband Filter
Two-Channel Analysis Subband Filter
Decompose a signal into a high-frequency subband (Hi band) and a low-frequency subband (Lo band) using the specified highpass and lowpass FIR filters. Each subband has half the bandwidth and half the sample rate of the original signal. Usually, the highpass and lowpass filters should be half-band filters designed to complement each other. This block accepts sample- and frame-based inputs of all sizes.
To implement a multilevel filter bank that decomposes a signal into more than two subbands, use the Dyadic Analysis Filter Bank block in the Multirate Filters library (accepts only frame-based inputs of certain sizes).
For some fixed-point modes, the FIR filter coefficients fraction length or slope is automatically set for you to "Best precision." In these cases, the scaling is set to the best possible precision given the real-world values and word length of the coefficients. This is equivalent to the "Best Precision: Matrix-wise" scaling option used in some Simulink fixed-point blocks.
Main
Data Types
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input.
Fixed-point operational parameters
Rounding mode: Floor
Overflow mode: Wrap
Fixed-point data types
Data Type
Coefficients Same word length as input
Product output Inherit via internal rule
Accumulator Inherit via internal rule
Output Same as accumulator
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Rounding mode

Select the rounding mode for fixed-point operations. The filter coefficients do not obey this parameter; they always round to `Nearest`.

Overflow mode

Select the overflow mode for fixed-point operations. The filter coefficients do not obey this parameter; they are always saturated.

Coefficients

Choose how you specify the word length and the fraction length of the FIR filter coefficients:

When you select `Same word length as input`, the word length of the filter coefficients match that of the input to the block. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

When you select `Specify word length`, you can enter the word length of the coefficients, in bits. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

When you select `Binary point scaling`, you can enter the word length and the fraction length of the coefficients, in bits.

When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the coefficients. This block requires power-of-two slope and a bias of zero.

The filter coefficients do not obey the **Rounding mode** and the **Overflow mode** parameters; they are always saturated and rounded to `Nearest`.

Product output

Use this parameter to specify how you would like to designate the product output word and fraction lengths. See "Fixed-Point Data Types" on page 2-522 of the FIR Decimation reference page and "Multiplication Data Types" for illustrations depicting the use of the product output data type in the FIR Decimation blocks of this block:

When you select `Inherit via internal rule`, the product output word length and fraction length are calculated automatically. For information about how the product output word and fraction lengths are calculated when an internal rule is used, see "Inherit via Internal Rule".

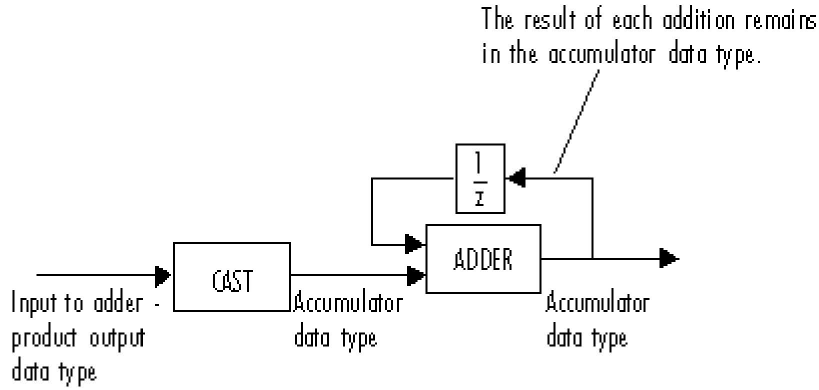
Note The actual product output word length may be equal to or greater than the calculated ideal product output word length, depending on the settings on the **Hardware Implementation** pane of the Configuration Parameters dialog box.

When you select `Same as input`, these characteristics match those of the input to the block.

When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator



As depicted above, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how you would like to designate this accumulator word and fraction lengths.

You also use this parameter to specify the accumulator word and fraction lengths resulting from a complex-complex multiplication in the FIR Decimation blocks in this block. See "Multiplication Data Types" for more information:

When you select `Inherit via internal rule`, the accumulator word length and fraction length are calculated automatically. For information about how the accumulator word and fraction lengths are calculated when an internal rule is used, see "Inherit via Internal Rule".

When you select `Same as product output`, these characteristics match those of the product output

When you select `Same as input`, these characteristics match those of the input to the block.

When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Choose how you specify the output word length and fraction length of the FIR Decimation blocks, as well as of the final overall filter output:

When you select `Same as accumulator`, these characteristics match those of the accumulator.

A special case occurs when `Inherit via internal rule` is specified for **Accumulator**, and block inputs and coefficients are complex. In that case, the output word length is one less than the accumulator word length.

When you select `Same as product output`, these characteristics match those of the product output

When you select `Same as input`, these characteristics match those of the input to the block.

When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

References

Fliege, N. J. *Multirate Digital Signal Processing: Multirate Systems, Filter Banks, Wavelets*. West Sussex, England: John Wiley & Sons, 1994.

Strang, G. and T. Nguyen. *Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1996.

Vaidyanathan, P. P. *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice Hall, 1993.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed only)
- 8-, 16-, and 32-bit signed integers

See Also

DWT	Signal Processing Blockset
Dyadic Analysis Filter Bank	Signal Processing Blockset
FIR Decimation	Signal Processing Blockset
IDWT	Signal Processing Blockset
Two-Channel Synthesis Subband Filter	Signal Processing Blockset
`fir1`	Signal Processing Toolbox
`fir2`	Signal Processing Toolbox
`firls`	Signal Processing Toolbox
`wfilters`	Wavelet Toolbox

For related information, see “Multirate Filters”.

Two-Channel Synthesis Subband Filter

Purpose

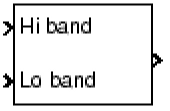
Reconstruct signal from high-frequency subband and low-frequency subband

Library

Filtering / Multirate Filters

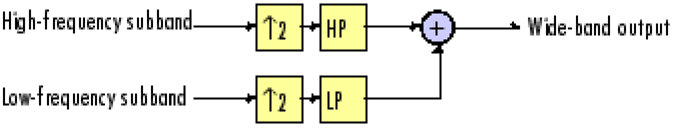
`dspmlti4`

Description



The Two-Channel Synthesis Subband Filter block reconstructs a signal from its high-frequency subband and low-frequency subband, each with half the bandwidth and half the sample rate of the original signal. Use this block to reconstruct signals decomposed by the Two-Channel Analysis Subband Filter block.

The block upsamples the high- and low-frequency subbands by 2, and then filters the results with a pair of highpass and lowpass FIR filters, as illustrated in the following figure.



Note that the block implements the FIR filtering and downsampling steps together using a polyphase filter structure, which is more efficient than the straightforward interpolate-then-filter algorithm illustrated above.

You must provide the vector of filter coefficients for the two filters. Each filter should be a half-band filter that passes the frequency band that the other filter stops. To use this block to reconstruct the output of a Two-Channel Analysis Subband Filter block, the filters in this block *must* be designed to perfectly reconstruct the outputs of the analysis filters.

Note By connecting many copies of this block, you can implement a multilevel dyadic synthesis filter bank. In some cases, it is more efficient to use the Dyadic Synthesis Filter Bank block instead. For more information, see "Creating Multilevel Dyadic Synthesis Filter Banks" on page 2-1358.

Sections of This Reference Page

Specifying the FIR Filters

You must provide the vector of numerator coefficients for the lowpass and highpass filters in the **Lowpass FIR filter coefficients** and **Highpass FIR filter coefficients** parameters.

For example, to specify a filter with the following transfer function, enter the vector `[b(1) b(2) ...  b(m)]`.

$$H(z) = B(z) = b_1 + b_2 z^{-1} + \ldots + b_m z^{-(m-1)}$$

Each filter should be a half-band filter that passes the frequency band that the other filter stops. To use this block to reconstruct the output of a Two-Channel Analysis Subband Filter block, the filters in this block must be designed to perfectly reconstruct the outputs of the analysis filters.

The best way to design perfect reconstruction filters is to use the Wavelet Toolbox `wfilters` function for the filters in both this block *and* in the corresponding Two-Channel Analysis Subband Filter block. You can also use Filter Design Toolbox functions and Signal Processing Toolbox functions. To learn how to design your own perfect reconstruction filters, see "References" on page 2-1369.

The block initializes all filter states to zero.

Sample-Based Operation

- "Valid Sample-Based Inputs" on page 2-1356
- "Sample-Based Outputs" on page 2-1356

Valid Sample-Based Inputs

The block accepts any two M-by-N sample-based matrices with the same sample rates. The block treats each M-by-N matrix as MxN independent subbands, where MxN is the product of the matrix dimensions. Each matrix element is the high- or low-frequency subband of the corresponding channel in the output matrix. The input to the topmost input port should contain the high-frequency subbands.

Sample-Based Outputs

Given valid sample-based inputs, the block outputs one sample-based matrix with the same dimensions as the inputs. The output sample rate is twice that of the input. Each element of the output is a single channel, reconstructed from the corresponding elements in each input matrix. Depending on the Simulink configuration parameters, some sample-based outputs can have one sample of latency, as described in "Latency" on page 2-1357.

Frame-Based Operation

- "Valid Frame-Based Inputs" on page 2-1357

- "Frame-Based Outputs" on page 2-1357

Valid Frame-Based Inputs

The block accepts any two M-by-N frame-based matrices with the same frame rates. The block treats each input column as the high- or low-frequency subbands of the corresponding output channel. The input to the topmost input port should contain the high-frequency subbands.

Frame-Based Outputs

Given valid frame-based inputs, the block outputs a frame-based matrix. Each output column is a single channel, reconstructed from the corresponding columns in each input matrix.

The sample rate of the output is twice that of the input. The **Framing** parameter sets whether the block doubles the sample rate by doubling the output frame size, or doubling the output frame rate:

- `Maintain input frame size` — The input and output frame *sizes* are the same, but the frame *rate* of the output is twice that of the input. So, the overall sample rate of the output is twice that of the input. This setting causes the block to have one frame of latency, as described in "Latency" on page 2-1341.
- `Maintain input frame rate` — The input and output frame *rates* are the same, but the frame *size* of the output is twice that of the input. So, the overall sample rate of the output is twice that of the input.

Latency

In some cases, the block has nonzero tasking latency, which means that there is a constant delay between the time that the block receives an input, and produces the corresponding output, as summarized below and in the following table:

- For sample-based inputs, there are cases where the block exhibits *one-sample latency*. In such cases, when the block receives the *n*th input sample, it produces the outputs corresponding to the *n-1*th input sample. When the block receives the first input sample, the block outputs an initial value of zero in each output channel.

- For frame-based inputs, there are cases where the block exhibits *one-frame latency*. In such cases, when the block receives the *n*th input frame, it produces the outputs corresponding to the *n-1*th input frame. When the block receives the first input frame, the block outputs a frame of zeros.

Note For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

Amount of Block Latency for All Possible Block Settings

Input	Latency	No Latency
Sample based	One sample of latency when the **Tasking mode for periodic sample times** parameter is set to `MultiTasking` or `Auto` in the **Solver** pane of the Configuration Parameters dialog box. The first output sample of each channel is always `0`.	The **Tasking mode for periodic sample times** parameter is set to `SingleTasking` in the **Solver** pane of the Configuration Parameters dialog box.
Frame based	One frame of latency when the **Framing** parameter is set to `Maintain input frame size`. The first output frame is always all zeros.	The **Framing** parameter is set to `Maintain input frame rate`.

Creating Multilevel Dyadic Synthesis Filter Banks

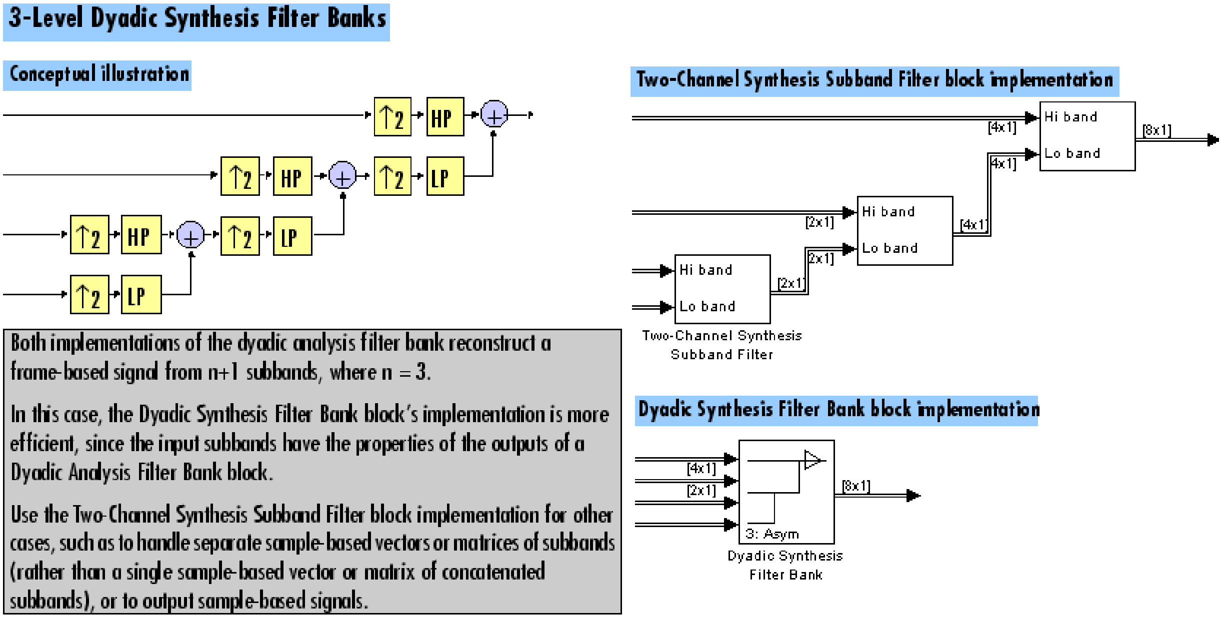
The Two-Channel Synthesis Subband Filter block is the basic unit of a dyadic synthesis filter bank. You can connect several of these blocks to implement an n-level filter bank, as illustrated in the following figure. For a review of dyadic synthesis filter banks, see the Dyadic Synthesis Filter Bank block reference page.

When you create a filter bank by connecting multiple copies of this block, the output values of the filter bank differ depending on whether there is latency. See the previous table, Amount of Block Latency for All Possible Block Settings on page 2-1358.

For instance, for frame-based inputs, the filter bank output values differ depending on whether you set the **Framing** parameter to `Maintain input frame rate` (no latency), or `Maintain input frame size` (one frame of latency for every block). Though the output values differ, both sets of values are valid; the difference arises from changes in latency.

In some cases, rather than connecting several Two-Channel Synthesis Subband Filter blocks, it is faster and requires less memory to use the Dyadic Synthesis Filter Bank block. In particular, use the Dyadic Synthesis Filter Bank block to reconstruct a frame-based signal (with frame size a multiple of 2^n) from 2^n or n+1 subbands whose properties match those of the Dyadic Analysis Filter Bank block's outputs. These properties are described in the Dyadic Analysis Filter Bank reference page.

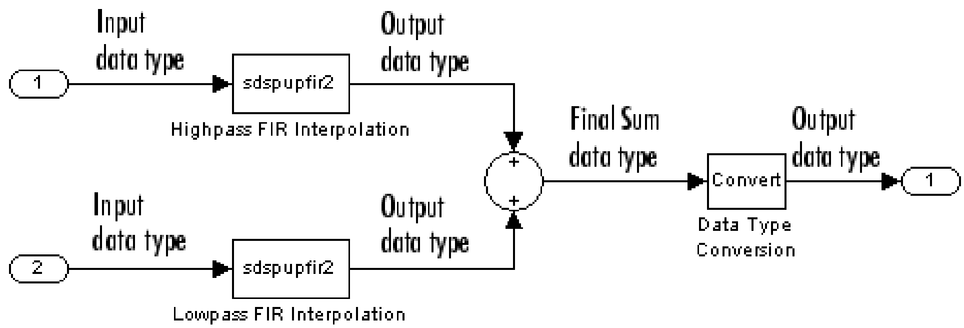
Two-Channel Synthesis Subband Filter



The Dyadic Synthesis Filter Bank block allows you to specify the filter bank filters by providing vectors of filter coefficients, just as this block does. The Dyadic Synthesis Filter Bank block provides an additional option of using wavelet-based filters that the block designs by using a wavelet you specify.

Fixed-Point Data Types

The Two-Channel Synthesis Subband Filter block is comprised of two FIR Interpolation blocks as shown in the following diagram.

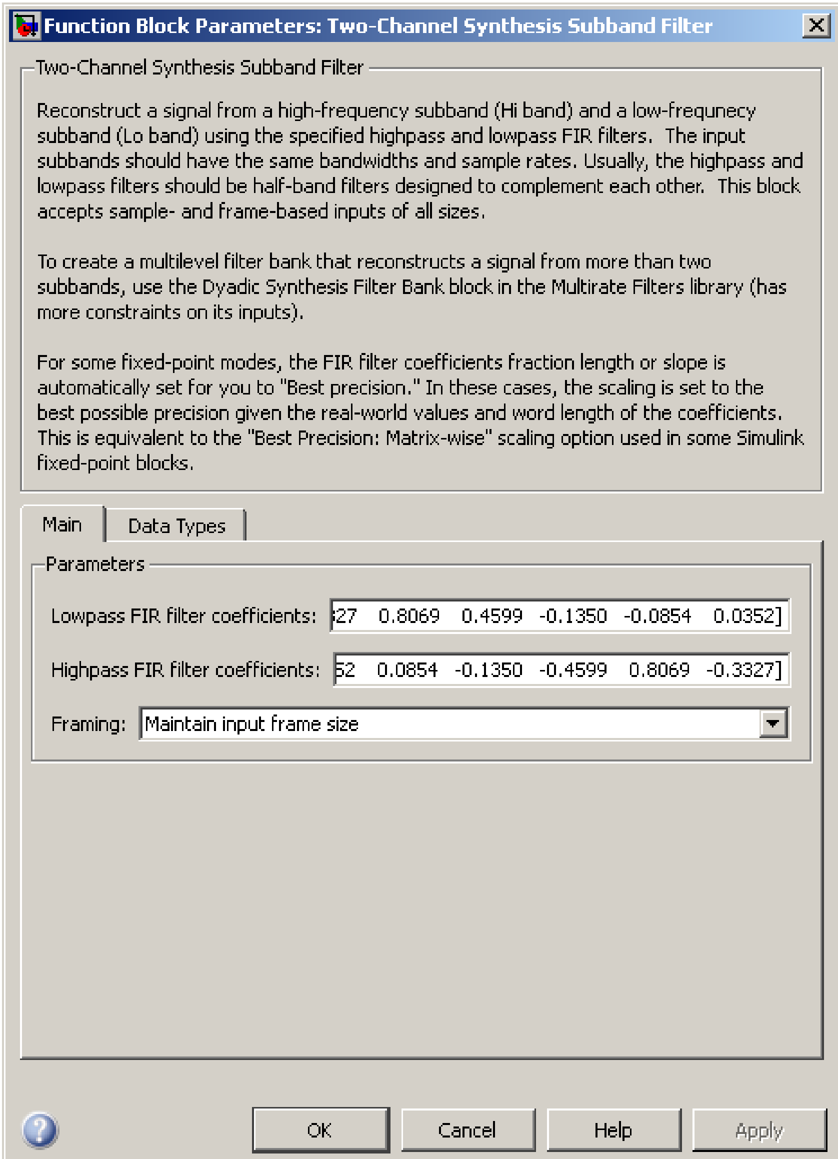


For fixed-point signals, you can set the coefficient, product output, accumulator, and output data types used in the FIR Interpolation blocks as discussed in "Dialog Box" on page 2-1362 below. For a diagram showing the usage of these data types within the FIR blocks, see the FIR Interpolation block reference page.

In addition, the inputs to the Sum block in the diagram above are accumulated using the accumulator data type. The output of the Sum block is then cast from the accumulator data type to the output data type. Therefore the output of the Two-Channel Synthesis Subband Filter block is in the output data type. You also set these data types in the block dialog as discussed in "Dialog Box" on page 2-1362 below.

Dialog Box

The **Main** pane of the Two-Channel Synthesis Subband Filter block dialog appears as follows.



Lowpass FIR filter coefficients

A vector of lowpass FIR filter coefficients, in descending powers of *z*. The lowpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the **Highpass FIR filter coefficients** parameter. To use this block to reconstruct the output of a Two-Channel Analysis Subband Filter block, you must design the filters in this block to perfectly reconstruct the outputs of the analysis filters. For more information, see "Specifying the FIR Filters" on page 2-1355.

Highpass FIR filter coefficients

A vector of highpass FIR filter coefficients, in descending powers of *z*. The highpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the **Lowpass FIR filter coefficients** parameter. To use this block to reconstruct the output of a Two-Channel Analysis Subband Filter block, you must design the filters in this block to perfectly reconstruct the outputs of the analysis filters. For more information, see "Specifying the FIR Filters" on page 2-1355.

Framing

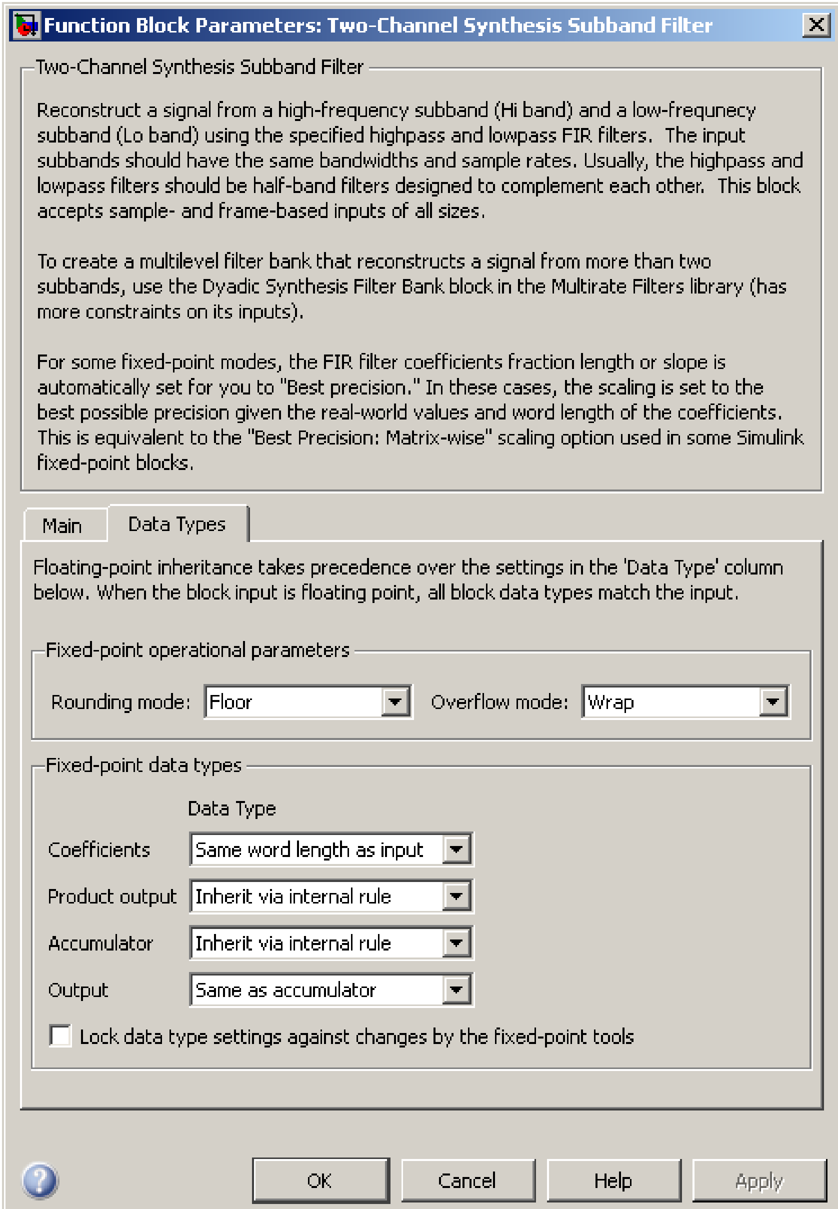
Select the method by which to implement the interpolation for frame-based inputs:

Select `Maintain input frame size` to double the output frame rate

Select `Maintain input frame rate` to double the output frame size

For more information, see "Frame-Based Operation" on page 2-1340. Some settings of this parameter causes the block to have nonzero latency, as described in "Latency" on page 2-1341.

The **Data Types** pane of the Two-Channel Synthesis Subband Filter block dialog appears as follows.


Function Block Parameters: Two-Channel Synthesis Subband Filter
Two-Channel Synthesis Subband Filter
Reconstruct a signal from a high-frequency subband (Hi band) and a low-frequency subband (Lo band) using the specified highpass and lowpass FIR filters. The input subbands should have the same bandwidths and sample rates. Usually, the highpass and lowpass filters should be half-band filters designed to complement each other. This block accepts sample- and frame-based inputs of all sizes.
To create a multilevel filter bank that reconstructs a signal from more than two subbands, use the Dyadic Synthesis Filter Bank block in the Multirate Filters library (has more constraints on its inputs).
For some fixed-point modes, the FIR filter coefficients fraction length or slope is automatically set for you to "Best precision." In these cases, the scaling is set to the best possible precision given the real-world values and word length of the coefficients. This is equivalent to the "Best Precision: Matrix-wise" scaling option used in some Simulink fixed-point blocks.
Main
Data Types
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input.
Fixed-point operational parameters
Rounding mode:
Floor
Overflow mode:
Wrap
Fixed-point data types
Data Type
Coefficients
Same word length as input
Product output
Inherit via internal rule
Accumulator
Inherit via internal rule
Output
Same as accumulator
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Round mode
Select the rounding mode for fixed-point operations. The filter coefficients do not obey this parameter; they always round to `Nearest`.

Overflow mode
Select the overflow mode for fixed-point operations. The filter coefficients do not obey this parameter; they are always saturated.

Coefficients
Choose how you specify the word length and the fraction length of the FIR filter coefficients:

When you select `Same word length as input`, the word length of the filter coefficients match that of the input to the block. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

When you select `Specify word length`, you can enter the word length of the coefficients, in bits. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

When you select `Binary point scaling`, you can enter the word length and the fraction length of the coefficients, in bits.

When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the coefficients. This block requires power-of-two slope and a bias of zero.

The filter coefficients do not obey the **Rounding mode** and the **Overflow mode** parameters; they are always saturated and rounded to `Nearest`.

Product output

Use this parameter to specify how you would like to designate the product output word and fraction lengths. See “Fixed-Point Data Types” on page 2-545 of the FIR Interpolation reference page and “Multiplication Data Types” for illustrations depicting the use of the product output data type in the FIR Interpolation blocks of this block:

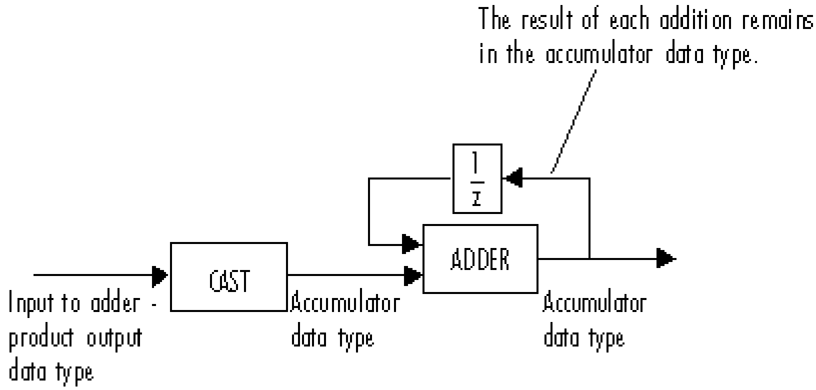
When you select `Inherit via internal rule`, the product output word length and fraction length are calculated automatically. For information about how the product output word and fraction lengths are calculated when an internal rule is used, see “Inherit via Internal Rule”.

When you select `Same as input`, these characteristics match those of the input to the block.

When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.

When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator



As depicted above, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how you would like to designate this accumulator word and fraction lengths.

You also use this parameter to specify the accumulator word and fraction lengths resulting from a complex-complex multiplication in the FIR Interpolation blocks in this block. See “Multiplication Data Types” for more information:

When you select `Inherit via internal rule`, the accumulator word length and fraction length are calculated automatically. For information about how the accumulator word and fraction lengths are calculated when an internal rule is used, see “Inherit via Internal Rule”.

When you select `Same as product output`, these characteristics match those of the product output

When you select `Same as input`, these characteristics match those of the input to the block.

When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.

When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Choose how you specify the output word length and fraction length of the FIR Interpolation blocks, as well as of the final overall filter output:

When you select `Same as accumulator`, these characteristics match those of the accumulator.

A special case occurs when `Inherit via internal rule` is specified for **Accumulator**, and block inputs and coefficients are complex. In that case, the output word length is one less than the accumulator word length.

When you select `Same as product output`, these characteristics match those of the product output

When you select `Same as input`, these characteristics match those of the input to the block.

When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.

When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

References

Fliege, N. J. *Multirate Digital Signal Processing: Multirate Systems, Filter Banks, Wavelets*. West Sussex, England: John Wiley & Sons, 1994.

Strang, G. and T. Nguyen. *Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1996.

Vaidyanathan, P. P. *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice Hall, 1993.

Supported Data Types

- Double-precision floating point
- Single-precision floating point
- Fixed point (signed only)
- 8-, 16-, and 32-bit signed integers

See Also

DWT	Signal Processing Blockset
Dyadic Synthesis Filter Bank	Signal Processing Blockset
FIR Interpolation	Signal Processing Blockset
IDWT	Signal Processing Blockset
Two-Channel Analysis Subband Filter	Signal Processing Blockset
`fir1`	Signal Processing Toolbox
`fir2`	Signal Processing Toolbox
`firls`	Signal Processing Toolbox
`wfilters`	Wavelet Toolbox

For related information, see “Multirate Filters”.

UDP Receive

Purpose

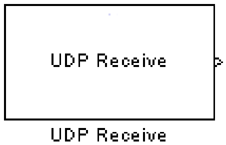
Receive `uint8` vector as UDP message

Library

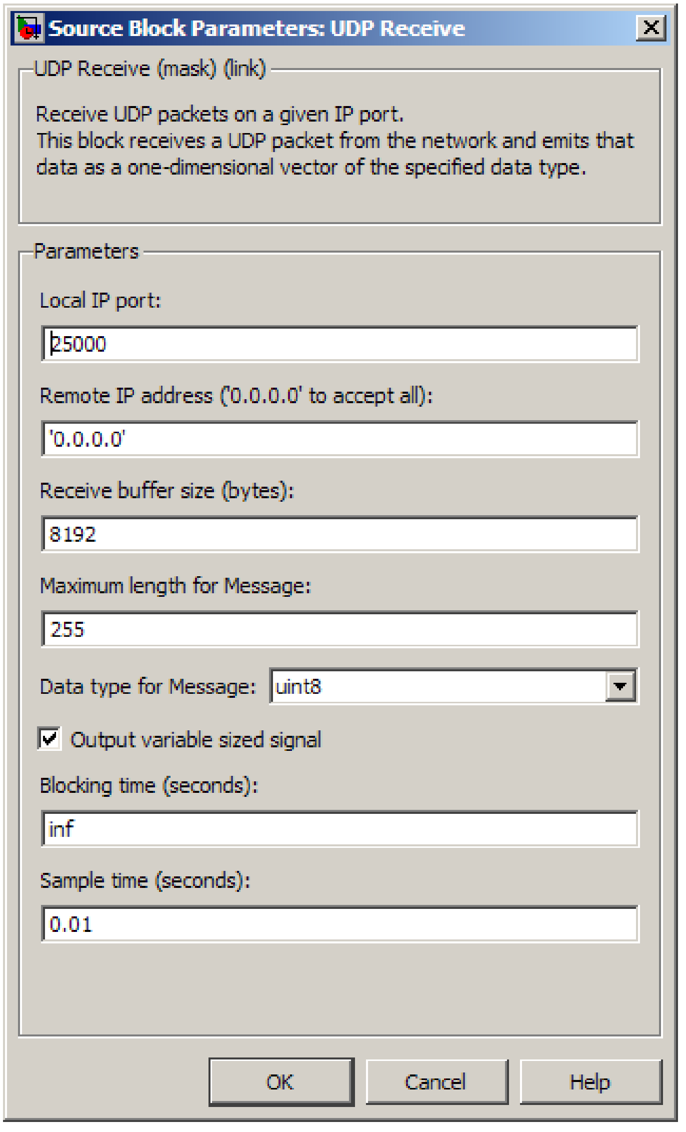
Signal Processing Sources

`dspsrcs4`

Description



The UDP Receive block receives UDP packets from an IP network port and saves them to its buffer. With each sample, the block output, emits the contents of a single UDP packet as a data vector.



Dialog

Local IP port

Specify the IP port number upon to receive UDP packets. This value defaults to 25000. The value can range 1–65535.

Remote IP address (0.0.0.0 to accept all)

Specify the IP address from which to accept packets. Entering a specific IP address blocks UDP packets from any other address. To accept packets from any IP address, enter `'0.0.0.0'`. This value defaults to `'0.0.0.0'`.

Receive buffer size (bytes)

Make the receive buffer large enough to avoid data loss caused by buffer overflows. This value defaults to `8192`.

Maximum length for Message

Specify the maximum length, in vector elements, of the data output vector. Set this parameter to a value equal or greater than the data size of any UDP packet. The system truncates data that exceeds this length. This value defaults to `255`.

If you disable **Output variable sized signal**, the block outputs a fixed-length output the same length as the **Maximum length for Message**.

Data type for Message

Set the data type of the vector elements in the Message output. Match the data type with the data input used to create the UDP packets. This option defaults to `uint8`.

Output variable sized signal

If your model supports signals of varying length, enable the **Output variable sized signal** parameter. This checkbox defaults to selected (enabled). In that case:

- The output vector varies in length, depending on the amount of data in the UDP packet.
- The block emits the data vector from a single unlabeled output.

If your model does not support signals of varying length, disable the **Output variable sized signal** parameter. In that case:

- The block emits a fixed-length output the same length as the **Maximum length for Message**.

- If the UDP packet contains less data than the fixed-length output, the difference contains invalid data.
- The block emits the data vector from the **Message** output.
- The block emits the length of the valid data from the **Length** output.
- The block dialog box displays the **Data type for Length** parameter.

In both cases, the block truncates data that exceeds the **Maximum length for Message**.

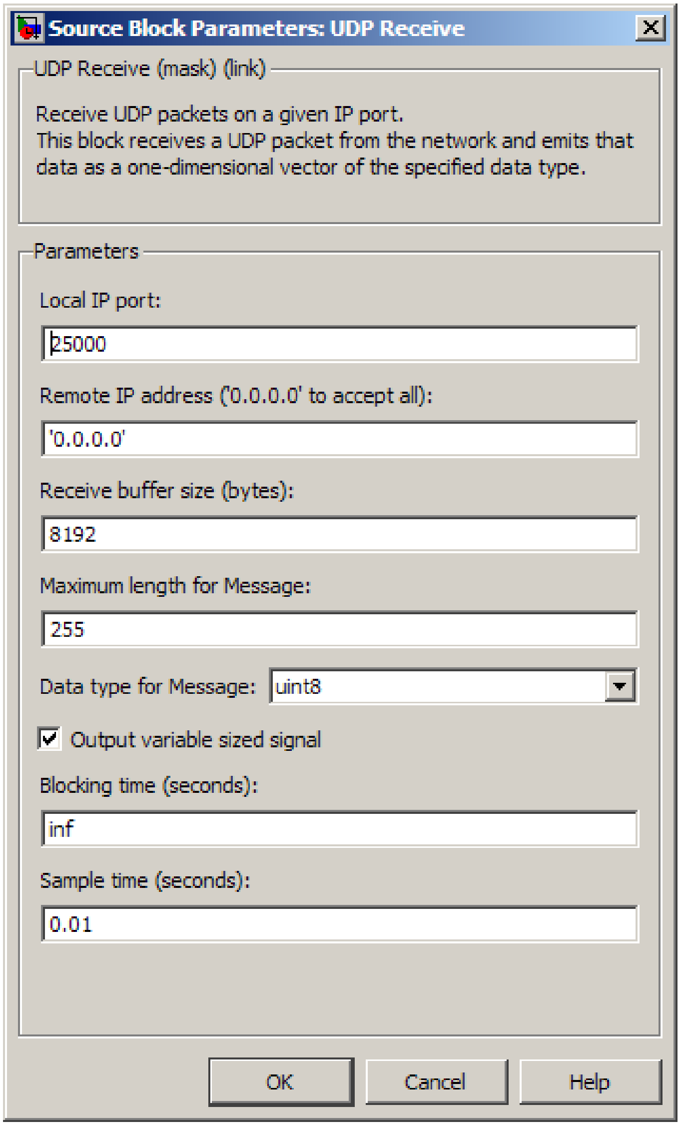
Data type for Length
Set the data type of the Length output. This option defaults to `double`.

Blocking time (seconds)
For each sample, wait this length of time for a UDP packet before returning control to the scheduler. This value defaults to `inf`, which indicates to wait indefinitely.

Note This parameter appears only in the UDP Receive block from the Target Support Package product.

Sample time (seconds)
Specify how often the scheduler runs this block. Enter a value greater than zero. In real-time operation, setting this option to a large value reduces the likelihood of dropped UDP messages. This value defaults to a sample time of 0.01 s.

Dialog



Local IP port

Specify the IP port number upon to receive UDP packets. This value defaults to `25000`. The value can range 1–65535.

Remote IP address (0.0.0.0 to accept all)
Specify the IP address from which to accept packets. Entering a specific IP address blocks UDP packets from any other address. To accept packets from any IP address, enter `'0.0.0.0'`. This value defaults to `'0.0.0.0'`.

Receive buffer size (bytes)
Make the receive buffer large enough to avoid data loss caused by buffer overflows. This value defaults to `8192`.

Maximum length for Message
Specify the maximum length, in vector elements, of the data output vector. Set this parameter to a value equal or greater than the data size of any UDP packet. The system truncates data that exceeds this length. This value defaults to `255`.

If you disable **Output variable sized signal**, the block outputs a fixed-length output the same length as the **Maximum length for Message**.

Data type for Message
Set the data type of the vector elements in the Message output. Match the data type with the data input used to create the UDP packets. This option defaults to `uint8`.

Output variable sized signal
If your model supports signals of varying length, enable the **Output variable sized signal** parameter. This checkbox defaults to selected (enabled). In that case:

- The output vector varies in length, depending on the amount of data in the UDP packet.
- The block emits the data vector from a single unlabeled output.

If your model does not support signals of varying length, disable the **Output variable sized signal** parameter. In that case:

- The block emits a fixed-length output the same length as the **Maximum length for Message**.

- If the UDP packet contains less data than the fixed-length output, the difference contains invalid data.
- The block emits the data vector from the **Message** output.
- The block emits the length of the valid data from the **Length** output.
- The block dialog box displays the **Data type for Length** parameter.

In both cases, the block truncates data that exceeds the **Maximum length for Message**.

Data type for Length
Set the data type of the Length output. This option defaults to `double`.

Blocking time (seconds)
For each sample, wait this length of time for a UDP packet before returning control to the scheduler. This value defaults to `inf`, which indicates to wait indefinitely.

Note This parameter appears only in the UDP Receive block from the Target Support Package product.

Sample time (seconds)
Specify how often the scheduler runs this block. Enter a value greater than zero. In real-time operation, setting this option to a large value reduces the likelihood of dropped UDP messages. This value defaults to a sample time of 0.01 s.

See Also

Byte Pack, Byte Reversal, Byte Unpack, UDP Send

Purpose

Send UDP message

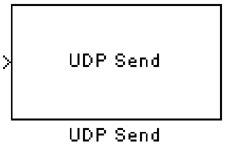
Library

Signal Processing Sinks

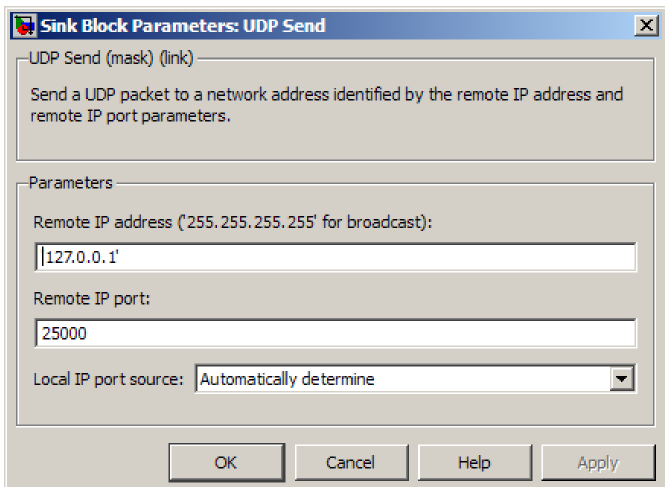
`dspsnks4`

Description

The UDP Send block transmits an input vector as a UDP message over an IP network port.



Dialog Box



IP address (255.255.255.255 for broadcast)

Specify the IP address or hostname to which the block sends the message. To broadcast the UDP message, retain the default value, `'255.255.255.255'`.

Remote IP port
Specify the port to which the block sends the message. The value defaults to 25000, but the values range from 1–65535.

Local IP port source
To let the system automatically assign the port number, select `Assign automatically`. To specify the IP port number using the **Local IP port** parameter, select `Specify`.

Local IP port
Specify the IP port number from which the block sends the message.

If the receiving address expects messages from a particular port number, enter that number here.

See Also

Byte PackByte Reversal, Byte Unpack, UDP Receive

Unbuffer

Purpose

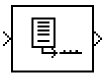
Unbuffer input frame into sequence of scalar outputs

Library

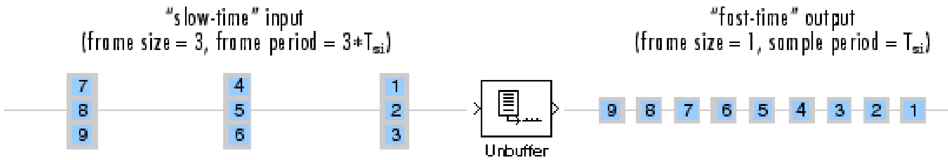
Signal Management / Buffers

`dspbuff3`

Description

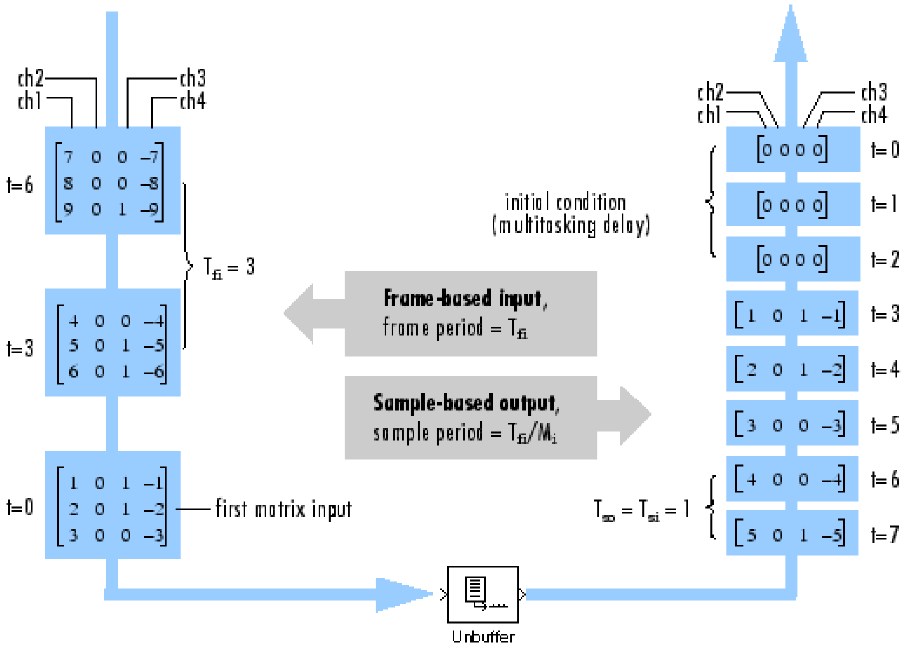


The Unbuffer block unbuffers an M_i-by-N frame-based input into a 1-by-N sample-based output. That is, inputs are unbuffered *row-wise* so that each matrix row becomes an independent time-sample in the output. The rate at which the block receives inputs is generally less than the rate at which the block produces outputs.



The block adjusts the output rate so that the *sample period* is the same at both the input and output, $T_{so}=T_{si}$. Therefore, the output sample period for an input of frame size M_i and frame period T_{fi} is T_{fi}/M_i, which represents a *rate* M_i times higher than the input frame rate. In the example above, the block receives inputs only once every three sample periods, but produces an output once every sample period. To rebuffer frame-based inputs to a larger or smaller frame size, use the Buffer block.

In the model below, the block unbuffers a four-channel frame-based input with frame size 3. The **Initial conditions** parameter is set to zero and the tasking mode is set to multitasking, so the first three outputs are zero vectors.



Zero Latency

The Unbuffer block has *zero-tasking latency* in the Simulink single-tasking mode. Zero-tasking latency means that the first input sample (received at *t*=0) appears as the first output sample.

Nonzero Latency

For *multitasking* operation, the Unbuffer block's buffer is initialized with the value specified by the **Initial condition** parameter, and the block begins unbuffering this frame at the start of the simulation. Inputs to the block are therefore delayed by one buffer length, or M_i samples.

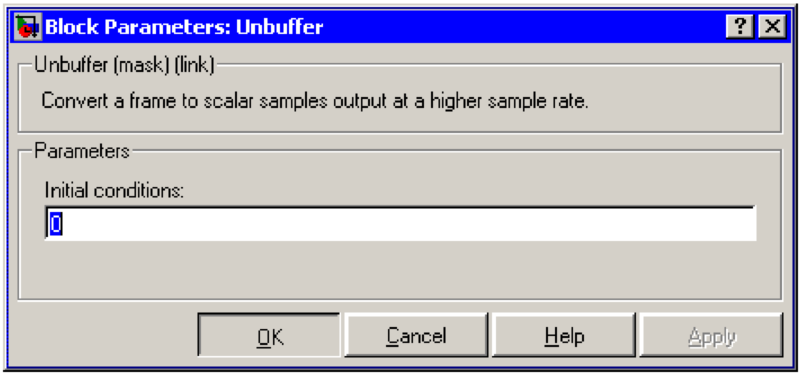
The **Initial condition** parameter can be one of the following:

- A scalar to be repeated for the first M_i output samples of every channel

- A length-M_i vector containing the values of the first M_i output samples for every channel
- An M_i-by-N matrix containing the values of the first M_i output samples in each of N channels

Note For more information on latency and the Simulink tasking modes, see “Excess Algorithmic Delay (Tasking Latency)” and “Scheduling Considerations” in the *Real-Time Workshop User’s Guide*.

Dialog Box



Initial conditions
: The value of the block’s initial output for cases of nonzero latency; a scalar, vector, or matrix.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Buffer	Signal Processing Blockset

See "Unbuffering Frame-Based Signals into Sample-Based Signals" for related information.

Purpose

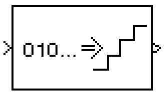
Decode integer input into floating-point output

Library

Quantizers

`dspquant2`

Description



The Uniform Decoder block performs the inverse operation of the Uniform Encoder block, and reconstructs quantized floating-point values from encoded integer input. The block adheres to the definition for uniform decoding specified in ITU-T Recommendation G.701.

Inputs can be real or complex values of the following six integer data types: `uint8`, `uint16`, `uint32`, `int8`, `int16`, or `int32`.

The block first casts the integer input values to floating-point values, and then uniquely maps (decodes) them to one of 2^B uniformly spaced floating-point values in the range [-V, $(1-2^{1-B})$V], where you specify B in the **Bits** parameter (as an integer between 2 and 32) and V is a floating-point value specified by the **Peak** parameter. The smallest input value representable by B bits (0 for an unsigned input data type; -2^{B-1} for a signed input data type) is mapped to the value -V. The largest input value representable by B bits (2^B-1 for an unsigned input data type; 2^{B-1}-1 for a signed input data type) is mapped to the value $(1-2^{1-B})$V. Intermediate input values are linearly mapped to the intermediate values in the range [-V, $(1-2^{1-B})$V].

To correctly decode values encoded by the Uniform Encoder block, the **Bits** and **Peak** parameters of the Uniform Decoder block should be set to the same values as the **Bits** and **Peak** parameters of the Uniform Encoder block. The **Overflow mode** parameter specifies the Uniform Decoder block's behavior when the integer input is outside the range representable by B bits. When you select **Saturate**, *unsigned* input values greater than 2^B-1 saturate at 2^B-1; *signed* input values greater than 2^{B-1}-1 or less than -2^{B-1} saturate at those limits. The real and imaginary components of complex inputs saturate independently.

When you select **Wrap**, *unsigned* input values, `u`, greater than 2^B-1 are wrapped back into the range [0, 2^B-1] using mod-2^B arithmetic.

```
u = mod(u,2^B)
```

Signed input values, u, greater than 2^{B-1}-1 or less than -2^{B-1} are wrapped back into that range using mod-2^B arithmetic.

```
u = (mod(u+2^B/2,2^B)-(2^B/2))
```

The real and imaginary components of complex inputs wrap independently.

The **Output type** parameter specifies whether the decoded floating-point output is single or double precision. Either level of output precision can be used with any of the six integer input data types.

Examples

Consider a Uniform Decoder block with the following parameter settings:

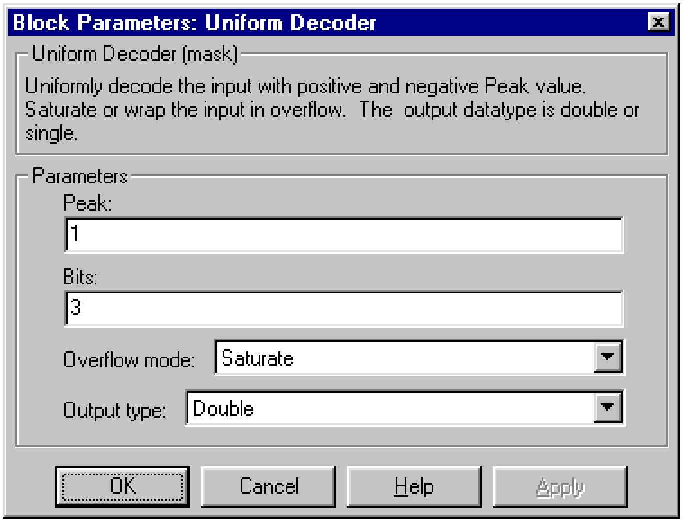
- **Peak** = 2
- **Bits** = 3

The input to the block is the uint8 output of a Uniform Encoder block with comparable settings: **Peak** = 2, **Bits** = 3, and **Output type** = Unsigned. (Comparable settings ensure that inputs to the Uniform Decoder block do not saturate or wrap. See the example on the Uniform Encoder block reference page for more about these settings.)

The real and complex components of each input are independently mapped to one of 2^3 distinct levels in the range [-2.0,1.5].

```
0  is mapped to  -2.0
1  is mapped to  -1.5
2  is mapped to  -1.0
3  is mapped to  -0.5
4  is mapped to   0.0
5  is mapped to   0.5
6  is mapped to   1.0
7  is mapped to   1.5
```

Dialog Box



Peak

Specify the largest amplitude represented in the encoded input. To correctly decode values encoded with the Uniform Encoder block, set the **Peak** parameters in both blocks to the same value.

Bits

Specify the number of input bits, B, used to encode the data. (This can be less than the total number of bits supplied by the input data type.) To correctly decode values encoded with the Uniform Encoder block, set the **Bits** parameters in both blocks to the same value.

Overflow mode

Specify the block's behavior when the integer input is outside the range representable by B bits. Out-of-range inputs can either saturate at the extreme value, or wrap back into range.

Output type

Specify the precision of the floating-point output, `single` or `double`.

References

General Aspects of Digital Transmission Systems: Vocabulary of Digital Transmission and Multiplexing, and Pulse Code Modulation (PCM) Terms, International Telecommunication Union, ITU-T Recommendation G.701, March, 1993

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Data Type Conversion	Simulink
Quantizer	Simulink
Scalar Quantizer Decoder	Signal Processing Blockset
Uniform Encoder	Signal Processing Blockset
`udecode`	Signal Processing Toolbox
`uencode`	Signal Processing Toolbox

Purpose

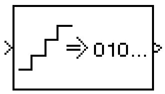
Quantize and encode floating-point input into integer output

Library

Quantizers

`dspquant2`

Description



The Uniform Encoder block performs the following two operations on each floating-point sample in the input vector or matrix:

1. Quantizes the value using the same precision
2. Encodes the quantized floating-point value to an integer value

In the first step, the block quantizes an input value to one of 2^B uniformly spaced levels in the range $[-V, (1\text{-}2^{1-B})V]$, where you specify B in the **Bits** parameter and you specify V in the **Peak** parameter. The quantization process rounds both positive and negative inputs *downward* to the nearest quantization level, with the exception of those that fall exactly on a quantization boundary. The real and imaginary components of complex inputs are quantized independently.

The number of bits, B, can be any integer value between 2 and 32, inclusive. Inputs greater than $(1\text{-}2^{1-B})V$ or less than $-V$ saturate at those respective values. The real and imaginary components of complex inputs saturate independently.

In the second step, the quantized floating-point value is uniquely mapped (encoded) to one of 2^B integer values. When the **Output type** is set to `Unsigned integer`, the smallest quantized floating-point value, $-V$, is mapped to the integer 0, and the largest quantized floating-point value, $(1\text{-}2^{1-B})V$, is mapped to the integer 2^B-1. Intermediate quantized floating-point values are linearly (uniformly) mapped to the intermediate integers in the range $[0, 2^B\text{-}1]$. For efficiency, the block automatically selects an *unsigned* output data type (`uint8`, `uint16`, or `uint32`) with the minimum number of bits equal to or greater than B.

When the **Output type** is set to `Signed integer`, the smallest quantized floating-point value, $-V$, is mapped to the integer -2^{B-1}, and the largest quantized floating-point value, $(1\text{-}2^{1-B})V$, is mapped to the integer 2^{B-1}-1. Intermediate quantized floating-point values are linearly mapped to the intermediate integers in the range [-2^{B-1}, 2^{B-1}-1]. The block automatically selects a *signed* output data type (`int8`, `int16`, or `int32`) with the minimum number of bits equal to or greater than B.

Inputs can be real or complex, double or single precision. The output data types that the block uses are shown in the table below. Note that most of the Signal Processing Blockset blocks accept only double-precision inputs. Use the Simulink Data Type Conversion block to convert integer data types to double precision. See "Working with Data Types" in the Simulink documentation for a complete discussion of data types, as well as a list of Simulink blocks capable of reduced-precision operations.

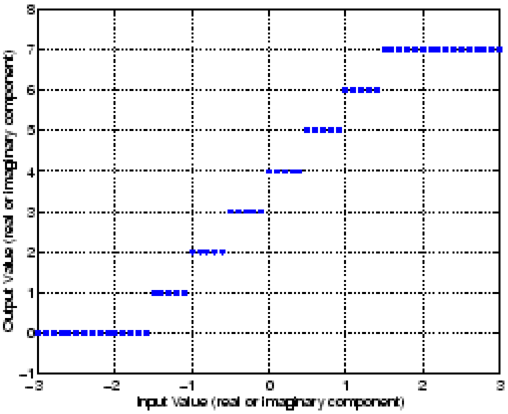
Bits	Unsigned Integer	Signed Integer
2 to 8	uint8	int8
9 to 16	uint16	int16
17 to 32	uint32	int32

The Uniform Encoder block operations adhere to the definition for uniform encoding specified in ITU-T Recommendation G.701.

Examples

The following figure illustrates uniform encoding with the following parameter settings:

- **Peak** = 2
- **Bits** = 3
- **Output type** = `Unsigned`



The real and complex components of each input (horizontal axis) are independently quantized to one of 2^3 distinct levels in the range `[-2,1.5]` and then mapped to one of 2^3 integer values in the range `[0,7]`.

```
-2.0  is mapped to  0
-1.5  is mapped to  1
-1.0  is mapped to  2
-0.5  is mapped to  3
 0.0  is mapped to  4
 0.5  is mapped to  5
 1.0  is mapped to  6
 1.5  is mapped to  7
```

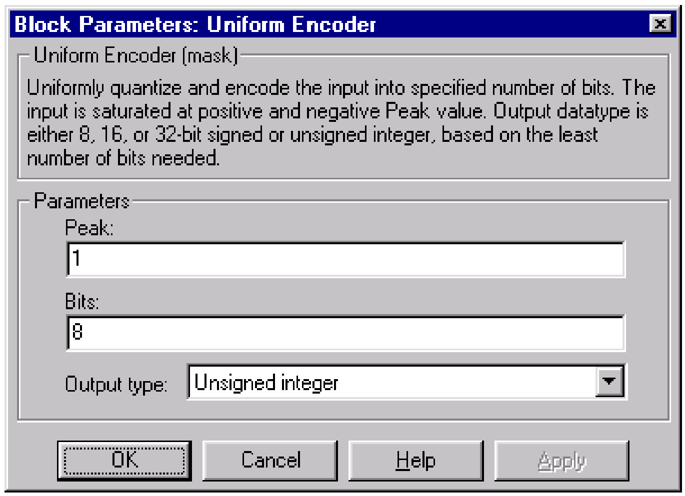
The table below shows the results for a few particular inputs.

Input	Quantized Input	Output	Notes
`1.6`	`1.5+0.0i`	`7+4i`	
`-0.4`	`-0.5+0.0i`	`3+4i`	
`-3.2`	`-2.0+0.0i`	`4i`	Saturation (real)

Input	Quantized Input	Output	Notes
`0.4-1.2i`	`0.0-1.5i`	`4+i`	
`0.4-6.0i`	`0.0-2.0i`	`4`	Saturation (imaginary)
`-4.2+3.5i`	`-2.0+2.0i`	`7i`	Saturation (real and imaginary)

The output data type is automatically set to `uint8`, the most efficient format for this input range.

Dialog Box



Peak

The largest input amplitude to be encoded, V. Real or imaginary input values greater than $(1\text{-}2^{1\text{-}B})V$ or less than $-V$ saturate (independently for complex inputs) at those limits.

Bits

Specify the number of bits, B, needed to represent the integer output. The number of levels at which the block quantizes the floating-point input is 2^B.

Output type

The data type of the block's output, `Unsigned integer` or `Signed integer`. Unsigned outputs are `uint8`, `uint16`, or `uint32`, while signed outputs are `int8`, `int16`, or `int32`.

References

General Aspects of Digital Transmission Systems: Vocabulary of Digital Transmission and Multiplexing, and Pulse Code Modulation (PCM) Terms, International Telecommunication Union, ITU-T Recommendation G.701, March, 1993

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

Data Type Conversion	Simulink
Quantizer	Simulink
Scalar Quantizer Decoder	Signal Processing Blockset
Uniform Decoder	Signal Processing Blockset
`udecode`	Signal Processing Toolbox
`uencode`	Signal Processing Toolbox

Unwrap

Purpose

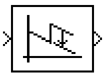
Unwrap signal phase

Library

Signal Operations

`dspsigops`

Description



The Unwrap block unwraps each input channel by adding or subtracting appropriate multiples of 2π to each channel element. The input can be any matrix or 1-D vector, and must have radian phase entries. The block recognizes phase discontinuities larger than the **Tolerance** parameter setting.

The block preserves the input size, dimension, and frame status, and the output port rate equals the input port rate. For a detailed discussion of the Unwrap block, see other sections of this reference page.

Sections of This Reference Page

- "Acceptable Inputs and Corresponding Output Characteristics"
- "The Two Unwrap Modes"
- "Unwrap Method"
- "Definition of Phase Unwrap"

Acceptable Inputs and Corresponding Output Characteristics

The Unwrap block preserves the input size, dimension, and frame status, and the output port rate equals the input port rate.

Characteristics of Valid Input	Characteristics of Corresponding Output
Input elements must be phase values in radians. Sample- or frame-based M-by-N 2-D matrix or a 1-D vector	Output elements are phase values in radians. Same frame status as input Same size and dimension as input Output port rate = input port rate

The Two Unwrap Modes

You must specify the unwrap mode by setting the parameter, **Do not unwrap phase discontinuities between successive frames**. The unwrap modes are summarized in the next table.

Two Unwrap Modes	
In both unwrap modes, the block adds 2 *k* to each input channel's elements, where it updates *k* at each phase discontinuity. (For more on the updating of *k*, see "Unwrap Method" on page 2-1396.) The number of times that *k* is reset to 0 depends on the unwrap mode.	
Default Unwrap Mode: Initialize k to 0 for Only the First Input Frame	**Nondefault Unwrap Mode: Set k to 0 for Each Successive Input Matrix or Input Vector**
☐ Do not unwrap phase discontinuities between successive frames In this mode, k is initialized to 0 for only the first input matrix or input vector. As k gets updated, the value of k is retained between successive input matrices or input vectors. That is, the block unwraps each input's channel by considering phase discontinuities in all previous frames and the current frame.	☑ Do not unwrap phase discontinuities between successive frames In this mode, k is reset to 0 for each successive input matrix or input vector. As k gets updated, the value of k is only retained within the current input matrix or vector. That is, the block unwraps each input's channel by considering phase discontinuities in the current input matrix or input vector only, ignoring discontinuities in previous inputs.
In this mode, the block unwraps the columns or each individual element of the input: • Frame-based inputs — unwrap columns • Sample-based inputs — unwrap each element of the input. • 1-D vector inputs — treat as frame-based column	In this mode, the block unwraps the columns or rows of the input: • Frame-based inputs — unwrap columns • Sample-based nonrow inputs — unwrap columns • Sample-based row vector inputs — unwrap the row.

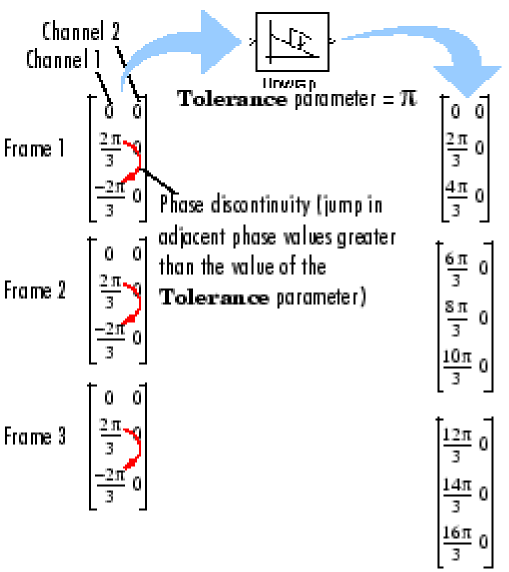
Two Unwrap Modes	
In both unwrap modes, the block adds 2 *k* to each input channel's elements, where it updates *k* at each phase discontinuity. (For more on the updating of *k*, see "Unwrap Method" on page 2-1396.) The number of times that *k* is reset to 0 depends on the unwrap mode.	
Default Unwrap Mode: Initialize k to 0 for Only the First Input Frame	**Nondefault Unwrap Mode: Set k to 0 for Each Successive Input Matrix or Input Vector**
	• 1-D vector inputs — treat as frame-based column
See the following diagrams.	See the following diagrams.

The following diagrams illustrate how the two unwrap modes operate on various inputs.

Default Unwrap Mode Operation: ☐ Do not unwrap phase discontinuities between successive frames

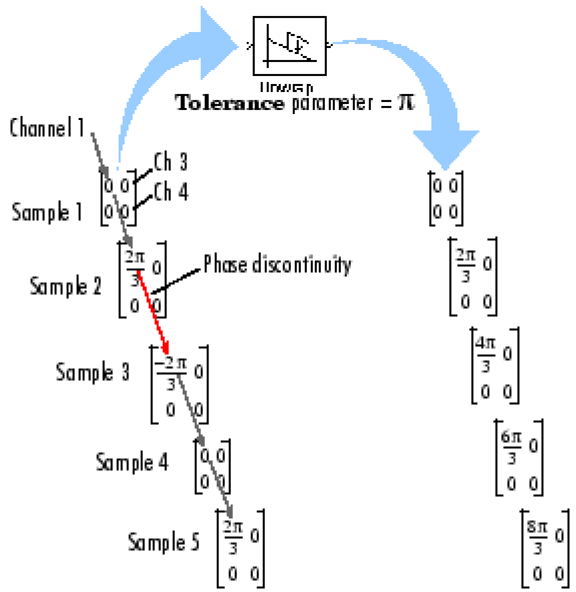
Frame-Based Inputs

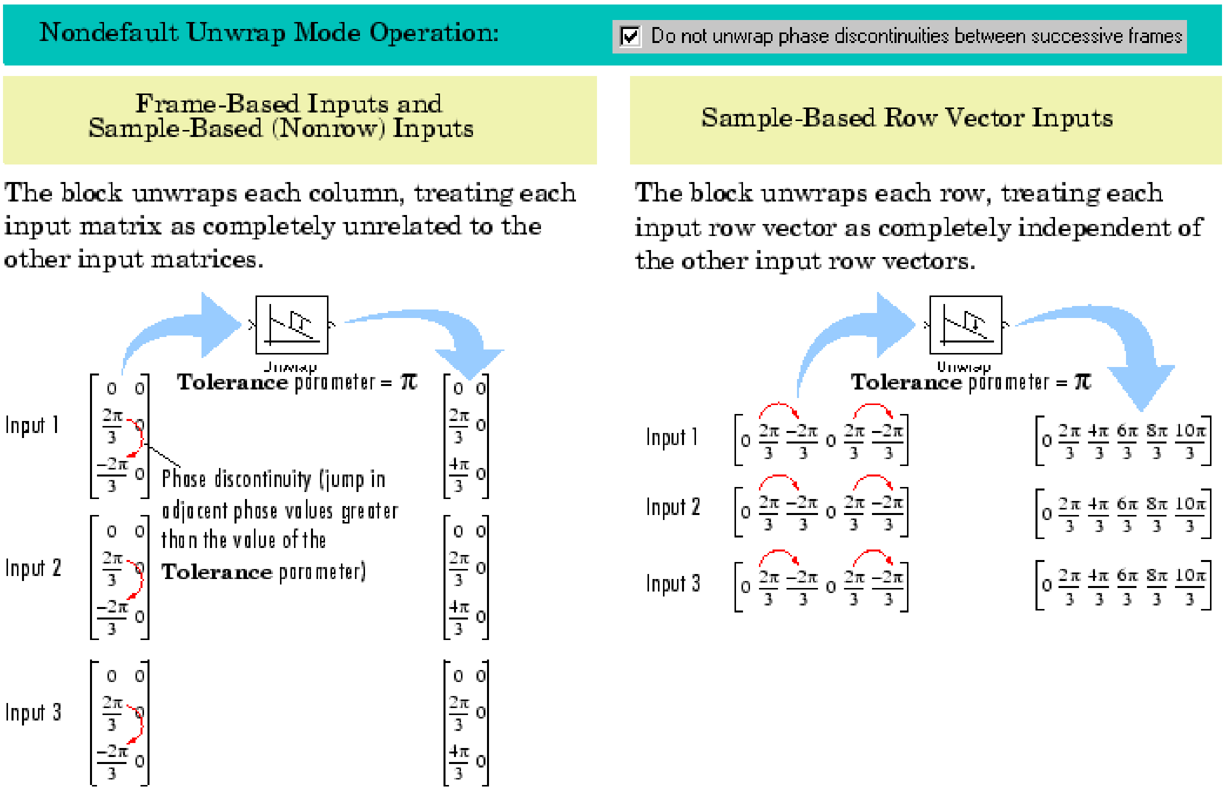
The block treats each input column as an independent channel. It unwraps by treating Channel 1 of Frame 2 as a continuation of Channel 1 of Frame 1.



Sample-Based Inputs

The block treats each element of the input matrix as an independent channel. (The first sample in Channel 1 is in the upper left corner of the Sample 1 matrix. The second sample of Channel 1 is in the corresponding corner of the Sample 2 matrix, and so on.)





Unwrap Method

The Unwrap block unwraps each channel of its input matrix or input vector by adding $2\pi k$ to each successive channel element, and updating k at each *phase jump*. See the following steps to the unwrap method for details.

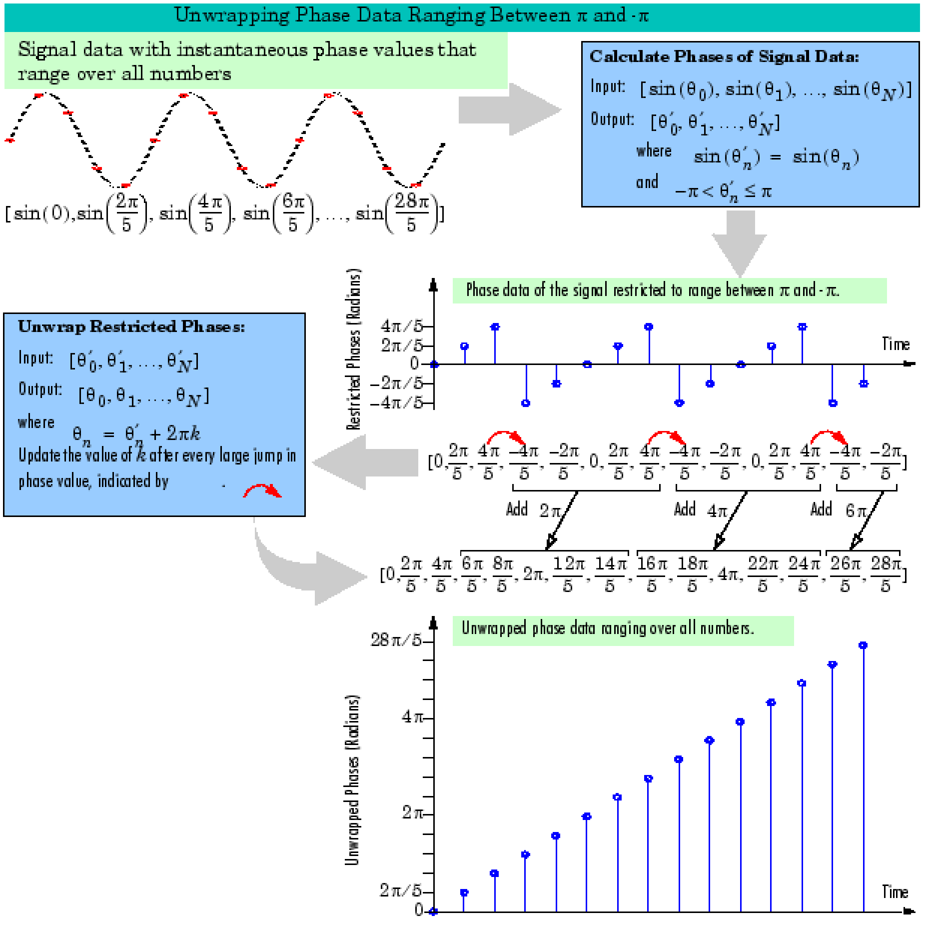
Relevant Unwrap Terms:
• u_i — ith element of the input channel on which the algorithm operates • a —**Tolerance** parameter value • Phase jump or phase discontinuity — difference between phase values of two adjacent channel entries that exceeds a. The diagram in the next section indicates phase jumps with red arrows.
Steps to the Unwrap Method:
1 Set k to 0 (See "The Two Unwrap Modes" on page 2-1393 for more on how often this step occurs.) **2** Check for a phase jump between adjacent channel elements u_i and u_{i+1}: • When there is no phase jump between u_i and u_{i+1} $\left(\lvert u_{i+1}-u_i\rvert \le \lvert\alpha\rvert\right)$, add $2\pi k$ to u_i, and then repeat step 2 to continue checking for phase jumps. • When there is a phase jump between u_i and u_{i+1} $\left(\lvert u_{i+1}-u_i\rvert > \lvert\alpha\rvert\right)$, add $2\pi k$ to u_i, and then go to step 3 to update k. **3** Update k as follows when there is a phase jump between u_i and u_{i+1}. Then go back to step 2 to add the updated $2\pi k$ value to u_{i+1} and succeeding channel elements until the next phase jump: • When $u_{i+1} < u_i$ (phase jump is negative), increment k. • When $u_{i+1} > u_i$ (phase jump is positive), decrement k.

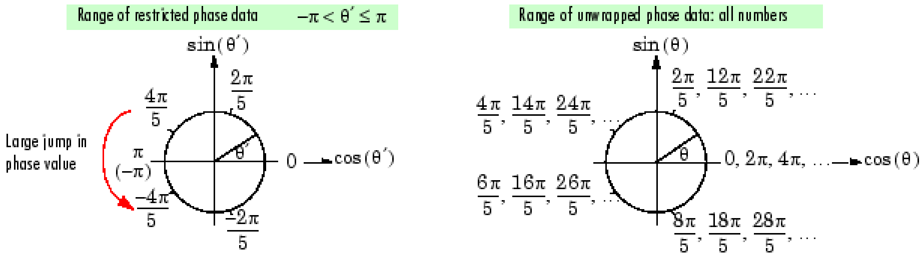
Definition of Phase Unwrap

Algorithms that compute the phase of a signal often only output phases between –π and π. For instance, such algorithms compute the phase of sin(2π + 3) to be 3, since sin(3) = sin(2π + 3), and since the actual phase,

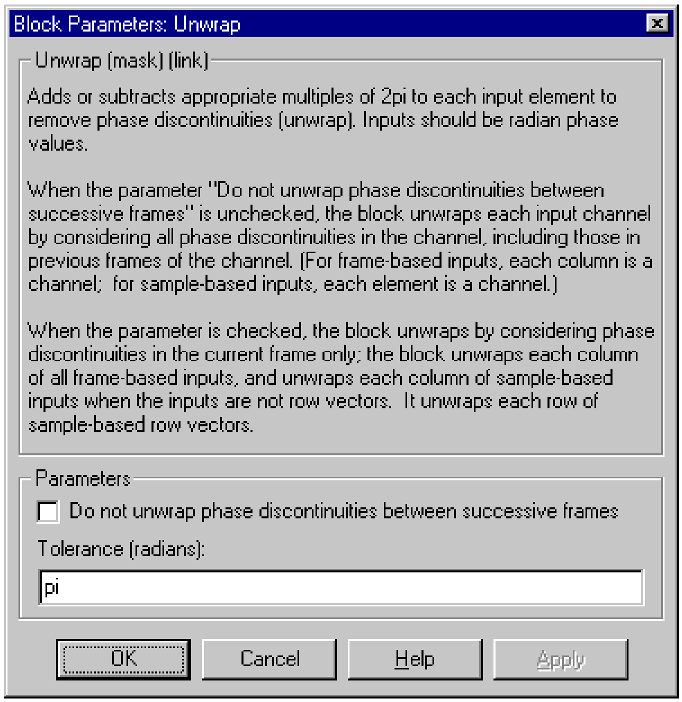
$2\pi + 3$, is not between $-\pi$ and π. Such algorithms compute the phases of $\sin(-4\pi + 3)$ and $\sin(16\pi + 3)$ to be 3 as well.

Phase unwrap or unwrap is a process often used to reconstruct a signal's original phase. Unwrap algorithms add appropriate multiples of 2π to each phase input to restore original phase values, as illustrated in the following diagram. For more on phase unwrap, see the previous section, "Unwrap Method" on page 2-1396.





Dialog Box



Do not unwrap phase discontinuities between successive frames

When this parameter is cleared, the block unwraps each input's channels (the input channels are the columns of frame-based inputs and each element of sample-based inputs). When you select this parameter, the block unwraps each row of sample-based row vector inputs, and unwraps the columns of all other inputs, where each input matrix or input vector is treated as completely unrelated to the other input matrices or input vectors. 1-D vector inputs are always treated as frame-based column vectors. See "The Two Unwrap Modes" on page 2-1393.

Tolerance

The jump size that the block recognizes as a true phase discontinuity. The default is set to π (rather than a smaller value) to avoid altering legitimate signal features. To increase the block's sensitivity, set **Tolerance** to a value slightly less than π.

Supported Data Types

- Double-precision floating point
- Single-precision floating point

See Also

`unwrap`	MATLAB

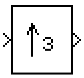
Upsample

Purpose Resample input at higher rate by inserting zeros

Library Signal Operations

`dspsigops`

Description



The Upsample block resamples each channel of the M_i-by-N input at a rate L times higher than the input sample rate by inserting L-1 zeros between consecutive samples. You specify the integer L in the **Upsample factor** parameter. The **Sample offset** parameter delays the output samples by an integer number of sample periods D, where

$0 \leq D < (L-1)$, so that any of the L possible output phases can be selected.

This block supports triggered subsystems if, for **Frame-based mode**, you select `Maintain input frame rate`.

Sample-Based Operation

When the input is sample based, the block treats each of the M*N matrix elements as an independent channel, and upsamples each channel over time. The **Frame-based mode** parameter must be set to `Maintain input frame size`. The output sample rate is L times higher than the input sample rate ($T_{so} = T_{si}/L$), and the input and output sizes are identical.

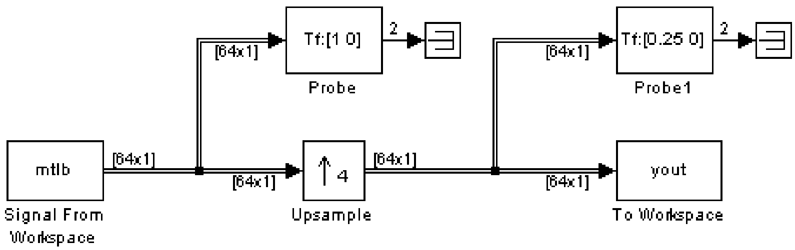
Frame-Based Operation

When the input is frame based, the block treats each of the N input columns as a frame containing M_i sequential time samples from an independent channel. The block upsamples each channel independently by inserting L-1 rows of zeros between each row in the input matrix. The **Frame-based mode** parameter determines how the block adjusts the rate at the output to accommodate the added rows. There are two available options:

- `Maintain input frame size`

The block generates the output at the faster (upsampled) rate by using a proportionally shorter frame *period* at the output port than at the input port. For upsampling by a factor of L, the output frame period is L times shorter than the input frame period ($T_{fo} = T_{fi}/L$), but the input and output frame sizes are equal.

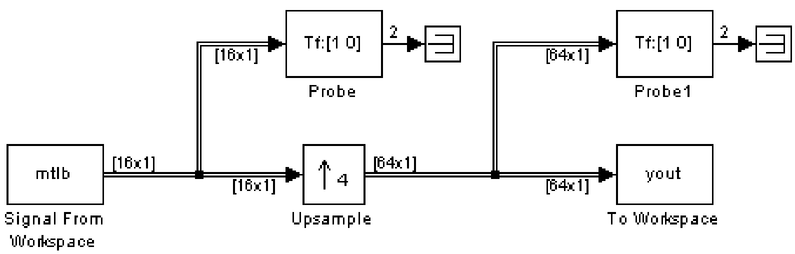
The model below shows a single-channel input with a frame period of 1 second being upsampled by a factor of 4 to a frame period of 0.25 second. The input and output frame sizes are identical.



- `Maintain input frame rate`

  The block generates the output at the faster (upsampled) rate by using a proportionally larger frame *size* than the input. For upsampling by a factor of L, the output frame size is L times larger than the input frame size ($M_o = M_i*L$), but the input and output frame rates are equal.

  The model below shows a single-channel input of frame size 16 being upsampled by a factor of 4 to a frame size of 64. The input and output frame rates are identical.



Zero Latency

The Upsample block has *zero-tasking latency* for all single-rate operations. The block is single-rate for the particular combinations of sampling mode and parameter settings shown in the table below.

Sampling Mode	Parameter Settings
Sample based	**Upsample factor** parameter, L, is `1`.
Frame based	**Upsample factor** parameter, L, is `1`, *or* **Frame-based mode** parameter is `Maintain input frame rate`.

The block also has zero latency for all multirate operations in the Simulink single-tasking mode.

Zero-tasking latency means that the block propagates the first input (received at t=0) immediately following the D consecutive zeros specified by the **Sample offset** parameter. This output (D+1) is followed in turn by the L-1 inserted zeros and the next input sample. The **Initial condition** parameter value is not used.

Nonzero Latency

The Upsample block has tasking latency only for multirate operation in the Simulink multitasking mode:

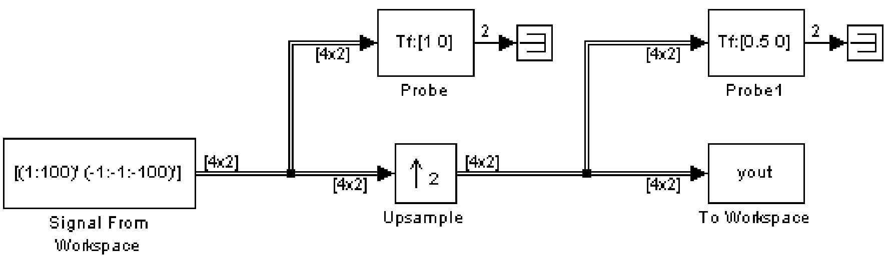
- In sample-based mode, the initial condition for each channel appears as output sample D+1, and is followed by L-1 inserted zeros. The channel's first input appears as output sample D+L+1. The **Initial condition** value can be an M_i-by-N matrix containing one value for each channel, or a scalar to be applied to all signal channels.
- In frame-based mode, the first row of the initial condition matrix appears as output sample D+1, and is followed by L-1 inserted rows of zeros, the second row of the initial condition matrix, and so on. The first row of the first input matrix appears in the output as sample M_iL+D+1. The **Initial condition** value can be an M_i-by-N matrix, or

a scalar to be repeated across all elements of the M_i-by-N matrix. See the example below for an illustration of this case.

Note For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and "Scheduling Considerations" in the *Real-Time Workshop User's Guide*.

Examples

Construct the frame-based model shown below.



Adjust the block parameters as follows:

- Configure the Signal From Workspace block to generate a two-channel signal with frame size of 4 and sample period of 0.25. This represents an output frame period of 1 (0.25*4). The first channel should contain the positive ramp signal 1, 2, ..., 100, and the second channel should contain the negative ramp signal -1, -2, ..., -100.
  - **Signal** = `[(1:100)' (-1:-1:-100)']`
  - **Sample time** = `0.25`
  - **Samples per frame** = `4`
- Configure the Upsample block to upsample the two-channel input by increasing the output frame rate by a factor of 2 relative to the input frame rate. Set a sample offset of 1, and an initial condition matrix of

$$\begin{bmatrix} 11 & -11 \\ 12 & -12 \\ 13 & -13 \\ 14 & -14 \end{bmatrix}$$

- **Upsample factor** = `2`
- **Sample offset** = `1`
- **Initial condition** = `[11 -11;12 -12;13 -13;14 -14]`
- **Frame-based mode** = `Maintain input frame size`

• Configure the Probe blocks by clearing the **Probe width** and **Probe complex signal** check boxes (if desired).

This model is multirate because there are at least two distinct frame rates, as shown by the two Probe blocks. To run this model in the Simulink multitasking mode, open the Configuration Parameters dialog box. In the **Select** pane, click **Solver**. From the **Type** list, select `Fixed-step`, and from the **Solver** list, select `Discrete (no continuous states)`. From the **Tasking mode for periodic sample times** list, select `MultiTasking`. Also set the **Stop time** to `30`.

Run the model and look at the output, `yout`. The first few samples of each channel are shown below.

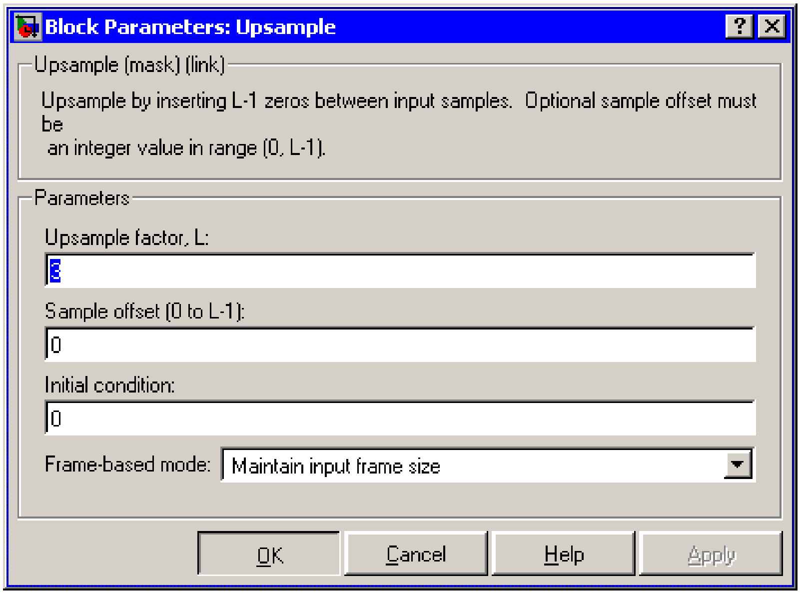
```
yout =

     0     0
    11   -11
     0     0
    12   -12
     0     0
    13   -13
     0     0
    14   -14
     0     0
     1    -1
```

0	0
2	-2
0	0
3	-3
0	0
4	-4
0	0
5	-5
0	0

Since we ran this frame-based multirate model in multitasking mode, the first row of the initial condition matrix appears as output sample 2 (that is, sample D+1, where D is the **Sample offset** value). It is followed by the other three initial condition rows, each separated by L-1 inserted rows of zeros, where L is the **Upsample factor** value of 2. The first row of the first input matrix appears in the output as sample 10 (that is, sample M_iL+D+1, where M_i is the input frame size).

Dialog Box



Upsample factor
The integer factor, L, by which to increase the input sample rate.

Sample offset
The sample offset, D, which must be an integer in the range [0,L-1].

Initial conditions
The value with which the block is initialized for cases of nonzero latency, a scalar or matrix. This value (first row in frame-based mode) appears in the output as sample D+1.

Frame-based mode
For frame-based operation, the method by which to implement the upsampling: `Maintain input frame size` (that is, increase the frame rate), or `Maintain input frame rate` (that is, increase the frame size). The **Framing** parameter must be set to `Maintain input frame size` for sample-base inputs.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Downsample	Signal Processing Blockset
FIR Interpolation	Signal Processing Blockset
FIR Rate Conversion	Signal Processing Blockset
Repeat	Signal Processing Blockset

Purpose

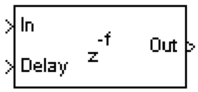
Delay input by time-varying fractional number of sample periods

Library

Signal Operations

`dspsigops`

Description



The Variable Fractional Delay block delays each element of the discrete-time N-D input array, u, by a variable number of sample intervals. The input delay values can be integer or noninteger values. The block provides three different interpolation modes: `Linear`, `FIR`, and `Farrow`.

The block computes the value for each channel of the output based on the stored samples in memory most closely indexed by the `Delay` input, v, and the interpolation method specified by the **Interpolation mode** parameter.

- In `Linear` interpolation mode, the block stores the D_{max}+1 most recent samples received at the In port for each channel, where D_{max} is the value specified for the **Maximum delay (Dmax) in samples** parameter.
- In `FIR` interpolation mode, the block stores the $D_{max}+P+1$ most recent samples received at the In port for each channel, where P is the value specified for the **Interpolation filter half-length (P)** parameter.
- In `Farrow` interpolation mode, the block stores the $D_{max}+\frac{N}{2}+1$ most recent samples received at the In port for each channel, where N is the value specified for the **Farrow filter length (N)** parameter.

The Variable Fractional Delay block assumes that the input values at the `Delay` port are between D_{min} and D_{max}, where D_{min} appears in the `Valid delay range` section on the **Main** pane of the block mask, and D_{max} is the value of the **Maximum delay (Dmax) in samples** parameter. The block clips delay values less than D_{min} to D_{min} and delay values greater than D_{max} to D_{max}.

If you are working with frame-based signals and select the **Disable direct feedthrough by increasing the minimum possible delay by one** check box, the smallest possible delay value is increased by *frame-size* – 1. Thus, all input delay values less than Dmin + *frame-size* – 1 are clipped to Dmin + *frame-size* – 1.

You must consider additional factors when selecting valid Delay values for the FIR and Farrow interpolation modes. For more information about these considerations, refer to FIR Interpolation Mode on page 1412 or Farrow Interpolation Mode on page 1414, respectively.

The Variable Fractional Delay block is similar to the Variable Integer Delay block, in that they both store a minimum of D_{max}+1 past samples in memory. The Variable Fractional Delay block differs only in the way that these stored samples are accessed; a fractional delay requires the computation of a value by interpolation from the nearby samples in memory.

Sample-Based Operation

For sample-based inputs, the block treats each element of the N-D input array, u, as an independent channel. The input to the Delay port, v, must either be an N-D array of the same size and dimension as the input u, or be a scalar value, such that $D_{min} \leq v \leq D_{max}$.

For example, consider an M-by-N input matrix. The block treats each of the $M*N$ matrix elements as independent channels. The input to the Delay port can be an M-by-N matrix of floating-point values in the range $D_{min} \leq v \leq D_{max}$ that specifies the number of sample intervals to delay each channel of the input, or it can be a scalar floating-point value, $D_{min} \leq v \leq D_{max}$, by which to equally delay all channels.

The block treats a 1-D vector input as an M-by-1 matrix, and outputs a 1-D vector.

The **Initial conditions** parameter specifies the values in the block's memory at the start of the simulation in the same manner as the Variable Integer Delay block. See the Variable Integer Delay block reference page for more information.

Frame-Based Operation

For frame-based inputs, the block treats each of the N input columns as a frame containing M_i sequential time samples from an independent channel.

The input to the Delay port, v, contains floating-point values that specify the number of sample intervals to delay the current input. When you clear the **Disable direct feedthrough by increasing minimum possible delay by one** check box, the valid range of delay values is $D_{min} \leq v \leq D_{max}$. When you select the **Disable direct feedthrough by increasing minimum possible delay by one** check box, the valid range of delay values is $D_{min} + \texttt{frame-size} - 1 \leq v \leq D_{max}$.

The input to the Delay port can be a scalar value to uniformly delay every sample in every channel. It can also be a column-based length-M vector, containing one delay for each sample in the input frame. The block applies the set of delays contained in the vector identically to every channel of a multichannel input. The Delay port entry can also be a row-based length-N vector, containing one delay for each channel. Finally, the Delay port entry can be an M-by-N matrix, containing a different delay for each corresponding element of the input.

For example, if v is the M_i-by-1 matrix `[v(1) v(2) ...  v(Mi)]'`, the earliest sample in the current frame is delayed by `v(1)` fractional sample intervals, the following sample in the frame is delayed by `v(2)` fractional sample intervals, and so on. The block applies the set of fractional delays contained in v identically to every channel of a multichannel input.

The **Initial conditions** parameter specifies the values in the block's memory at the start of the simulation in the same manner as the Variable Integer Delay block. See the Variable Integer Delay block reference page for more information.

Interpolation Modes

The delay value specified at the Delay port serves as an index into the block's memory, `U`, which stores, at a minimum, the D_{max}+1 most recent samples received at the In port for each channel. For example, an integer delay of `5` on a scalar input sequence retrieves and outputs the fifth most recent input sample from the block's memory, `U(6)`. The block computes fractional delays by interpolating between stored samples; the three available interpolation modes are `Linear`, `FIR` and `Farrow`.

Linear Interpolation Mode

For noninteger delays, at each sample time, the `Linear Interpolation` mode uses the two samples in memory nearest to the specified delay to compute a value for the sample at that time. If v is the specified fractional delay for a scalar input, the output sample, y, is computed as follows.

```
vi = floor(v)     % vi = integer delay
vf = v-vi     % vf = fractional delay
y = (1-vf)*U(vi+1) + vf*U(vi)
```

FIR Interpolation Mode

In `FIR Interpolation` mode, the block provides a discrete set of fractional delays described by:

$$v + \frac{i}{L}, \qquad v \geq P-1, \;\; i = 0,1,\ldots,L-1$$

If v is less than P-1, the block's behavior depends on the setting of the **For small input delay values** parameter. You can specify the block's behavior when the input delay value is too small to center the kernel (less than P-1), by setting the **For small input delay values** parameter:

- If you select `Clip to the minimum value necessary for centered kernel`, the block remains in `FIR` interpolation mode by clipping small input delay values to the smallest value necessary to center the kernel.

  To determine the minimum delay value, select `Clip to the minimum value necessary for centered kernel`, and click **Apply** on the block mask. All input delay values less than the value displayed for D_{min} will be clipped to D_{min}.

- If you select `Switch to linear interpolation if kernel cannot be centered`, the block computes fractional delays using linear interpolation when the input delay value is less than P-1.

To add an extra delay to the minimum possible delay value, select the **Disable direct feedthrough by increasing minimum possible delay by one** check box. Checking this box prevents algebraic loops from occurring when you use the block inside a feedback loop.

Note If the input to the block is frame based and you select the **Disable direct feedthrough by increasing minimum possible delay by one** check box, the minimum possible delay (`Dmin`) increases by *`frame-size`* – 1.

In `FIR Interpolation` mode, the block implements a polyphase structure to compute a value for each sample at the desired delay. Each arm of the structure corresponds to a different delay value and the output computed for each sample corresponds to the output of the arm with a delay value nearest to the desired input delay. Thus, only a discrete set of delays is actually possible. The number of coefficients in each of the L filter arms of the polyphase structure is $2P$. In most cases, using values of P between 4 and 6 will provide you with reasonably accurate interpolation values.

In this mode, the Signal Processing Toolbox `intfilt` function computes an FIR filter for interpolation.

For example, when you set the parameters on the block mask to the following values:

- Interpolation filter half-length (P): 4
- Interpolation points per input sample: 10
- Normalized input bandwidth: 1

The filter coefficients are given by:

```
b = intfilt(10,4,1);
```

The block then implements this filter as a polyphase structure, as described previously.

Increasing the **Interpolation filter half length (P)** increases the accuracy of the interpolation, but also increases the number of computations performed per input sample, as well as the amount of memory needed to store the filter coefficients. Increasing the **Interpolation points per input sample** (*L*) increases the number of representable discrete delay points, but also increases the simulation's memory requirements and does not affect the computational load per sample.

The **Normalized input bandwidth (0 to 1)** parameter allows you to take advantage of the bandlimited frequency content of the input. For example, if you know that the input signal does not have frequency content above $F_s/4$, you can specify a value of `0.5` for the **Normalized input bandwidth (0 to 1)** to constrain the frequency content of the output to that range.

Note You can consider each of the *L* interpolation filters to correspond to one output phase of an "upsample-by-*L*" FIR filter. Thus, the **Normalized input bandwidth (0 to 1)** value improves the stopband in critical regions, and relaxes the stopband requirements in frequency regions where there is no signal energy.

Farrow Interpolation Mode

In `Farrow` interpolation mode, the block uses the LaGrange method to interpolate values.

To add an extra delay of 1 to the minimum possible delay value, select the **Disable direct feedthrough by increasing minimum possible delay by one** check box. Checking this box prevents algebraic loops from occurring when you use the block inside a feedback loop.

Note If the input to the block is frame-based and you select the **Disable direct feedthrough by increasing minimum possible delay by one** check box, the minimum possible delay (`Dmin`) increases by *`frame-size`* – 1.

To specify the block's behavior when the input delay value is too small to center the kernel (less than $\frac{N}{2}$ -1), set the **For small input delay values** parameter:

- If you select `Clip to the minimum value necessary for centered kernel`, the block clips small input delay values to the smallest value necessary to keep the kernel centered. This increases D_{min} but yields more accurate interpolation values.

  To determine the minimum delay value, select `Clip to the minimum value necessary for centered kernel`, and click **Apply** on the block mask. All input delay values less than the value displayed for D_{min} will be clipped to D_{min}.

- If you select `Use off-centered kernel`, the block computes fractional delays using a Farrow filter with an off-centered kernel. This mode does not increase D_{min}, but if there are input delay values less than $\frac{N}{2}$ -1, the results are less accurate than the results achieved by keeping the kernel centered.

Fixed-Point Data Types

The diagrams in the following sections show the data types used within the Variable Fractional Delay block for fixed-point signals.

Although you can specify most of these data types on the **Data Types** pane of the block mask, the following data types are computed internally by the block and cannot be directly specified on the block mask.

Data Type	Word Length	Fraction Length
vf data type	Same word length as the **Coefficients**	Same as the word length
HoldInteger data type	Same word length as the input delay value	0 bits
Integer data type	32 bits	0 bits

Note When the block input is fixed point, all internal data types are signed fixed point.

To compute the integer (v_i) and fractional (v_f) parts of the input delay value (v), the Variable Fractional Delay block uses the following equations:

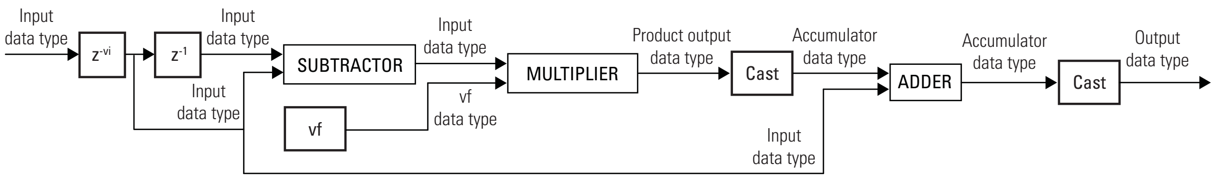
$$D_{min} < v < D_{max} \Rightarrow \begin{cases} v_i = \text{floor}(v) \\ v_f = v - vi \end{cases}$$

$$v \le D_{min} \Rightarrow \begin{cases} v_i = D_{min} \\ v_f = 0 \end{cases}$$

$$v \ge D_{max} \Rightarrow \begin{cases} v_i = D_{max} \\ v_f = 0 \end{cases}$$

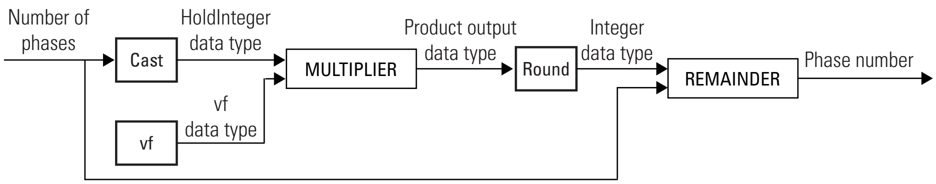
Linear Interpolation Mode

The following diagram shows the fixed-point data types used by the `Linear` interpolation mode of the Variable Fractional Delay block.



FIR Interpolation Mode

The following diagram illustrates how the Variable Fractional Delay block selects the arm of the polyphase filter structure that most closely matches the fractional delay value (v_f).

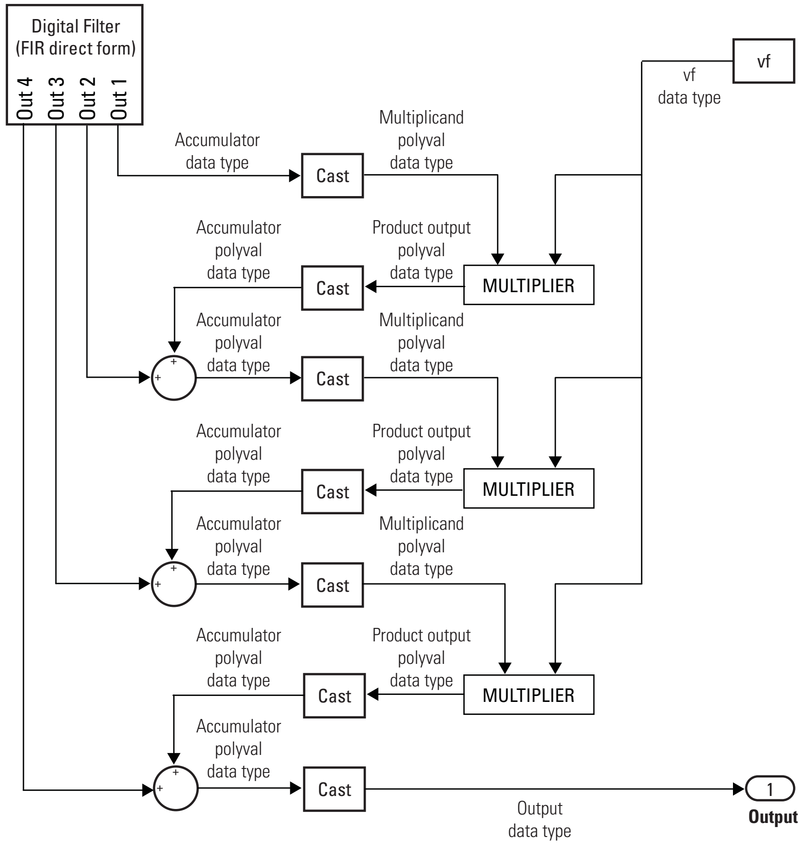


See the “Fixed-Point Data Types” on page 2-545 section of the FIR Interpolation reference page for a diagram showing the fixed-point data types used by the Variable Fractional Delay block in `FIR` interpolation mode.

Farrow Interpolation Mode

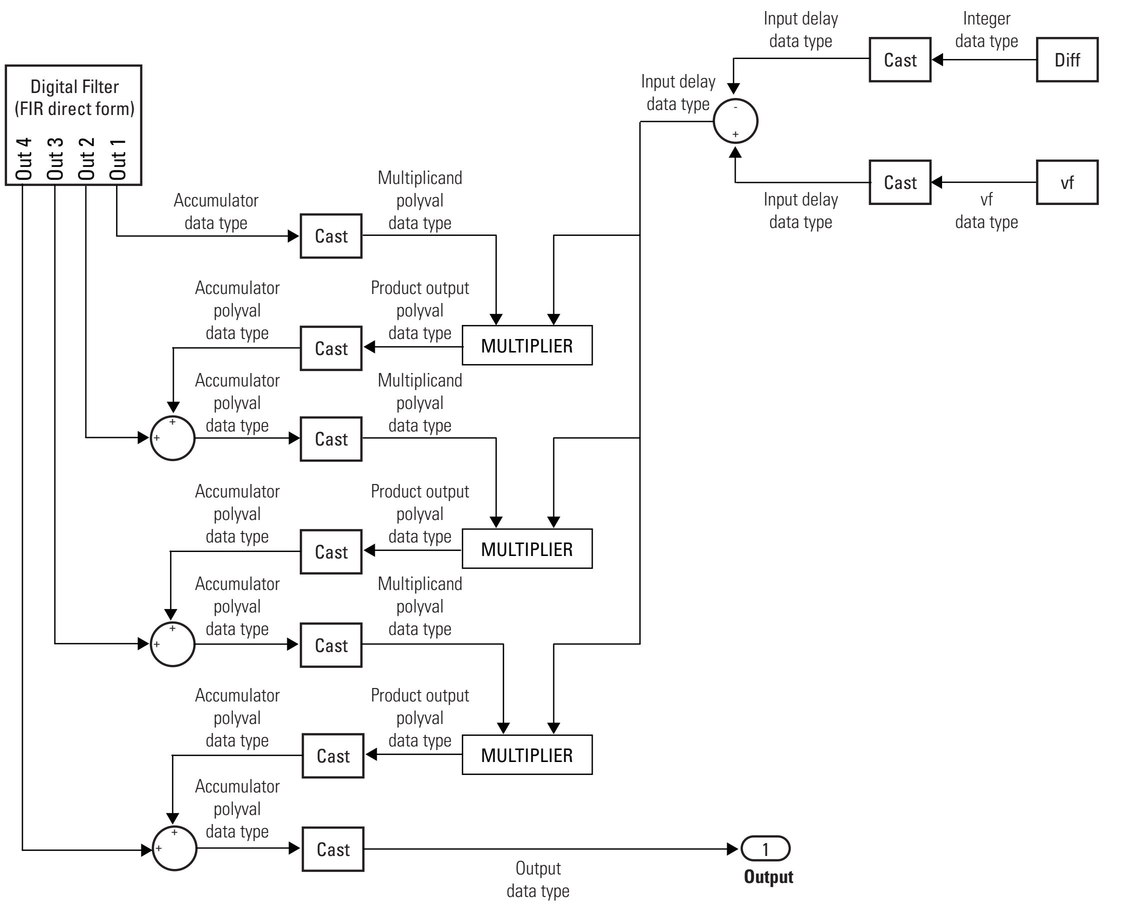
The following diagram shows the fixed-point data types used by the `Farrow` interpolation mode of the Variable Fractional Delay block where:

- **Farrow filter length (N)** = `4`
- **For small input delay values** = `Clip to the minimum value necessary to for centered kernel`



The following diagram shows the fixed-point data types used by the `Farrow` interpolation mode of the Variable Fractional Delay block where:

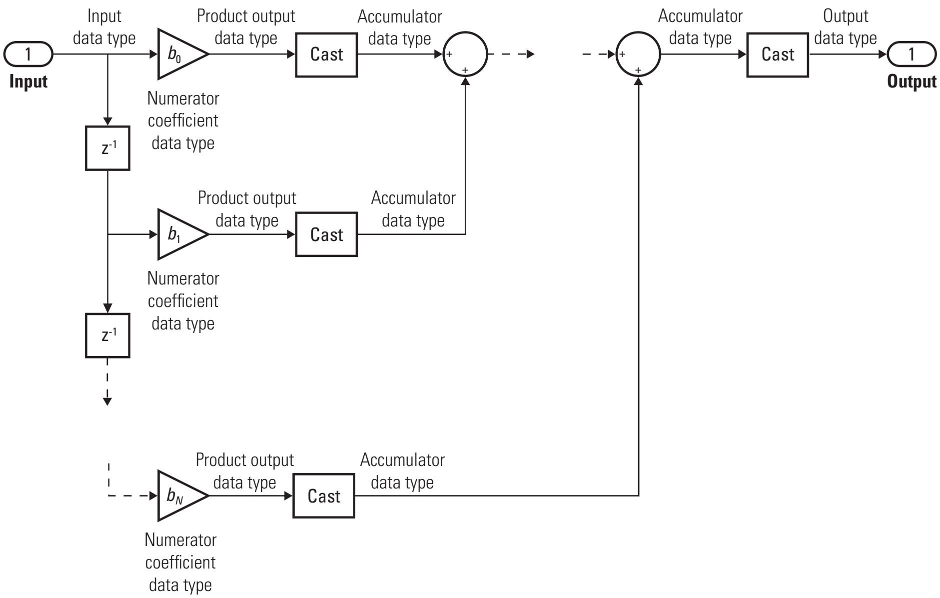
- **Farrow filter length (N)** = `4`
- **For small input delay values** = `Use off-centered kernel`



`Diff` is computed from the integer part of the delay value (v_i) and the **Farrow filter length (N)** according to the following equation:

$$Diff = v_i - \left(\frac{N-1}{2}\right)$$
$$Diff \geq 0 \Rightarrow Diff = 0$$
$$Diff < 0 \Rightarrow Diff = -Diff$$

The following diagram shows the fixed-point data types used by the Digital Filter block's FIR direct form filter.



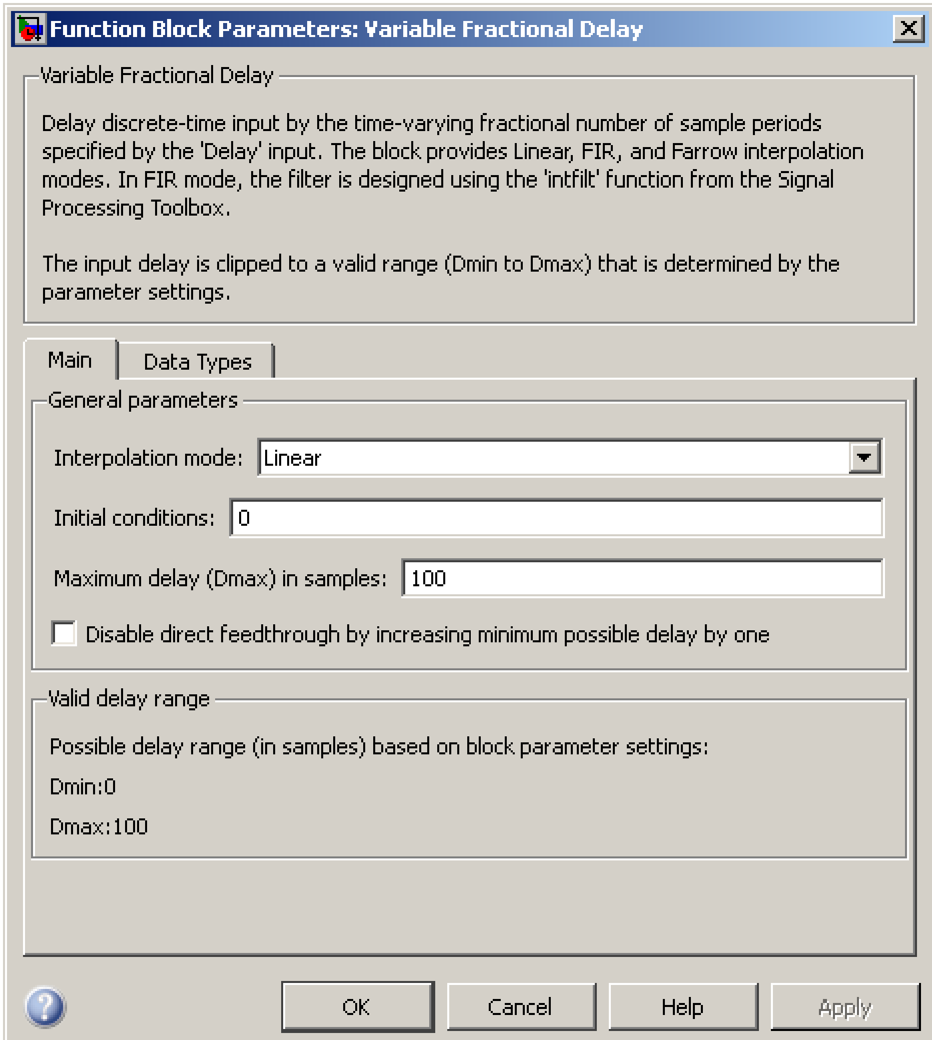
Examples

The `dspaudioeffects` demo illustrates three audio effects applied to a short segment of music. When you set the **Audio effect** of the Effect block to `Flanging`, the model uses the Variable Fractional Delay block to mix the original signal with a delayed version of itself.

To see the Flanging subsystem, right-click the Effect block, and select **Look Under Mask**. Next, double-click the Flanging block in the Effect block subsystem that just opened. The Flanging subsystem opens, and you can see the parameters of the Variable Fractional Delay block.

Dialog Box

The **Main** pane of the Variable Fractional Delay block dialog appears as follows.



Interpolation mode
The method by which to interpolate between adjacent stored samples to obtain a value for the sample indexed by the input at the Delay port.

Interpolation filter half-length (P)
Half the number of input samples to use in the FIR interpolation filter. This parameter is only visible when the **Interpolation mode** is set to `FIR`.

Farrow filter length (N)
The number of input samples to use in the Farrow interpolation filter. This parameter is only visible when the **Interpolation mode** is set to `Farrow`.

Interpolation points per input sample
The number of points per input sample, L, at which a unique FIR interpolation filter is computed. This parameter is only visible when the **Interpolation mode** is set to `FIR`.

Normalized input bandwidth (0 to 1)
The bandwidth to which the interpolated output samples should be constrained. The value must be a real scalar between `0` and `1`. A value of `1` specifies half the sample frequency. This parameter is only visible when the **Interpolation mode** is set to `FIR`.

Initial conditions
The values with which the block's memory is initialized. See the Variable Integer Delay block for more information.

Maximum delay (Dmax) in samples
The maximum delay that the block can produce, D_{max}. Input delay values exceeding this maximum are clipped to D_{max}.

Disable direct feedthrough by increasing minimum possible delay by one
Select this box to disable direct feedthrough by adding one to the minimum possible delay value. Checking this box allows the Variable Fractional Delay block to be used in feedback loops.

For small input delay values

Specify the block's behavior when the input delay values are too small to center the kernel. This parameter is only visible when the **Interpolation mode** is set to `FIR` or `Farrow`.

You can specify how the block handles input delay values that are too small for the kernel to be centered using one of the following choices:

- In both `FIR` and `Farrow` interpolation modes, you can select `Clip to the minimum value necessary for centered kernel`. This option forces the block to increase D_{min} to the smallest value necessary to keep the kernel centered.
- In `FIR` interpolation mode, you can select `Switch to linear interpolation if kernel cannot be centered`. This option forces the block to preserve the value of D_{min} and compute all interpolated values using `Linear` interpolation.
- In `Farrow` interpolation mode, you can select `Use off-centered kernel`. This option forces the block to preserve the value of D_{min} and compute the interpolated values using a farrow filter with an off-centered kernel.

Valid delay range

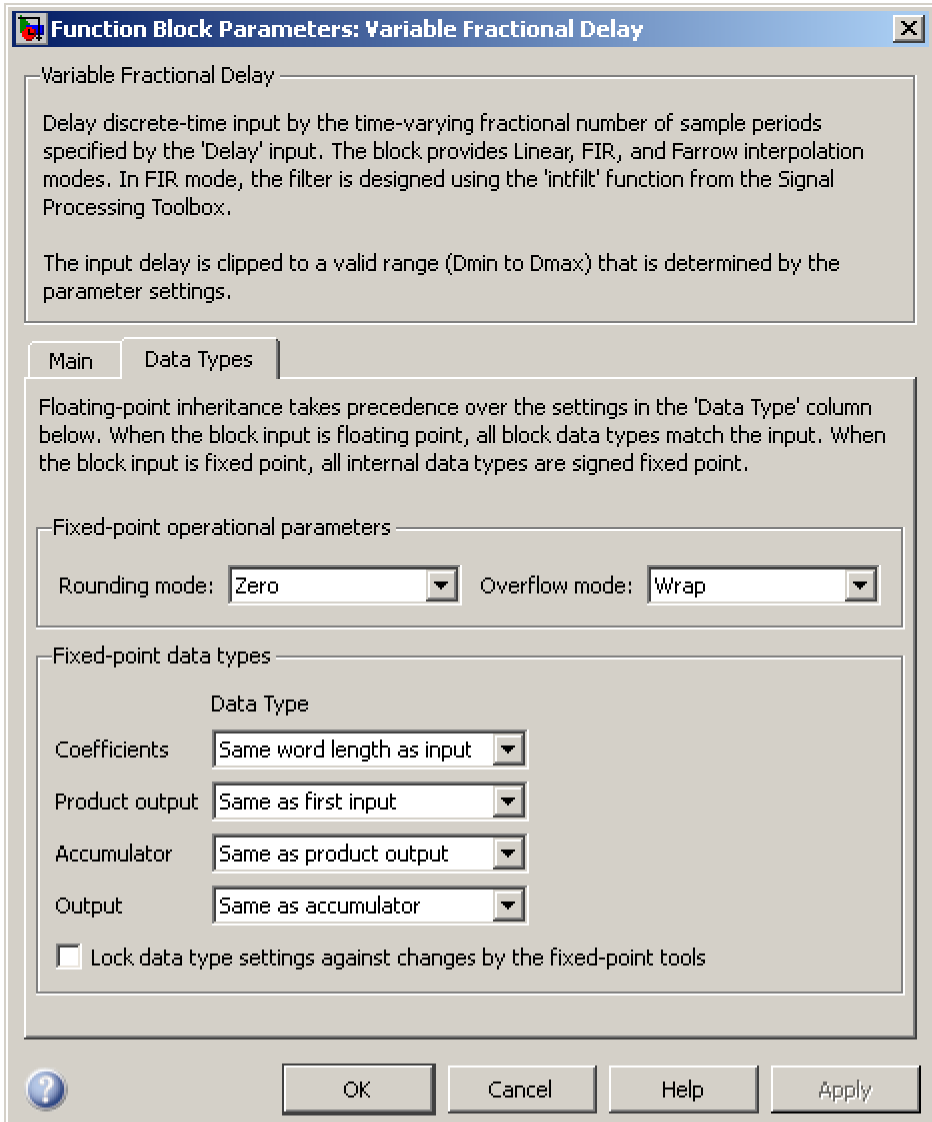
The values displayed in this section of the **Main** pane are calculated (in samples) by the block based on the current parameter settings. To update the values displayed in this section after changing parameter values on the block mask, click **Apply**.

- `Dmin` is the smallest possible valid delay value (in samples) based on the current settings of the block parameters. All input delay values less than `Dmin` are clipped to `Dmin`.

Note If you are working with frame-based signals and have the **Disable direct feedthrough by increasing the minimum possible delay by one** check box selected, you must compute the smallest possible valid delay value by adding `frame-size` – 1 to `Dmin`.

- `Dmax` is the maximum valid delay value (in samples) based on the current settings of the block parameters. All input delay values greater than `Dmax` are clipped to `Dmax`.

The **Data Types** pane of the Variable Fractional Delay block dialog appears as follows.


Function Block Parameters: Variable Fractional Delay
Variable Fractional Delay
Delay discrete-time input by the time-varying fractional number of sample periods specified by the 'Delay' input. The block provides Linear, FIR, and Farrow interpolation modes. In FIR mode, the filter is designed using the 'intfilt' function from the Signal Processing Toolbox.
The input delay is clipped to a valid range (Dmin to Dmax) that is determined by the parameter settings.
Main
Data Types
Floating-point inheritance takes precedence over the settings in the 'Data Type' column below. When the block input is floating point, all block data types match the input. When the block input is fixed point, all internal data types are signed fixed point.
Fixed-point operational parameters
Rounding mode:
Zero
Overflow mode:
Wrap
Fixed-point data types
Data Type
Coefficients
Same word length as input
Product output
Same as first input
Accumulator
Same as product output
Output
Same as accumulator
Lock data type settings against changes by the fixed-point tools
OK
Cancel
Help
Apply

Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode for fixed-point operations.

Coefficients

Choose how you specify the word length and fraction length of the filter coefficients.

- When you select `Same word length as input`, the word length of the filter coefficients match that of the input to the block. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.
- When you select `Specify word length`, you can enter the word length of the coefficients, in bits. In this mode, the fraction length of the coefficients is automatically set to the binary-point only scaling that provides you with the best precision possible given the value and word length of the coefficients.

Product output

Use this parameter to specify how you would like to designate the product output word and fraction lengths. See "Fixed-Point Data Types" on page 2-1415 and "Multiplication Data Types" for illustrations depicting the use of the product output data type in this block.

- When you select `Same as first input`, these characteristics match those of the first input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Accumulator

Use this parameter to specify how you would like to designate the accumulator word and fraction lengths. See "Fixed-Point Data Types" on page 2-1415 and "Multiplication Data Types" for illustrations depicting the use of the accumulator data type in this block:

- When you select `Same as product output`, these characteristics match those of the product output.
- When you select `Same as first input`, these characteristics match those of the first input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Choose how you specify the output word length and fraction length:

- When you select `Same as accumulator`, these characteristics match those of the accumulator.
- When you select `Same as first input`, these characteristics match those of the first input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Product output polyval

Choose how you specify the word length and fraction length of the product output polyval data type. This parameter is only visible when the **Interpolation mode** is set to `Farrow`.

- When you select `Same as first input`, these characteristics match those of the first input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output polyval in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output polyval. This block requires power-of-two slope and a bias of zero.

Accumulator polyval

Choose how you specify the word length and fraction length of the accumulator polyval data type. This parameter is only visible when the **Interpolation mode** is set to `Farrow`.

- When you select `Same as first input`, these characteristics match those of the first input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator polyval in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator polyval. This block requires power-of-two slope and a bias of zero.

Multiplicand polyval

Choose how you specify the word length and fraction length of the multiplicand polyval data type. This parameter is only visible when the **Interpolation mode** is set to `Farrow`.

- When you select `Same as first input`, these characteristics match those of the first input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the multiplicand polyval, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the multiplicand polyval. This block requires power-of-two slope and a bias of zero.

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Delay	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Delay	Signal Processing Blockset
Unit Delay	Simulink
Variable Integer Delay	Signal Processing Blockset

Variable Integer Delay

Purpose

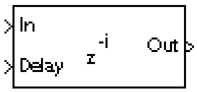
Delay input by time-varying integer number of sample periods

Library

Signal Operations

`dspsigops`

Description



The Variable Integer Delay block delays the discrete-time input at the In port by the integer number of sample intervals specified by the input to the Delay port. The sample rate of the input signal at the Delay port must be the same as the sample rate of the input signal at the In port. When these sample rates are not the same, you need to insert a Zero-Order Hold or Rate Transition block in order to make the sample rates identical. The delay for an N-D sample-based input can be a scalar value to uniformly delay every sample in every channel, or a matrix containing one delay value for each channel of the input. The delay for a frame-based input sequence can be a scalar value to uniformly delay every sample in every channel, a vector containing one delay value for each sample in the input frame, or a vector containing one delay value for each channel in the input frame.

The delay values should be in the range of 0 to D, where D is the **Maximum delay**. Delay values greater than D or less than 0 are clipped to those respective values and noninteger delays are rounded to the nearest integer value.

The Variable Integer Delay block differs from the Delay block in the following ways.

Variable Integer Delay Block	Delay Block
The delay is provided as an input to the Delay port.	You specify the delay as a parameter setting in the dialog box.
Delay can vary with time; for example, for a frame-based input, the *n*th element's delay in the first input frame can differ from the *n*th element's delay in the second input frame.	Delay cannot vary with time; for example, for a frame-based input, the *n*th element's delay is the same for every input frame.
When the Variable Integer Delay block is used in a feedback loop, you must check the **Disable direct feedthrough by increasing minimum possible delay by one** checkbox. This prevents the occurrence of an algebraic loop when the delay of the Variable Integer Delay block is driven to zero.	You can use the Delay block to break an algebraic loop.

Sample-Based Operation

The Variable Integer Delay block supports N-D input arrays. When the input is an M-by-N-by-P sample-based array, the block treats each of the $M*N*P$ elements as independent channels, and applies the delay at the Delay port to each channel.

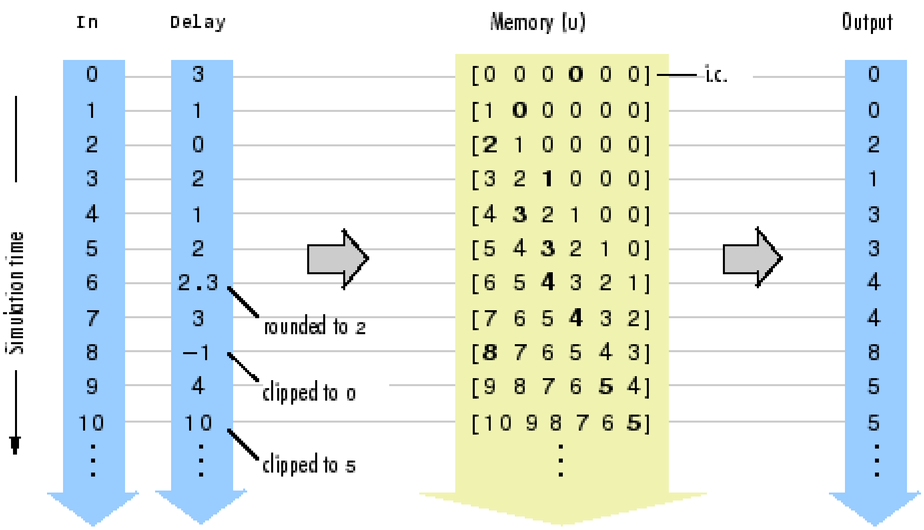
The Variable Integer Delay block stores the D+1 most recent samples received at the In port for each channel. At each sample time the block outputs the stored sample(s) indexed by the input to the Delay port.

For example, when the input to the In port, `u`, is a scalar signal, the block stores a vector, `U`, of the D+1 most recent signal samples. When the current input sample is `U(1)`, the previous input sample is `U(2)`, and so on, then the block's output is

```
y = U(v+1);     % Equivalent MATLAB code
```

where `v` is the input to the Delay port. A delay value of 0 (v=0) causes the block to pass through the sample at the In port in the same simulation step that it is received. The block's memory is initialized to the **Initial conditions** value at the start of the simulation (see below).

The next figure shows the block output for a scalar ramp sequence at the In port, a **Maximum delay** of 5, an **Initial conditions** of 0, and a variety of different delays at the Delay port.



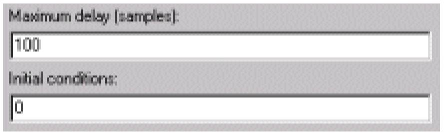
The current input at each time step is immediately stored in memory as `U(1)`. This allows the current input to be available at the output for a delay of 0 (v=0).

The **Initial conditions** parameter specifies the values in the block's memory at the start of the simulation. Unlike the Delay block, the Variable Integer Delay block does not have a fixed initial delay period during which the initial conditions appear at the output. Instead, the initial conditions are propagated to the output only when they are indexed in memory by the value at the Delay port. Both fixed and time-varying initial conditions can be specified in a variety of ways to suit the dimensions of the input sequence.

Fixed Initial Conditions

The settings in this section specify fixed initial conditions. For a fixed initial condition, the block initializes each of D samples in memory to the value entered in the **Initial conditions** parameter. A fixed initial condition in sample-based mode can be specified in one of the following ways:

- Scalar value with which to initialize every sample of every channel in memory. For a general M-by-N input and the parameter settings in this figure,

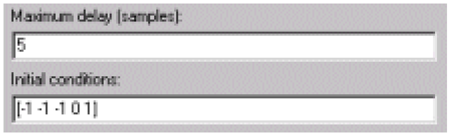
  the block initializes 100 M-by-N matrices in memory with zeros.

- Array of size M-by-N-by-D. In this case, you can specify different fixed initial conditions for each channel. See the Array bullet in "Time-Varying Initial Conditions" on page 2-1433 below for details.

Time-Varying Initial Conditions

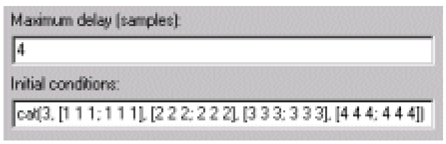
The following settings specify time-varying initial conditions. For a time-varying initial condition, the block initializes each of D samples in memory to one of the values entered in the **Initial conditions** parameter. This allows you to specify a unique output value for each sample in memory. A time-varying initial condition in sample-based mode can be specified in one of the following ways:

- Vector containing D elements with which to initialize memory samples `U(2:D+1)`, where D is the **Maximum delay**. For a scalar input and the parameters in the next figure, the block initializes `U(2:6)` with values `[-1, -1, -1, 0, 1]`.



- Array of dimension M-by-N-by-D with which to initialize memory samples `U(2:D+1)`, where D is the **Maximum delay** and M and N are the number of rows and columns, respectively, in the input matrix. For a 2-by-3 input and the following parameters, the block initializes memory locations `U(2:5)` with values

$$\mathbf{U}(2) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \mathbf{U}(3) = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}, \mathbf{U}(4) = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix}, \mathbf{U}(5) = \begin{bmatrix} 4 & 4 & 4 \\ 4 & 4 & 4 \end{bmatrix}$$



An M-by-N-by-P-by-D matrix can be entered for the **Initial Conditions** parameter when the input is an M-by-N-by-P array. The (M,N,P,T)th sample of the **Initial Conditions** matrix provides the initial condition value for the (M,N,P)th channel of the input matrix at delay = D–t+1 samples.

Frame-Based Operation

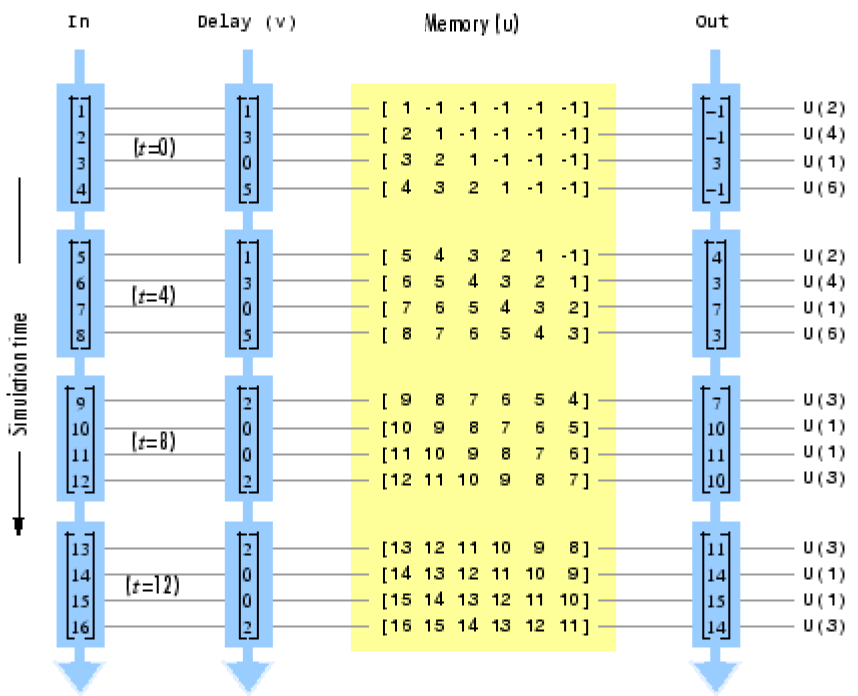
When the input is an M-by-N frame-based matrix, the block treats each of the N input columns as a frame containing M sequential time samples from an independent channel.

In frame-based mode, the input at the Delay port can be a scalar value to uniformly delay every sample in every channel. It can also be a column-based length-M vector, containing one delay for each sample in the input frame(s). The set of delays contained in the vector is applied identically to every channel of a multichannel input. The Delay port entry can also be a row-based length-N vector, containing one delay for each channel. Finally, the Delay port entry can also be an M-by-N matrix, containing a different delay for each corresponding element of the input.

Vector `v` does not specify when the samples in the current input frame will appear in the output. Rather, `v` indicates which previous input samples (stored in memory) should be included in the current output frame. The first sample in the current output frame is the input sample `v(1)` intervals earlier in the sequence, the second sample in the current output frame is the input sample `v(2)` intervals earlier in the sequence, and so on.

The illustration below shows how this works for an input with a sample period of 1 and frame size of 4. The **Maximum delay** (`Dmax`) is `5`, and the **Initial conditions** parameter is set to `-1`. The delay input changes from `[1 3 0 5]` to `[2 0 0 2]` after the second input frame. The samples in each output frame are the values in memory indexed by the elements of `v`:

```
y(1) = U(v(1)+1)
y(2) = U(v(2)+1)
y(3) = U(v(3)+1)
y(4) = U(v(4)+1)
```

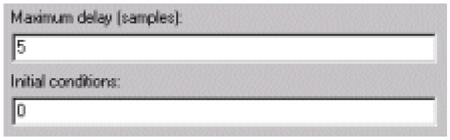


The **Initial conditions** parameter specifies the values in the block's memory at the start of the simulation. Both fixed and time-varying initial conditions can be specified.

Fixed Initial Conditions

The settings shown in this section specify fixed initial conditions. For a fixed initial condition, the block initializes each of *D* samples in memory to the value entered in the **Initial conditions** parameter. A fixed initial condition in frame-based mode can be one of the following:

- Scalar value with which to initialize every sample of every channel in memory. For a general *M*-by-*N* input with the parameter settings below, the block initializes five samples in memory with zeros.



- Array of size 1-by-N-by-D. In this case, you can specify different fixed initial conditions for each channel. See the Array bullet in "Time-Varying Initial Conditions" on page 2-1437 below for details.

Time-Varying Initial Conditions

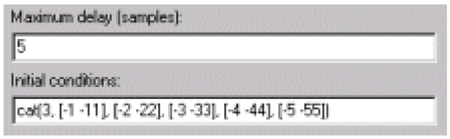
The following setting specifies a time-varying initial condition. For a time-varying initial condition, the block initializes each of D samples in memory to one of the values entered in the **Initial conditions** parameter. This allows you to specify a unique output value for each sample in memory. A time-varying initial condition in frame-based mode can be specified in the following ways:

- Vector containing D elements. In this case, all channels have the same set of time-varying initial conditions specified by the entries of the vector. For the ramp input `[1:100; 1:100]'` with a frame size of `4`, delay of `5`, and the following parameter settings, the block outputs the following sequence of frames at the start of the simulation:

$$\begin{bmatrix} -1 & -1 \\ -2 & -2 \\ -3 & -3 \\ -4 & -4 \end{bmatrix}, \begin{bmatrix} -5 & -5 \\ 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 \\ 5 & 5 \\ 6 & 6 \\ 7 & 7 \end{bmatrix}, \ldots$$

- Array of size 1-by-N-by-D. In this case, you can specify different time-varying initial conditions for each channel. For the ramp input `[1:100; 1:100]'` with a frame size of `4`, delay of `5`, and the following parameter settings, the block outputs the following sequence of frames at the start of the simulation:

$$\begin{bmatrix} -1 & -11 \\ -2 & -22 \\ -3 & -33 \\ -4 & -44 \end{bmatrix}, \begin{bmatrix} -5 & -55 \\ 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 \\ 5 & 5 \\ 6 & 6 \\ 7 & 7 \end{bmatrix}, \ldots$$

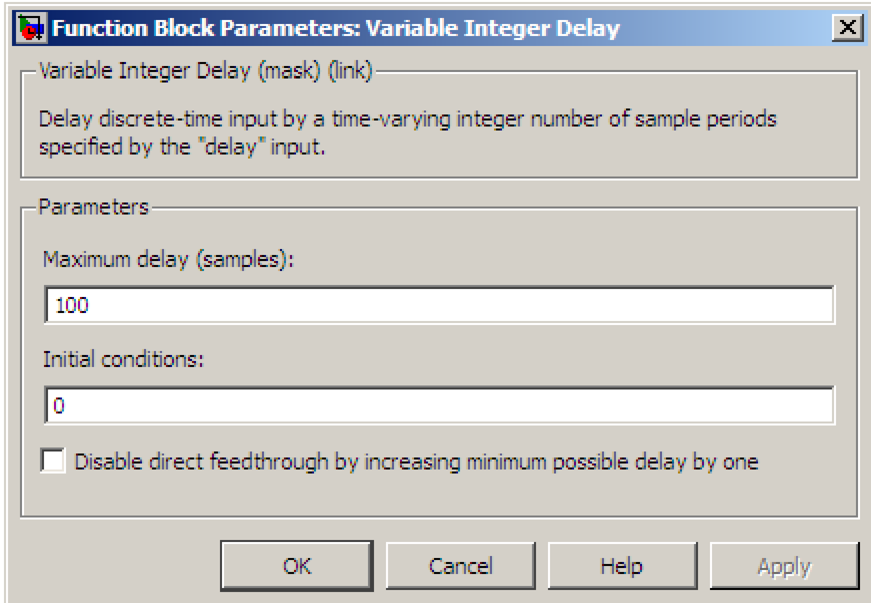


By specifying a 1-by-N-by-D initial condition array such that each 1-by-N vector entry is identical, you can implement different fixed initial conditions for each channel.

Examples

See “Basic Algorithmic Delay” in the *Signal Processing Blockset User’s Guide*.

Dialog Box



Maximum delay

The maximum delay that the block can produce for any sample. Delay input values exceeding this maximum are clipped at the maximum.

Initial conditions

The values with which the block's memory is initialized.

Disable direct feedthrough by increasing minimum possible delay by one

Select this box to disable direct feedthrough by adding one to the minimum possible delay value. Checking this box allows the Variable Integer Delay block to be used in feedback loops.

Supported Data Types

Port	Supported Data Types
In	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Delay	• Double-precision floating point • Single-precision floating point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Out	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Delay	Signal Processing Blockset
Variable Fractional Delay	Signal Processing Blockset

Purpose

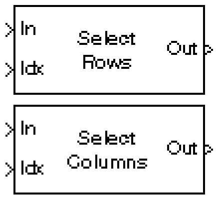
Select subset of rows or columns from input

Library

Signal Management / Indexing

`dspindex`

Description



The Variable Selector block extracts a subset of rows or columns from the M-by-N input matrix `u` at each input port. You specify the number of input and output ports in the **Number of input signals** parameter.

When the **Select** parameter is set to `Rows`, the Variable Selector block extracts rows from each input matrix, while if the **Select** parameter is set to `Columns`, the block extracts columns.

When the **Selector mode** parameter is set to `Variable`, the length-L vector input to the Idx port selects L rows or columns of each input to pass through to the output. The elements of the indexing vector can be updated at each sample time, but the vector length must remain the same throughout the simulation.

When the **Selector mode** parameter is set to `Fixed`, the Idx port is disabled, and the length-L vector specified in the **Elements** parameter selects L rows or columns of each input to pass through to the output. The **Elements** parameter is tunable, so you can change the values of the indexing vector elements at any time during the simulation; however, the vector length must remain the same.

For both variable and fixed indexing modes, the row selection operation is equivalent to

```
y = u(idx,:)      % Equivalent MATLAB code
```

and the column selection operation is equivalent to

```
y = u(:,idx)      % Equivalent MATLAB code
```

where `idx` is the length-L indexing vector. The row selection output size is L-by-N and the column selection output size is M-by-L. Input rows or columns can appear any number of times in the output, or not at all.

When the input is a 1-D vector, the **Select** parameter is ignored; the output is a 1-D vector of length L containing those elements specified by the length-L indexing vector.

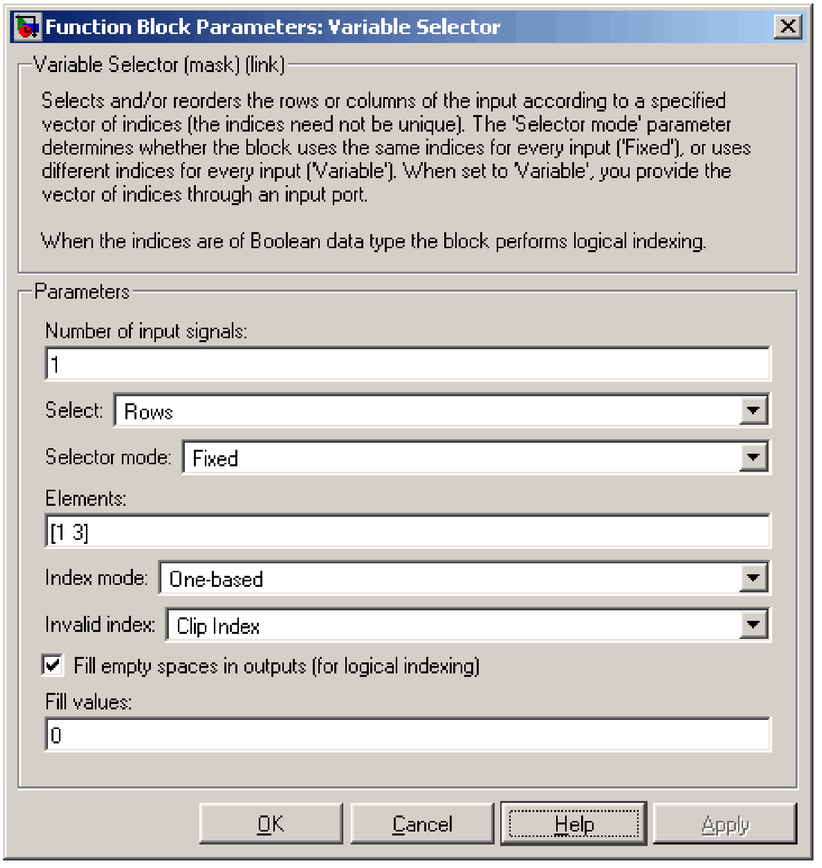
When an element of the indexing vector references a nonexistent row or column of the input, the block reacts with the behavior specified by the **Invalid index** parameter. The following options are available:

- `Clip index` — Clip the index to the nearest valid value, and do not issue an alert. Example: For a 64-by-N input, an index of 72 is clipped to 64; an index of -2 is clipped to 1.
- `Clip and warn` — Display a warning message in the MATLAB Command Window, and clip as above.
- `Generate error` — Display an error dialog box and terminate the simulation.

When the indexing vector elements are of Boolean data type, the block performs logical indexing. Select **Fill empty spaces in outputs (for logical indexing)** to access the **Fill values** parameter. These values are appended to the output to make it as long as the input elements.

Note The Variable Selector block always copies the selected input rows to a contiguous block of memory (unlike the Simulink Selector block).

Dialog Box



Number of input signals

Specify the number of input signals. An input port is created on the block for each input signal.

Select

The dimension of the input to select, `Rows` or `Columns`.

Selector mode

The type of indexing operation to perform, `Variable` or `Fixed`. Variable indexing uses the input at the Idx port to select rows or columns from the input at the `In` port. Fixed indexing uses the **Elements** parameter value to select rows from the input at the In port, and disables the Idx port.

Elements
A vector containing the indices of the input rows or columns that will appear in the output matrix. This parameter is only visible when you select `Fixed` for the **Selector mode** parameter.

Index mode
When set to `One-based`, an index value of `1` refers to the first row or column of the input. When set to `Zero-based`, an index value of `0` refers to the first row or column of the input.

Invalid index
Response to an invalid index value. Tunable.

Fill empty spaces in outputs (for logical indexing)
When the indexing vector elements are of Boolean data type, the block performs logical indexing. This can cause empty spaces in the output. Select this parameter to designate values to be appended to the output in the **Fill values** parameter.

Fill values
Specify the fill values when the block performs logical indexing. This parameter is only visible when the **Fill empty spaces in outputs (for logical indexing)** parameter is selected.

Supported Data Types

Port	Supported Data Types
In	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
	• Enumerated
Idx	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated
Out	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers • Enumerated

See Also

Multiport Selector	Signal Processing Blockset
Permute Matrix	Signal Processing Blockset
Selector	Simulink
Submatrix	Signal Processing Blockset

Variance

Purpose

Compute variance of input or sequence of inputs

Library

Statistics

`dspstat3`

Description

The Variance block computes the unbiased variance of each row or column of the input, along vectors of a specified dimension of the input, or of the entire input. The Variance block can also track the variance of a sequence of inputs over a period of time. The **Running variance** parameter selects between basic operation and running operation.

Basic Operation

When you do not select the **Running variance** check box, the block computes the variance of each row or column of the input, along vectors of a specified dimension of the input, or of the entire input at each individual sample time, and outputs the array y. Each element in y is the variance of the corresponding column, row, vector, or entire input. The output y depends on the setting of the **Find the variance value over** parameter. For example, consider a 3-dimensional input signal of size M-by-N-by-P:

- `Entire input` — The output at each sample time is a scalar that contains the variance of the entire input. In this mode, the output is always sample based.

  ```
  y = var(u(:))     % Equivalent MATLAB code
  ```

- `Each row` — The output at each sample time consists of an M-by-1-by-P array, where each element contains the variance of each vector over the second dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is an M-by-1 column vector. In this mode, the frame status of the output is the same as that of the input.

  ```
  y = var(u,0,2)     % Equivalent MATLAB code
  ```

- `Each column` — The output at each sample time consists of a 1-by-N-by-P array, where each element contains the variance of each vector over the first dimension of the input. For an input that is an M-by-N matrix, the output at each sample time is a 1-by-N row vector. In this mode, the frame status of the output is the same as that of the input.

  ```
  y = var(u,0,1)     % Equivalent MATLAB code
  ```

  For convenience, length-M 1-D vector inputs are treated as M-by-1 column vectors when the block is in this mode. Sample-based length-M row vector inputs are also treated as M-by-1 column vectors when the **Treat sample-based row input as a column** check box is selected.

- `Specified dimension` — The output at each sample time depends on **Dimension**. If **Dimension** is set to 1, the output is the same as that when you select `Each column`. If **Dimension** is set to `2`, the output is the same as when you select `Each row`. If **Dimension** is set to `3`, the output at each sample time is an M-by-N matrix containing the variance of each vector over the third dimension of the input. In this mode, the frame status of the output is the same as that of the input.

  ```
  y = var(u,0,Dimension)     % Equivalent MATLAB code
  ```

For purely real or purely imaginary inputs, the variance of an M-by-N matrix is the square of the standard deviation:

$$y = \sigma^2 = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N}\left|u_{ij}\right|^2 - \frac{\left|\sum_{i=1}^{M}\sum_{j=1}^{N}u_{ij}\right|^2}{M*N}}{M*N-1}$$

For complex inputs, the variance is given by the following equation:

$$\sigma^2 = \sigma_{\mathrm{Re}}{}^2 + \sigma_{\mathrm{Im}}{}^2$$

Running Operation

When you select the **Running variance** check box, the block tracks the variance of successive inputs to the block. For sample-based M-by-N inputs, the output is a sample-based M-by-N matrix with each element y_{ij} containing the variance of element u_{ij} over all inputs since the last reset. For frame-based M-by-N inputs, the output is a frame-based M-by-N matrix with each element y_{ij} containing the variance of the jth column over all inputs since the last reset, up to and including element u_{ij} of the current input.

N-D signals cannot be frame based. When the block is set to `Running` mode, each element of the N-D signal is treated as a separate channel.

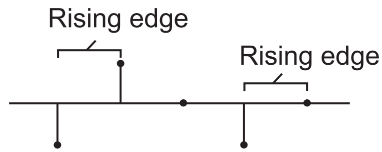
There are $\prod d_i$ channels, where d_i is the size of the ith dimension.

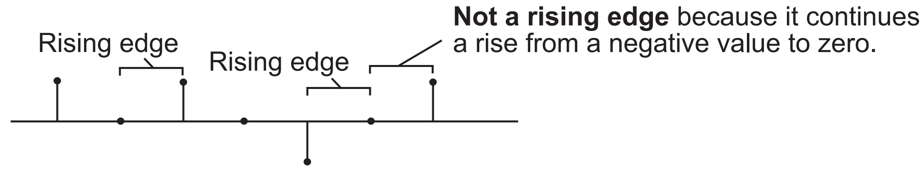
Resetting the Running Variance

The block resets the running variance whenever a reset event is detected at the optional Rst port. The reset sample time must be a positive integer multiple of the input sample time.

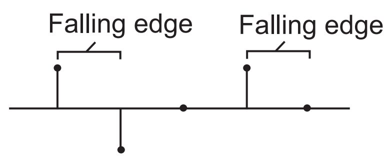
You specify the reset event in the **Reset port** parameter:

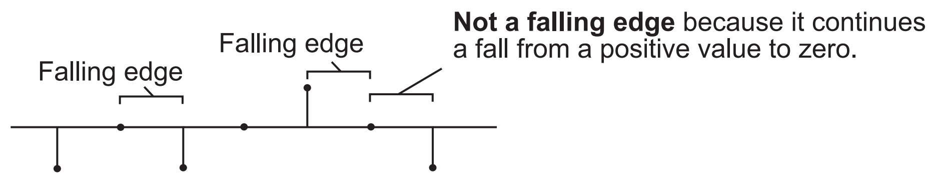
- `None` disables the Rst port.
- `Rising edge` — Triggers a reset operation when the Rst input does one of the following:
  - Rises from a negative value to a positive value or zero
  - Rises from zero to a positive value, where the rise is not a continuation of a rise from a negative value to zero (see the following figure)





- `Falling edge` — Triggers a reset operation when the Rst input does one of the following:
  - Falls from a positive value to a negative value or zero
  - Falls from zero to a negative value, where the fall is not a continuation of a fall from a positive value to zero (see the following figure)





- `Either edge` — Triggers a reset operation when the Rst input is a `Rising edge` or `Falling edge` (as described earlier)
- `Non-zero sample` — Triggers a reset operation at each sample time that the Rst input is not zero

---

Note When running simulations in the Simulink MultiTasking mode, reset signals have a one-sample latency. Therefore, when the block detects a reset event, there is a one-sample delay at the reset port rate before the block applies the reset. For more information on latency and the Simulink tasking modes, see “Excess Algorithmic Delay (Tasking Latency)” and the topic on models with multiple sample rates in the Real-Time Workshop documentation.

ROI Processing

To calculate the statistical value within a particular region of interest (ROI) of the input, select the **Enable ROI processing** check box. This option is only available when the **Find the variance value over** parameter is set to `Entire input` and the **Running variance** check box is not selected. ROI processing is only supported for 2-D inputs.

Note Full ROI processing is only available to users who have a Video and Image Processing Blockset license. If you only have a Signal Processing Blockset license, you can still use ROI processing, but are limited to the **ROI type** `Rectangles`.

Use the **ROI type** parameter to specify whether the ROI is a binary mask, label matrix, rectangle, or line. ROI processing is only supported for 2-D inputs.

- A binary mask is a binary image that enables you to specify which pixels to highlight, or select.
- In a label matrix, pixels equal to 0 represent the background, pixels equal to 1 represent the first object, pixels equal to 2 represent the second object, and so on. When the **ROI type** parameter is set to `Label matrix`, the Label and Label Numbers ports appear on the block. Use the Label Numbers port to specify the objects in the label matrix for which the block calculates statistics. The input to this

port must be a vector of scalar values that correspond to the labeled regions in the label matrix.

- For more information about the format of the input to the ROI port when the ROI is a rectangle or a line, see the Draw Shapes reference page.

Note For rectangular ROIs, use the **ROI portion to process** parameter to specify whether to calculate the statistical value for the entire ROI or just the ROI perimeter.

Use the **Output** parameter to specify the block output. The block can output separate statistical values for each ROI or the statistical value for all specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select `Binary mask`.

If, for the **ROI type** parameter, you select `Rectangles` or `Lines`, the **Output flag indicating if ROI is within image bounds** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual Statistics for Each ROI

Flag Port Output	Description
0	ROI is completely outside the input image.
1	ROI is completely or partially inside the input image.

Output = Single Statistic for All ROIs

Flag Port Output	Description
0	All ROIs are completely outside the input image.
1	At least one ROI is completely or partially inside the input image.

If the ROI is partially outside the image, the block only computes the statistical values for the portion of the ROI that is within the image.

If, for the **ROI type** parameter, you select `Label matrix`, the **Output flag indicating if input label numbers are valid** check box appears in the dialog box. If you select this check box, the Flag port appears on the block. The following tables describe the Flag port output based on the block parameters.

Output = Individual Statistics for Each ROI

Flag Port Output	Description
0	Label number is not in the label matrix.
1	Label number is in the label matrix.

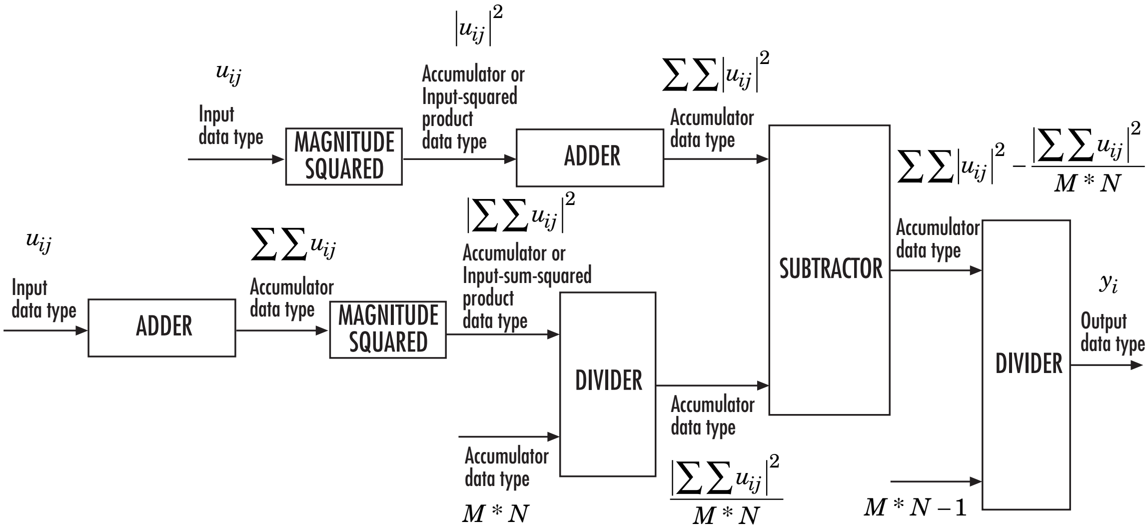
Output = Single Statistic for All ROIs

Flag Port Output	Description
0	None of the label numbers are in the label matrix.
1	At least one of the label numbers is in the label matrix.

Fixed-Point Data Types

The parameters on the **Data Types** pane of the block dialog are only used for fixed-point inputs. For purely real or purely imaginary inputs, the variance of the input is the square of its standard deviation. For complex inputs, the output is the sum of the variance of the real and imaginary parts of the input.

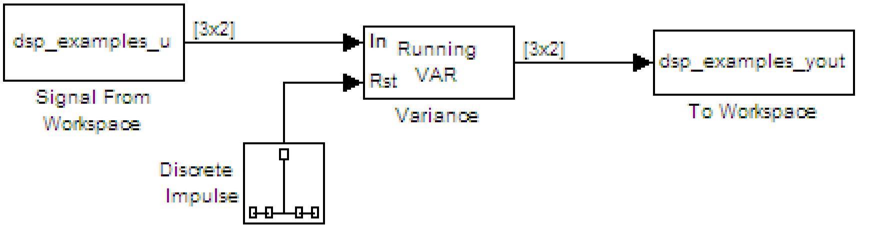
The following diagram shows the data types used within the Variance block for fixed-point signals.



The results of the magnitude-squared calculations in the figure are in the product output data type. You can set the accumulator, product output, and output data types in the block dialog as discussed in "Dialog Box" on page 2-1456.

Examples

The Variance block in the next model calculates the running variance of a frame-based 3-by-2 (two-channel) matrix input, `u`. The running variance is reset at *t*=2 by an impulse to the block's Rst port.



The Variance block has the following settings:

- **Running variance** = ☑
- **Reset port** = `Non-zero sample`

The Signal From Workspace block has the following settings:

- **Signal** = `dsp_examples_u`
- **Sample time** = `1/3`
- **Samples per frame** = `3`

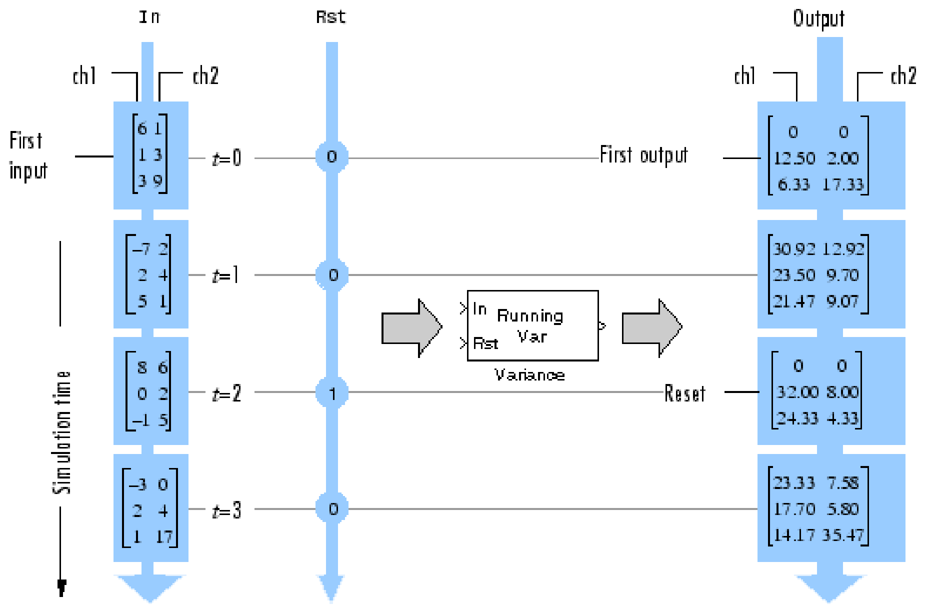
where

```
dsp_examples_u = [6 1 3 -7 2 5 8 0 -1 -3 2 1;1 3 9 2 4 1 6 2 5 0 4 17]'
```

The Discrete Impulse block has the following settings:

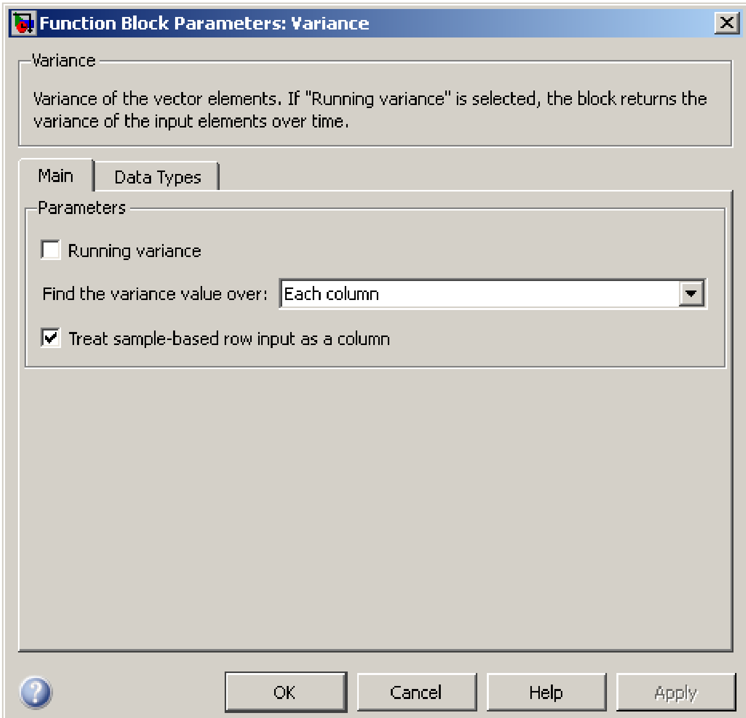
- **Delay (samples)** = `2`
- **Sample time** = `1`
- **Samples per frame** = `1`

The next figure shows the block's operation.


In
Rst
Output
ch1
ch2
ch1
ch2
First input
First output
Reset
Simulation time
t=0
t=1
t=2
t=3
In
Rst
Running Var
Variance

Dialog Box

The **Main** pane of the Variance block dialog appears as follows.



Running variance
: Enables running operation when selected.

Reset port
: Determines the reset event that causes the block to reset the running variance. The rate of the reset signal must be a positive integer multiple of the rate of the data signal input. This parameter is enabled only when you select the **Running variance** parameter. For more information, see “Resetting the Running Variance” on page 2-1448

Find the variance value over
Specify whether to find the variance along rows, columns, entire input, or the dimension specified in the **Dimension** parameter. For more information, see "Basic Operation" on page 2-1446.

Treat sample-based row input as a column
Select to treat sample-based length-M row vector inputs as M-by-1 column vectors. This parameter is only visible when the **Find the variance value over** parameter is set to `Each column`.

Dimension
Specify the dimension (one-based value) of the input signal, over which the variance is computed. The value of this parameter cannot exceed the number of dimensions in the input signal. This parameter is only visible when the **Find the variance value over** parameter is set to `Specified dimension`.

Enable ROI Processing
Select this check box to calculate the statistical value within a particular region of each image. This parameter is only available when the **Find the variance value over** parameter is set to `Entire input`, and the block is not in running mode.

Note Full ROI processing is only available to users who have a Video and Image Processing Blockset license. If you only have a Signal Processing Blockset license, you can still use ROI processing, but are limited to the **ROI type** `Rectangles`.

ROI type
Specify the type of ROI you want to use. Your choices are `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`.

ROI portion to process
Specify whether you want to calculate the statistical value for the entire ROI or just the ROI perimeter. This parameter is only visible if, for the **ROI type** parameter, you specify `Rectangles`.

Output

Specify the block output. The block can output a vector of separate statistical values for each ROI or a scalar value that represents the statistical value for all the specified ROIs. This parameter is not available if, for the **ROI type** parameter, you select `Binary mask`.

Output flag

☑ **Output flag indicating if ROI is within image bounds**

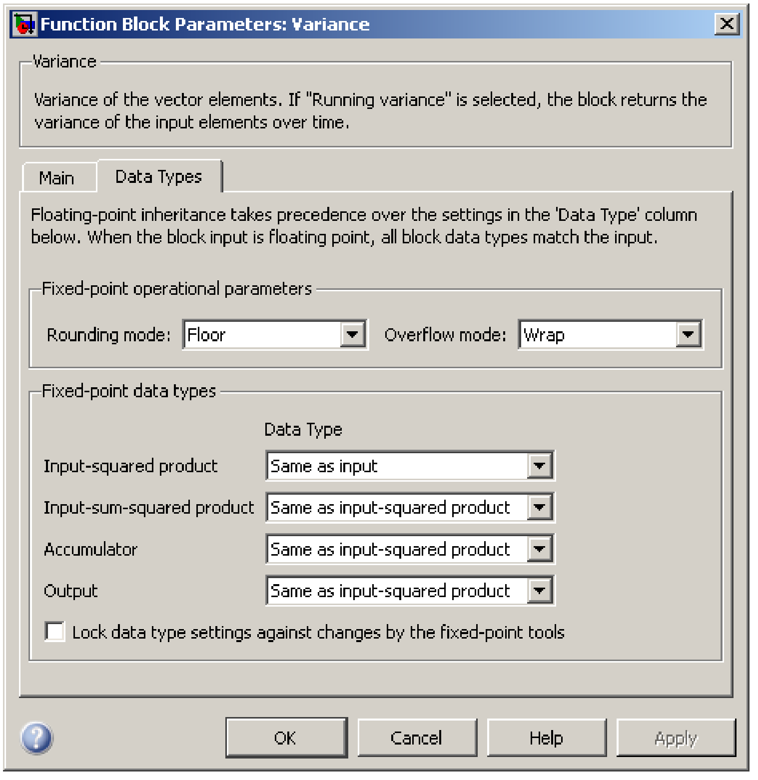
☑ **Output flag indicating if label numbers are valid**

When you select either of these check boxes, the Flag port appears on the block. For a description of the Flag port output, see the tables in “ROI Processing” on page 2-1450.

The **Output flag indicating if ROI is within image bounds** check box is only visible when you select `Rectangles` or `Lines` as the **ROI type**.

The **Output flag indicating if label numbers are valid** check box is only visible when you select `Label matrix` for the **ROI type** parameter.

The **Data Types** pane of the Variance block dialog appears as follows.



Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode for fixed-point operations.

Note See "Fixed-Point Data Types" on page 2-1453 for more information on how the product output, accumulator, and output data types are used in this block.

Input-squared product

Use this parameter to specify how to designate the input-squared product word and fraction lengths:

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the input-squared product, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the input-squared product. This block requires power-of-two slope and a bias of zero.

Input-sum-squared product

Use this parameter to specify how to designate the input-sum-squared product word and fraction lengths:

- When you select `Same as input-squared product`, these characteristics match those of the input-squared product.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the input-sum-squared product, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the input-sum-squared product. This block requires power-of-two slope and a bias of zero.

Accumulator

Use this parameter to specify the accumulator word and fraction lengths resulting from a complex-complex multiplication in the block:

- When you select `Same as input-squared product`, these characteristics match those of the input-squared product.
- When you select `Same as input`, these characteristics match those of the input to the block.

- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and a bias of zero.

Output

Choose how you specify the output word length and fraction length:

- When you select `Same as accumulator`, these characteristics match those of the accumulator.
- When you select `Same as input-squared product`, these characteristics match those of the input-squared product.
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock data type settings against changes by the fixed-point tools

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Reset	• Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
ROI	Rectangles and lines: • Double-precision floating point • Single-precision floating point • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers Binary Mask: • Boolean
Label	• 8-, 16-, and 32-bit unsigned integers
Label Numbers	• 8-, 16-, and 32-bit unsigned integers

Port	Supported Data Types
Output	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Flag	• Boolean

See Also

Mean	Signal Processing Blockset
RMS	Signal Processing Blockset
Standard Deviation	Signal Processing Blockset
`var`	MATLAB

Vector Quantizer Decoder

Purpose

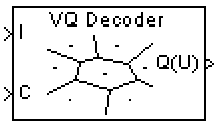
Find vector quantizer codeword that corresponds to given, zero-based index value

Library

Quantizers

`dspquant2`

Description



Vector Quantizer Decoder

The Vector Quantizer Decoder block associates each input index value with a codeword, a column vector of quantized output values defined in the **Codebook values** parameter. When you input multiple index values into this block, the block outputs a matrix of quantized output vectors. This matrix is created by horizontally concatenating the codeword vectors that correspond to each index value.

You can select how you want to enter the codebook values using the **Source of codebook** parameter. When you select `Specify via dialog`, you can type the codebook values into the block parameters dialog box. Select `Input port` and port C appears on the block. The block uses the input to port C as the **Codebook values** parameter.

The **Codebook values** parameter is a k-by-N matrix of values, where $k \geq 1$ and $N \geq 1$. Each column of this matrix is a codeword vector, and each codeword vector corresponds to an index value. The index values are zero based; therefore, the first codeword vector corresponds to an index value of 0, the second codeword vector corresponds to an index value of 1, and so on.

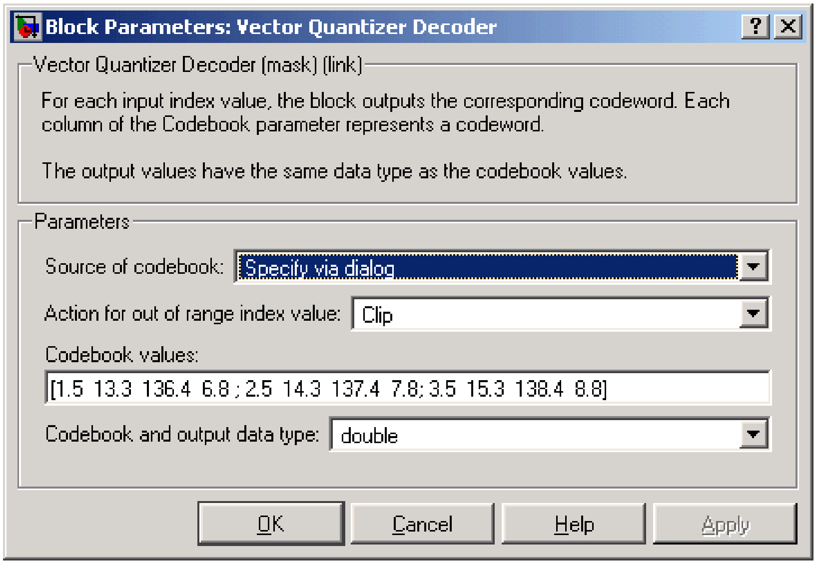
The input to this block is a vector of index values, where $0 \leq index < N$ and N is the number of columns of the codebook matrix. Use the **Action for out of range index value** parameter to determine how the block behaves when an input index value is out of this range. When you want any index values less than 0 to be set to 0 and any index values greater than or equal to N to be set to N-1, select `Clip`. When you want to be warned when any index values less than 0 are set to 0 and any index values greater than or equal to N are set to N-1, select `Clip and warn`. When you want the simulation to stop and display an error when the index values are out of range, select `Error`.

Data Type Support

The input to the block can be the index values and the codebook values. The data type of the index input to the block at port I can be `uint8`, `uint16`, `uint32`, `int8`, `int16`, or `int32`. The data type of the codebook values can be `double`, `single`, or Fixed-point.

The output of the block is the quantized output values. These quantized output values always have the same data type as the codebook values. When the codebook values are specified via an input port, the block assigns the same data type to the Q(U) output port. When the codebook values are specified via the dialog, use the **Codebook and output data type** parameter to specify the data type of the Q(U) output port. The data type of the codebook and quantized output can be `Same as input`, `double`, `single`, `Fixed-point`, `User-defined`, or `Inherit via back propagation`.

Dialog Box



Source of codebook

Choose `Specify via dialog` to type the codebook values into the block parameters dialog box. Select `Input port` to specify the codebook values using the block's input port, C.

Action for out of range index value

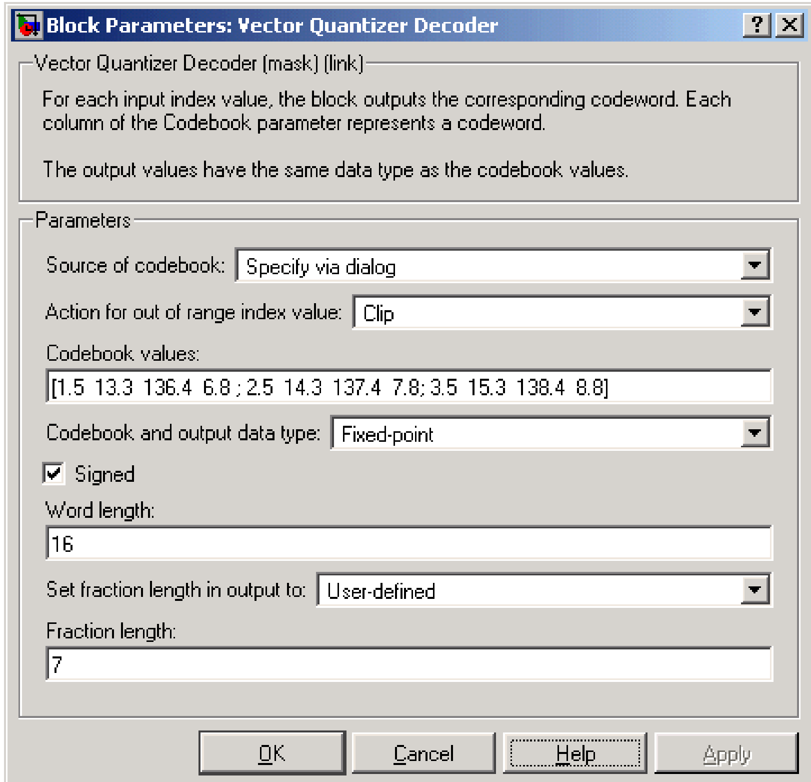
Choose the behavior of the block when an input index value is out of range, where $0 \leq index < N$ and N is the length of the codebook vector. Select `Clip` when you want any index values less than 0 to be set to 0 and any index values greater than or equal to N to be set to N-1. Select `Clip and warn` when you want to be warned when any index values less than 0 are set to 0 and any index values greater than or equal to N are set to N-1. Select `Error` when you want the simulation to stop and display an error when the index values are out of range.

Codebook values

Enter a k-by-N matrix of quantized output values, where $1 \leq k$ and $1 \leq N$. Each column of your matrix corresponds to an index value. This parameter is visible if, from the **Source of codebook** list, you select `Specify via dialog`.

Codebook and output data type

Use this parameter to specify the data type of the codebook and quantized output values. The data type can be `Same as input`, `double`, `single`, `Fixed-point`, `User-defined`, or`Inherit via back propagation`. This parameter becomes visible when you select `Specify via dialog` for the **Source of codebook** parameter.



Signed
: Select to output a signed fixed-point signal. Otherwise, the signal is unsigned. This parameter is only visible if, from the **Codebook and output data type** list, you select `Fixed-point`.

Word length
: Specify the word length, in bits, of the fixed-point output data type. This parameter is only visible if, from the **Codebook and output data type** list, you select `Fixed-point`.

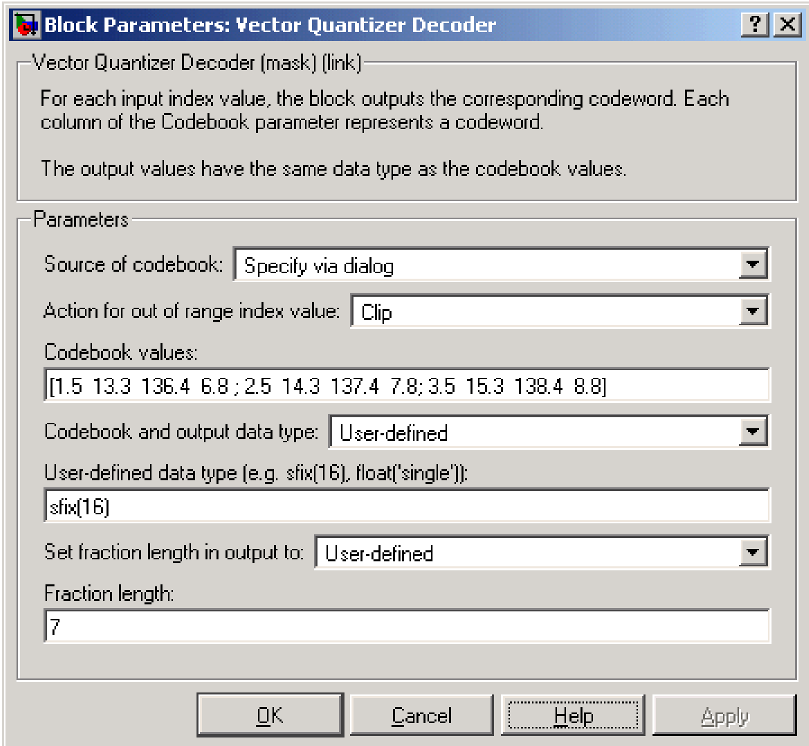
Set fraction length in output to
: Specify the scaling of the fixed-point output by either of the following two methods:

- Choose `Best precision` to have the output scaling automatically set such that the output signal has the best possible precision.
- Choose `User-defined` to specify the output scaling in the **Fraction length** parameter.

This parameter is only visible if, from the **Codebook and output data type** list, you select `Fixed-point` or when you select `User-defined` and the specified output data type is a fixed-point data type.

Fraction length

For fixed-point output data types, specify the number of fractional bits, or bits to the right of the binary point. This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Codebook and output data type** parameter and `User-defined` for the **Set fraction length in output to** parameter.



User-defined data type

Specify any built-in or fixed-point data type. You can specify fixed-point data types using the following Simulink Fixed Point functions: `sfix`, `ufix`, `sint`, `uint`, `sfrac`, and `ufrac`. This parameter is only visible when you select `User-defined` for the **Codebook and output data type** parameter.

References

Gersho, A. and R. Gray. *Vector Quantization and Signal Compression*. Boston: Kluwer Academic Publishers, 1992.

Supported Data Types

Port	Supported Data Types
I	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
C	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers
Q(U)	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Quantizer	Simulink
Scalar Quantizer Decoder	Signal Processing Blockset
Scalar Quantizer Design	Signal Processing Blockset
Uniform Encoder	Signal Processing Blockset
Uniform Decoder	Signal Processing Blockset
Vector Quantizer Encoder	Signal Processing Blockset

Purpose

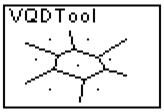
Design vector quantizer using Vector Quantizer Design Tool (VQDTool)

Library

Quantizers

`dspquant2`

Description



Vector Quantizer Design

Double-click on the Vector Quantizer Design block to start VQDTool, a GUI that allows you to design and implement a vector quantizer. You can also start VQDTool by typing `vqdtool` at the MATLAB command prompt. Based on your specifications, VQDTool iteratively calculates the codebook values that minimize the mean squared error between the training set and the codebook until the stopping criteria for the design process is satisfied. The block uses the resulting codebook values to implement your vector quantizer.

For the **Training Set** parameter, enter a k-by-M matrix of values you want to use to train the quantizer codebook. The variable k, where $k \geq 1$, is the length of each training vector. It also represents the dimension of your quantizer. The variable M, where $M \geq 2$, is the number of training vectors. This data can be created using a MATLAB function, such as the default value `randn(10,1000)`, or it can be any variable defined in the MATLAB workspace.

You have two choices for the **Source of initial codebook** parameter. Select `Auto-generate` to have the block choose the values of the initial codebook. In this case, the block picks N random training vectors as the initial codebook, where N is the **Number of levels** parameter and $N \geq 2$. When you select `User defined`, enter the initial codebook values in the **Initial codebook** field. The initial codebook matrix must have the same number of rows as the training set. Each column of the codebook is a codeword, and your codebook must have at least two codewords.

For the given training set and initial codebook, the block performs an iterative process, using the Generalized Lloyd Algorithm (GLA), to design a final codebook. For each iteration of the GLA, the block first associates each training vector with its nearest codeword by calculating the distortion. You can specify one of the two possible methods for calculating distortion using the **Distortion measure** parameter.

When you select `Squared error` for the **Distortion measure** parameter, the block finds the nearest codeword by calculating the squared error (unweighted). Consider the codebook

$CB = [CW_1 \quad CW_2 \quad \ldots \quad CW_N]$. This codebook has N codewords; each codeword has k elements. The i-th codeword is defined as

$CW_i = [a_{1i} \quad a_{2i} \quad \ldots \quad a_{ki}]$. The training set has M columns and is defined as $U = [U_1 \quad U_2 \quad \ldots \quad U_M]$, where the p-th training vector

is $U_p = \left[u_{1p} \quad u_{2p} \quad \ldots \quad u_{kp}\right]'$. The squared error (unweighted) is calculated using the equation

$$D = \sum_{j=1}^{k} \left(a_{ji} - u_{jp}\right)^2$$

When you select `Weighted squared error` for the **Distortion measure** parameter, enter a vector or matrix for the **Weighting factor** parameter. When the weighting factor is a vector, its length must be equal to the number of rows in the training set. This weighting factor is used for each training vector. When the weighting factor is a matrix, it must be the same size as the training set matrix. The block finds the nearest codeword by calculating the weighted squared error. If the weighting factor for the p-th column of the training vector, U_p,

is defined as $Wp = \left[w_{1p} \quad w_{2p} \quad \ldots \quad w_{kp}\right]'$, then the weighted squared error is defined by the equation

$$D = \sum_{j=1}^{k} w_{jp} \left(a_{ji} - u_{jp}\right)^2$$

Once the block has associated all the training vectors with their nearest codeword vectors, the block calculates the mean squared error for the codebook and checks to see if the stopping criteria for the process has been satisfied.

The two possible options for the **Stopping criteria** parameter are `Relative threshold` and `Maximum iteration`. When you want the design process to stop when the fractional drop in the squared error is below a certain value, select `Relative threshold`. Then, type the maximum acceptable fractional drop in the **Relative threshold** field. The fraction drop in the squared error is defined as

$$\frac{\text{error at previous iteration} - \text{error at current iteration}}{\text{error at previous iteration}}$$

When you want the design process to stop after a certain number of iterations, choose `Maximum iteration`. Then, enter the maximum number of iterations you want the block to perform in the **Maximum iteration** field. For **Stopping criteria**, you can also choose `Whichever comes first` and enter **Relative threshold** and **Maximum iteration** values. The block stops iterating as soon as one of these conditions is satisfied.

When a training vector has the same distortion for two different codeword vectors, the algorithm uses the **Tie-breaking rule** parameter to determine which codeword vector the training vector is associated with. When you want the training vector to be associated with the lower indexed codeword, select `Lower indexed codeword`. To associate the training vector with the higher indexed codeword, select `Higher indexed codeword`.

With each iteration, the block updates the codeword values in order to minimize the distortion. The **Codebook update method** parameter defines the way the block calculates these new codebook values.

Note If, for the **Distortion measure** parameter, you choose `Squared error`, the **Codebook update method** parameter is set to `Mean`.

If, for the **Distortion measure** parameter, you choose `Weighted squared error` and you choose `Mean` for the **Codebook update method** parameter, the new codeword vector is found as follows. Suppose there are three training vectors associated with one codeword vector. The training vectors are

$$TS_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix},\ TS_3 = \begin{bmatrix} 10 \\ 12 \end{bmatrix},\ \text{and}\ TS_7 = \begin{bmatrix} 11 \\ 12 \end{bmatrix}.$$

The new codeword vector is calculated as

$$CW_{new} = \begin{bmatrix} \dfrac{1+10+11}{3} \\ \dfrac{2+12+12}{3} \end{bmatrix}$$

where the denominator is the number of training vectors associated with this codeword. If, for the **Codebook update method** parameter, you choose `Centroid` and you specify the weighting factors $W_1 = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$, $W_3 = \begin{bmatrix} 1 \\ 0.6 \end{bmatrix}$, and $W_7 = \begin{bmatrix} 0.3 \\ 0.4 \end{bmatrix}$, the new codeword vector is calculated as

$$CW_{new} = \begin{bmatrix} \dfrac{(0.1)(1)+(1)(10)+(0.3)(11)}{0.1+1+0.3} \\ \dfrac{(0.2)(2)+(0.6)(12)+(0.4)(12)}{0.2+0.6+0.4} \end{bmatrix}$$

Click **Design and Plot** to design the quantizer with the parameter values specified on the left side of the GUI. The performance curve and the entropy of the quantizer are updated and displayed in the figures on the right side of the GUI.

Note You must click **Design and Plot** to apply any changes you make to the parameter values in the VQDTool dialog box.

The following is an example of how the block calculates the entropy of the quantizer at each iteration. Suppose you have a codebook with four codewords and a training set with 200 training vectors. Also suppose that, at the *i*-th iteration, 40 training vectors are associated with the first codeword, 60 training vectors are associated with the second codeword, 20 training vectors are associated with the third codeword, and 80 training vectors are associated with the fourth codeword. The probability that a training vector is associated with the first codeword is $\frac{40}{200}$. The probabilities that training vectors are associated with the second, third, and fourth codewords are $\frac{60}{200}$, $\frac{20}{200}$, and $\frac{80}{200}$, respectively. The GUI uses these probabilities to calculate the entropy according to the equation

$$H = \sum_{i=1}^{N} -p_i \log_2 p_i$$

where N is the number of codewords. Based on these probabilities, the GUI calculates the entropy of the quantizer at the *i*-th iteration as

$$H = -\left(\frac{40}{200}\log_2\frac{40}{200} + \frac{60}{200}\log_2\frac{60}{200} + \frac{20}{200}\log_2\frac{20}{200} + \frac{80}{200}\log_2\frac{80}{200}\right)$$
$$H = 1.8464$$

VQDTool can export parameter values that correspond to the figures displayed in the GUI. Click the **Export Outputs** button, or press **Ctrl+E**, to export the **Final Codebook**, **Mean Square Error**, and **Entropy** values to the workspace, a text file, or a MAT-file.

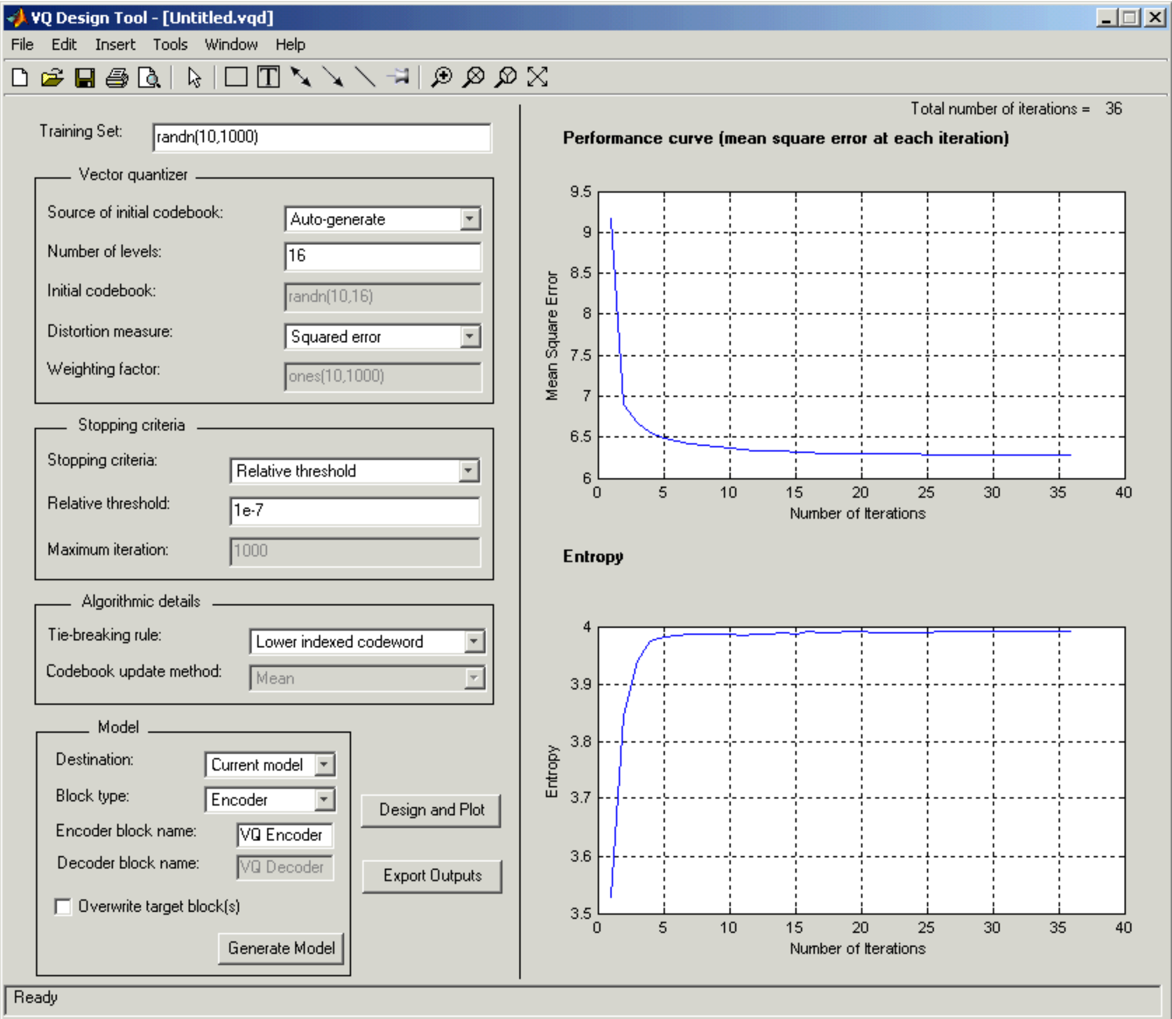
In the **Model** section of the GUI, specify the destination of the block that will contain the parameters of your quantizer. For **Destination**, select `Current model` to create a block with your parameters in the model you most recently selected. Type `gcs` in the MATLAB Command Window to display the name of your current model. Select `New model` to create a block in a new model file.

From the **Block type** list, select `Encoder` to design a Vector Quantizer Encoder block. Select `Decoder` to design a Vector Quantizer Decoder block. Select `Both` to design a Vector Quantizer Encoder block and a Vector Quantizer Decoder block.

In the **Encoder block name** field, enter a name for the Vector Quantizer Encoder block. In the **Decoder block name** field, enter a name for the Vector Quantizer Decoder block. When you have a Vector Quantizer Encoder and/or Decoder block in your destination model with the same name, select the **Overwrite target block** check box to replace the block's parameters with the current parameters. When you do not select this check box, a new Vector Quantizer Encoder and/or Decoder block is created in your destination model.

Click **Generate Model**. VQDTool uses the parameters that correspond to the current plots to set the parameters of the Vector Quantizer Encoder and/or Decoder blocks.

Dialog Box


VQ Design Tool - [Untitled.vqd]
File Edit Insert Tools Window Help
Total number of iterations = 36
Training Set: randn(10,1000)
Vector quantizer
Source of initial codebook: Auto-generate
Number of levels: 16
Initial codebook: randn(10,16)
Distortion measure: Squared error
Weighting factor: ones(10,1000)
Stopping criteria
Stopping criteria: Relative threshold
Relative threshold: 1e-7
Maximum iteration: 1000
Algorithmic details
Tie-breaking rule: Lower indexed codeword
Codebook update method: Mean
Model
Destination: Current model
Block type: Encoder
Encoder block name: VQ Encoder
Decoder block name: VQ Decoder
Overwrite target block(s)
Generate Model
Design and Plot
Export Outputs
Performance curve (mean square error at each iteration)
Mean Square Error
Number of Iterations
Entropy
Entropy
Number of Iterations
Ready

Training Set
Enter the samples of the signal you would like to quantize. This data set can be a MATLAB function or a variable defined in the MATLAB workspace. The typical length of this data vector is `1e5`.

Source of initial codebook
Select `Auto-generate` to have the block choose the initial codebook values. Choose `User defined` to enter your own initial codebook values.

Number of levels
Enter the number of codeword vectors, N, in your codebook matrix, where $N \geq 2$.

Initial codebook
Enter your initial codebook values. From the **Source of initial codebook** list, select `User defined` in order to activate this parameter. The codebook must have the same number of rows as the training set. You must provide at least two codeword vectors.

Distortion measure
When you select `Squared error`, the block finds the nearest codeword by calculating the squared error (unweighted). When you select `Weighted squared error`, the block finds the nearest codeword by calculating the weighted squared error.

Weighting factor
Enter a vector or matrix. The block uses these values to compute the weighted squared error. When the weighting factor is a vector, its length must be equal to the number of rows in the training set. This weighting factor is used for each training vector. When the weighting factor is a matrix, it must be the same size as the training set matrix. The individual weighting factors cannot be negative. The weighting factor vector or matrix cannot contain all zeros.

Stopping criteria
Choose `Relative threshold` to enter the maximum acceptable fractional drop in the squared quantization error. Choose `Maximum iteration` to specify the number of iterations at which to stop.

Choose `Whichever comes first` and the block stops the iteration process as soon as the relative threshold or maximum iteration value is attained.

Relative threshold
This parameter is available when you choose `Relative threshold` or `Whichever comes first` for the **Stopping criteria** parameter. Enter the value that is the maximum acceptable fractional drop in the squared quantization error.

Maximum iteration
This parameter is available when you choose `Maximum iteration` or `Whichever comes first` for the **Stopping criteria** parameter. Enter the maximum number of iterations you want the block to perform.

Tie-breaking rules
When a training vector has the same distortion for two different codeword vectors, select `Lower indexed codeword` to associate the training vector with the lower indexed codeword. Select `Higher indexed codeword` to associate the training vector with the lower indexed codeword.

Codebook update method
When you choose `Mean`, the new codeword vector is calculated by taking the average of all the training vector values that were associated with the original codeword vector. When you choose `Centroid`, the block calculates the new codeword vector by taking the weighted average of all the training vector values that were associated with the original codeword vector Note that if, for the **Distortion measure** parameter, you choose `Squared error`, the **Codebook update method** parameter is set to `Mean`.

Destination
Choose `Current model` to create a Vector Quantizer block in the model you most recently selected. Type `gcs` in the MATLAB Command Window to display the name of your current model. Choose `New model` to create a block in a new model file.

Block type
Select `Encoder` to design a Vector Quantizer Encoder block. Select `Decoder` to design a Vector Quantizer Decoder block. Select `Both` to design a Vector Quantizer Encoder block and a Vector Quantizer Decoder block.

Encoder block name
Enter a name for the Vector Quantizer Encoder block.

Decoder block name
Enter a name for the Vector Quantizer Decoder block.

Overwrite target block
When you do not select this check box and a Vector Quantizer Encoder and/or Decoder block with the same block name exists in the destination model, a new Vector Quantizer Encoder and/or Decoder block is created in the destination model. When you select this check box and a Vector Quantizer Encoder and/or Decoder block with the same block name exists in the destination model, the parameters of these blocks are overwritten by new parameters.

Generate Model
Click this button and VQDTool uses the parameters that correspond to the current plots to set the parameters of the Vector Quantizer Encoder and/or Decoder blocks.

Design and Plot
Click this button to design a quantizer using the parameters on the left side of the GUI and to update the performance curve and entropy plots on the right side of the GUI.

You must click **Design and Plot** to apply any changes you make to the parameter values in the VQDTool GUI.

Export Outputs
Click this button, or press **Ctrl+E**, to export the **Final Codebook**, **Mean Squared Error**, and **Entropy** values to the workspace, a text file, or a MAT-file.

References

Gersho, A. and R. Gray. *Vector Quantization and Signal Compression*. Boston: Kluwer Academic Publishers, 1992.

Supported Data Types

- Double-precision floating point

See Also

Quantizer	Simulink
Scalar Quantizer Decoder	Signal Processing Blockset
Scalar Quantizer Design	Signal Processing Blockset
Uniform Encoder	Signal Processing Blockset
Uniform Decoder	Signal Processing Blockset
Vector Quantizer Decoder	Signal Processing Blockset
Vector Quantizer Encoder	Signal Processing Blockset

Vector Quantizer Encoder

Purpose

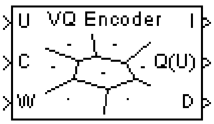
For given input, find index of nearest codeword based on Euclidean or weighted Euclidean distance measure

Library

Quantizers

`dspquant2`

Description



Vector Quantizer Encoder

The Vector Quantizer Encoder block compares each input column vector to the codeword vectors in the codebook matrix. Each column of this codebook matrix is a codeword. The block finds the codeword vector nearest to the input column vector and returns its zero-based index. This block supports real floating-point and fixed-point signals on all input ports.

The block finds the nearest codeword by calculating the distortion. The block uses two methods for calculating distortion: Euclidean squared error (unweighted) and weighted Euclidean squared error. Consider the codebook, $CB = [CW_1 \quad CW_2 \quad \ldots \quad CW_N]$. This codebook has N codewords; each codeword has k elements. The i-th codeword is defined as a column vector, $CW_i = [a_{1i} \quad a_{2i} \quad \ldots \quad a_{ki}]$. The multichannel input has M columns and is defined as $U = [U_1 \quad U_2 \quad \ldots \quad U_M]$, where the p-th input column vector is $U_p = \left[u_{1p} \quad u_{2p} \quad \ldots \quad u_{kp}\right]'$. The squared error (unweighted) is calculated using the equation

$$D = \sum_{j=1}^{k} \left(a_{ji} - u_{jp}\right)^2$$

The weighted squared error is calculated using the equation

$$D = \sum_{j=1}^{k} w_j \left(a_{ji} - u_{jp}\right)^2$$

where the weighting factor is defined as $W = [w_1 \quad w_2 \quad \ldots \quad w_k]$. The index of the codeword that is associated with the minimum distortion is assigned to the input column vector.

You can select how you want to enter the codebook values using the **Source of codebook** parameter. When you select `Specify via dialog`, you can type the codebook values into the block parameters dialog box. Select `Input port` and port C appears on the block. The block uses the input to port C as the **Codebook** parameter.

The **Codebook** parameter is an k-by-N matrix of values, where $k \geq 1$ and $N \geq 1$. Each input column vector is compared to this codebook. Each column of the codebook matrix is a codeword, and each codeword has an index value. The first codeword vector corresponds to an index value of 0, the second codeword vector corresponds to an index value of 1, and so on. The codeword vectors must have the same number of rows as the input, U.

For the **Distortion measure** parameter, select `Squared error` when you want the block to calculate the distortion by evaluating the Euclidean distance between the input column vector and each codeword in the codebook. Select `Weighted squared error` when you want to use a weighting factor to emphasize or deemphasize certain input values.

For the **Source of weighting factor** parameter, select `Specify via dialog` to enter a weighting factor vector in the dialog box. Choose `Input port` to specify the weighting factor using port W.

Use the **Weighting factor** parameter to emphasize or deemphasize certain input values when calculating the distortion measure. For example, consider the p-th input column vector, U_p, as previously defined. When you want to neglect the effect of the first element of this vector, enter `[0 1 1 ...  1]` as the **Weighting factor** parameter. This weighting factor is used to calculate the weighted squared error using the equation

$$D = \sum_{j=1}^{k} w_j \left(a_{ji} - u_{jp}\right)^2$$

Because of the weighting factor used in this example, the weighted squared error is not affected by the first element of the input matrix. Therefore, the first element of the input column vector no longer impacts the choice of index value output by the Vector Quantizer Encoder block.

Use the **Index output data type** parameter to specify the data type of the index values output at port I. The data type of the index values can be `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`.

When an input vector is equidistant from two codewords, the block uses the **Tie-breaking rule** parameter to determine which index value the block chooses. When you want the input vector to be represented by the lower index valued codeword, select `Choose the lower index`. To represent the input column vector by the higher index valued codeword, select `Choose the higher index`.

Select the **Output codeword** check box to output at port Q(U) the codeword vectors that correspond to each index value. When the input is a matrix, the corresponding codeword vectors are horizontally concatenated into a matrix.

Select the **Output quantization error** check box to output at port D the quantization error that results when the block represents the input column vector by its nearest codeword. When the input is a matrix, the quantization error values are horizontally concatenated.

The Vector Quantizer Encoder block accepts real floating-point and fixed-point inputs. For more information on the data types accepted by each port, see "Data Type Support" on page 2-1484 or "Supported Data Types" on page 2-1491.

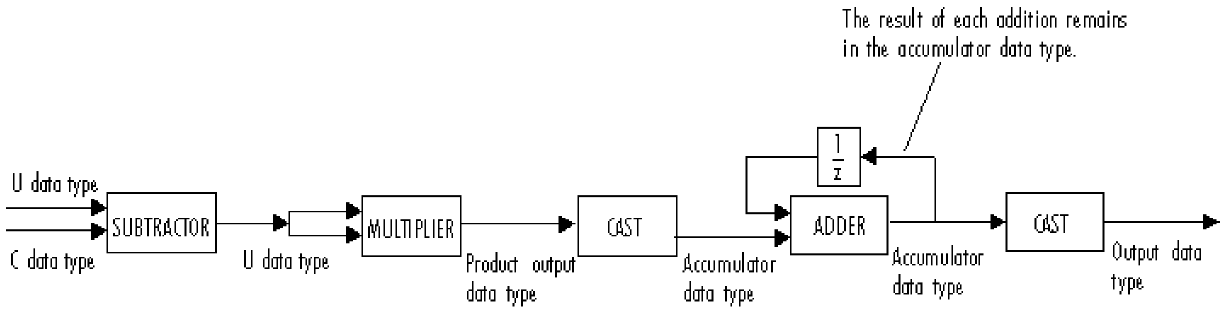
Data Type Support

The input data values, codebook values, and weighting factor values are input to the block at ports U, C, and W, respectively. The data type of the input data values, codebook values, and weighting factor values can be `double`, `single`, or Fixed-point. The input data, codebook values, and weighting factor must be the same data type.

The outputs of the block are the index values, output codewords, and quantization error. Use the **Index output data type** parameter to specify the data type of the index output from the block at port I. The data type of the index can be `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`. The data type of the output codewords and the quantization error can be `double`, `single`, or Fixed-point. The block assigns the data type of the output codewords and the quantization error based on the data type of the input data.

Fixed-Point Data Types

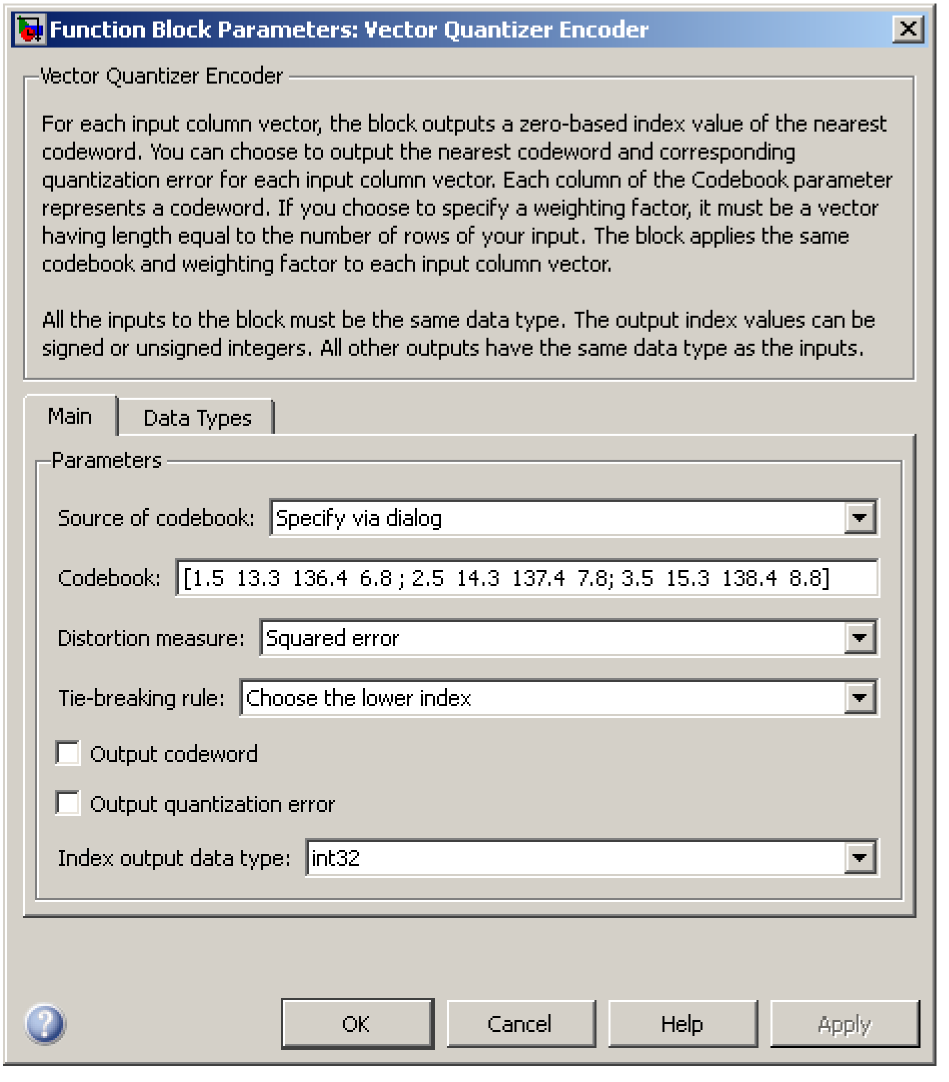
The following diagram shows the data types used within the Vector Quantizer Encoder block for fixed-point signals.



You can set the product output, accumulator, and index output data types in the block dialog as discussed below.

Dialog Box

The **Main** pane of the Vector Quantizer Encoder block dialog appears as follows.



Source of codebook
Choose `Specify via dialog` to type the codebook values into the block parameters dialog box. Select `Input port` to specify the codebook values using the block's input port, C.

Codebook
Enter a k-by-N matrix of values, where $1 \leq k$ and $1 \leq N$, to which your input column vector or matrix is compared. This parameter is visible if, from the **Source of codebook** list, you select `Specify via dialog`.

Distortion measure
Select `Squared error` when you want the block to calculate the distortion by evaluating the Euclidean distance between the input column vector and each codeword in the codebook. Select `Weighted squared error` when you want the block to calculate the distortion by evaluating a weighted Euclidean distance using a weighting factor to emphasize or deemphasize certain input values.

Source of weighting factor
Select `Specify via dialog` to enter a value for the weighting factor in the dialog box. Choose `Input port` and specify the weighting factor using port W on the block. This parameter is visible if, for the **Distortion measure** parameter, you select `Weighted squared error`.

Weighting factor
Enter a vector of values. This vector must have length equal to the number of rows of the input, U. This parameter is visible if, for the **Source of weighting factor** parameter, you select `Specify via dialog`.

Tie-breaking rule
Set this parameter to determine the behavior of the block when an input column vector is equidistant from two codewords. When you want the input column vector to be represented by the lower index valued codeword, select `Choose the lower index`. To represent

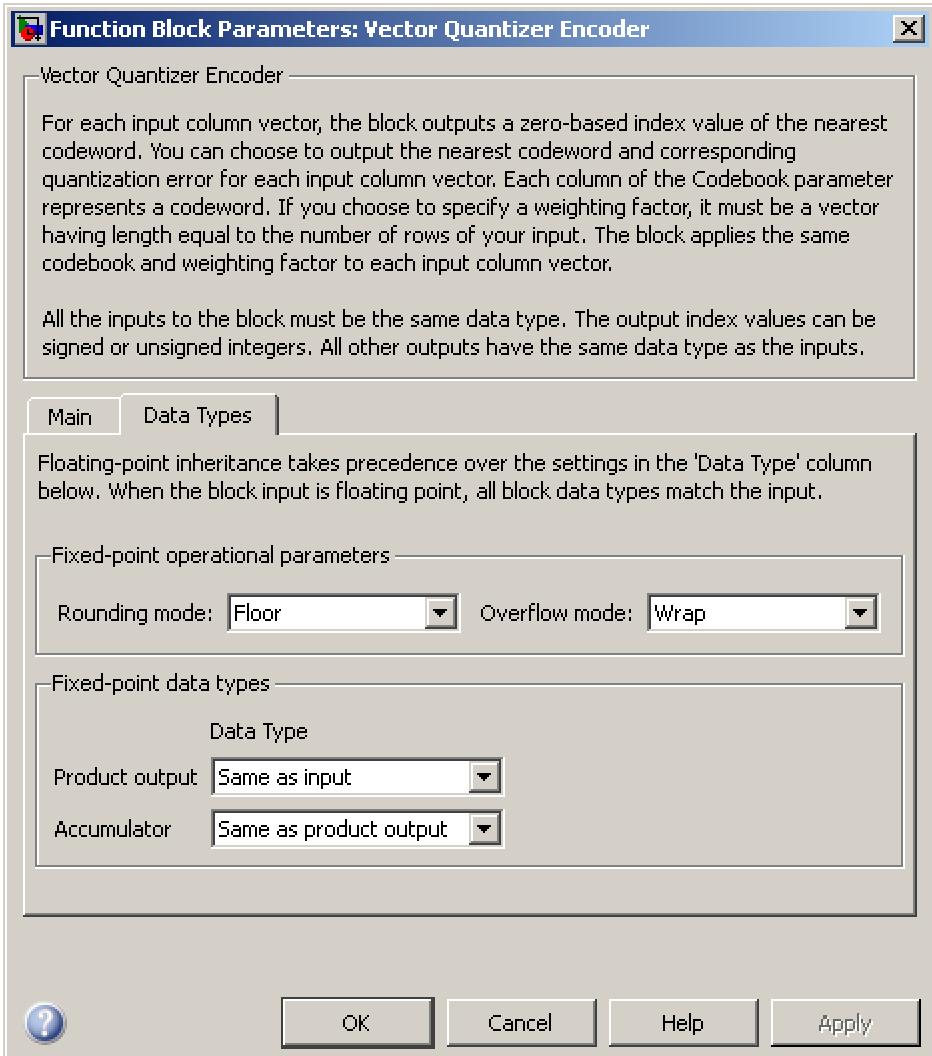
the input column vector by the higher index valued codeword, select `Choose the higher index`.

Output codeword
: Select this check box to output the codeword vectors nearest to the input column vectors.

Output quantization error
: Select this check box to output the quantization error value that results when the block represents the input column vector by the nearest codeword.

Index output data type
: Select `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32` as the data type of the index output at port I.

The **Data Types** pane of the Vector Quantizer Encoder block dialog appears as follows.



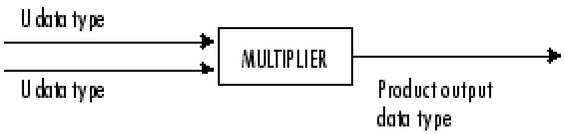
Rounding mode

Select the rounding mode for fixed-point operations.

Overflow mode

Select the overflow mode to be used when block inputs are fixed point.

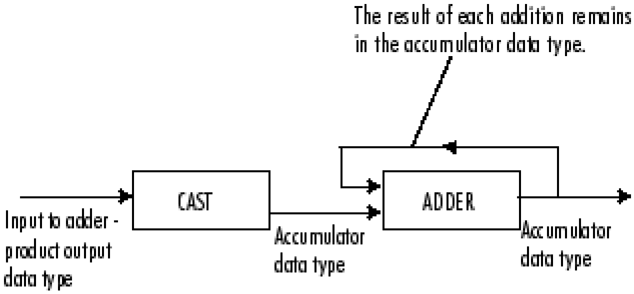
Product output



As depicted above, the output of the multiplier is placed into the product output data type and scaling. Use this parameter to specify how you would like to designate this product output word and fraction lengths.

- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and zero bias.

Accumulator



As depicted above, inputs to the accumulator are cast to the accumulator data type. The output of the adder remains in the accumulator data type as each element of the input is added to it. Use this parameter to specify how you would like to designate the accumulator word and fraction lengths.

- When you select `Same as product output`, these characteristics match those of the product output.
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the accumulator, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the accumulator. This block requires power-of-two slope and zero bias.

References

Gersho, A. and R. Gray. *Vector Quantization and Signal Compression*. Boston: Kluwer Academic Publishers, 1992.

Supported Data Types

Port	Supported Data Types
U	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
C	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

Port	Supported Data Types
W	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
I	• 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Q(U)	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
D	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers

See Also

Quantizer	Simulink
Scalar Quantizer Decoder	Signal Processing Blockset
Scalar Quantizer Design	Signal Processing Blockset
Uniform Encoder	Signal Processing Blockset
Uniform Decoder	Signal Processing Blockset
Vector Quantizer Decoder	Signal Processing Blockset

Purpose

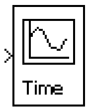
Display vector or matrix of time-domain, frequency-domain, or user-defined data

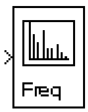
Library

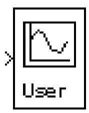
Signal Processing Sinks

`dspsnks4`

Description







The Vector Scope block is a comprehensive display tool similar to a digital oscilloscope. The block can display time-domain, frequency-domain, or user-defined signals. You can use the Vector Scope block to plot consecutive time samples from a frame-based vector, or to plot vectors containing data such as filter coefficients or spectral magnitudes. To compute and plot the periodogram of a signal with a single block, use the Spectrum Scope block.

The input to the Vector Scope block can be any real-valued M-by-N matrix, column or row vector, or 1-D (unoriented) vector, where 1-D vectors are treated as column vectors. Regardless of the input frame status, the block treats each column of an M-by-N input as an independent channel of data with M consecutive samples.

The block plots each sample of each input channel sequentially across the horizontal axis of the plot.

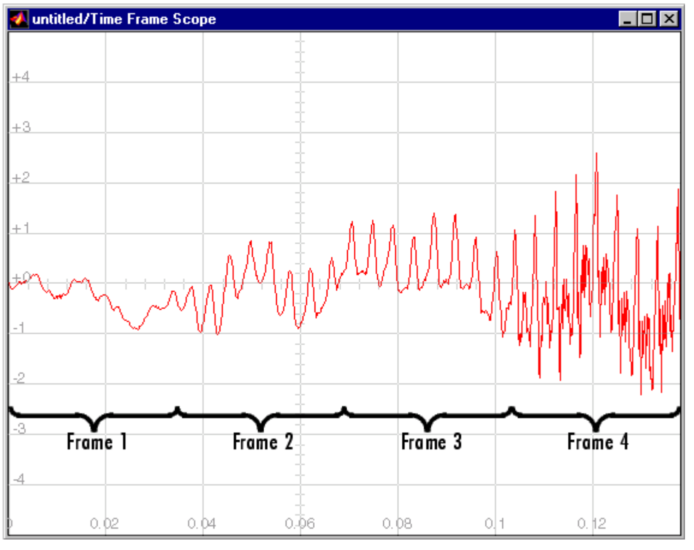
Scope Properties Pane

The **Scope Properties** pane enables you to plot time-domain, frequency-domain, or user-defined data, and adjust the horizontal display span of the plot. The scope displays frames of data, and updates the display for each new input frame.

The **Input domain** parameter specifies the domain of the input data. If you select `Time`, for M-by-N inputs containing time-domain data, the block treats each of the N input frames (columns) as a succession of M consecutive samples taken from a time series. That is, each data point in the input frame is assumed to correspond to a unique time value. Also, the **Time display span (number of frames)** parameter appears on the pane. Enter a scalar value greater than or equal to one that corresponds to the number of frames to be displayed across the width of the scope window.

If you select `Frequency` for the **Input domain** parameter, for *M*-by-*N* inputs containing frequency-domain data, the block treats each of the *N* input frames (columns) as a vector of spectral magnitude data corresponding to *M* consecutive ascending frequency indices. That is, when the input is a single column vector, u, each value in the input frame, `u(i)`, is assumed to correspond to a unique frequency value, `f(i)`, where `f(i+1)>f(i)`.

If you select `User-defined` for the **Input domain** parameter, the block does not assume that the input frame data is time-domain or frequency-domain data. You can plot the data in the appropriate manner. Also, the **Horizontal display span (number of frames)** parameter appears on the pane. Enter a scalar value greater than or equal to one that corresponds to the number of frames to be displayed across the width of the scope window.



Display Properties Pane

The **Display Properties** pane enables you to control how the block displays your data.

The **Show grid** parameter toggles the background grid on and off.

If you select the **Persistence** check box, the window maintains successive displays. That is, the scope does not erase the display after each frame (or collection of frames), but overlays successive input frames in the scope display.

If you select the **Frame number** check box, the block displays the number of the current frame in the input sequence on the scope window, and the block increments the count as each new input is received. Counting starts at 1 with the first input frame, and continues until the simulation stops.

If you select the **Channel legend** check box, a legend indicating the line color, style, and marker of each channel's data is added. When the input signal is labeled, that label appears in the channel legend. When the input signal is not labeled, but comes from a Concatenate block or a Mux block with labeled inputs, those labels appear in the channel legend. Otherwise, each channel in the legend is labeled with the channel number (`CH 1`, `CH 2`, etc.). Click and drag the legend to reposition it in the scope window; double-click on the line label to edit the text. If you rerun the simulation, the labels revert to the defaults.

If you select the **Compact display** check box, the scope completely fills the figure window. The scope does not display menus and axis titles, but it does show the numerical axis labels within the axes. If you clear the **Compact display** check box, the scope displays the axis labels and titles in a gray border surrounding the scope axes, and the window's menus and toolbar become visible.

If you select the **Open scope at start of simulation** check box, the scope opens at the start of the simulation. If you clear this parameter, the scope does not open automatically during the simulation. You can use this feature when you have several scope blocks in a model, and you do not want to view all the associated scopes during the simulation.

If you want to view a scope window that is not open during simulation, click **Open scope immediately** on the **Display Properties** pane of the desired Scope block.

The **Scope position** parameter specifies a four-element vector of the form

```
[left bottom width height]
```

specifying the position of the scope window on the screen, where (0,0) is the lower-left corner of the display. See the MATLAB `figure` function for more information.

Axis Properties Pane

The parameters that are available on the **Axis Properties** pane depend on the setting of the **Input domain** parameter on the **Scope Properties** pane.

Time Domain Inputs

When **Time display limits** is set to `Auto`, the block scales the horizontal axis of time-domain signals automatically. The range of the time axis is $[0, S * T_{fi}]$, where T_{fi} is the input frame period, and S is the **Time display span (number of frames)** parameter on the **Scope Properties** pane. The spacing between time points is $T_{fi}/(M-1)$, where M is the number of samples in each consecutive input frame.

When **Time display limits** is set to `User-defined`, the **Minimum X-limit (s)** and **Maximum X-limit (s)** parameters set the range of the horizontal axis.

Minimum Y-limit and **Maximum Y-limit** parameters set the range of the vertical axis. Setting these parameters is analogous to setting the `ymin` and `ymax` values of the MATLAB axis function.

The **Y-axis label** is the text displayed to the left of the y-axis.

Frequency Domain Inputs

The **Frequency units** parameter specifies whether the frequency axis values should be in units of Hertz or rad/sec. When the **Frequency units** parameter is set to `Hertz`, the spacing between frequency points is $1/(M * T_s)$, where T_s is the sample time of the original time-domain signal. When the **Frequency units** parameter is set to `rad/sec`, the spacing between frequency points is $2\pi/(M * T_s)$.

The **Frequency range** parameter specifies the range of frequencies over which the magnitudes in the input should be plotted. The available options are `[0..Fs/2]`, `[-Fs/2..Fs/2]`, and `[0..Fs]`, where `Fs` is the original time-domain signal's sample frequency. The Vector Scope block assumes that the input data spans the range $[0, F_s)$, which is the same as the output from an FFT. To plot over the range `[0..Fs/2]` the scope truncates the input vector, leaving only the first half of the data, then plots these remaining samples over half the frequency range. To plot over the range `[-Fs/2..Fs/2]`, the scope reorders the input vector elements such that the last half of the data becomes the first half, and vice versa; then it relabels the *x*-axis accordingly.

If you select the **Inherit sample time from input** check box for frequency domain inputs, the block scales the frequency axis by reconstructing the frequency data from the frame-period of the frequency-domain input. This is valid when the following conditions hold:

- Each frame of frequency-domain data shares the same length as the frame of time-domain data from which it was generated; for example, when the FFT is computed on the same number of points as are contained in the time-domain input.
- The sample period of the time-domain signal in the simulation is equal to the period with which the physical signal was originally sampled.

- Consecutive frames containing the time-domain signal do not overlap each other; that is, a particular signal sample does not appear in more than one sequential frame.

In cases where not all of these conditions hold, specify the appropriate value for the **Sample time of original time series** parameter.

When **Frequency display limits** is set to `Auto`, the block scales the horizontal axis of frequency-domain signals automatically. To do this, the Vector Scope block needs to know the sample period of the original time-domain sequence represented by the frequency-domain data. Specify this period by entering a value for the **Sample time of original time series** parameter.

When **Frequency display limits** is set to `User-defined`, the **Minimum frequency** and **Maximum frequency** parameters set the range of the horizontal axis.

The **Y-axis scaling** parameter allows you to select `Magnitude` or `dB` scaling along the *y*-axis.

Minimum Y-limit and **Maximum Y-limit** parameters set the range of the vertical axis. Setting these parameters is analogous to setting the `ymin` and `ymax` values of the MATLAB axis function.

The **Y-axis label** is the text displayed to the left of the *y*-axis.

User-Defined Inputs

If you select the **Inherit sample increment from input** check box for user-defined input domains, the block scales the horizontal axis by computing the horizontal interval between samples in the input frame from the frame period of the input. For example, when the input frame period is 1, and there are 64 samples per input frame, the interval between samples is computed to be 1/64. Computing the interval this way is usually only valid when the following conditions hold:

- The input is a nonoverlapping time series; the *x*-axis on the scope represents time.

- The input's sample period (1/64 in the above example) is equal to the period with which the physical signal was originally sampled.

In cases where not all of these conditions hold, use the **X display offset (samples)** and **Increment per sample in input** parameters.

The **X-axis title** is the text displayed below the *x*-axis.

When **X display limits** is set to `Auto`, the block scales the horizontal axis of user-defined domain signals automatically. To do this, the Vector Scope block needs to know the spacing of the input data. Specify this spacing using the **Increment per sample in input** parameter, I_s. This parameter represents the numerical interval between adjacent *x*-axis points corresponding to the input data. The range of the horizontal axis is $[0, M * I_s * S]$, where M is the number of samples in each consecutive input frame, and S is the **Horizontal display span (number of frames)** parameter that you specify in the **Scope Properties** pane.

When **X display limits** is set to `User-defined`, the **Minimum X-limit (samples)** and **Maximum X-limit (samples)** parameters set the range of the horizontal axis.

Minimum Y-limit and **Maximum Y-limit** parameters set the range of the vertical axis. Setting these parameters is analogous to setting the `ymin` and `ymax` values of the MATLAB `axis` function.

The **Y-axis label** is the text displayed to the left of the *y*-axis.

Line Properties Pane

Use the parameters on the **Line Properties** pane to help you distinguish between two or more independent channels of data on the scope.

The **Line visibilities** parameter specifies which channel's data is displayed on the scope, and which is hidden. The syntax specifies the visibilities in list form, where the term `on` or `off` as a list entry specifies the visibility of the corresponding channel's data. The list entries are separated by the pipe symbol, `|`.

For example, a five-channel signal would ordinarily generate five distinct plots on the scope. To disable plotting of the third and fifth lines, enter the following visibility specification in the **Line visibilities** parameter.

```
 on | on | off | on | off
ch 1  ch 2  ch 3  ch 4  ch 5
```

Note that the first (leftmost) list item corresponds to the first signal channel (leftmost column of the input matrix).

The **Line styles** parameter specifies the line style with which each channel's data is displayed on the scope. The syntax specifies the channel line styles in list form, with each list entry specifying a style for the corresponding channel's data. The list entries are separated by the pipe symbol, `|`.

For example, a five-channel signal would ordinarily generate all five plots with a solid line style. To plot each line with a different style, enter

```
  -  | -- | : | -. | -
ch 1  ch 2  ch 3 ch 4  ch 5
```

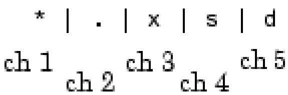
These settings plot the signal channels with the following styles.

Line Style	Command to Type in Line Style Parameter	Appearance
Solid	`-`	
Dashed	`--`	
Dotted	`:`	
Dash-dot	`-.`	
No line	`none`	No line appears

Note that the first (leftmost) list item, `'-'`, corresponds to the first signal channel (leftmost column of the input matrix). See the LineStyle property of the MATLAB `line` function for more information about the style syntax.

The **Line markers** parameter specifies the marker style with which each channel's samples are represented on the scope. The syntax specifies the channels' marker styles in list form, with each list entry specifying a marker for the corresponding channel's data. The list entries are separated by the pipe symbol, `|`.

For example, a five-channel signal would ordinarily generate all five plots with no marker symbol (that is, the individual sample points are not marked on the scope). To instead plot each line with a different marker style, you could enter



These settings plot the signal channels with the following styles.

Marker Style	Command to Type in Marker Style Parameter	Appearance
Asterisk	`*`	
Point	`.`	
Cross	`x`	

Marker Style	Command to Type in Marker Style Parameter	Appearance
Square	`s`	
Diamond	`d`	

Note that the leftmost list item, `'*'`, corresponds to the first signal channel or leftmost column of the input matrix. See the LineSpec property of the MATLAB `line` function for more information about the available markers.

To produce a stem plot for the data in a particular channel, type the word `stem` instead of one of the basic marker shapes.

The **Line colors** parameter specifies the color in which each channel's data is displayed on the scope. The syntax specifies the channel colors in list form, with each list entry specifying a color (in one of the MATLAB `ColorSpec` formats) for the corresponding channel's data. The list entries are separated by the pipe symbol, `|`.

For example, a five-channel signal would ordinarily generate all five plots in the color black. To instead plot the lines with the color order below, enter

```
[0 0 0] | [0 0 1] | [1 0 0 ] | [0 1 0] | [.7529 0 .7529]
 ch 1      ch 2      ch 3       ch 4         ch 5
```

or

```
 'k' | 'b' | 'r' | 'g' | [.7529 0 .7529]
ch 1  ch 2  ch 3  ch 4     ch 5
```

These settings plot the signal channels in the following colors (8-bit RGB equivalents shown in the center column).

Color	RGB Equivalent	Appearance
Black	(0,0,0)	
Blue	(0,0,255)	
Red	(255,0,0)	
Green	(0,255,0)	
Dark purple	(192,0,192)	

Note that the leftmost list item, `'k'`, corresponds to the first signal channel or leftmost column of the input matrix. See the MATLAB function `ColorSpec` for more information about the color syntax.

Vector Scope Window

The title that appears in the title bar of the scope window is the same as the block title. In addition to the standard MATLAB figure window menus such as **File**, **Window**, and **Help**, the Vector Scope window contains **View**, **Axes**, and **Channels** menus.

The options in the **View** menu allow you to zoom in and out of the scope window:

- To zoom in on the scope window, you must first select **View > Zoom In** or click the corresponding `Zoom In` toolbar button ( ). You can then zoom in by clicking in the center of your area of interest, or by clicking and dragging your cursor to draw a rectangular area of interest inside of the scope window.
- To zoom in on the *x*-axis of the scope window, you must first select **View > Zoom X**, or click the corresponding `Zoom X-Axis` toolbar

button ( ) on the scope window. You can then zoom in on the *x*-axis with a single click inside the scope window or by clicking and dragging the cursor along the *x*-axis over your area of interest.

- To zoom in on the *y*-axis of the scope window, you must first select **View > Zoom Y** or click the corresponding `Zoom Y-Axis` toolbar button ( ). You can then zoom in on the *y*-axis with a single click inside the scope window or by clicking and dragging the cursor along the *y*-axis over your area of interest.
- To return to the original view of the scope window, you have the following options:
  - Select **Full View** from the **View** menu .
  - Click the `Restore default view` toolbar button ( ) on the Vector Scope window.
  - Right-click inside the scope window, and select `Reset to Original View`.

Note To zoom out in smaller increments, you can right-click inside of the scope window and select `Zoom Out`. You can also zoom out by holding down the **Shift** key and clicking the left mouse button inside the scope window.

The parameters that you set using the **Axes** menu apply to all channels. Many of the parameters in this menu are also accessible through the block parameters dialog box. For descriptions of these parameters, see “Display Properties Pane” on page 2-1494. Below are descriptions of other parameters in the **Axes** menu:

- **Refresh** erases all data on the scope display, except for the most recent trace. This command is useful in conjunction with the **Persistence** setting.
- **Autoscale** resizes the *y*-axis to best fit the vertical range of the data.

Note The **Minimum Y-limit** and **Maximum Y-limit** parameters on the **Axis properties** pane of the block dialog are not updated to display the numerical limits selected by the autoscale feature. See "Zoom Capability for Spectrum Scope and Vector Scope Blocks" in the Signal Processing Blockset Release Notes for more information.

- **Save Axes Settings** allows you to save the current axes settings. When you select this option, the **Minimum Y-limit** and **Maximum Y-limit** parameters of the **Axes Properties** pane update with the current *y*-axes limits. The **Time display limits** (or **Frequency display limits**) parameter is set to `User-defined`, and the current *x*-axes limits are saved in the **Minimum X-limit** and **Maximum X-limit** (or **Minimum Frequency** and **Maximum Frequency**) parameters. To save these axes settings for your next MATLAB session, you need to resave your model.
- **Save Scope Position** updates the **Scope position** parameter on the **Display Properties** pane of the block dialog to reflect the scope window's current position and size. To make the scope window open at a particular location on the screen when the simulation runs, drag the window to the desired location, resize it, and select **Save Scope Position** from the **Axes** menu.

The properties listed in the **Channels** menu apply to a particular channel. All of the parameters in this menu are also accessible through the block parameters dialog box. For descriptions of these parameters, see "Line Properties Pane" on page 2-1499.

Many of these options can also be accessed by right-clicking with the mouse anywhere on the scope display. The menu that is displayed contains a combination of the options available in the **View**, **Axes** and **Channels** menus.

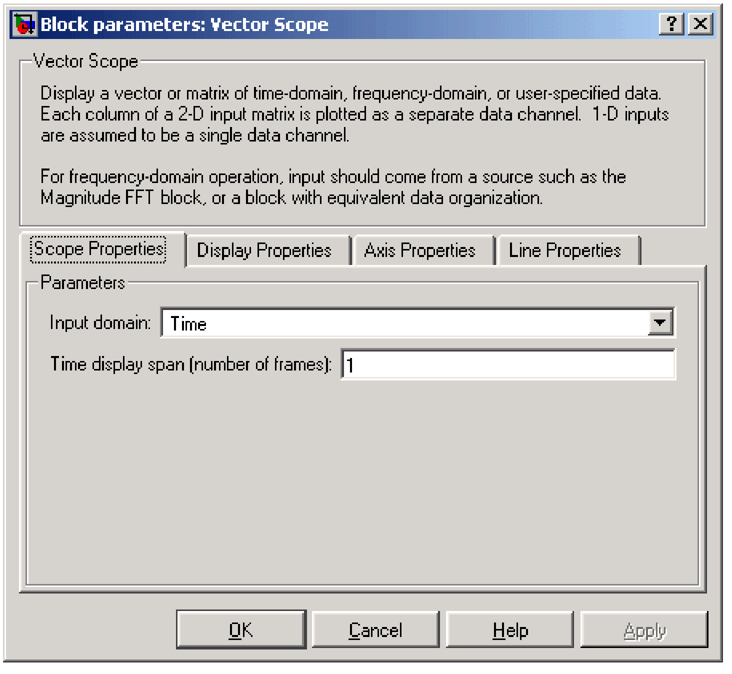
Note When you select **Compact Display** from the **Axes** menu, the scope window menus are no longer visible. Right-click in the Vector Scope window and click **Compact Display** in order to make the menus reappear.

Examples

See “Displaying Time Domain Data in the Vector Scope” in the *Signal Processing Blockset User’s Guide*.

Dialog Box

Scope Properties Pane



Input domain
Select the domain of the input. Your choices are `Time`, `Frequency`, or `User-defined`. Tunable.

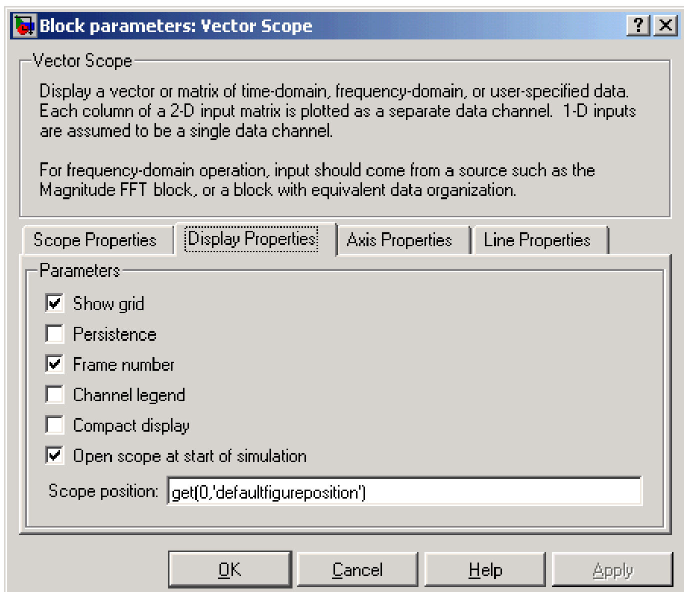
Time display span (number of frames)

The number of consecutive frames to display (horizontally) on the scope at any one time. This parameter is visible when the **Input domain** parameter is set to `Time`.

Horizontal display span (number of frames)

The number of consecutive frames to display (horizontally) on the scope at any one time. This parameter is visible when the **Input domain** parameter is set to `User-defined`.

Display Properties Pane



Show grid

Toggle the scope grid on and off. Tunable.

Persistence

Select this check box to maintain successive displays. That is, the scope does not erase the display after each frame (or collection of frames), but overlays successive input frames in the scope display. Tunable.

Frame number
If you select this check box, the number of the current frame in the input sequence appears in the Vector Scope window. Tunable.

Channel legend
Toggles the legend on and off. Tunable.

Compact display
Resizes the scope to fill the window. Tunable.

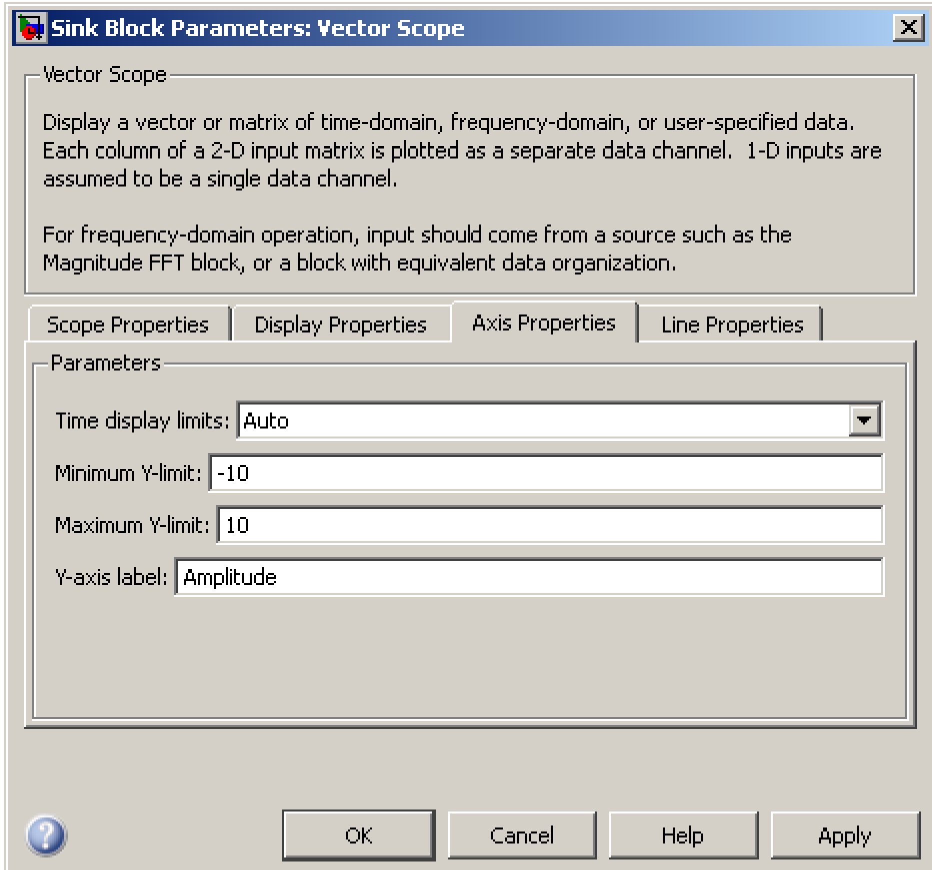
Open scope at start of simulation
Select this check box to open the scope at the start of the simulation. When this parameter is cleared, the scope does not open automatically during the simulation. Tunable.

Open scope immediately
If the scope is not open during simulation, select this check box to open it. This parameter is visible only while the simulation is running.

Scope position
A four-element vector of the form `[left bottom width height]` specifying the position of the scope window. `(0,0)` is the lower-left corner of the display. Tunable.

Axis Properties Pane

The parameters that are available on the **Axis Properties** pane depend on the setting of the **Input domain** parameter on the **Scope Properties** pane. When `Time` is selected for the **Input domain** parameter, the following parameters are available on the **Axis Properties** pane:



Time display limits

Select `Auto` to have the limits of the *x*-axis set for you automatically, or `User-defined` to set the limits yourself in the **Minimum X-limit (s)** and **Maximum X-limit (s)** parameters.

Minimum X-limit (s)

Specify the minimum value of the *x*-axis in seconds. This parameter is only visible if the **Time display limits** parameter is set to `User-defined`. Tunable.

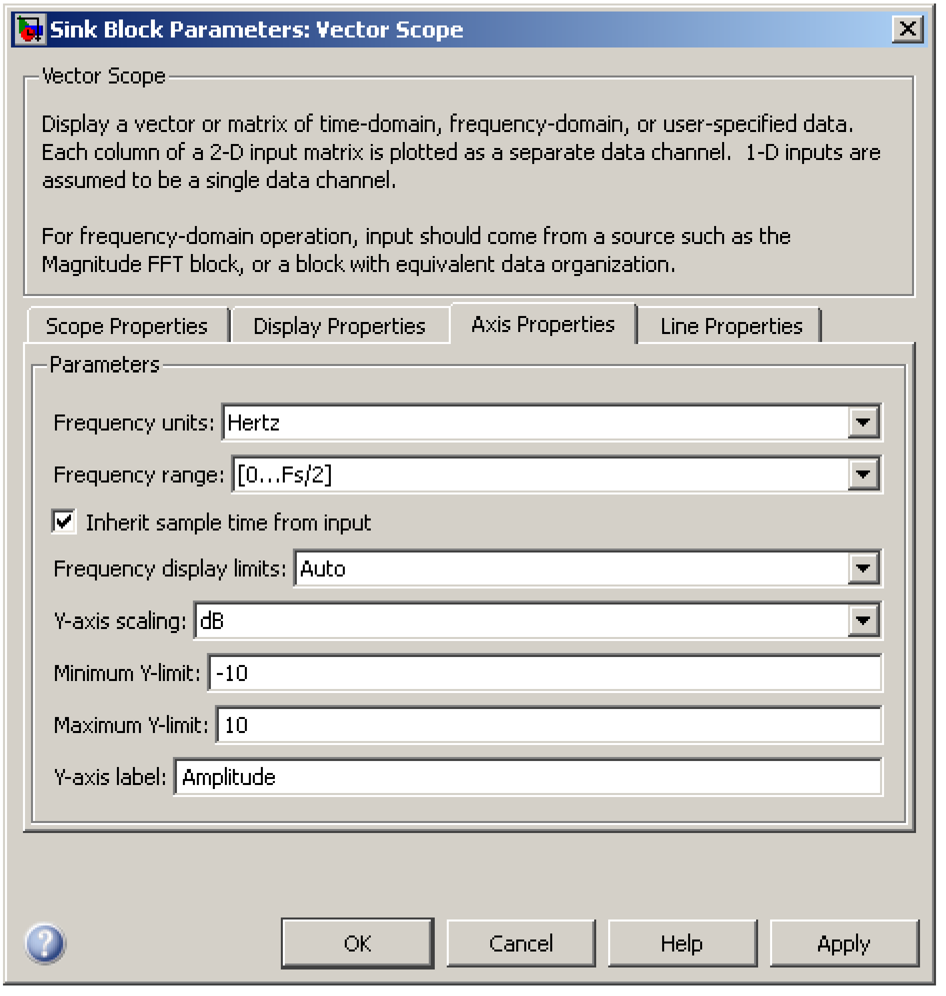
Maximum X-limit (s)
Specify the maximum value of the *x*-axis in seconds. This parameter is only visible if the **Time display limits** parameter is set to `User-defined`. Tunable.

Minimum Y-limit
Specify the minimum value of the *y*-axis. Setting this parameter is analogous to setting the `ymin` value of the MATLAB `axis` function. Tunable.

Maximum Y-limit
Specify the maximum value of the *y*-axis. Setting this parameter is analogous to setting the `ymax` value of the MATLAB `axis` function.Tunable.

Y-axis label
Specify text to be displayed to the left of the *y*-axis. Tunable.

When `Frequency` is selected for the **Input domain** parameter, the following parameters are available on the **Axis Properties** pane:



Frequency units

Choose the frequency units for the *x*-axis, `Hertz` or `rad/sec`. Tunable.

Frequency range
Specify the frequency range over which to plot the data. Tunable.

Inherit sample time from input
If you select this check box, the block computes the time-domain sample period from the frame period and frame size of the frequency-domain input. Use this parameter only when the length of the each frame of frequency-domain data is the same as the length of the frame of time-domain data from which it was generated. Tunable.

Sample time of original time series
Enter the sample period, T_s, of the original time-domain signal. This parameter is only visible when the **Inherit sample time from input** check box is not selected. Tunable.

Frequency display limits
Select `Auto` to have the limits of the *x*-axis set for you automatically, or `User-defined` to set the limits yourself in the **Minimum frequency** and **Maximum frequency** parameters.

Minimum frequency
Specify the minimum frequency value of the *x*-axis in Hertz or rad/sec. This parameter is only visible if the **Frequency display limits** parameter is set to `User-defined`. Tunable.

Maximum frequency
Specify the maximum frequency value of the *x*-axis in Hertz or rad/sec. This parameter is only visible if the **Frequency display limits** parameter is set to `User-defined`. Tunable.

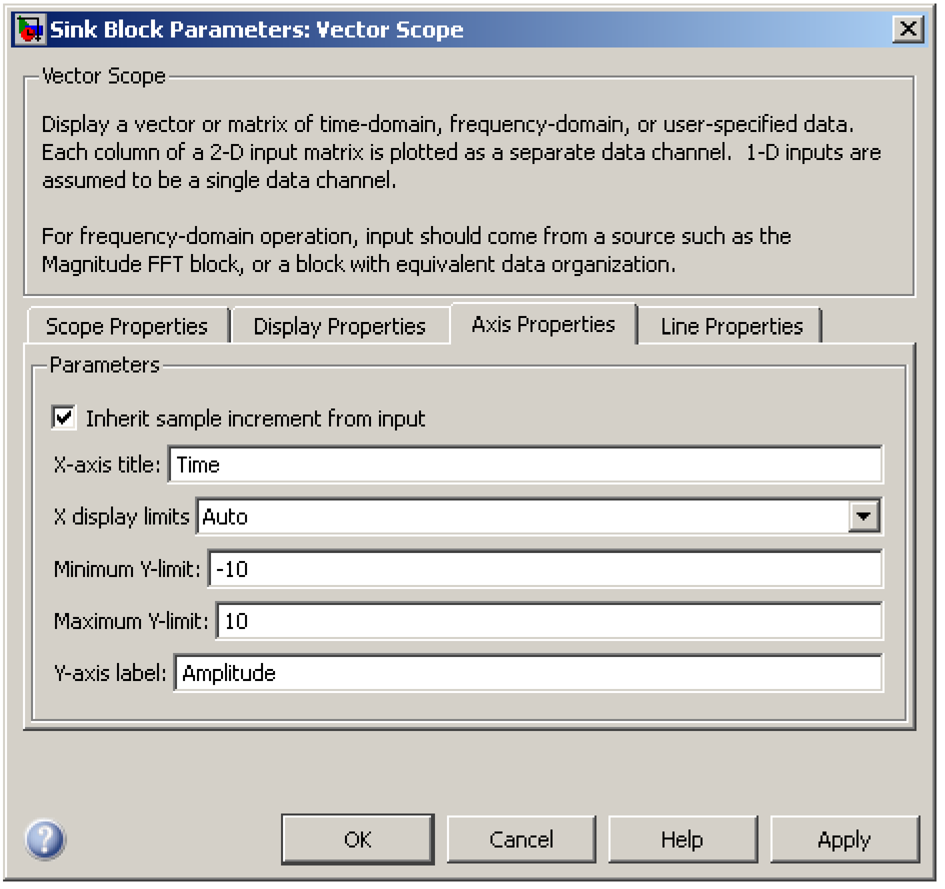
Y-axis scaling
Choose either `dB` (decibel) or `Magnitude` scaling for the *y*-axis. Tunable.

Minimum Y-limit
Specify the minimum value of the *y*-axis. Setting this parameter is analogous to setting the `ymin` value of the MATLAB `axis` function. Tunable.

Maximum Y-limit
: Specify the maximum value of the *y*-axis. Setting this parameter is analogous to setting the `ymax` value of the MATLAB `axis` function. Tunable.

Y-axis label
: Specify text to be displayed to the left of the *y*-axis. Tunable.

When `User-defined` is selected for the **Input domain** parameter, the following parameters are available on the **Axis Properties** pane:



Inherit sample increment from input

When you select this check box, the block scales the horizontal axis by computing the horizontal interval between samples in the input frame from the frame period of the input. Use this parameter only when the input's sample period is equal to the period with which the physical signal was originally sampled. Tunable.

X display offset (samples)
Specify an offset for the *x*-axis display in samples. This parameter is only visible when the **Inherit sample increment from input** check box is not selected. Tunable.

Increment per sample in input
Enter the numerical interval between adjacent *x*-axis points corresponding to the user-defined input data. This parameter is only visible when the **Inherit sample increment from input** check box is not selected. Tunable.

X-axis title
Enter the text to be displayed below the *x*-axis. Tunable.

X display limits
Select `Auto` to have the limits of the *x*-axis set for you automatically, or `User-defined` to set the limits yourself in the **Minimum X-limit (samples)** and **Maximum X-limit (samples)** parameters.

Minimum X-limit (samples)
Specify the minimum value of the *x*-axis in samples. This parameter is only visible if the **X display limits** parameter is set to `User-defined`. Tunable.

Maximum X-limit (samples)
Specify the maximum value of the *x*-axis in samples. This parameter is only visible if the **X display limits** parameter is set to `User-defined`. Tunable.

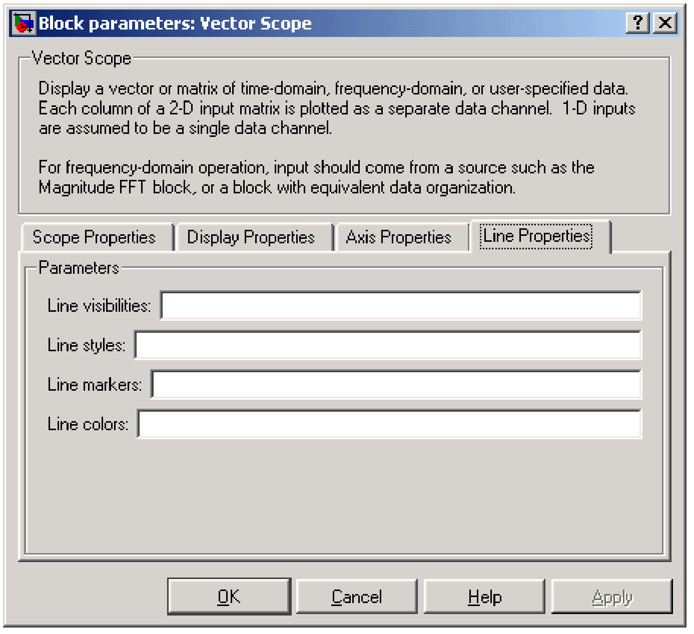
Minimum Y-limit
Specify the minimum value of the *y*-axis. Setting this parameter is analogous to setting the `ymin` value of the MATLAB `axis` function. Tunable.

Maximum Y-limit
Specify the maximum value of the *y*-axis. Setting this parameter is analogous to setting the `ymax` value of the MATLAB `axis` function.Tunable.

Y-axis label

Specify text to be displayed to the left of the *y*-axis. Tunable.

Line Properties Pane



Line visibilities

Enter `on` or `off` to specify the visibility of the various channels' scope traces. Separate your choices for each channel with by a pipe (`|`) symbol. Tunable.

Line styles

Enter the line styles of the various channels' scope traces. Separate your choices for each channel with by a pipe (`|`) symbol. Tunable.

Line markers

Enter the line markers of the various channels' scope traces. Separate your choices for each channel with by a pipe (`|`) symbol. Tunable.

Line colors

Enter the colors of the various channels' scope traces using the `ColorSpec` formats. Separate your choices for each channel with by a pipe (`|`) symbol. Tunable.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • Boolean • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

See Also

Matrix Viewer	Signal Processing Blockset
Spectrum Scope	Signal Processing Blockset

Waterfall

Purpose View vectors of data over time

Library Signal Processing Sinks

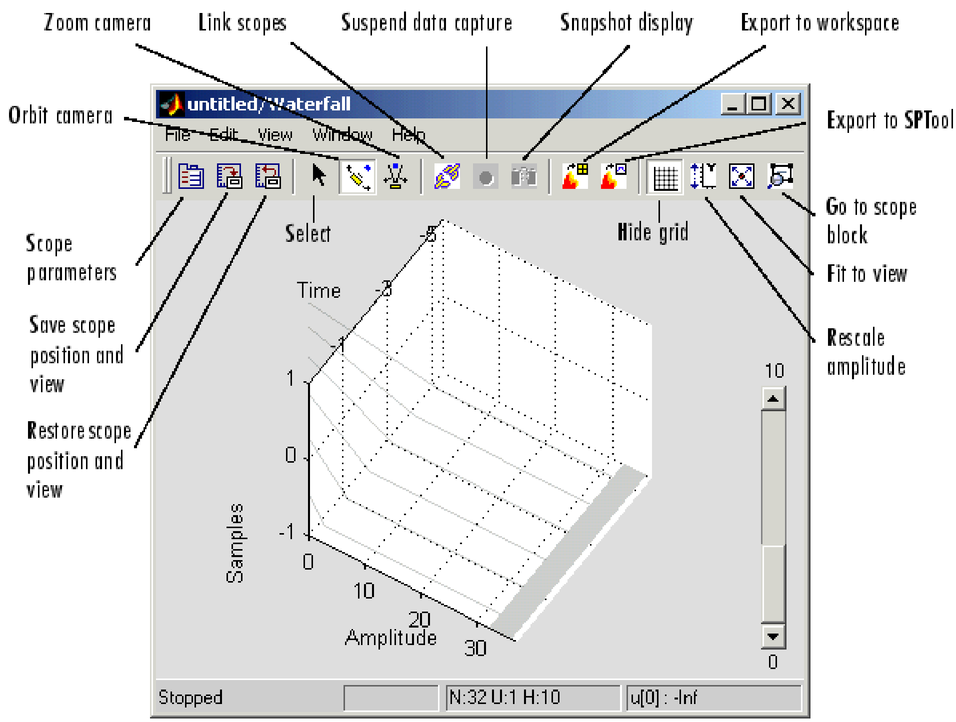
`dspsnks4`

Description

Waterfall

The Waterfall block displays multiple vectors of data at one time. These vectors represent the input data at consecutive sample times. The input to the block can be real or complex-valued data vectors of any data type including fixed-point data types. However, the input is converted to double-precision before the block processes the data. The Waterfall block displays only real-valued, double-precision vectors of data.

The data is displayed in a three-dimensional axis in the Waterfall window. By default, the x-axis represents amplitude, the y-axis represents samples, and the z-axis represents time. You can adjust the number of sample vectors that the block displays, move and resize the Waterfall window, and modify block parameter values during the simulation. The Waterfall window has toolbar buttons that enable you to zoom in on displayed data, suspend data capture, freeze the scope's display, save the scope position, and export data to the workspace. The toolbar buttons are labeled in the following figure, which shows the Waterfall window as it appears when you double-click a Waterfall block.



Sections of This Reference Page

- "Waterfall Parameters" on page 2-1520
- "Display Parameters" on page 2-1521
- "Axes Parameters" on page 2-1522
- "Data History Parameters" on page 2-1523
- "Triggering Parameters" on page 2-1524
- "Scope Trigger Function" on page 2-1527
- "Transform Parameters" on page 2-1530
- "Scope Transform Function" on page 2-1532

- “Examples” on page 2-1532

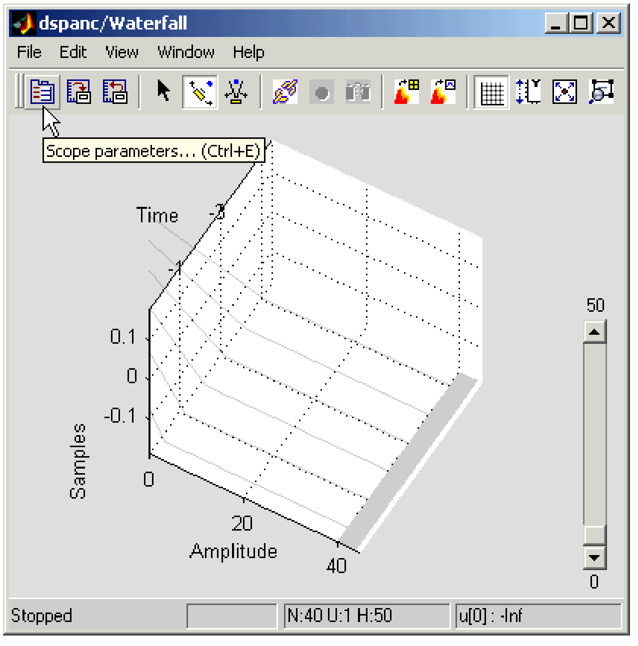
Waterfall Parameters

You can control the display and behavior of the Waterfall window using the Parameters dialog box.

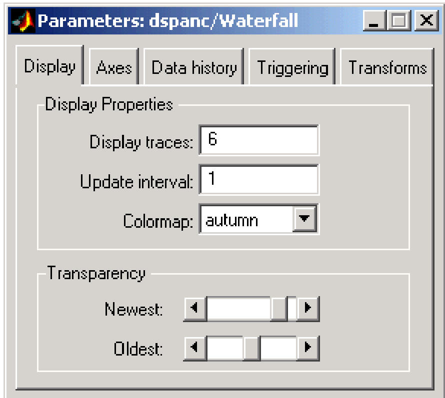
Note You can alter the Waterfall parameters while the simulation is running. However, when you make changes to values in text boxes, you must click **Enter** or click outside the text box before the block accepts your changes.

---

1 To open the Parameters dialog box, click the **Scope parameters** button.



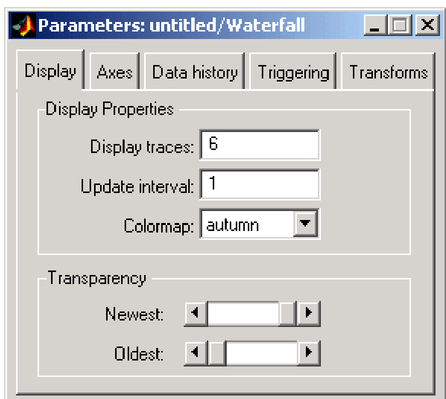
The Parameters dialog box appears.



2 Click on the different panes to enter parameter settings.

Display Parameters

The following parameters control the Waterfall window's display.



Display traces

Enter the number of vectors of data to be displayed in the Waterfall window.

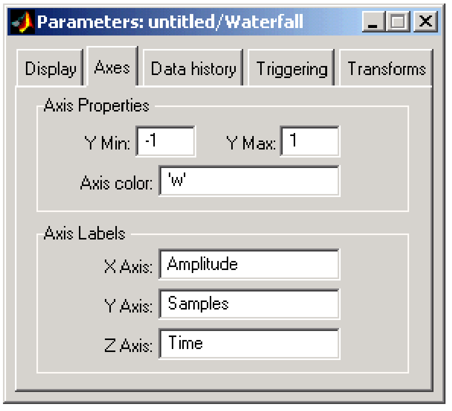
Update interval
Enter the number of vectors the block should store before it displays them to the window.

Colormap
Choose a colormap for the displayed data.

Transparency
Specify the transparency of the newest and oldest data vectors. Placing the slider in the left-most position tells the block to make the data vector transparent. Placing the slider in the right-most position tells the block to make the data vector opaque. The intermediate data vectors transition between the two chosen transparency values.

Axes Parameters

The following parameters control the axes in the Waterfall window.



Y Min
Enter the minimum value of the y-axis.

Y Max
Enter the maximum value of the y-axis.

Axis color

Enter a background color for the axes. Specify the color using a character string. For example, to specify black, enter `'k'`.

X Axis

Enter the *x*-axis label.

Y Axis

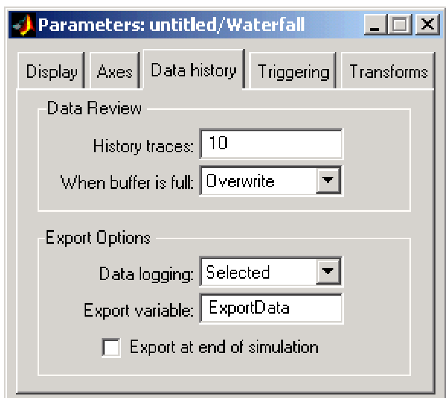
Enter the *y*-axis label.

Z Axis

Enter the *z*-axis label.

Data History Parameters

The following parameters control how many input data vectors the Waterfall block stores. They also control how the data is exported to the MATLAB workspace or SPTool.



History traces

Enter the number of vectors (traces) that you want the block to store.

When the buffer is full

Use this parameter to control the behavior of the block when the buffer is filled:

- `Overwrite` — The old data is replaced with the new data.
- `Suspend` — The block stops storing data in the buffer; however, the simulation continues to run.
- `Extend` — The block extends the buffer so that it can continue to store all the input data.

Data logging
Use this parameter to control which data is exported from the block:

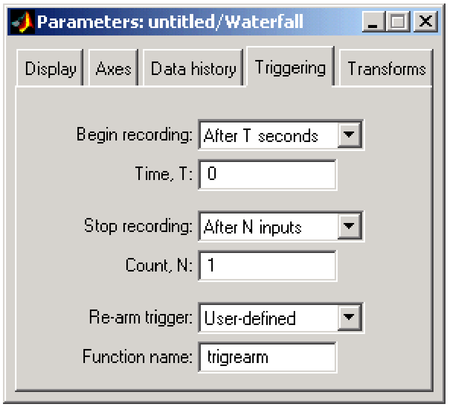
- `Selected` — The selected data vector is exported.
- `All visible` — All of the data vectors displayed in the Waterfall window are exported.
- `All history` — All of the data vectors stored in the block's history buffer are exported.

Export variable
Enter the name of the variable that represents your data in the MATLAB workspace or SPTool. The default variable name is `ExportData`.

Export at end of simulation
Select this check box to automatically export the data to the MATLAB workspace when the simulation stops.

Triggering Parameters

The following parameters control when the Waterfall block starts and stops capturing data.



Begin recording

This parameter controls when the Waterfall block starts capturing data:

- `Immediately` — The Waterfall window captures the input data as soon as the simulation starts.
- `After T seconds` — The **Time, T** parameter appears in the dialog box. Enter the number of seconds the block should wait before it begins capturing data.
- `After N inputs` — The **Count, N** parameter appears in the dialog box. Enter the number of inputs the block should receive before it begins capturing data.
- `User-defined` — The **Function name** parameter appears in the dialog box. Enter the name of a MATLAB function that defines when the block should begin capturing data. For more information about how you define this function, see "Scope Trigger Function" on page 2-1527.

Stop recording

This parameter controls when the Waterfall block stops capturing data:

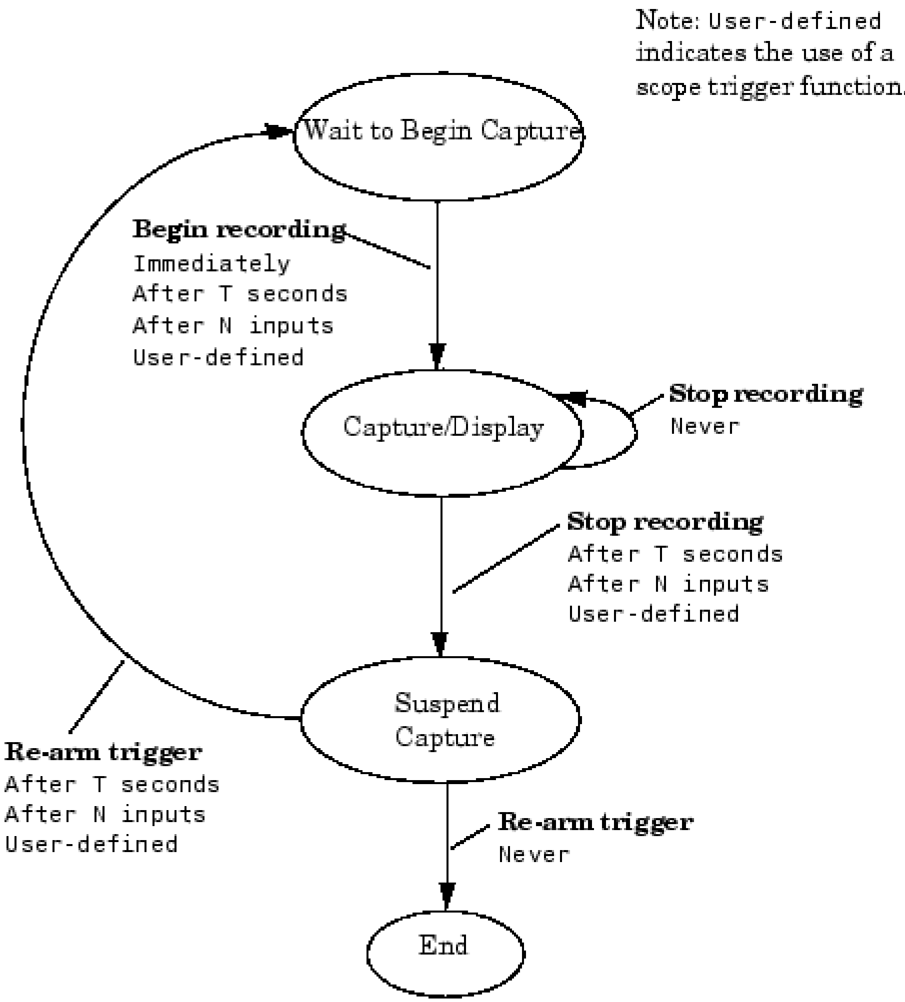
- `Never` — The block captures the input data as long as the simulation is running.
- `After T seconds` — The **Time, T** parameter appears in the dialog box. Enter the number of seconds the block should wait before it stops capturing data.
- `After N inputs` —The **Count, N** parameter appears in the dialog box. Enter the number of inputs the block should receive before it stops capturing data.
- `User-defined` — The **Function name** parameter appears in the dialog box. Enter the name of a MATLAB function that defines when the block should stop capturing data. For more information about how you define this function, see "Scope Trigger Function" on page 2-1527.

Re-arm trigger

This parameter controls when the Waterfall block begins waiting to capture data. It is available only when you select `After T seconds`, `After N inputs`, or `User-defined` for the **Stop recording** parameter:

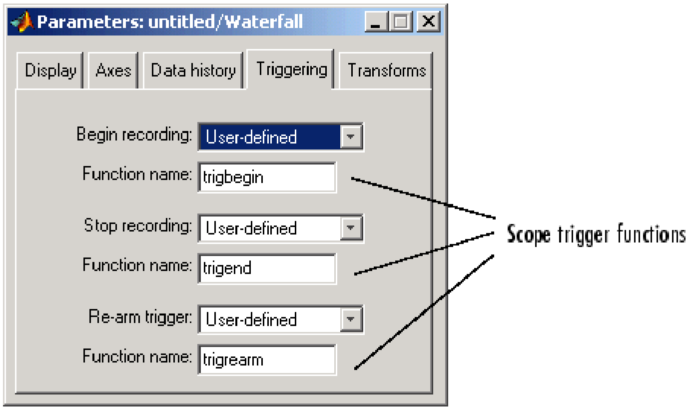
- `Never` — The Waterfall Scope block starts and stops capturing data as defined by the **Begin recording** and **Stop recording** parameters.
- `After T seconds` — The **Time, T** parameter appears in the dialog box. Enter the number of seconds the block should wait before it begins waiting to capture data.
- `After N inputs` — The **Count, N** parameter appears in the dialog box. Enter the number of inputs the block should receive before it begins waiting to capture data.
- `User-defined` — The **Function name** parameter appears in the dialog box. Enter the name of a MATLAB function that defines when the block should begin waiting to capture data. For more information about how you define this function, see "Scope Trigger Function" on page 2-1527.

The triggering process is illustrated in the state diagram below.



Scope Trigger Function

You can create custom scope trigger functions to control when the scope starts, stops, or begins waiting to capture data.



These functions must be valid MATLAB functions and be located either in the current folder or on the MATLAB path.

Each scope trigger function must have the following form

```
y = functionname(blk,t,u),
```

where `functionname` refers to the name you give your scope trigger function. The variable `blk` is the Simulink block handle. When the scope trigger function is called by the block, Simulink automatically populates this variable with the handle of the Waterfall block. The variable `t` is the current simulation time, represented by a real, double-precision, scalar value. The variable `u` is the vector input to the block. The output of the scope trigger function, `y`, is interpreted as a logical signal. It is either true or false:

- Begin recording scope trigger function
  - When the output of this scope trigger function is true, the Waterfall block starts capturing data.
  - When the output is false, the block remains in its current state.
- Stop recording scope trigger function
  - When the output of this scope trigger function is true, the block stops capturing data.

  - When the output is false, the block remains in its current state.
- Re-arm trigger scope trigger function
  - When the output of this scope trigger function is true, the block waits for a begin recording event.
  - When the output is false, the block remains in its current state.

Note The Waterfall block passes its input data directly to the scope trigger functions. These functions do not use the transformed data defined by the Transform parameters.

The following is an example of a scope trigger function. This function, called `trigPower` detects when the energy in u exceeds a certain threshold.

```
function y = trigPower(blk, t, u)

y = (u'*u > 2300);
```

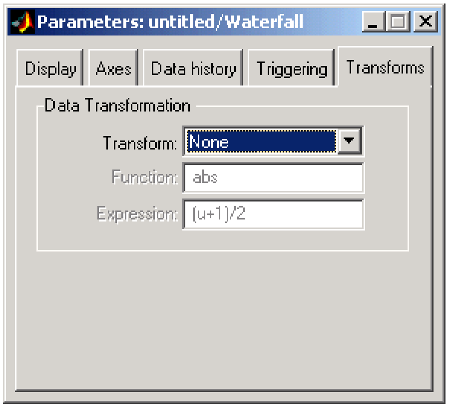
The following is another example of a scope trigger function. This function, called `count3`, triggers the scope once three vectors with positive means are input to the block. Then, the function resets itself and begins searching for the next three input vectors with positive means. This scope trigger function is valid only when one Waterfall block is present in your model.

```
function y = count3(blk, t, u)

persistent state;
if isempty(state); state = 0; end
if mean(u)>0; state = state+1; end
y = (state>=3);
if y; state = 0; end
```

Transform Parameters

The following parameters transform the input data to the Waterfall block. The result of the transform is displayed in the Waterfall window.



Note The block assumes that the input to the block corresponds to the **Transform** parameter you select. For example, when you choose `Complex-> Angle`, the block assumes that the input is complex. The block does not produce an error when the input is not complex. Therefore, you must verify the format of your input data to guarantee that a meaningful result is displayed in the Waterfall window.

Transform

Choose a transform that you would like to apply to the input of the Waterfall block:

- `None` — The input is displayed as it is received by the block.
- `Amplitude-> dB` — The block converts the input amplitude into decibels.
- `Complex-> Mag Lin` — The block converts the complex input into linear magnitude.

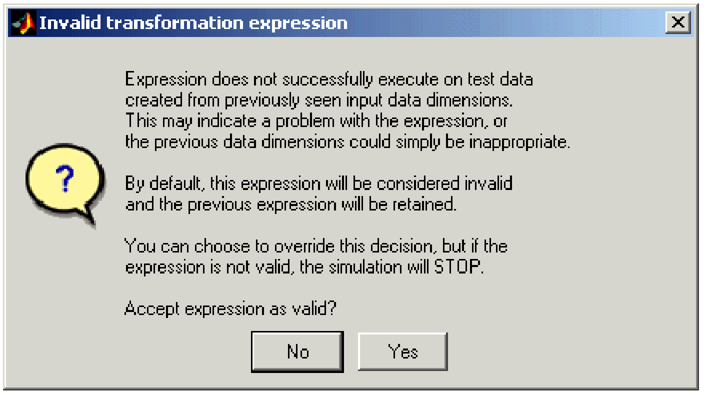
- `Complex-> Mag dB` — The block converts the complex input into magnitude in decibels.
- `Complex-> Angle` — The block converts the complex input into phase.
- `FFT-> Mag Lin Fs/2` — The block takes the linear magnitude of the FFT input and plots it from 0 to the Nyquist frequency.
- `FFT-> Mag dB Fs/2` — The block takes the magnitude of the FFT input, converts it to decibels, and plots it from 0 to the Nyquist frequency.
- `FFT-> Angle Fs/2` — The block converts the FFT input into phase and plots it from 0 to the Nyquist frequency.
- `Power-> dB` — The block converts the input power into decibels.

Function

This parameter is only available when you select `User-defined fcn` for the **Transform** parameter. Enter a function that you would like to apply to the input of the Waterfall block. For more information about how you define this function, see “Scope Transform Function” on page 2-1532.

Expression

This parameter is only available when you select `User-defined expr` for the **Transform** parameter. Enter an expression that you would like to apply to the input of the Waterfall block. The result of this expression must be real-valued. When you write the expression, be sure to include only one unknown variable. The block assumes this unknown variable represents the input to the block. When the block believes your expression is invalid, the following window appears.



When you click **No**, your expression is not applied to the input. When you click **Yes** and your expression is invalid, your simulation stops and Simulink displays an error.

Scope Transform Function

You can create a scope transform function to control how the Waterfall block transforms your input data. This function must have a valid MATLAB function name and be located either in the current folder or on the MATLAB path.

Your scope transform function must have the following form

```
y = functionname(u),
```

where `functionname` refers to the name you give your function. The variable `u` is the real or complex vector input to the block. The output of the scope transform function, `y`, must be a double-precision, real-valued vector. When it is not, the simulation stops and Simulink displays an error. Note that the output vector does not need to be the same size as the input vector.

Examples

The following examples illustrate some capabilities of the Waterfall block.

- “Exporting Data” on page 2-1533

Exporting Data

You can use the Waterfall block to export data to the MATLAB workspace or to SPTool:

1 Open and run the `dspanc` demo.

2 While the simulation is running, click the **Export to Workspace** button.

3 Type `whos` at the MATLAB command line.

   The variable `ExportData` appears in your MATLAB workspace. `ExportData` is a 40-by-6 matrix. This matrix represents the six data vectors that were present in the Waterfall window at the time you clicked the **Export to Workspace** button. Each column of this matrix contains 40 filter coefficients. The columns of data were captured at six consecutive instants in time.

   You can control what data is exported using the **Data logging** parameter in **Data history** pane of the Parameters dialog box. For more information, see “Data History Parameters” on page 2-1523.

4 While the simulation is running, click the **Export to SPTool** button.

   The SPTool GUI opens and the variable `ExportData` is displayed in the **Signals** list.

For more information about SPTool, see the Signal Processing Toolbox documentation.

Capturing Data

You can use the Waterfall block to interact with your data while it is being captured:

1. Open and run the `dspanc` demo.

2. While the simulation is running, click the **Suspend data capture** button.

   The Waterfall block no longer captures or displays the data coming from the Downsample block.

3. To continue capturing data, click the **Resume data capture** button.

4. To freeze the data display while continuing to capture data, click the **Snapshot display** button.

5. To view the Waterfall block that the data is coming from, click the **Go to scope block** button.

   In the Simulink model window, the Waterfall block that corresponds to the active Waterfall window flashes. This feature is helpful when you have more than one Waterfall block in a model and you want to clarify which data is being displayed.

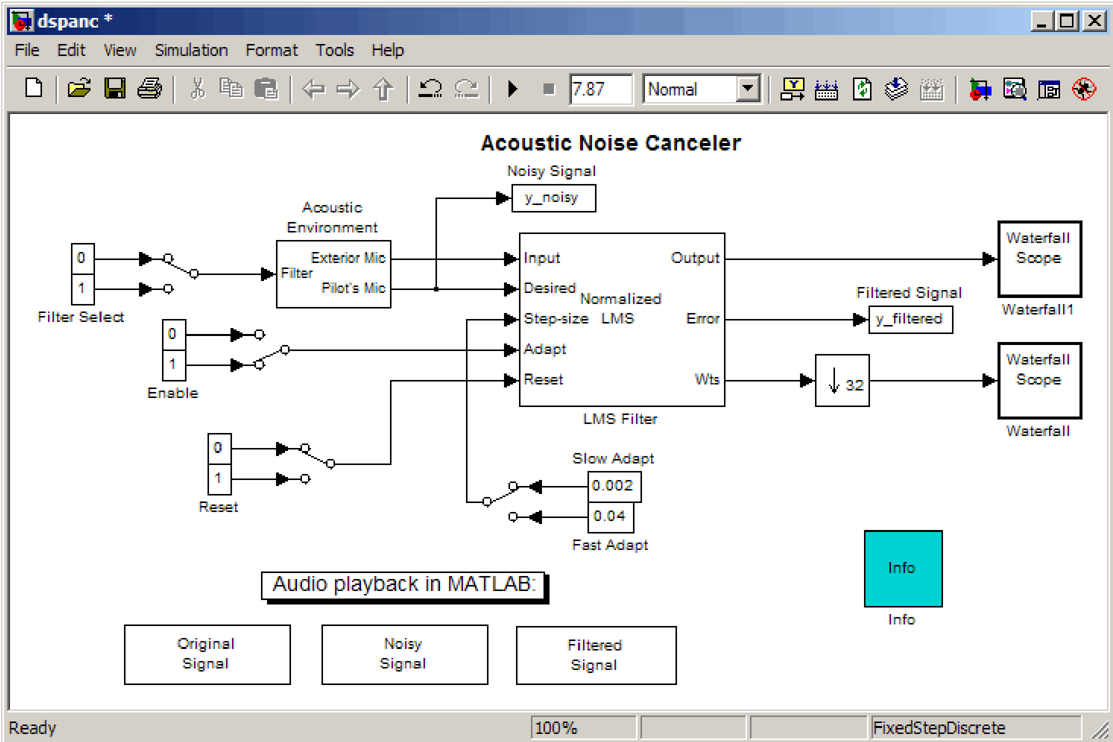
Linking Scopes

You can link several Waterfall blocks together in order to capture the effect of a model event in all of the Waterfall windows in the model:

1. Open the `dspanc` demo.

2. Drag a second Waterfall block into the demo model.

3. Connect this block to the Output port of the LMS Filter block as shown in the figure below.



4 Run the model and view the model behavior in both Waterfall windows.

5 In the dspanc/Waterfall window, click the **Link scopes** button.

6 In the same window, click the **Suspend data capture** button.

The data capture is suspended in both scope windows.

7 Click the **Resume data capture** button.

The data capture resumes in both scope windows.

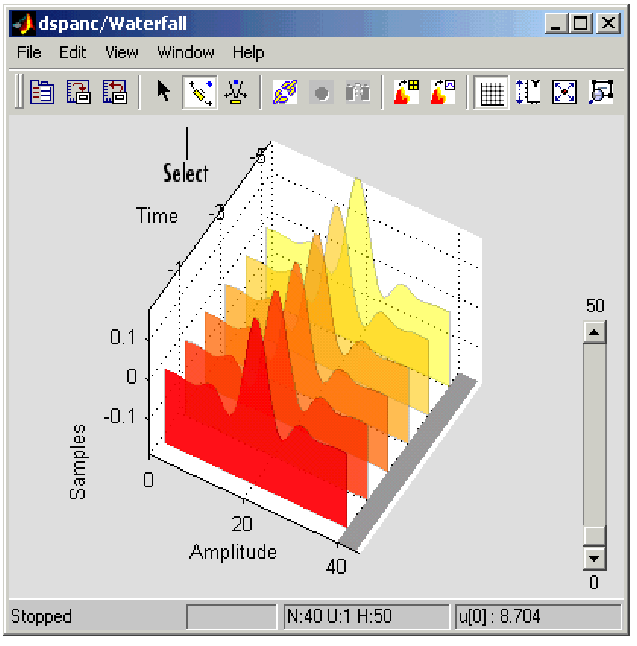
8 In the dspanc/Waterfall window, click the **Snapshot display** button.

In both scope windows, the data display freezes while the block continues to capture data.

9 To continue displaying the captured data, click the **Resume display** button.

Selecting Data

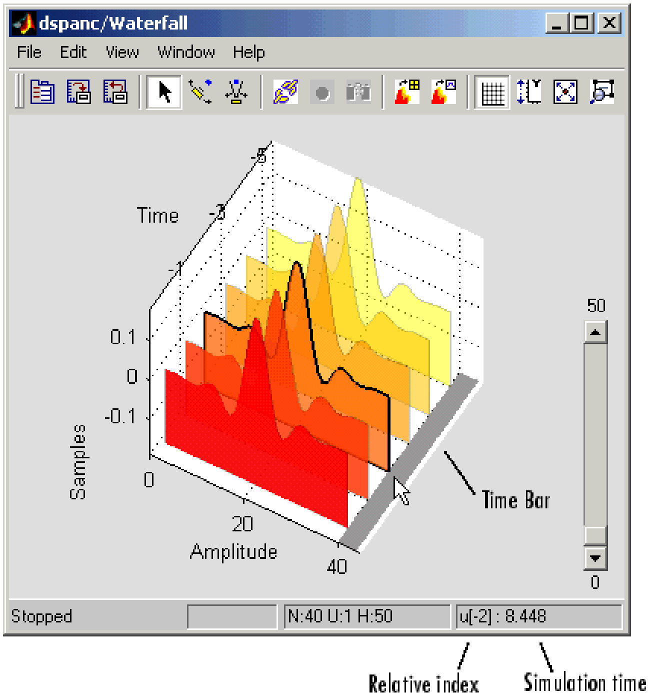
The following figure shows the Waterfall window displaying the output of the `dspanc` demo:



1 To select a particular set of data, click the **Select** button.

2 Click on the Time Bar at the bottom right of the axes to select a vector of data.

The Waterfall block highlights the selected trace.



While the simulation is running, in the bottom right corner, the Waterfall window displays the relative index of the selected trace. For example, in the previous figure, the selected vector is two sample times away from the most current data vector. When the simulation is stopped, the Waterfall window displays both the relative index and the simulation time associated with the selected trace.

3 To deselect the data vector, click it again.

4 Click-and-drag along the Time Bar.

Your selection follows the movement of the pointer.

You can use this feature to choose a particular vector to export to the MATLAB workspace or SPTool. For more information, see "Data History Parameters" on page 2-1523.

Zooming

You can use the Waterfall window to zoom in on data:

1. Click the **Zoom camera** button.
2. In the Waterfall window, click and hold down the left mouse button.
3. Move the mouse up and down and side-to-side to move closer and farther away from the axes.
4. To resize the axes to fit the Waterfall window, click the **Fit to view** button.

Rotating the Display

You can rotate the data displayed in the Waterfall window:

1. Click on the **Orbit camera** button.
2. In the Waterfall window, click and hold down the left mouse button.
3. Move the mouse in a circular motion to rotate the axes.
4. To return to the position of the original axes, click the **Restore scope position and view** button.

Scaling the Axes

You can use the Waterfall window to rescale the *y*-axis values:

1. Open and run the `dspanc` demo.
2. Click the **Rescale amplitude** button.

   The *y*-axis changes so that its minimum value is zero. The maximum value is scaled to fit the data displayed.

Alternatively, you can scale the *y*-axis using the **Y Min** and **Y Max** parameters in the **Axes** pane of the Parameters dialog box. This is helpful when you want to undo the effects of rescaling the amplitude. For more information, see "Axes Parameters" on page 2-1522.

Saving Scope Settings

The Waterfall block can save the screen position and viewpoint of the Waterfall window:

1. Click the **Save scope position and view** button.
2. Close the Waterfall window.
3. Reopen the Waterfall window.

   It reopens at the same place on your screen. The viewpoint of the axes also remains the same.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers

The Waterfall block accepts any of these data types as input. However, the input is converted to double-precision before the block processes the data. The Waterfall block displays only real-valued, double-precision vectors of data.

See Also

Scope	Simulink
Time Scope	Signal Processing Blockset

Vector Scope	Signal Processing Blockset
Spectrum Scope	Signal Processing Blockset
Matrix Viewer	Signal Processing Blockset
Signal To Workspace	Signal Processing Blockset
Triggered To Workspace	Signal Processing Blockset

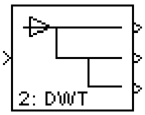
Purpose

Decompose signal into components of logarithmically decreasing frequency intervals and sample rates (requires the Wavelet Toolbox product)

Library

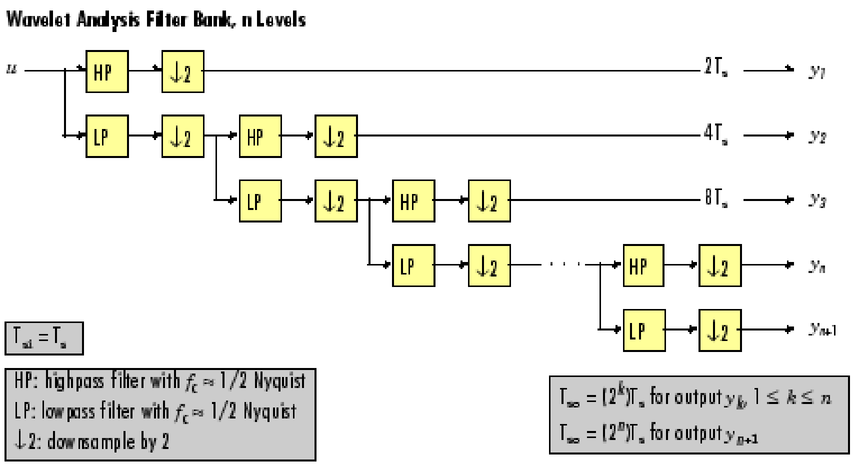
`dspobslib`

Description



Note The Wavelet Analysis block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the DWT block.

The Wavelet Analysis block uses the Wavelet Toolbox `wfilters` function to construct a dyadic analysis filter bank that decomposes a broadband signal into a collection of successively more bandlimited components. An n-level filter bank structure is shown below, where n is specified by the **Number of levels** parameter.



At each level, the low-frequency output of the previous level is decomposed into adjacent high- and low-frequency subbands by a highpass (HP) and lowpass (LP) filter pair. Each of the two output subbands is half the bandwidth of the input to that level. The bandlimited output of each filter is maximally decimated by a factor of 2 to preserve the bit rate of the original signal.

Filter Coefficients

The filter coefficients for the highpass and lowpass filters are computed by the Wavelet Toolbox function `wfilters`, based on the wavelet specified in the **Wavelet name** parameter. The table below lists the available options.

Wavelet Name	Sample Wavelet Function Syntax
Haar	`wfilters('haar')`
Daubechies	`wfilters('db4')`
Symlets	`wfilters('sym3')`
Coiflets	`wfilters('coif1')`
Biorthogonal	`wfilters('bior3.1')`
Reverse Biorthogonal	`wfilters('rbio3.1')`
Discrete Meyer	`wfilters('dmey')`

The **Daubechies**, **Symlets**, and **Coiflets** options enable a secondary **Wavelet order** parameter that allows you to specify the wavelet order. For example, if you specify a **Daubechies** wavelet with **Wavelet order** equal to `6`, the Wavelet Analysis block calls the `wfilters` function with input argument `'db6'`.

The **Biorthogonal** and **Reverse Biorthogonal** options enable a secondary **Filter order [synthesis / analysis]** parameter that allows you to independently specify the wavelet order for the analysis and synthesis filter stages. For example, if you specify a **Biorthogonal** wavelet with **Filter order [synthesis / analysis]** equal to `[2 / 6]`, the Wavelet Analysis block calls the `wfilters` function with input argument `'bior2.6'`.

See the Wavelet Toolbox documentation for more information about the `wfilters` function. If you want to explicitly specify the FIR coefficients for the analysis filter bank, use the Dyadic Analysis Filter Bank block.

Tree Structure

The wavelet tree structure has n+1 outputs, where n is the number of levels. The sample rate and bandwidth of the top output are half the input sample rate and bandwidth. The sample rate and bandwidth of each additional output (except the last) are half that of the output from the previous level. In general, for an input with sample period $T_{si} = T_s$, and bandwidth BW, output y_k has sample period $T_{so,k}$ and bandwidth BW_k.

$$T_{so,k} = \begin{cases} \left(2^k\right)T_s & (1 \le k \le n) \\ \left(2^n\right)T_s & (k = n+1) \end{cases}$$

$$BW_k = \begin{cases} \dfrac{BW}{2^k} & (1 \le k \le n) \\ \dfrac{BW}{2^n} & (k = n+1) \end{cases}$$

Note that in frame-based mode, the change in the sample period of output y_k is reflected by its frame size $M_{o,k}$, rather than by its frame rate.

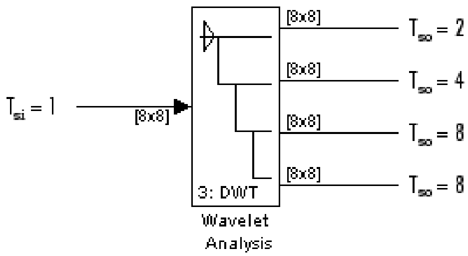
$$M_{o,k} = \begin{cases} \dfrac{M_i}{2^k} & (1 \le k \le n) \\ \dfrac{M_i}{2^n} & (k = n+1) \end{cases}$$

The bottom two outputs (y_n and y_{n+1}) share the same sample period, bandwidth, and frame size because they originate at the same tree level.

Sample-Based Operation

An M-by-N sample-based matrix input is treated as $M*N$ independent channels, and the block filters each channel independently over time. The output at each port is the same size as the input, one output channel for each input channel. As described earlier, each output port has a different sample period.

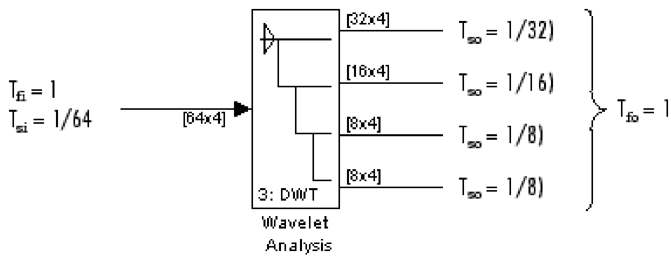
The figure below shows the input and output sample periods for a 64-channel sample-based input to a three-level filter bank. The input has a period of 1, so the fastest output has a period of 2.



Frame-Based Operation

An M_i-by-N frame-based matrix input is treated as N independent channels, and the block filters each channel independently over time. The input frame size M_i must be a multiple of 2^n, and n is the number of filter bank levels. For example, a frame size of 8 would be appropriate for a three-level tree (2^3=8). The number of columns in each output is the same as the number of columns in the input.

Each output port has the same frame period as the input. The reduction in the output sample rates results from the smaller output frame sizes, as shown in the example below for a four-channel input to a three-level filter bank.



Zero Latency

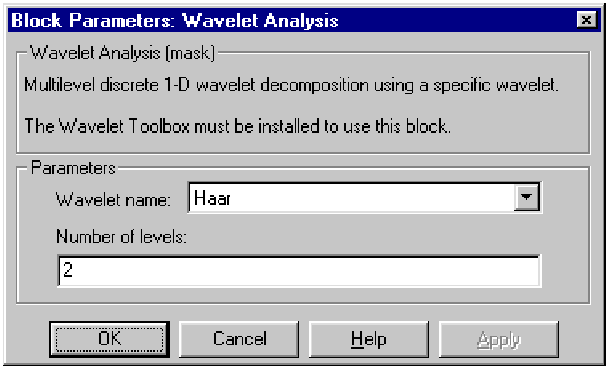
The Wavelet Analysis block has no tasking latency for frame-based operation, which is always single-rate. The block therefore analyzes the first input sample (received at t=0) to produce the first output sample at each port.

Nonzero Latency

For sample-based operation, the Wavelet Analysis block is multirate and has 2^{n-1} samples of latency in both Simulink tasking modes. As a result, the block repeats a zero initial condition in each channel for the first 2^{n-1} output samples, before propagating the first analyzed input sample (computed from the input received at t=0).

Note For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and the topic on models with multiple sample rates in the Real-Time Workshop documentation.

Dialog Box



The parameters displayed in the dialog box vary for different wavelet types. Only some of the parameters listed below are visible in the dialog box at any one time.

Wavelet name
The wavelet used in the analysis.

Wavelet order
The order for the **Daubechies**, **Symlets**, and **Coiflets** wavelets. This parameter is available only when one of these wavelets is selected in the **Wavelet name** menu.

Filter order [synthesis / analysis]
The filter orders for the synthesis and analysis stages of the **Biorthogonal** and **Reverse Biorthogonal** wavelets. For example, `[2 / 6]` selects a second-order synthesis stage and a sixth-order analysis stage. The **Filter order** parameter is available only when one of the above wavelets is selected in the **Wavelet name** menu.

Number of levels
The number of filter bank levels. An n-level structure has $n+1$ outputs.

References

Fliege, N. J. *Multirate Digital Signal Processing: Multirate Systems, Filter Banks, Wavelets*. West Sussex, England: John Wiley & Sons, 1994.

Strang, G. and T. Nguyen. *Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1996.

Vaidyanathan, P. P. *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice Hall, 1993.

Supported Data Types

- Double-precision floating point

See Also

Dyadic Analysis Filter Bank	Signal Processing Blockset
Wavelet Synthesis (Obsolete)	Signal Processing Blockset
`wfilters`	Wavelet Toolbox

Wavelet Synthesis (Obsolete)

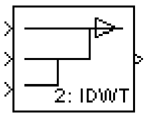
Purpose

Reconstruct signal from its multirate bandlimited components (requires the Wavelet Toolbox product)

Library

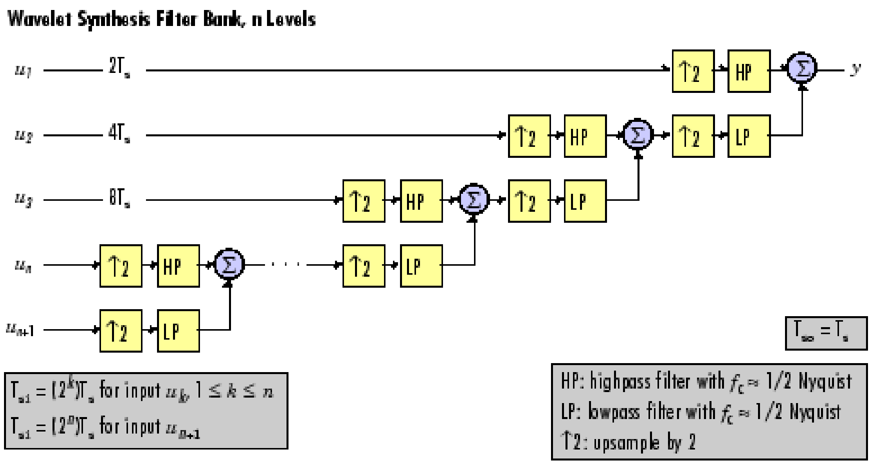
`dspobslib`

Description



Note The Wavelet Synthesis block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the IDWT block.

The Wavelet Synthesis block uses the Wavelet Toolbox `wfilters` function to reconstruct a signal that was decomposed by the Wavelet Analysis (Obsolete) block. The reconstruction or synthesis process is the inverse of the analysis process, and restores the original signal by upsampling, filtering, and summing the bandlimited inputs in stages corresponding to the analysis process. An n-level synthesis filter bank structure is shown below, where n is specified by the **Number of levels** parameter.



At each level, the two bandlimited inputs (one low-frequency, one high-frequency, both with the same sample rate) are upsampled by a factor of 2 to match the sample rate of the input to the next stage. They are then filtered by a highpass (HP) and lowpass (LP) filter pair with coefficients calculated to cancel (in the subsequent summation) the aliasing introduced in the corresponding analysis filter stage. The output from each (upsample-filter-sum) level has twice the bandwidth and twice the sample rate of the input to that level.

For perfect reconstruction, the Wavelet Synthesis and Wavelet Analysis blocks must have the same parameter settings.

Filter Coefficients

The filter coefficients for the highpass and lowpass filters are computed by the Wavelet Toolbox function `wfilters`, based on the wavelet specified in the **Wavelet name** parameter. The table below lists the available options.

Wavelet Name	Sample Wavelet Function Syntax
Haar	`wfilters('haar')`
Daubechies	`wfilters('db4')`
Symlets	`wfilters('sym3')`
Coiflets	`wfilters('coif1')`
Biorthogonal	`wfilters('bior3.1')`
Reverse Biorthogonal	`wfilters('rbio3.1')`
Discrete Meyer	`wfilters('dmey')`

The **Daubechies**, **Symlets**, and **Coiflets** options enable a secondary **Wavelet order** parameter that allows you to specify the wavelet order. For example, if you specify a **Daubechies** wavelet with **Wavelet order** equal to `6`, the Wavelet Synthesis block calls the `wfilters` function with input argument `'db6'`.

The **Biorthogonal** and **Reverse Biorthogonal** options enable a secondary **Filter order [synthesis / analysis]** parameter that allows you to independently specify the wavelet order for the analysis and synthesis filter stages. For example, if you specify a **Biorthogonal** wavelet with **Filter order [synthesis / analysis]** equal to `[2 / 6]`, the Wavelet Synthesis block calls the `wfilters` function with input argument `'bior2.6'`.

See the Wavelet Toolbox documentation for more information about the `wfilters` function. If you want to explicitly specify the FIR coefficients for the synthesis filter bank, use the Dyadic Synthesis Filter Bank block.

Tree Structure

The wavelet tree structure has n+1 inputs, where n is the number of levels. The sample rate and bandwidth of the output are twice the sample rate and bandwidth of the top input. The sample rate and bandwidth of each additional input (except the last) are half that of the input to the previous level.

$$T_{si,k+1} = 2T_{si,k} \quad 1 \le k < n$$

$$BW_{k+1} = \frac{BW_k}{2} \quad 1 \le k < n$$

The bottom two inputs (u_n and u_{n+1}) should have the same sample rate and bandwidth since they are processed by the same level.

$$T_{si,n+1} = T_{si,n}$$

$$BW_{n+1} = BW_n$$

Note that in frame-based mode, the sample period of input u_k is reflected by its frame size$M_{i,k}$, rather than by its frame rate.

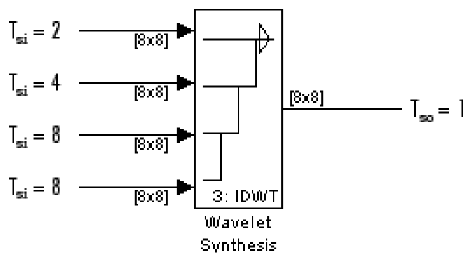
$$M_{i,k+1} = \frac{M_{i,k}}{2} \qquad 1 \le k < n$$

$$M_{i,n+1} = M_{i,n}$$

Sample-Based Operation

An M-by-N sample-based matrix input is treated as $M*N$ independent channels, and the block filters each channel independently over time. The output is the same size as the input at each port, one output channel for each input channel. As described earlier, each input port has a different sample period.

The figure below shows the input and output sample periods for the four 64-channel sample-based inputs to a three-level filter bank. The fastest input has a period of 2, so the output period is 1.

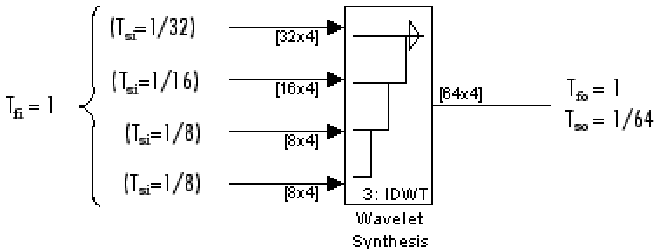


Frame-Based Operation

An M_i-by-N frame-based matrix input is treated as N independent channels, and the block filters each channel independently over time. The number of columns in the output is the same as the number of columns in the input.

All inputs must have the same frame period, which is also the output frame period. The different input sample rates should be represented by the input frame sizes: If the input to the top port has frame size M_i, the input to the second-from-top port should have frame size $M_i/2$, the input to the third-from-top port should have frame size $M_i/4$, and so on. The input to the bottom port should have the same frame size as the second-from-bottom port. The increase in the sample rate of the output is also represented by its frame size, which is twice the largest input frame size.

The relationship between sample periods, frame periods, and frame sizes is shown below for a four-channel frame-based input to a 3-level filter bank.



Zero Latency

The Wavelet Synthesis block has no tasking latency for frame-based operation, which is always single-rate. The block therefore uses the first input samples (received at t=0) to synthesize the first output sample.

Nonzero Latency

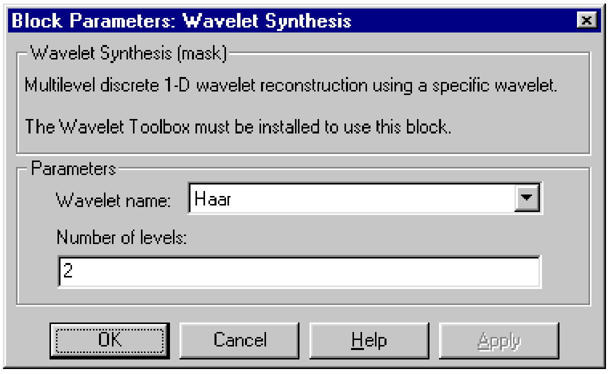
For sample-based operation, the Wavelet Synthesis block is multirate and has the following tasking latencies:

- 2^n-2 samples in Simulink's single-tasking mode
- 2^n samples in Simulink's multitasking mode

In the above cases, the block repeats a zero initial condition in each channel for the first D output samples, where D is the latency shown above. For example, in single-tasking mode the block generates 2^n-2 zero-valued output samples in each channel before propagating the first synthesized output sample (computed from the inputs received at t=0).

Note For more information on latency and the Simulink tasking modes, see "Excess Algorithmic Delay (Tasking Latency)" and the topic on models with multiple sample rates in the Real-Time Workshop documentation.

Dialog Box



The parameters displayed in the dialog box vary for different wavelet types. Only some of the parameters listed below are visible in the dialog box at any one time.

Wavelet name
The wavelet used in the synthesis.

Wavelet order
The order for the **Daubechies**, **Symlets**, and **Coiflets** wavelets. This parameter is available only when one of these wavelets is selected in the **Wavelet name** menu.

Filter order [synthesis / analysis]
The filter orders for the synthesis and analysis stages of the **Biorthogonal** and **Reverse Biorthogonal** wavelets. For example, `[2 / 6]` selects a second-order synthesis stage and a sixth-order analysis stage. The **Filter order** parameter is available only when one of the above wavelets is selected in the **Wavelet name** menu.

Number of levels

The number of filter bank levels. An n-level structure has n+1 outputs.

References

Fliege, N. J. *Multirate Digital Signal Processing: Multirate Systems, Filter Banks, Wavelets*. West Sussex, England: John Wiley & Sons, 1994.

Strang, G. and T. Nguyen. *Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1996.

Vaidyanathan, P. P. *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice Hall, 1993.

Supported Data Types

- Double-precision floating point

See Also

Dyadic Synthesis Filter Bank	Signal Processing Blockset
Wavelet Analysis (Obsolete)	Signal Processing Blockset
`wfilters`	Wavelet Toolbox

Purpose

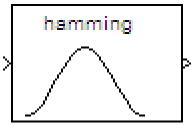
Compute and/or apply window to input signal

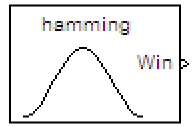
Library

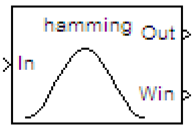
Signal Operations

`dspsigops`

Description







The Window Function block computes a window and/or applies a window to an input signal. The input signal can be a frame-based matrix, or a sample-based N-D array. The Window Function block supports real and complex floating-point and fixed-point inputs.

Operation Modes

The Window Function block has three modes of operation that you can select via the **Operation** parameter. In each mode, the block first creates a window vector w by sampling the window specified in the **Window type** parameter at M discrete points. The operation modes are:

- `Apply window to input`

  In this mode, the block computes an M-by-1 window vector w and applies it to the input. The output y always has the same dimension as the input. When the input is frame based, the output is frame based; otherwise, the output is sample based.

  When the input is an M-by-N matrix u, the window is multiplied element-wise with each of the N channels in the input matrix u. This is equivalent to the following MATLAB code:

  ```
  y = repmat(w,1,N) .* u        % Equivalent MATLAB code
  ```

  When the input is a sample-based N-D array, the window is always applied to the first dimension:

  $$y(i,j,\ldots,k) = w(i) * u(i,j,\ldots,k) \qquad i = 1,\ldots,M,\ \ j = 1,\ldots,N,\ \ \ldots,\ \ k = 1,\ldots,P$$

  A length-M 1-D vector input is treated as an M-by-1 matrix.

- `Generate window`

In this mode, the block generates a sample-based 1-D window vector w with length M specified by the **Window length** parameter. The In port is disabled for this mode.

- `Generate and apply window`

  In this mode, the block generates an M-by-1 window vector w and applies it to the input. The block produces two outputs:

  - At the Out port, the block produces the result of the multiplication y, which has the same dimension as the input. When the input is frame based, the output y is frame based; otherwise, the output y is sample based.
  - At the Win port, the block produces the M-by-1 window vector w. The output vector w is always sample based.

  When the input is an M-by-N matrix u, the window is multiplied element-wise with each of the N channels in the input matrix u. This is equivalent to the following MATLAB code:

  ```
  y = repmat(w,1,N) .* u        % Equivalent MATLAB code
  ```

  When the input is a sample-based N-D array, the window is always applied to the first dimension:

  $$y(i,j,\ldots,k) = w(i) * u(i,j,\ldots,k) \qquad i = 1,\ldots,M,\ \ j = 1,\ldots,N,\ \ \ldots,\ \ k = 1,\ldots,P$$

  A length-M 1-D vector input is treated as an M-by-1 matrix.

Window Type

The following table lists the available window types. For complete information about the window functions, consult the “Signal Processing Toolbox” documentation.

Window Type	Description
Bartlett	Computes a Bartlett window. `w = bartlett(M)`
Blackman	Computes a Blackman window. `w = blackman(M)`
Boxcar	Computes a rectangular window. `w = rectwin(M)`
Chebyshev	Computes a Chebyshev window with stopband ripple `R`. `w = chebwin(M,R)`
Hamming	Computes a Hamming window. `w = hamming(M)`
Hann	Computes a Hann window (also known as a Hanning window). `w = hann(M)`
Hanning	Obsolete. This window type is included only for compatibility with older models. Use the `Hann` **Window type** instead of `Hanning` whenever possible.
Kaiser	Computes a Kaiser window with the Kaiser parameter `beta`. `w = kaiser(M,beta)`

Window Type	Description
Taylor	Computes a Taylor window. `w = taylorwin(M)`
Triang	Computes a triangular window. `w = triang(M)`

User Defined	Computes the user-defined window function specified by the entry in the **Window function name** parameter, `usrwin`.
	
	`w = usrwin(M) % Window takes no extra parameters`
	`w = usrwin(M,x_1,...,x_n) % Window takes extra parameters {x_1 ... x_n}`

Window Sampling

For the generalized-cosine windows (`Blackman`, `Hamming`, `Hann`, and `Hanning`), the **Sampling** parameter determines whether the window samples are computed in a periodic or a symmetric manner. For example, when **Sampling** is set to `Symmetric`, a Hamming window of length M is computed as

```
w = hamming(M)     % Symmetric (aperiodic) window
```

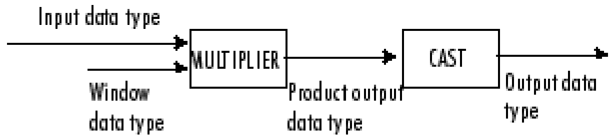
When **Sampling** is set to `Periodic`, the same window is computed as

```
w = hamming(M+1)     % Periodic (asymmetric) window
w = w(1:M)
```

Fixed-Point Data Types

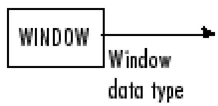
The following diagram shows the data types used within the Window Function block for fixed-point signals for each of the three operating modes.

Apply window to input



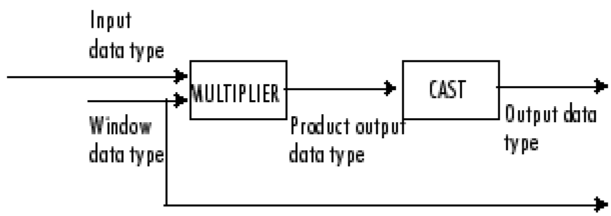
The input data type comes from the driving block. You can set the window, product output, and output data types in the block dialog. In this mode, the window vector is not output from the block.

Generate window



In this mode, the block acts as a source. The window vector is output in the window data type you specify in the block dialog.

Generate and apply window



The input data type comes from the driving block. You can set the window, product output, and output data types in the block dialog. In this mode, the window vector is output from the block.

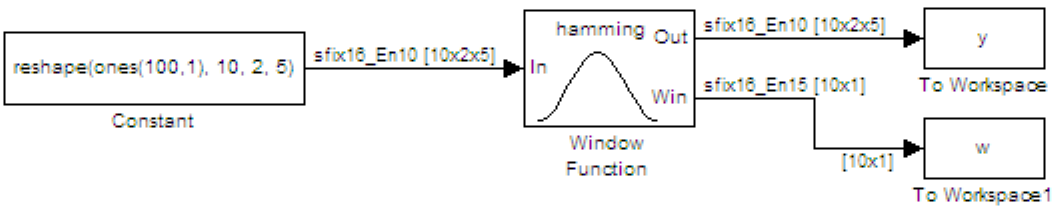
You can set the window, product output, and output data types in the block dialog box. For more information see the “Dialog Box” on page 2-1562 section.

Examples

The following model uses the Window Function block to generate and apply a Hamming window to a sample-based 3-dimensional input array.

In this example, set the **Operation** mode of the Window Function block to `Generate and apply window`, so the block provides two outputs: the window vector *w* at the Win port, and the result of the multiplication *y* at the Out port.

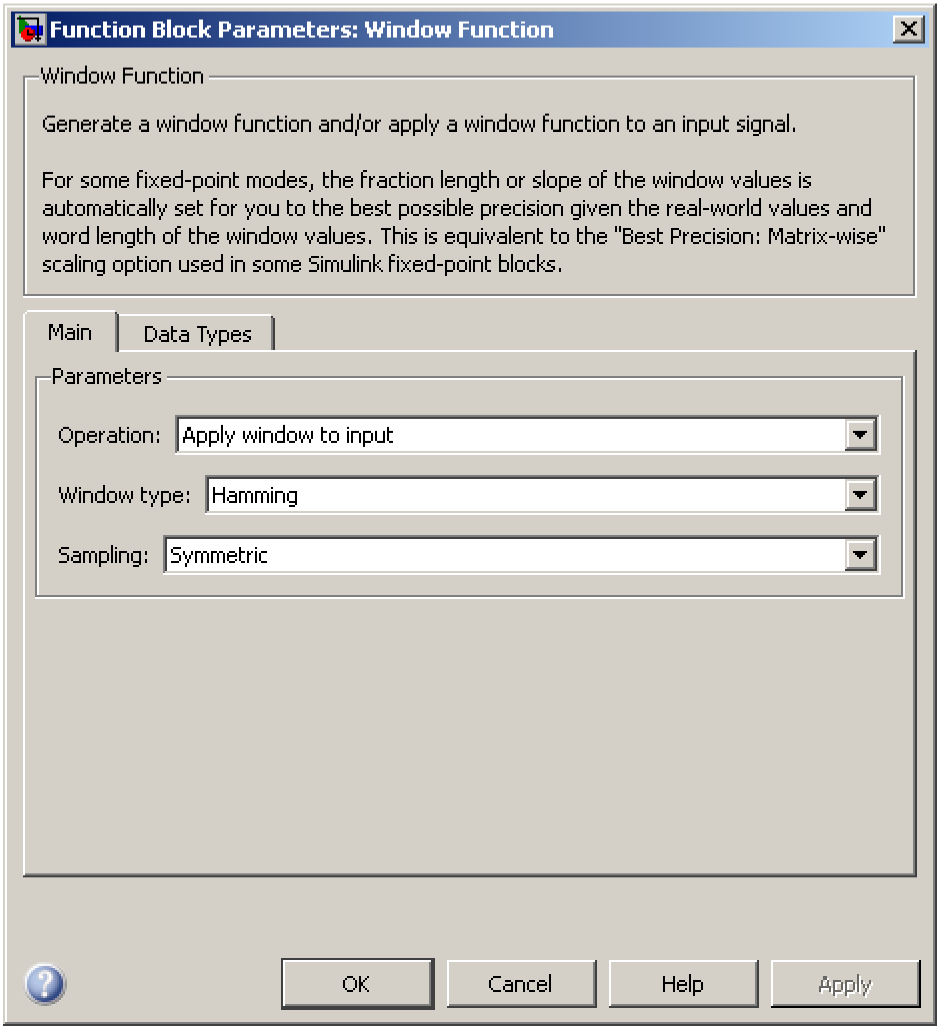
Open the model by typing `doc_windowfunction_ref` at the MATLAB command line, and run it.



- The length of the first dimension of the input array is 10, so the Window Function block generates and outputs a Hamming window vector of length 10. To see the window vector generated by the Window Function block, type `w` at the MATLAB command line.
- To see the result of the multiplication, type `y` at the MATLAB command line.

Dialog Box

The **Main** pane of the Window Function block dialog appears as follows.



Operation

Specify the block's operation, as discussed in "Operation Modes" on page 2-1555. The port configuration of the block is updated to match the setting of this parameter.

Window type

Specify the window type to apply, as listed in "Window Type" on page 2-1556. Tunable in simulation only.

Sampling

Specify the window sampling for generalized-cosine windows. This parameter is only visible when you select `Blackman`, `Hamming`, `Hann`, or `Hanning` for the **Window type** parameter. Tunable in simulation only.

Sample Mode

Specify the sample mode for the block, `Continuous` or `Discrete`, when it is in `Generate Window` mode. In the `Apply window to output` and `Generate and apply window` modes, the block inherits the sample time from its driving block. Therefore, this parameter is only visible when you select `Generate window` for the **Operation** parameter.

Sample time

Specify the sample time for the block when it is in `Generate window` and `Discrete` modes. In `Apply window to output` and `Generate and apply window` modes, the block inherits the sample time from its driving block. This parameter is only visible when you select `Discrete` for the **Sample Mode** parameter.

Window length

Specify the length of the window to apply. This parameter is only visible when you select `Generate window` for the **Operation** parameter. Otherwise, the window vector length is computed to match the length of the first dimension of the input.

Stopband attenuation in dB

Specify the level of stopband attenuation, R_s, in decibels. This parameter is only visible when you select `Chebyshev` for the **Window type** parameter. Tunable in simulation only.

Beta

Specify the Kaiser window β parameter. Increasing β widens the mainlobe and decreases the amplitude of the window sidelobes in the window's frequency magnitude response. This parameter is only visible when you select Kaiser for the **Window type** parameter. Tunable in simulation only.

Number of sidelobes

Specify the number of sidelobes as a scalar integer value greater than zero. This parameter is only visible when you select Taylor for the **Window type** parameter.

Maximum sidelobe level relative to mainlobe (dB)

Specify, in decibels, the maximum sidelobe level relative to the mainlobe. This parameter must be a scalar less than or equal to zero. The default value of –30 produces sidelobes with peaks 30 dB down from the mainlobe peak. This parameter is only visible when you select Taylor for the **Window type** parameter.

Window function name

Specify the name of the user-defined window function to be calculated by the block. This parameter is only visible when you select User defined for the **Window type** parameter.

Specify additional arguments to the hamming function

Select to enable the **Cell array of additional arguments** parameter, when the user-defined window requires parameters other than the window length. This parameter is only visible when you select User defined for the **Window type** parameter.

Cell array of additional arguments

Specify the extra parameters required by the user-defined window function, besides the window length. This parameter is only available when you select the **Specify additional arguments to the hamming function** parameter. The entry must be a cell array.

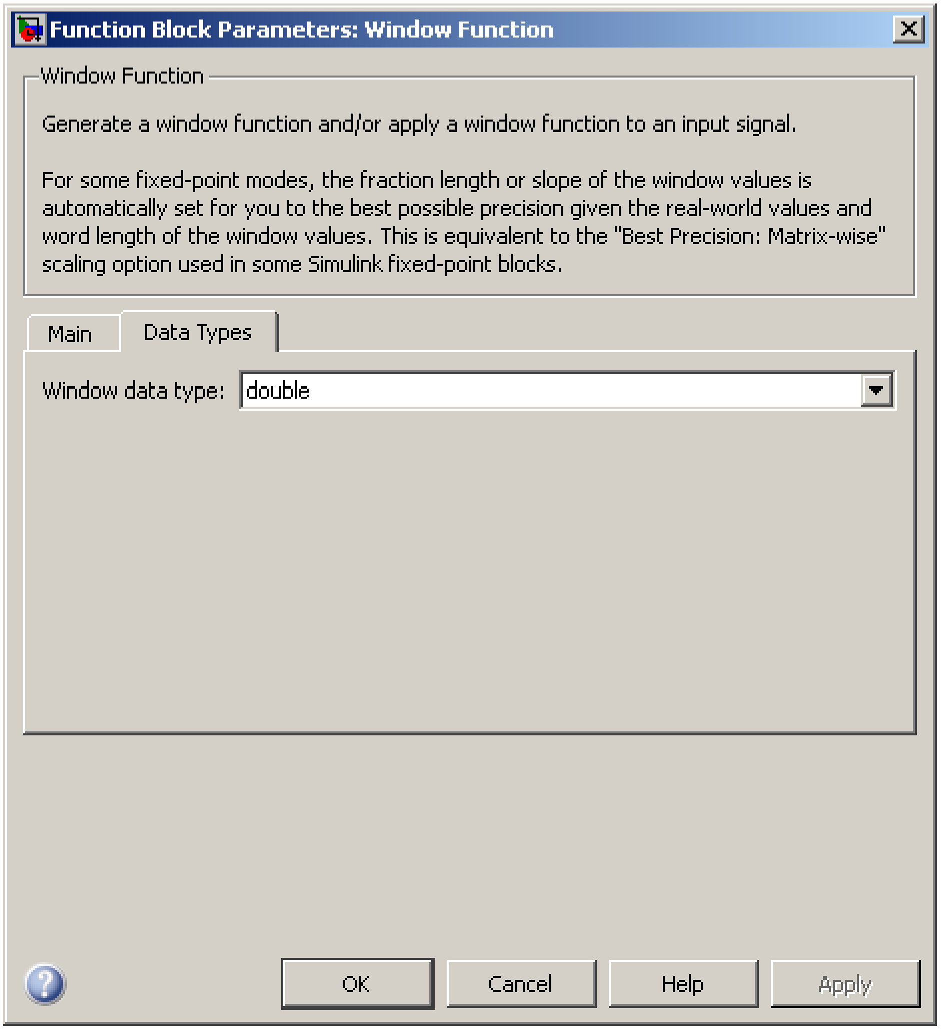
The **Data Types** pane of the Window Function block dialog is discussed in the following sections:

"Parameters for Generate Window Only Mode" on page 2-1565

"Parameters for Apply Window Modes" on page 2-1568

Parameters for Generate Window Only Mode

The **Data Types** pane of the Window Function block dialog appears as follows when the **Operation** parameter is set to `Generate window`.



Window data type

Specify the window data type in one of the following ways:

- Choose `double` or `single` from the list.

- Choose `Fixed-point` to specify the window data type and scaling in the **Signed**, **Word length**, **Set fraction length in output to,** and **Fraction length** parameters.
- Choose `User-defined` to specify the window data type and scaling in the **User-defined data type**, **Set fraction length in output to,** and **Fraction length** parameters.
- Choose `Inherit via back propagation` to set the window data type and scaling to match the following block.

Signed
Select to output a signed fixed-point signal. Otherwise, the signal is unsigned.

Word length
Specify the word length, in bits, of the fixed-point window data type. This parameter is only visible when you select `Fixed-point` for the **Window data type** parameter.

User-defined data type
Specify any built-in or fixed-point data type. You can specify fixed-point data types using the Simulink Fixed Point functions `sfix`, `ufix`, `sint`, `uint`, `sfrac`, and `ufrac`. This parameter is only visible when you select `User-defined` for the **Window data type** parameter.

Set fraction length in output to
Specify the scaling of the fixed-point window data type by either of the following two methods:

- Choose `Best precision` to have the window data type scaling automatically set such that the output signal has the best possible precision.
- Choose `User-defined` to specify the window data type scaling in the **Fraction length** parameter.

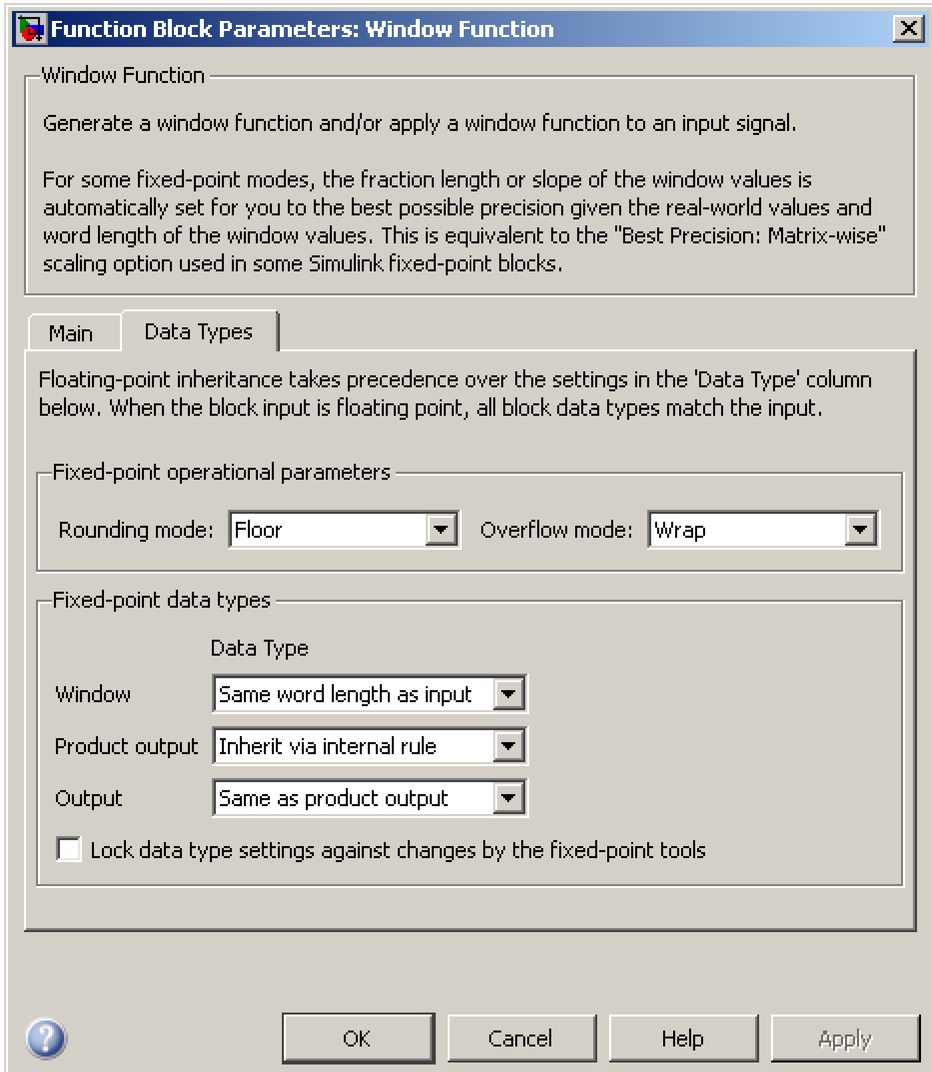
This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Window data type** parameter, and when the specified window data type is a fixed-point data type.

Fraction length

Specify the fraction length, in bits, of the fixed-point window data type. This parameter is only visible when you select `Fixed-point` or `User-defined` for the **Window data type** parameter and `User-defined` for the **Set fraction length in output to** parameter.

Parameters for Apply Window Modes

The **Data Types** pane of the Window Function block dialog appears as follows when the **Operation** parameter is set to either `Apply window to input` or `Generate and apply window`.



Rounding mode

Select the rounding mode for fixed-point operations.

The window vector *w* does not obey this parameter; it always rounds to `Nearest`.

Overflow mode
Select the overflow mode for fixed-point operations.

The window vector *w* does not obey this parameter; it is always saturated.

Window
Choose how you specify the word length and fraction length of the window vector *w*.

When you select `Same word length as input`, the word length of the window vector elements is the same as the word length of the input. The fraction length is automatically set to the best precision possible.

When you select `Specify word length`, you can enter the word length of the window vector elements in bits. The fraction length is automatically set to the best precision possible.

When you select `Binary point scaling`, you can enter the word length and the fraction length of the window vector elements in bits.

When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the window vector elements. This block requires power-of-two slope and a bias of zero.

The window vector does not obey the **Rounding mode** and **Overflow mode** parameters; it is always saturated and rounded to `Nearest`.

Product output
Use this parameter to specify how you want to designate the product output word and fraction lengths. See "Fixed-Point Data

Types" on page 2-1453 for illustrations depicting the use of the product output data type in this block:

- When you select `Inherit via internal rule`, the product output word length and fraction length are calculated automatically. For information about how the product output word and fraction lengths are calculated when an internal rule is used, see "Inherit via Internal Rule".
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the product output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the product output. This block requires power-of-two slope and a bias of zero.

Output
Choose how you specify the word length and fraction length of the output of the block:

- When you select `Same as product output`, these characteristics match those of the product output.
- When you select `Same as input`, these characteristics match those of the input to the block.
- When you select `Binary point scaling`, you can enter the word length and the fraction length of the output, in bits.
- When you select `Slope and bias scaling`, you can enter the word length, in bits, and the slope of the output. This block requires power-of-two slope and a bias of zero.

Lock data type settings against changes by the fixed-point tools
Select this parameter to prevent the fixed-point tools from overriding the data types you specify on the block mask.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
Output	• Double-precision floating point • Single-precision floating point • Fixed point (signed only) • 8-, 16-, and 32-bit signed integers
Win	• Double-precision floating point • Single-precision floating point • Fixed point • 8-, 16-, and 32-bit integers

See Also

FFT	Signal Processing Blockset
`bartlett`	Signal Processing Toolbox
`blackman`	Signal Processing Toolbox
`rectwin`	Signal Processing Toolbox
`chebwin`	Signal Processing Toolbox
`hamming`	Signal Processing Toolbox
`hann`	Signal Processing Toolbox
`kaiser`	Signal Processing Toolbox
`taylorwin`	Signal Processing Toolbox
`triang`	Signal Processing Toolbox

Purpose

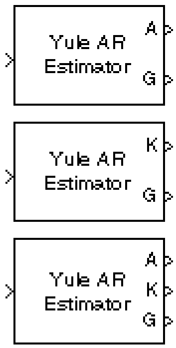
Compute estimate of autoregressive (AR) model parameters using Yule-Walker method

Library

Estimation / Parametric Estimation

`dspparest3`

Description



The Yule-Walker AR Estimator block uses the Yule-Walker AR method, also called the autocorrelation method, to fit an autoregressive (AR) model to the windowed input data by minimizing the forward prediction error in the least squares sense. This formulation leads to the Yule-Walker equations, which are solved by the Levinson-Durbin recursion. Block outputs are always nonsingular.

The Yule-Walker AR Estimator block can output the AR model coefficients as polynomial coefficients, reflection coefficients, or both. Each channel of the input is a sample-based vector (row, column, or 1-D) or frame-based vector (column only) representing a frame of consecutive time samples from a signal that is assumed to be the output of an AR system driven by white noise. The block computes the normalized estimate of the AR system parameters, $A(z)$, independently for each successive input frame.

$$H(z) = \frac{\sqrt{G}}{A(z)} = \frac{\sqrt{G}}{1 + a(2)z^{-1} + \ldots + a(p+1)z^{-p}}$$

When you select **Inherit estimation order from input dimensions**, the order p of the all-pole model is one less than the length of each input channel. Otherwise, the order is the value specified by the **Estimation order** parameter. To guarantee a valid output, you must set the **Estimation order** parameter to be a scalar less than or equal to half the input channel length. The Yule-Walker AR Estimator and Burg AR Estimator blocks return similar results for large frame sizes.

When **Output(s)** is set to `A`, port A is enabled. For each channel, port A outputs a column of length $p+1$ that contains the normalized estimate of the AR model coefficients in descending powers of z

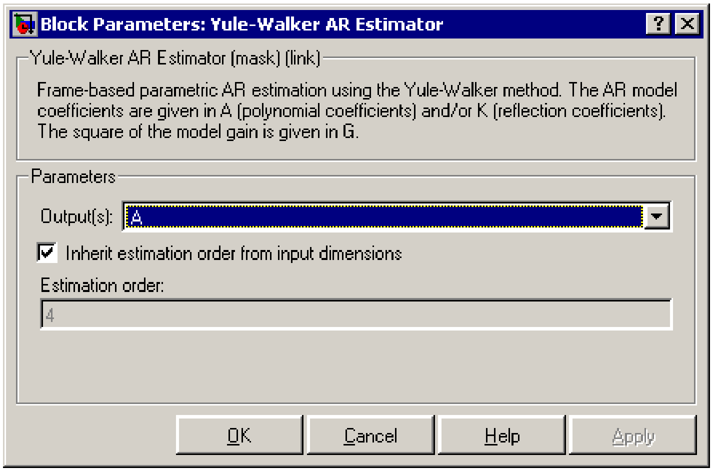
```
[1 a(2) ... a(p+1)]
```

When **Output(s)** is set to `K`, port K is enabled. For each channel, port K outputs a length-*p* column whose elements are the AR model reflection coefficients. When **Output(s)** is set to `A and K`, both port A and K are enabled, and each port outputs the respective AR model coefficients for each channel.

The square of the model gain, *G*, is provided at port G. *G* is a scalar for each channel.

See the Burg AR Estimator block reference page for a comparison of the Burg AR Estimator, Covariance AR Estimator, Modified Covariance AR Estimator, and Yule-Walker AR Estimator blocks.

Dialog Box



Output(s)

The type of AR model coefficients output by the block. The block can output polynomial coefficients (`A`), reflection coefficients (`K`), or both (`A and K`).

Inherit estimation order from input dimensions
When selected, sets the estimation order *p* to one less than the length of each input channel.

Estimation order
The order of the AR model, *p*. This parameter is enabled when you do not select **Inherit estimation order from input dimensions**.

References

Kay, S. M. *Modern Spectral Estimation: Theory and Application.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

Marple, S. L., Jr., *Digital Spectral Analysis with Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1987.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point
A	• Double-precision floating point • Single-precision floating point
K	• Double-precision floating point • Single-precision floating point
G	• Double-precision floating point • Single-precision floating point

See Also

Burg AR Estimator	Signal Processing Blockset
Covariance AR Estimator	Signal Processing Blockset
Modified Covariance AR Estimator	Signal Processing Blockset

Yule-Walker Method	Signal Processing Blockset
`aryule`	Signal Processing Toolbox

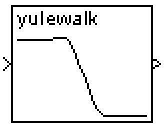
Purpose

Design and apply IIR filter

Library

`dspobslib`

Description



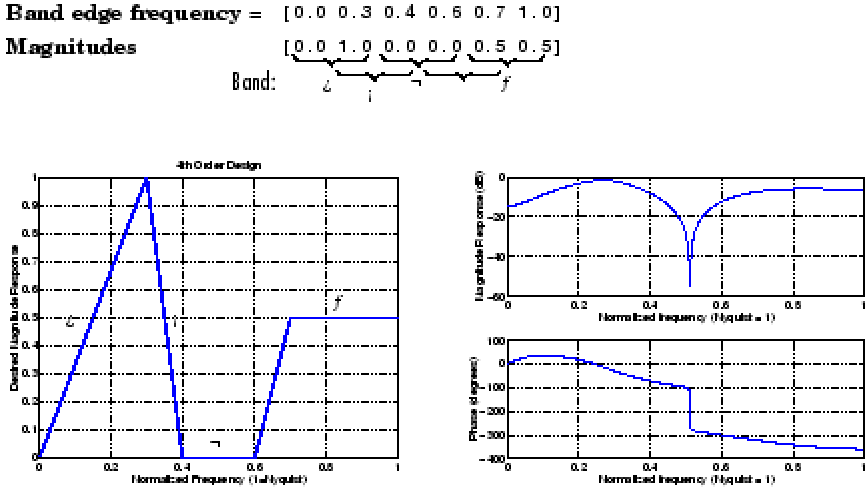
Note The Yule-Walker IIR Filter Design block is still supported but is likely to be obsoleted in a future release. We strongly recommend replacing this block with the Digital Filter block.

The Yule-Walker IIR Filter Design block designs a recursive (ARMA) digital filter with arbitrary multiband magnitude response, and applies it to a discrete-time input using the Direct-Form II Transpose Filter block. The filter design, which uses the Signal Processing Toolbox `yulewalk` function, performs a least-squares fit to the specified frequency response.

An M-by-N sample-based matrix input is treated as $M*N$ independent channels, and an M-by-N frame-based matrix input is treated as N independent channels. In both cases, the block filters each channel independently over time, and the output has the same size and frame status as the input.

The **Band-edge frequency vector** parameter is a vector of frequency points in the range `0` to `1`, where `1` corresponds to half the sample frequency. The first element of this vector must be `0` and the last element `1`, and intermediate points must appear in ascending order. The **Magnitudes at these frequencies** parameter is a vector containing the desired magnitude response at the points specified in the **Band-edge frequency vector**.

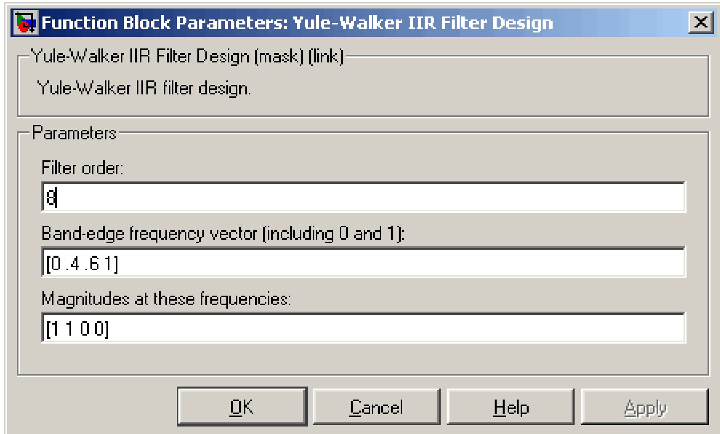
Note that, unlike the Remez FIR Filter Design block, each frequency-magnitude pair specifies the junction of two adjacent frequency bands, so there are no “don’t care” regions.



When specifying the **Band-edge frequency vector** and **Magnitudes at these frequencies** vectors, avoid excessively sharp transitions from passband to stopband. You may need to experiment with the slope of the transition region to get the best filter design.

For more details on the Yule-Walker filter design algorithm, see the description of the `yulewalk` function in the Signal Processing Toolbox documentation.

Dialog Box



Filter order
The order of the filter.

Band-edge frequency vector
A vector of frequency points. The value 1 corresponds to half the sample frequency. The first element of this vector must be 0 and the last element 1. Tunable.

Magnitudes at these frequencies
A vector of frequency response magnitudes corresponding to the points in the **Band-edge frequency vector**. This vector must be the same length as the **Band-edge frequency vector**. Tunable.

References

Oppenheim, A. V. and R. W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

Proakis, J. and D. Manolakis. *Digital Signal Processing*. 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996.

Yule-Walker Method

Purpose

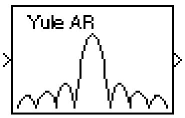
Power spectral density estimate using Yule-Walker method

Library

Estimation / Power Spectrum Estimation

`dspspect3`

Description



The Yule-Walker Method block estimates the power spectral density (PSD) of the input using the Yule-Walker AR method. This method, also called the *autocorrelation method*, fits an autoregressive (AR) model to the windowed input data. It does so by minimizing the forward prediction error in the least squares sense. This formulation leads to the Yule-Walker equations, which the Levinson-Durbin recursion solves. Block outputs are always nonsingular.

The input is a sample-based vector (row, column, or 1-D) or frame-based vector (column only). This input represents a frame of consecutive time samples from a single-channel signal. The block outputs a column vector containing the estimate of the power spectral density of the signal at N_{fft} equally spaced frequency points. The frequency points are in the range $[0,F_s)$, where F_s is the sampling frequency of the signal.

When you select **Inherit estimation order from input dimensions**, the order of the all-pole model is one less that the input frame size. Otherwise, the **Estimation order** parameter value specifies the order. To guarantee a valid output, the **Estimation order** parameter must be less than or equal to half the input vector length. The block computes the spectrum from the FFT of the estimated AR model parameters.

Selecting the **Inherit FFT length from estimation order** parameter specifies that N_{fft} is one greater than the estimation order. Clearing the **Inherit FFT length from estimation order** check box allows you to use the **FFT length** parameter to specify N_{fft} as a power of 2. The block zero-pads or wraps the input to N_{fft} before computing the FFT. The output is always sample based.

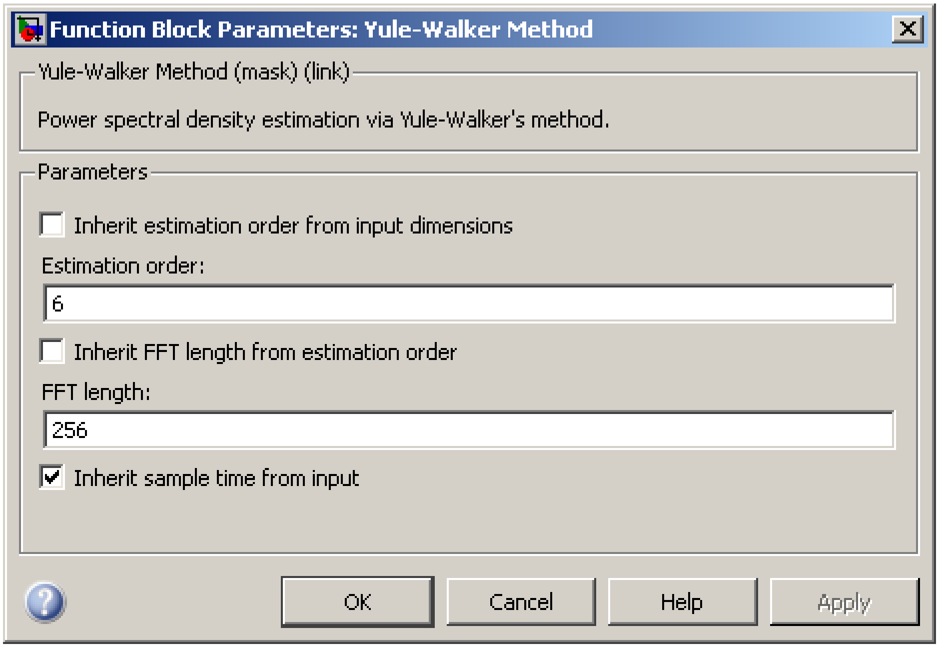
When you select the **Inherit sample time from input** check box, the block computes the frequency data from the sample period of the input signal. For the block to produce valid output, the following conditions must hold:

- The input to the block is the original signal, with no samples added or deleted (by insertion of zeros, for example).
- The sample period of the time-domain signal in the simulation equals the sample period of the original time series.

If these conditions do not hold, clear the **Inherit sample time from input** check box. You can then specify a sample time using the **Sample time of original time series** parameter.

See the Burg Method block reference for a comparison of the Burg Method, Covariance Method, Modified Covariance Method, and Yule-Walker AR Estimator blocks. The Yule-Walker AR Estimator and Burg Method blocks return similar results for large buffer lengths.

Dialog Box



Inherit estimation order from input dimensions
When you select this option, it sets the estimation order to one less than the length of the input vector.

Estimation order
Specify the order of the AR model. This parameter is only visible when you clear the **Inherit estimation order from input dimensions** check box.

Inherit FFT length from estimation order
When you select the **Inherit FFT length from estimation order** check box, the FFT length is one greater than the estimation order. To specify the number of points on which to perform the FFT, clear the **Inherit FFT length from estimation order** check box. You can then specify a power-of-two FFT length using the **FFT length** parameter.

FFT length
Enter the number of data points on which to perform the FFT, N_{fft}. When N_{fft} is larger than the input frame size, the block zero-pads each frame as needed. When N_{fft} is smaller than the input frame size, the block wraps each frame as needed. This parameter becomes visible only when you clear the **Inherit FFT length from input dimensions** check box.

Inherit sample time from input
When you select the **Inherit sample time from input** check box, the block computes the frequency data from the sample period of the input signal. For the block to produce valid output, the following conditions must hold:

- The input to the block is the original signal, with no samples added or deleted (by insertion of zeros, for example).
- The sample period of the time-domain signal in the simulation equals the sample period of the original time series.

If these conditions do not hold, clear the **Inherit sample time from input** check box. You can then specify a sample time using the **Sample time of original time series** parameter.

Sample time of original time series
: Specify the sample time of the original time-domain signal. This parameter becomes visible only when you clear the **Inherit sample time from input** check box.

References

Kay, S. M. *Modern Spectral Estimation: Theory and Application*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

Marple, S. L. Jr., *Digital Spectral Analysis with Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

Orfanidis, S. J. *Introduction to Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating point • Single-precision floating point
Output	• Double-precision floating point • Single-precision floating point

The output data type is the same as the input data type.

See Also

Burg Method	Signal Processing Blockset
Covariance Method	Signal Processing Blockset
Levinson-Durbin	Signal Processing Blockset
Autocorrelation LPC	Signal Processing Blockset
Short-Time FFT	Signal Processing Blockset
Yule-Walker AR Estimator	Signal Processing Blockset
`spectrum.yulear`	Signal Processing Toolbox

See "Power Spectrum Estimation" for related information.

Zero Crossing

Purpose

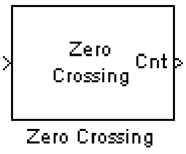
Count number of times signal crosses zero in single time step

Library

Signal Operations

`dspsigops`

Description



The Zero Crossing block concludes that a signal in a given channel has passed through zero if it meets any of the following criteria, where x_i is the current signal value, x_{i-1} is the previous signal value, and so on:

- $x_i < 0$ and $x_{i-1} > 0$
- $x_i > 0$ and $x_{i-1} < 0$
- For some positive integer L, $x_i < 0$, $x_{i-l} = 0$, and $x_{i-L-1} > 0$, where $0 \le l \le L$.
- For some positive integer L, $x_i > 0$, $x_{i-l} = 0$, and $x_{i-L-1} < 0$, where $0 \le l \le L$.

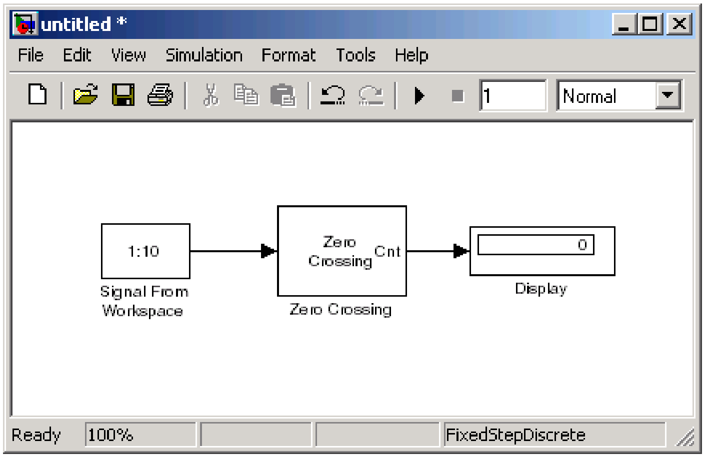
For the first input value, x_{i-1} and x_{i-2} are zero. The block outputs the number of times the signal crosses zero in a single time step at the Cnt port.

The input to this block must be a real-valued fixed-point or floating-point signal. If the input is a sample-based vector or matrix, then each entry is treated as a time-varying channel. If the input is a frame-based row vector of length N, it is treated as N independent channels. If the input is a frame-based column vector, it is treated as a single channel.

Examples

The following example illustrates the behavior of the Zero Crossing block.

1 Create the following Simulink model.



2 Use the Signal From Workspace block to create a frame-based signal. Set the parameters as follows:

- **Signal** = `[-3:3]'`
- **Sample time** = `1/7`
- **Samples per frame** = `7`
- **Form output after final data value by** = `Cyclic repetition`

The block outputs a single frame of the frame-based signal at the first time step, and identical frames at each additional time step.

3 Use the Zero Crossing block to detect the number of zero crossing in each time step. Use the default parameters.

4 Use the Display block to view the number of zero crossings.

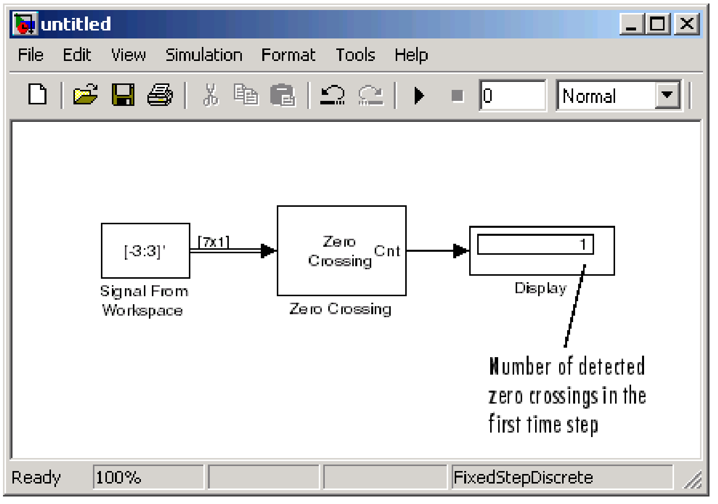
5 To run the model for one time step, set the configuration parameters. Open the Configuration Parameters dialog box by selecting **Configuration Parameters** from the **Simulation** menu. In the **Solver** pane, set the parameters as follows:

- **Stop time** = `0`

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

6 Run the model.

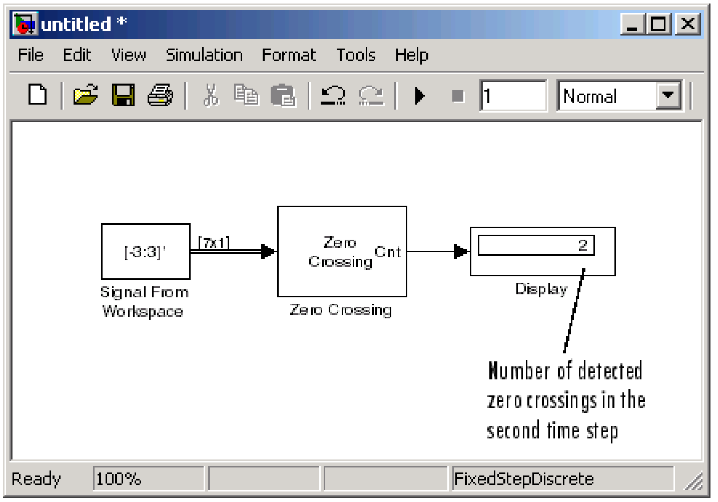
Because the signal passes through zero once during the first time step, the Zero Crossing block finds one zero crossing as shown in the figure below.



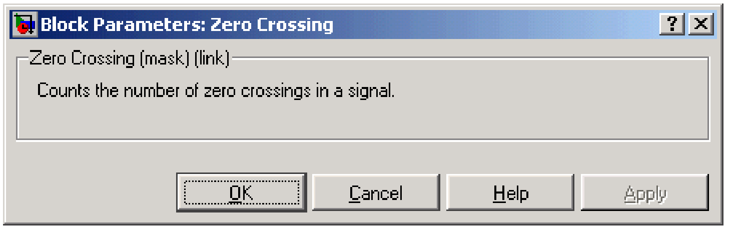
7 To run the model for two time steps, change the simulation **Stop time** to `1`.

8 Run the model.

The Zero Crossing block remembers that the last value of the last frame was 3. Therefore, the signal passes through zero twice during the second time step. It passes through zero while going from 3 to -3, and it passes through zero again while going from -3 to 3. The Zero Crossing block finds two zero crossings in the second time step as shown in the figure below.



Dialog Box



Supported Data Types

Port	Supported Data Types
Input	• Double-precision floating-point • Single-precision floating-point • Fixed point (signed and unsigned) • 8-, 16-, and 32-bit signed integers • 8-, 16-, and 32-bit unsigned integers
Cnt	• 32-bit unsigned integers

See Also Hit Crossing Simulink

3

System Object Reference

Estimation

signalblks.BurgAREstimator	Compute estimate of autoregressive (AR) model parameters using Burg method
signalblks.BurgSpectrumEstimator	Compute parametric spectral estimate using Burg method
signalblks.CepstralToLPC	Convert cepstral coefficients to linear prediction coefficients
signalblks.LPCToAutocorrelation	Convert linear prediction coefficients to autocorrelation coefficients
signalblks.LPCToCepstral	Convert linear prediction coefficients to cepstral coefficients
signalblks.LPCToLSF	Convert linear prediction coefficients to line spectral frequencies
signalblks.LPCToLSP	Convert linear prediction coefficients to line spectral pairs
signalblks.LPCToRC	Convert linear prediction coefficients to reflection coefficients
signalblks.LSFToLPC	Convert line spectral frequencies to linear prediction coefficients
signalblks.LSPToLPC	Convert line spectral pairs to linear prediction coefficients
signalblks.RCToAutocorrelation	Convert reflection coefficients to autocorrelation coefficients
signalblks.RCToLPC	Convert reflection coefficients to linear prediction coefficients

Filtering

signalblks.BiquadFilter	IIR filter using biquadratic structures
signalblks.BlockLMSFilter	Compute output, error, and weights using Block LMS adaptive algorithm
signalblks.CICDecimator	Decimate input using Cascaded Integrator-Comb filter
signalblks.CICInterpolator	Interpolate signal using Cascaded Integrator-Comb filter
signalblks.DigitalFilter	Static or time-varying digital filter
signalblks.DyadicAnalysisFilterBank	Dyadic analysis filter bank
signalblks.DyadicSynthesisFilterBank	Reconstruct signals from subbands with smaller bandwidths and smaller sample rates
signalblks.FIRDecimator	FIR polyphase decimator
signalblks.FIRInterpolator	Polyphase FIR interpolator
signalblks.FIRRateConverter	Sample rate converter
signalblks.LMSFilter	LMS adaptive filter
signalblks.SubbandAnalysisFilter	Decompose signal into high-frequency and low-frequency subbands
signalblks.SubbandSynthesisFilter	Reconstruct signal from high-frequency and low-frequency subbands

Math Functions

signalblks.ArrayVectorAdder	Add array to vector along specified dimension
signalblks.ArrayVectorDivider	Divide array by vector along specified dimension
signalblks.ArrayVectorMultiplier	Multiply array by vector along specified dimension
signalblks.ArrayVectorSubtractor	Subtract vector from array along specified dimension
signalblks.CumulativeProduct	Compute cumulative product of channel, column, or row elements
signalblks.CumulativeSum	Compute cumulative sum of channel, column, or row elements
signalblks.LDLFactor	Factor square Hermitian positive definite matrices into lower, upper, and diagonal components
signalblks.LevinsonSolver	Solve linear system of equations using Levinson-Durbin recursion
signalblks.LowerTriangularSolver	Solve lower-triangular matrix equation
signalblks.LUFactor	Factor square matrix into lower and upper triangular matrices
signalblks.Normalizer	Perform vector normalization along the specified dimension
signalblks.UpperTriangularSolver	Solve upper-triangular matrix equation

Quantizers

signalblks.ScalarQuantizerDecoder	Convert each index value into quantized output value
signalblks.ScalarQuantizerEncoder	Encode each input value by associating it with index value of quantization region
signalblks.UniformDecoder	Decode integer input into floating-point output
signalblks.UniformEncoder	Quantize and encode floating-point input into integer output
signalblks.VectorQuantizerDecoder	Return vector quantizer codeword for given index value
signalblks.VectorQuantizerEncoder	Perform vector quantization encoding

Signal Management

signalblks.Buffer	Buffer input signal
signalblks.Counter	Count up or down through specified range of numbers
signalblks.DelayLine	Rebuffer sequence of inputs with one-sample shift

Signal Operations

signalblks.Convolver	Convolution of two inputs
signalblks.Delay	Delay input by specified number of samples or frames
signalblks.Interpolator	Linear or FIR interpolation
signalblks.NCO	Generate real or complex sinusoidal signals
signalblks.PeakFinder	Determine extrema (maxima or minima) in input signal
signalblks.PhaseUnwrapper	Unwrap signal phase
signalblks.VariableFractionalDelay	Delay input by time-varying fractional number of sample periods
signalblks.VariableIntegerDelay	Delay input by time-varying integer number of sample periods
signalblks.Window	Window object
signalblks.ZeroCrossingDetector	Zero crossing detector

Signal Processing Sinks

signalblks.AudioPlayer	Play audio data using the computer's audio device
signalblks.MultimediaFileWriter	Write video frames, audio samples, or both to multimedia file
signalblks.SignalLogger	Log simulation data in buffer

Signal Processing Sources

signalblks.AudioRecorder	Record audio data using computer's audio device
signalblks.Chirp	Generate swept-frequency cosine (chirp) signal
signalblks.MultimediaFileReader	Read video frames, audio samples, or both from multimedia file
signalblks.SignalReader	Import variable from workspace
signalblks.SineWave	Discrete-time sinusoid

Statistics

signalblks.Autocorrelator	Autocorrelation sequence
signalblks.Crosscorrelator	Cross-correlation of two inputs
signalblks.Histogram	Generate histogram of input or sequence of inputs
signalblks.Maximum	Find maximum value of input or sequence of inputs
signalblks.Mean	Find mean value of input or sequence of inputs
signalblks.Median	Median value of input
signalblks.Minimum	Find minimum values of input or sequence of inputs
signalblks.RMS	Compute root-mean-square of the vector elements
signalblks.StandardDeviation	Compute standard deviation of input or sequence of inputs
signalblks.Variance	Variance of input or sequence of inputs

Transforms

signalblks.AnalyticSignal	Compute analytic signals of discrete-time inputs
signalblks.DCT	Discrete cosine transform (DCT)
signalblks.FFT	Discrete Fourier transform
signalblks.IDCT	Inverse discrete cosine transform (IDCT)
signalblks.IFFT	Inverse discrete Fourier transform (IDFFT)

Alphabetical List

signalblks.AnalyticSignal class

Purpose

Compute analytic signals of discrete-time inputs

Description

The `AnalyticSignal` object computes analytic signals of discrete-time inputs. The real part of the analytic signal in each channel is a replica of the real input in that channel, and the imaginary part is the Hilbert transform of the input. In the frequency domain, the analytic signal doubles the positive frequency content of the original signal while zeroing-out negative frequencies and retaining the DC component. The object computes the Hilbert transform using an equiripple FIR filter.

Construction

`H = signalblks.AnalyticSignal` returns an analytic signal object, `H`, that computes the complex analytic signal corresponding to each channel of a real M-by-N input matrix.

`H =`
`signalblks.AnalyticSignal('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an analytic signal object, `H`, with each specified property set to the specified value.

`H =`
`signalblks.AnalyticSignal(order,'`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an analytic signal object, `H`, with the `FilterOrder` property set to `order` and other specified properties set to the specified values.

Properties

`FilterOrder`

Filter order used to compute Hilbert transform

Specify the order of the equiripple FIR filter used in computing the Hilbert transform as an even integer scalar. The default value of this property is `100`.

`FrameBasedProcessing`

Process input as frames or samples

Set this property to `true` to enable frame-based processing. Set this property to `false` to enable sample-based processing. The default value of this property is `true`.

Methods

clone	Create analytic signal object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Return logical value to indicate whether input attributes and non-tunable properties are locked
reset	Reset internal states of analytic signal object
step	Compute analytic signal

Examples

Compute the analytic signal of a sinusoidal input.

```
t = (-1:0.01:1)';
x = sin(4*pi*t);
hanlytc = signalblks.AnalyticSignal(200);
% Filter order = 200
y = step(hanlytc, x);
subplot(2,1,1), plot(t, x);
title('Original Signal');
subplot(2,1,2), plot(t, [real(y) imag(y)]);
title('Analytic signal of the input')
legend('Real signal','Imaginary signal',...
'Location','best');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Analytic Signal block reference page. The object properties correspond to the block parameters.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See “What Are Sample- and Frame-Based Processing?” for more information.

See Also

`signalblks.FFT` | `signalblks.IFFT`

Purpose Create analytic signal object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an `AnalyticSignal` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

signalblks.AnalyticSignal.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Number of outputs from step method

Syntax

```
getNumOutputs(H)
```

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.AnalyticSignal.isLocked

Purpose Return logical value to indicate whether input attributes and non-tunable properties are locked

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `AnalyticSignal` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset internal states of analytic signal object

Syntax

```
reset(H)
```

Description `reset(H)` sets the internal states of the `AnalyticSignal` object `H` to their initial values.

signalblks.AnalyticSignal.step

Purpose Compute analytic signal

Syntax `Y = step(H,X)`

Description `Y = step(H,X)` computes the analytic signal, `Y`, of the M-by-N input matrix `X`, according to the equation

$$\mathbf{Y} = \mathbf{X} + jH\{\mathbf{X}\}$$

where j is the imaginary unit and $H\{\mathbf{X}\}$ denotes the Hilbert transform.

When you set the `FrameBasedProcessing` property to `false`, each of the M-by-N matrix elements represents an independent channel. Thus, the method computes the analytic signal for each element of `X`. When you set the `FrameBasedProcessing` property to `true`, each of the N columns in `X` contains M sequential time samples from an independent channel, and the method computes the analytic signal for each channel.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.ArrayVectorAdder class

Purpose

Add array to vector along specified dimension

Description

The `ArrayVectorAdder` object adds an N-D array to a vector along a specified dimension. The length of the vector must be the same as the size of the N–D array along the specified dimension.

Construction

`H = signalblks.ArrayVectorAdder` returns an array-vector addition object, `H`, that adds a vector to an N-D array along the first dimension.

`H = signalblks.ArrayVectorAdder('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns an array-vector addition object, `H`, with each property set to the specified value.

Properties

`Dimension`

Dimension along which to add vector elements to input

Specify the dimension along which to add the input array to the elements of the vector as a positive integer. The length of the vector must match the size of the N-D array along the specified dimension. The default value for this property is `1`.

`VectorSource`

Source of vector

Specify the source of the vector values as `Input port` or `Property`. This default value of this property is `Input port`.

`Vector`

Vector values

Specify the vector values. This property applies only when you set the `VectorSource` property to `Property`. The default value for this property is `[0.5 0.25]`. This property is tunable.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor` , `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`VectorDataType`

Vector word and fraction lengths

Specify the vector fixed-point data type as one of `Same word length as input` or `Custom`. This property applies when you set the `VectorSource` property to `Property`. The default value of this property is `Same word length as input`.

`CustomVectorDataType`

Vector word and fraction lengths

Specify the vector fixed-point type as a `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `VectorSource` property to `Property` and the `VectorDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Internal rule` , `Same as first input`, or `Custom`. The default value of this property is `Internal rule`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `AccumulatorDataType` property is `Custom`. The default value of this property is `numerictype([],32,30)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as one of `Same as accumulator` , `Same as first input`, or `Custom`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `OutputDataType` property is `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create array-vector adder object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Return number of outputs of step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Add vector to N-D array

Examples

Add 2x1 vector to 2x2 matrix along the first dimension of the array:

```
hava = signalblks.ArrayVectorAdder;
a = ones(2);
x = [1 2]';
```

```
y = step(hava, a, x);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Array-Vector Add block reference page. The object properties correspond to the block parameters, except for:

- The array–vector addition object does not have **Minimum** or **Maximum** options for data output.

See Also

`signalblks.ArrayVectorMultiplier` | `signalblks.ArrayVectorDivider` | `signalblks.ArrayVectorSubtractor`

Purpose Create array-vector adder object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an array–vector adder object, `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.ArrayVectorAdder.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs of step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.ArrayVectorAdder.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the array-vector adder.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Add vector to N-D array

Syntax

```
Y = step(H,A)
Y = step(H,A,V)
```

Description

`Y = step(H,A)` returns `Y` , the result of adding the input array `A` to the elements of the vector specified in the `Vector` property along the specified dimension when the `VectorSource` property is `Property`. The length of the vector specified in the `Vector` property must be the same as the length of the specified dimension of `A`.

`Y = step(H,A,V)` returns `Y` , the result of adding the input array `A` to the elements of the input vector `V` along the specified dimension when the `VectorSource` property is `Input port`. The length of the input `V` must be the same as the length of the specified dimension of `A`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.ArrayVectorDivider class

Purpose Divide array by vector along specified dimension

Description The `ArrayVectorDivider` object divides an array by a vector along the specified dimension.

Construction `H = signalblks.ArrayVectorDivider` returns an array–vector division object, `H`, that divides an input array by the elements of a vector along the first dimension of the array.

`H = signalblks.ArrayVectorDivider('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an array–vector division object, `H`, with each property set to the specified value.

Properties

`Dimension`

Dimension along which to divide input by vector elements

Specify the dimension along which to divide the input array by the elements of a vector as a positive integer. The default value for this property is `1`.

`VectorSource`

Source of vector

Specify the source of the vector values as one of `Input port` or `Property`. This default value for this property is `Input port`.

`Vector`

Vector values

Specify the vector values. This property applies when you set the `VectorSource` property to `Property`. The default value for this property is `[0.5 0.25]`. This property is tunable.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling`, `Convergent`, `Floor` , `Nearest`, `Round`, `Simplest`, or `Zero`. This default value for this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` or `Saturate`. This default value for this property is `Wrap`.

`VectorDataType`

Vector word and fraction lengths

Specify the vector fixed-point mode as one of `Same word length as input` or `Specify numeric type`. This property applies when you set the `VectorSource` property to `Property`. The default value for this property is `Same word length as input`.

`CustomVectorDataType`

Vector word and fraction lengths

Specify the vector fixed-point data type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `VectorSource` property to `Property` and the `VectorDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as one of `Same as first input` or `Custom`. This default value for this property is `Same as first input`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create array-vector division object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Divide array by vector

Examples

Divide a matrix by a vector:

```
havd = signalblks.ArrayVectorDivider;
a = ones(2);
x = [2 3]';
y = step(havd, a, x);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Array-Vector Divide block reference page. The object properties correspond to the block parameters, except for:

- The array–vector division object does not have **Minimum** or **Maximum** options for data output.

See Also

`signalblks.ArrayVectorMultiplier` | `signalblks.ArrayVectorAdder` | `signalblks.ArrayVectorSubtractor`

Purpose Create array-vector division object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an array-vector division object, `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.ArrayVectorDivider.getNumInputs

Purpose

Return number of expected inputs to step method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.ArrayVectorDivider.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the array-vector divider.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Divide array by vector

Syntax

`Y = step(H,A,V)`
`Y = step(H,A)`

Description

`Y = step(H,A,V)` returns `Y` , the result of dividing the input array `A` by the elements of input vector `V` along the specified dimension when the `VectorSource` property is `Input port`. The length of the input `V` must be the same as the length of the specified dimension of `A`.

`Y = step(H,A)` returns `Y` , the result of dividing the input array `A` by the elements of the vector specified in the `Vector` property along the specified dimension when the `VectorSource` property is `Property`. The length of the vector specified in the `Vector` property must be the same as the length of the specified dimension of `A`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.ArrayVectorMultiplier class

Purpose

Multiply array by vector along specified dimension

Description

The `ArrayVectorMultiplier` object multiplies an array by a vector along a specified dimension.

Construction

`H = signalblks.ArrayVectorMultiplier` returns an array–vector multiplication object, `H`, that multiplies an input N-D array by the elements of a vector along the second dimension.

`H = signalblks.ArrayVectorMultiplier('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an array–vector multiplication object, `H`, with each property set to the specified value.

Properties

`Dimension`

Dimension along which to multiply input by vector elements

Specify the dimension along which to multiply the input array by the elements of vector as a positive integer. The default value for this property is `2`.

`VectorSource`

Source of vector

Specify the source of the vector values as one of `Input port` or `Property`. The default value for this property is `Input port`.

`Vector`

Vector to multiply array

Specify the vector by which to multiply the array. This property applies when you set the `VectorSource` property to `Property`. The default value for this property is `[0.5 0.25]`. This property is tunable.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as Ceiling, Convergent, Floor , Nearest, Round, Simplest', Zero. The default value for this property is floor.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as Wrap or Saturate. The default value for this property is Wrap.

VectorDataType

Vector word and fraction lengths

Specify the vector fixed-point data type as Same word length as input , Custom. This property applies when you set the VectorSource property to Property. The default value for this property is Same word length as input.

CustomVectorDataType

Vector word and fraction lengths

Specify the vector fixed-point type as a numerictype object with a Signedness of Auto. This property applies when you set the VectorSource property to Property and the VectorDataType property to Custom. The default value of this property is numerictype([],16,15).

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as Internal rule , Same as first input, or Custom. The default value for this property is Internal rule.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies when you set the ProductDataType property to Custom. The default value of this property is numerictype([],32,30).

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as Internal rule , Same as product, Same as first input, or Custom. The default value for this property is Internal rule.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies when you set the AccumulatorDataType property to Custom. The default value of this property is numerictype([],32,30).

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as Same as product, Same as first input, or Custom. The default value for this property is Same as product.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies when you set the OutputDataType property to Custom. The default value of this property is numerictype([],16,15).

Methods

clone	Create array-vector multiplication object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Multiply array by vector

Examples

Multiply a matrix by a vector:

```
havm = signalblks.ArrayVectorMultiplier;
a = ones(2);
x = [2 3]';
y = step(havm, a, x);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Array-Vector Multiply block reference page. The object properties correspond to the block parameters, except for:

- The array–vector multiplication object does not have **Minimum** or **Maximum** options for data output.

See Also

signalblks.ArrayVectorAdder | signalblks.ArrayVectorDivider | signalblks.ArrayVectorSubtractor

signalblks.ArrayVectorMultiplier.clone

Purpose Create array-vector multiplication object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an array-vector multiplication object, `C`, with the same property values as `H`. The clone method creates a new unlocked object.

Purpose

Return number of expected inputs to step method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.ArrayVectorMultiplier.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of output arguments from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax

```
isLocked(H)
```

Description `isLocked(H)` returns the locked state of the array-vector multiplication object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.ArrayVectorMultiplier.step

Purpose

Multiply array by vector

Syntax

```
Y = step(H,A,V)
Y = step(H,A)
```

Description

`Y = step(H,A,V)` returns `Y` , the result of multipling the input array `A` by the elements of input vector `V` along the specified dimension when the `VectorSource` property is `Input port`. The length of the input `V` must be the same as the length of the specified dimension of `A`.

`Y = step(H,A)` returns `Y` , the result of multipling the input array `A` by the elements of vector specified in `Vector` property along the specified dimension when the `VectorSource` property is set to `Property`. The length of the vector specified in `Vector` property must be the same as the length of the specified dimension of `A`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.ArrayVectorSubtractor class

Purpose

Subtract vector from array along specified dimension

Description

The `ArrayVectorSubtractor` object subtracts a vector from an N-D array along the specified dimension.

Construction

`H = signalblks.ArrayVectorSubtractor` returns an array–vector subtraction object, `H`, that subtracts the elements of a vector from an N-D input array along the first dimension.

`H = signalblks.ArrayVectorSubtractor('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an array–vector subtraction object, `H`, with each property set to the specified value.

Properties

`Dimension`

Dimension along which to subtract vector elements from input

Specify the dimension along which to subtract the elements of the vector from the input array as an integer-valued scalar greater than `0`. The default value for this property is `1`.

`VectorSource`

Source of vector

Specify the source of the vector values as one of `Input port` or `Property`. The default value for this property is `Input port`.

`Vector`

Vector values

Specify the vector values. This property applies when you set the `VectorSource` property to `Property`. The default value for this property is `[0.5 0.25]`. This property is tunable.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor` , `Nearest`, `Round`, `Simplest`, or `Zero`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` or `Saturate`. The default value for this property is `Wrap`.

`VectorDataType`

Vector word and fraction lengths

Specify the vector fixed-point data type as `Same word length as input` or `Custom`. This property applies when you set the `VectorSource` property to `Property`. The default value for this property is `Same word length as input`.

`CustomVectorDataType`

Vector word and fraction lengths

Specify the vector fixed-point type as a `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `VectorSource` property to `Property` and the `VectorDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Internal rule` , `Same as first input`, or `Custom`. The default value for this property is `Internal rule`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as `Same as accumulator`, `Same as first input`, or `Custom`. The default value for this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create array-vector subtractor with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Subtract vector from array along specified dimension

Examples

Subtract a vector from a matrix:

```
havs = signalblks.ArrayVectorSubtractor;
a = ones(2);
x = [1 2]';
```

```
y = step(havs, a, x);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Array-Vector Subtract block reference page. The object properties correspond to the block parameters, except for:

- The array–vector subtraction object does not have **Minimum** or **Maximum** options for data output.

See Also

`signalblks.ArrayVectorMultiplier` | `signalblks.ArrayVectorDivider` | `signalblks.ArrayVectorAdder`

Purpose Create array-vector subtractor with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an array–vector subtractor object, `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.ArrayVectorSubtractor.getNumInputs

Purpose

Return number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Number of outputs from step method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.ArrayVectorSubtractor.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the array-vector subtractor object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Subtract vector from array along specified dimension

Syntax

```
Y = step(H,A,V)
Y = step(H,A)
```

Description

`Y = step(H,A,V)` returns `Y` , the result of subtracting the elements of input vector `V` from the input array `A` along the specified dimension when the `VectorSource` property is `Input port`. The length of the input `V` must be the same as the length of the specified dimension of `A`.

`Y = step(H,A)` returns `Y` , the result of subtracting the elements of the vector specified in the `Vector` property from the input array `A` along the specified dimension when the `VectorSource` property is `Property`. The length of the vector specified in the `Vector` property must be the same as the length of the specified dimension of `A`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.AudioPlayer class

Purpose

Play audio data using the computer's audio device

Description

The `AudioPlayer` object plays audio data using the computer's audio device.

Construction

`H = signalblks.AudioPlayer` returns an audio player object, `H`, that plays audio samples using an audio output device in real-time.

`H = signalblks.AudioPlayer('PropertyName',PropertyValue, ...)` returns an audio player object, `H`, with each property set to the specified value.

`H = signalblks.AudioPlayer(SAMPLERATE,'PropertyName',PropertyValue, ...)` returns an audio player object, `H`, with the `SampleRate` property set to `SAMPLERATE` and other specified properties set to the specified values.

Properties

`DeviceName`

Device to which to send audio data

Specify the device to which to send the audio data. The default value for this property is `Default`, which is the computer's standard output device. You can use tab completion to query valid `DeviceName` assignments for your computer. To use the tab completion for this property, enter the following at the MATLAB command prompt:

```
H = signalblks.AudioPlayer;
% Type the following and hit the Tab key
% after the single quote.
H.DeviceName = '
```

The tab completion functionality gives all valid audio device names for your computer. Choose the appropriate device.

`SampleRate`

Number of samples per second sent to audio device

Specify the number of samples per second in the signal as an integer. The default value of this property is 44100.

DeviceDataType

Data type used by the device

Specify the data type used by the audio device to acquire audio data as Determine from input data type , 8-bit integer, 16-bit integer, 24-bit integer, or 32-bit float. The default value for this property is Determine from input data type.

BufferSizeSource

Source of Buffer Size

Specify how to determine the buffer size as Auto or Property. The default value for this property is Auto.

BufferSize

Buffer size

Specify the size of the buffer that the audio player object uses to communicate with the audio device as an integer. This property applies when you set the BufferSizeSource property to Property. The default value of this property is 4096.

QueueDuration

Size of queue in seconds

Specify the length of the audio queue, in seconds. The default value of this property is 1.0.

Methods

clone	Create audio player object with same property values
close	Release audio output device
getNumInputs	Number of expected inputs to step method

getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Write audio to audio output device

Examples

Read in an AVI audio file and play it back using the standard audio output device:

```
hmfr = signalblks.MultimediaFileReader;
hap = signalblks.AudioPlayer('SampleRate',22050);
while ~isDone(hmfr)
audio = step(hmfr);
step(hap, audio);
end
pause(1); % Wait until audio is played to the end
close(hmfr); % close the input file
close(hap); % close the audio output device
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the To Audio Device block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.AudioRecorder` | `signalblks.MultimediaFileWriter`.

Purpose Create audio player object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an audio player object, `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.AudioPlayer.close

Purpose Release audio output device

Syntax `close(H)`

Description `close(H)` releases the audio output device used by the audio player object.

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.AudioPlayer.getNumOutputs

Purpose Number of outputs from step method

Syntax

```
getNumOutputs(H)
```

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose
Locked status (logical) for input attributes and non-tunable properties

Syntax
`isLocked(H)`

Description
`isLocked(H)` returns the locked state of the audio player object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.AudioPlayer.step

Purpose Write audio to audio output device

Syntax `step(H,AUDIO)`

Description `step(H,AUDIO)` writes one frame of audio samples to the selected audio output device.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Record audio data using computer's audio device

Description

The `AudioRecorder` object records audio data using the computer's audio device.

Construction

`H = signalblks.AudioRecorder` returns an audio recorder object, `H`, that records audio samples using an audio input device in real-time.

`H = signalblks.AudioRecorder('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns an audio recorder object, `H`, with each property set to the specified value.

Properties

`DeviceName`

Device from which to acquire audio data

Specify the device from which to acquire audio data. The default value for this property is `Default`, which is the computer's standard input device. You can use tab completion to query valid `DeviceName` assignments for your computer. To use the tab completion for this property, enter the following at the MATLAB command prompt:

```
H = signalblks.AudioRecorder;
% Type the following and hit the Tab key
% after the single quote.
H.DeviceName = '
```

The tab completion functionality gives all valid audio device names for your computer. Choose the appropriate device.

`SampleRate`

Number of samples per second read from audio device

Specify the number of samples per second in the signal as an integer. The default value of this property is 44100.

`NumChannels`

Number of audio channels

Specify the number of audio channels as an integer. The default value of this property is 2.

`DeviceDataType`

Data type used by the device

Specify the data type used by the device to acquire audio data as `Determine from output data type` , `8-bit integer`, `16-bit integer`, `24-bit integer`, or `32-bit float`. The default value for this property is `Determine from output data type`.

`BufferSizeSource`

Source of Buffer Size

Specify how to determine the buffer size as `Auto` or `Property`. The default value for this property is `Auto`.

`BufferSize`

Buffer size

Specify as an integer the size of the buffer that the audio recorder object uses to communicate with the audio device. This property applies when you set the `BufferSizeSource` property to `Property`. The default value of this property is `4096`.

`QueueDuration`

Size of queue in seconds

Specify the length of the audio queue, in seconds. The default value of this property is 1.0.

`SamplesPerFrame`

Number of samples in the output signal

Specify the number of samples in the audio recorder's output as an integer. The default value of this property is 1024.

`OutputDataType`

Data type of the output

Select the output data type as `uint8`, `int16`, `int32` , `single` , or `double`. The default value for this property is `single`.

Methods

clone	Create audio recorder object with same property values
close	Release audio input source
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs of step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Record audio from computer's recording device

Examples

Record ten seconds of speech from a microphone and send the output to a .wav file:

```
har = signalblks.AudioRecorder;
hmfw = signalblks.MultimediaFileWriter('myspeech.wav');
disp('Speak into microphone now');
tic;
while toc < 10,
step(hmfw, step(har));
end
close(har);
close(hmfw);
disp('Recording complete');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the From Audio Device block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.AudioPlayer` | `signalblks.MultimediaFileReader`.

Purpose Create audio recorder object with same property values

Syntax `clone(H)`

Description `clone(H)` creates an audio recorder object, `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.AudioRecorder.close

Purpose Release audio input source

Syntax `close(H)`

Description `close(H)` releases the audio input source used by the audio recorder.

signalblks.AudioRecorder.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.AudioRecorder.getNumOutputs

Purpose Return number of outputs of step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the audio recorder.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.AudioRecorder.step

Purpose Record audio from computer's recording device

Syntax

```
AUDIO = step(H)
```

Description `AUDIO = step(H)` reads one frame of audio samples from the selected audio input device.

Purpose

Autocorrelation sequence

Description

The `Autocorrelator` object returns the autocorrelation sequence for a discrete-time deterministic input, or the autocorrelation sequence estimate for a discrete-time wide-sense stationary (WSS) random process at positive lags.

Construction

`H = signalblks.Autocorrelator` returns an autocorrelator, `H`, that computes the autocorrelation along the first dimension of an N-D array. By default, the autocorrelator computes the autocorrelation at lags from zero to *N-1*, where *N* is the length of the input vector or the row dimension of the input matrix. Inputting a row vector results in a row of zero-lag autocorrelation sequence values, one for each column of the row vector. The default autocorrelator returns the unscaled autocorrelation and performs the computation in the time domain.

`H = signalblks.Autocorrelator('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns an autocorrelator, `H`, with each property set to the specified value.

Properties

`MaximumLagSource`

Source of the maximum lag

Specify how to determine the range of lags for the autocorrelation as `Auto` or `Property`. If the value of `MaximumLagSource` is `Auto`, the autocorrelator computes the autocorrelation over all nonnegative lags in the interval `[0, `*`N`*`-1]`, where *N* is the length of the first dimension of the input. Otherwise, the object computes the autocorrelation using lags in the range `[0,MaximumLag]`. The default value for this property is `Auto`.

`MaximumLag`

Maximum positive lag

Specify the maximum lag as a positive integer. This property applies only when the `MaximumLagSource` property is `Property`.

The MaximumLag must be less than the length of the input data. The default value for this property is 1.

Scaling

Autocorrelation function scaling

Specify the scaling to apply to the output as None, Biased, Unbiased, or Unity at zero-lag. Set this property to None to generate the autocorrelation function without scaling. This option is appropriate if you are computing the autocorrelation of a nonrandom (deterministic) input.

The Biased option scales the autocorrelation by 1/N, where *N* is the length of the input data. Scaling by 1/N yields a biased finite-sample approximation to the theoretical autocorrelation of a WSS random process. In spite of the bias, scaling by 1/N has the desirable property that the sample autocorrelation matrix is nonnegative definite, a property possessed by the theoretical autocorrelation matrices of all wide-sense stationary random processes. The Fourier transform of the biased autocorrelation estimate is the *periodogram*, a widely used estimate of the power spectral density of a WSS process.

The Unbiased option scales the estimate of the autocorrelation by 1/N-1. Scaling by *N-1* produces an unbiased estimate of the theoretical autocorrelation. However, using the unbiased option, it is possible to obtain an estimate of the autocorrelation function that fails to have the nonnegative definite property.

Use the Unity at zero-lag option to normalize the autocorrelation estimate to be identically one at lag zero. See "Definitions" on page 4-69 for more detail on scaling. The default value for this property is None.

Method

Domain for computing autocorrelations

Specify the domain for computing autocorrelations as Time Domain or Frequency Domain. This property must be set to Time

Domain for fixed-point signals. The default value for this property is Time Domain.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as Ceiling, Convergent, Floor, Nearest, Round, Simplest, or Zero. This property applies only when you set the Method property to Time Domain. The default value for this property is Floor.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as Wrap or Saturate. This property applies only when you set the Method property to Time Domain. The default value for this property is Wrap.

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as one of Internal rule, Same as input, or Custom. This property applies only when you set the Method property to Time Domain. The default value for this property is Internal rule

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only when you set the Method property to Time Domain and the ProductDataType property to Custom. The default value of this property is numerictype([],32,30).

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Internal rule`, `Same as product`, `Same as input`, or `Custom`. This property applies only when the `Method` property is `Time Domain`. The default value for this property is `Internal rule`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `Method` property to `Time Domain` and the `AccumulatorDataType` property to `Custom`. The default value of this property is`numerictype([],32,30)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as `Same as accumulator`, `Same as product`, `Same as input`, or `Custom`. This property applies only when the `Method` property is `Time Domain`. The default value for this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `Method` property to `Time Domain` and the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create autocorrelator object with same property values
getNumInputs	Return number of expected inputs to step method

getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Compute autocorrelation sequence

Definitions

The autocorrelation of a deterministic discrete-time sequence, *x(n)* is:

$$r_x(h) = \sum_{n=0}^{N-h-1} x^*(n)x(n+h) \quad h = 0,1,\ldots,N-1$$

where h is the lag and * denotes the complex conjugate. If the input is a length N realization of a WSS stationary random process, $r_x(h)$ is an estimate of the theoretical autocorrelation:

$$\rho_x(h) = E\{x^*(n)x(n+h)\}$$

where E{ } is the expectation operator. The `Unity at zero-lag` normalization divides each sequence value by the autocorrelation or autocorrelation estimate at zero lag.

$$\frac{\rho_x(h)}{\rho_x(0)} = \frac{E\{x^*(n)x(n+h)\}}{E\{|x(0)|^2\}}$$

The most commonly used estimate of the theoretical autocorrelation of a WSS random process is the biased estimate:

$$\hat{\rho}_x(h) = \frac{1}{N}\sum_{k=0}^{N-h-1} x^*(n)x(n+h)$$

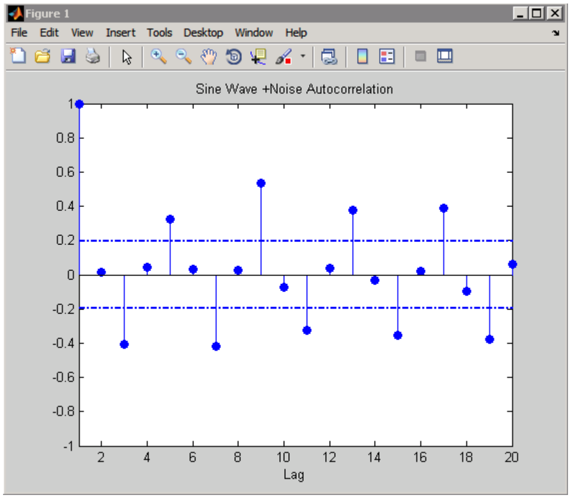
Examples

Compute autocorrelation for all positive lags

```
hac1 = signalblks.Autocorrelator;
% x is a column vector
x = [1:100]';
y = step(hac1, x);
```

Autocorrelation of sine wave in white Gaussian noise with approximate 95% upper and lower confidence limits:

```
S = RandStream('mt19937ar');
RandStream.setDefaultStream(S);
% Sine wave with period N=4
x = 1.4*cos(pi/2*(1:100))'+randn(100,1);
MaxLag = 20;
H = signalblks.Autocorrelator('MaximumLagSource',...
'Property','MaximumLag',MaxLag,'Scaling','Unity at zero-lag');
SigAutocorr = step(H,x);
stem(SigAutocorr,'b','markerfacecolor',[0 0 1]);
line(1:MaxLag+1,1.96/sqrt(100)*ones(MaxLag+1,1),...
     'linestyle','-.','linewidth',2);
line(1:MaxLag+1,-1.96/sqrt(100)*ones(MaxLag+1,1),...
     'linestyle','-.','linewidth',2);
axis([1 20 -1 1]);
title('Sine Wave +Noise Autocorrelation'); xlabel('Lag');
```



Note in the above figure that the autocorrelation estimate demonstrates the four sample periodic sine wave with excursions outside the 95% white Gaussian noise confidence limits every two samples.

Algorithm

This object implements the algorithm, inputs, and outputs described on the Autocorrelation block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.Crosscorrelator`

signalblks.Autocorrelator.clone

Purpose Create autocorrelator object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an autocorrelator object, `C`, with the same property values as `H`. The `clone` method creates a new unlocked object.

Purpose

Return number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.Autocorrelator.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs of the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the autocorrelator.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.Autocorrelator.step

Purpose Compute autocorrelation sequence

Syntax `Y = step(H,X)`

Description `Y = step(H,X)` computes the autocorrelation sequence `Y` for the columns of the input `X`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.BiquadFilter class

Purpose

IIR filter using biquadratic structures

Description

The `BiquadFilter` object implements an IIR filter structure using biquadratic, or second–order sections (SOS).

Construction

`H = signalblks.BiquadFilter` returns a default biquadratic IIR filter, `H`, which independently filters each channel of the input over time using the SOS section`[1 0.3 0.4 1 0.1 0.2]` with a direct-form II transposed structure.

`H = signalblks.BiquadFilter('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns a biquadratic filter object, `H`, with each property set to the specified value.

Properties

`Structure`

Filter structure

Specify the filter structure as `Direct form I`, `Direct form I transposed`, `Direct form II`, or `Direct form II transposed`. The default value of this property is `Direct form II transposed`.

`SOSMatrixSource`

SOS Matrix source

Specify the source of the SOS Matrix as `Property` or `Input port`. The default value of this property is `Property`.

`SOSMatrix`

SOS matrix

Specify the second-order section (SOS) matrix as an `N`-by-6 matrix, where `N` is the number of sections in the filter. Each row of the SOS matrix contains the numerator and denominator coefficients of the corresponding section in the filter. The system function, *H(z)*, of a biquad filter is:

$$H(z) = \frac{\sum_{k=0}^{2} b_k z^{-k}}{1 - \sum_{l=1}^{2} a_l z^{-l}}$$

The coefficients are ordered in the rows of the SOS matrix as (b_0, b_1, b_2, 1, $-a_1$, $-a_2$). The coefficients can be real– or complex-valued. This property applies only when you set the `SOSMatrixSource` property to `Property`. The biquad filter assumes that the leading denominator coefficient, a_0, is equal to one for each filter section, regardless of the specified value. This property is tunable when the `OptimizeUnityScaleValues` property is `false`. The default value of this property is `[1 0.3 0.4 1 0.1 0.2]`.

`ScaleValues`

Scale values for each biquad section

Specify the scale values to be applied before and after each section of a biquad filter. The scale values parameter must be a scalar or a vector of length `N+1`, where `N` is the number of sections. If this property is set to a scalar, the scalar value is used as the gain value only before the first filter section. The remainder of the gain values are set to `1`. If this property is set to a vector of `N+1`values, each value is used for a separate section of the filter. This property applies only when you set the `SOSMatrixSource` property to `Property`. This property is tunable when the `OptimizeUnityScaleValues` property is `false`. The default value of this property is `1`.

`InitialConditions`

Initial conditions for direct form II structures

Specify the initial conditions of the filter states when the `Structure` property is `Direct form II` or `Direct form II transposed`. The number of states or delay elements in a direct-form II biquad filter (zeros and poles) is equal to twice the number of filter sections. The initial conditions can be specified as a scalar, vector or matrix. If this property is set to a scalar value, the biquad filter initializes all delay elements in the filter to that value. If this property is set to a vector whose length is equal to the number of delay elements in the filter, each vector element specifies a unique initial condition for a corresponding delay element. The biquad filter applies the same vector of initial conditions to each channel of the input signal. If this property is set to a vector whose length is equal to the product of the number of input channels and the number of delay elements in the filter, each element specifies a unique initial condition for a corresponding delay element in a corresponding channel. If this property is set to a matrix with the same number of rows as the number of delay elements in the filter, and one column for each channel of the input signal, each element specifies a unique initial condition for a corresponding delay element in a corresponding channel. The default value of this property is `0`.

`NumeratorInitialConditions`

Initial conditions on zeros side

Specify the initial conditions of the filter states on the side of the filter structure with the zeros. This property applies only when you set the `Structure` property to `Direct form I` or `Direct form I transposed`. The number of states or delay elements in the numerator of a direct-form I biquad filter is equal to twice the number of filter sections. The initial conditions can be specified as a scalar, vector or matrix. If this property is set to a scalar value, the biquad filter initializes all delay elements on the zeros side in the filter to that value. If this property is set to a vector whose length is equal to the number of delay elements on the zeros side in the filter, each vector element specifies a unique initial condition for a corresponding delay element on the zeros side. The biquad filter applies the same vector of initial conditions to each channel of the input signal. If this property is set to a vector whose length is equal to the product of the number of input channels and the number of delay elements on the zeros side in the filter, each element specifies a unique initial condition for a corresponding delay element on the zeros side in a corresponding channel. If this property is set to a matrix with the same number of rows as the number of delay elements on the zeros side in the filter, and one column for each channel of the input signal, each element specifies a unique initial condition for a corresponding delay element on the zeros side in a corresponding channel. The default value of this property is `0`.

`DenominatorInitialConditions`

Initial conditions on poles side

Specify the initial conditions of the filter states on the side of the filter structure with the poles. This property only applies when you set the `Structure` property to `Direct form I` or `Direct form I transposed`. The number of denominator states, or delay elements, in a direct-form I (noncanonic) biquad filter is equal to twice the number of filter sections. The initial conditions can be specified as a scalar, vector or matrix. If this property is set to a scalar value, the biquad filter initializes all delay elements on the poles side of the filter to that value. If this property is set to a vector whose length is equal to the number of delay elements on the poles side in the filter, each vector element specifies a unique initial condition for a corresponding delay element on the poles side. The object applies the same vector of initial conditions to each channel of the input signal. If this property is set to a vector whose length is equal to the product of the number of input channels and the number of delay elements on the poles side in the filter, each element specifies a unique initial condition for a corresponding delay element on the poles side in a corresponding channel. If this property is set to a matrix with the same number of rows as the number of delay elements on the poles side in the filter, and one column for each channel of the input signal, each element specifies a unique initial condition for a corresponding delay element on the poles side in a corresponding channel. The default value of this property is `0`.

`OptimizeUnityScaleValues`

Optimize unity scale values

When this Boolean property is set to `true`, the biquad filter removes all unity scale gain computations. This reduces the number of computations and increases the fixed-point accuracy. This property applies only when you set the `SOSMatrixSource` property to `Property`. The default value of this property is `true`.

`ScaleValuesInputPort`

How to specify scale values

Select how scale values are specified. This property applies only when the `SOSMatrixSource` property is `Input port`. By default, this property is `true`, and the scale values are specified as an input port. If this property is `false`, the scale values are all assumed to be unity.

`FrameBasedProcessing`

Enable frame-based processing

Set this property to `true` to enable frame-based processing. When this property is `true`, the biquad filter treats each column as an independent channel. Set this property to `false` to enable sample-based processing. When this property is `false`, the biquad filter treats each element of the input as an individual channel. The default value of this property is `true`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`MultiplicandDataType`

Multiplicand word and fraction lengths

Specify the multiplicand fixed-point data type as `Same as output` or `Custom`. This property applies only when you set the `Structure` property to `Direct form I transposed`. The default value of this property is `Same as output`.

`CustomMultiplicandDataType`

Multiplicand word and fraction lengths

Specify the multiplicand fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `MultiplicandDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`SectionInputDataType`

Section input word and fraction lengths

Specify the section input fixed-point data type as `Same as input` or `Custom`. The default value of this property is `Same as input`.

`CustomSectionInputDataType`

Section input word and fraction lengths

Specify the section input fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `SectionInputDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

`SectionOutputDataType`

Section output word and fraction lengths

Specify the section output fixed-point data type as `Same as section input` or `Custom`. This default value of this property is `Same as section input`.

`CustomSectionOutputDataType`

Section output word and fraction lengths

Specify the section output fixed-point type as a signed, scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `SectionOutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

`NumeratorCoefficientsDataType`

Numerator coefficients word and fraction lengths

Specify the numerator coefficients fixed-point data type as `Same word length as input` or `Custom`. This property applies only when you set the `SOSMatrixSource` property to `Property`. Setting this property will also set the `DenominatorCoefficientsDataType` and `ScaleValuesDataType` properties to the same value. The default value of this property is `Same word length as input`.

`CustomNumeratorCoefficientsDataType`

Numerator coefficients word and fraction lengths

Specify the numerator coefficients fixed-point type as a `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `SOSMatrixSource` property to `Property` and the `NumeratorCoefficientsDataType` property to `Custom`. The word length of the `CustomNumeratorCoefficientsDataType`, `CustomDenominatorCoefficientsDataType` and `CustomScaleValuesDataType` properties must be the same. The default value of this property is `numerictype([],16,15)`.

`DenominatorCoefficientsDataType`

Denominator coefficients word and fraction lengths

Specify the denominator coefficients fixed-point data type as `Same word length as input` or `Custom`. This property applies only when you set the `SOSMatrixSource` property to `Property`. Setting this property will also set the `NumeratorCoefficientsDataType` and `ScaleValuesDataType` properties to the same value. The default value of this property is `Same word length as input`.

`CustomDenominatorCoefficientsDataType`

Denominator coefficients word and fraction lengths

Specify the denominator coefficients fixed-point type as a `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `SOSMatrixSource` property to `Property` and the `DenominatorCoefficientsDataType` property to `Custom`. The word length of the `CustomNumeratorCoefficientsDataType`, `CustomDenominatorCoefficientsDataType`, and `CustomScaleValuesDataType` properties must be the same. The default value of this property is `numerictype([],16,15)`.

`ScaleValuesDataType`

Scale values word and fraction lengths

Specify the scale values fixed-point data type as `Same word length as input` or `Custom`. This property applies only when you set the `SOSMatrixSource` property to `Property`. Setting this property also sets the `NumeratorCoefficientsDataType` and `DenominatorCoefficientsDataType` properties to the same value. The default value of this property is `Same word length as input`.

`CustomScaleValuesDataType`

Scale values word and fraction lengths

Specify the scale values fixed-point type as a `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `SOSMatrixSource` property to `Property` and the `ScaleValuesDataType` property to `Custom`. The word length of the `CustomNumeratorCoefficientsDataType`, `CustomDenominatorCoefficientsDataType`, and

CustomScaleValuesDataType properties must be the same. The default value of this property is numerictype([],16,15).

NumeratorProductDataType

Numerator product word and fraction lengths

Specify the product fixed-point data type as Same as input or Custom. Setting this property will set the DenominatorProductDataType property to the same value. The default value of this property is Same as input.

CustomNumeratorProductDataType

Numerator product word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only when you set the NumeratorProductDataType property to Custom. The word length of the CustomNumeratorProductDataType and CustomDenominatorProductDataType properties must be the same. The default value of this property is numerictype([],32,30).

DenominatorProductDataType

Denominator product word and fraction lengths

Specify the product fixed-point data type as Same as input or Custom. Setting this property will also set the NumeratorProductDataType property to the same value. The default value of this property is Same as input.

CustomDenominatorProductDataType

Denominator product word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only when you set the DenominatorProductDataType to Custom. The word length of the CustomNumeratorProductDataType and CustomDenominatorProductDataType properties must be the same. The default value of this property is numerictype([],32,30).

NumeratorAccumulatorDataType

Numerator accumulator word and fraction lengths

Specify the accumulator fixed-point data type as Same as input, Same as product, or Custom. Setting this property will also set the DenominatorAccumulatorDataType property to the same value. This default value of this property is Same as product.

CustomNumeratorAccumulatorDataType

Numerator accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only when you set the NumeratorAccumulatorDataType property to Custom. The word length of the CustomNumeratorAccumulatorDataType and CustomDenominatorAccumulatorDataType properties must be the same. The default value of this property is numerictype([],32,30).

DenominatorAccumulatorDataType

Denominator accumulator word and fraction lengths

Specify the accumulator fixed-point data type as Same as input, Same as product, or Custom. Setting this property will set also set the NumeratorAccumulatorDataType property to the same value. The default value of this property is Same as product.

CustomDenominatorAccumulatorDataType

Denominator accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only when you set the DenominatorAccumulatorDataType property to Custom. The word length of the CustomNumeratorAccumulatorDataType and CustomDenominatorAccumulatorDataType properties

must be the same. The default value of this property is `numerictype([],32,30)`.

`StateDataType`

State word and fraction lengths

Specify the state fixed-point data type as `Same as input`, `Same as accumulator`, or `Custom`. This property applies when you set the `Structure` property to `Direct form II` or `Direct form II transposed`. The default value of this property is `Same as accumulator`.

`CustomStateDataType`

State word and fraction lengths

Specify the state fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `StateDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

`NumeratorStateDataType`

Numerator state word and fraction lengths

Specify the state fixed-point data type as `Same as input`, `Same as accumulator`, or `Custom`. Setting this property will also set the `DenominatorStateDataType` property to the same value. This property applies only when you set the `Structure` property to `Direct form I transposed`. The default value of this property is `Same as accumulator`.

`CustomNumeratorStateDataType`

Numerator state word and fraction lengths

Specify the state fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `StateDataType` property to `Custom`. The word length of the `CustomNumeratorProductDataType` and `CustomDenominatorProductDataType` properties

must be the same. The default value of this property is `numerictype([],16,15)`.

`DenominatorStateDataType`

Denominator state word and fraction lengths

Specify the state fixed-point data type as `Same as input`, `Same as accumulator`, or `Custom`. Setting this property will also set the `NumeratorStateDataType` property to the same value. This property applies only when you set the `Structure` property to `Direct form I transposed`. The default value of this property is `Same as accumulator`.

`CustomDenominatorStateDataType`

Denominator state word and fraction lengths

Specify the state fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `StateDataType` property to `Custom`. The word length of the `CustomNumeratorStateDataType` and `CustomDenominatorStateDataType` properties must be the same. The default value of this property is `numerictype([],16,15)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as `Same as input`, `Same as accumulator`, or `Custom`. The default value of this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create biquad filter object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Return number of outputs of step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset states of biquad filter object
step	Filter input with biquad filter object

Examples

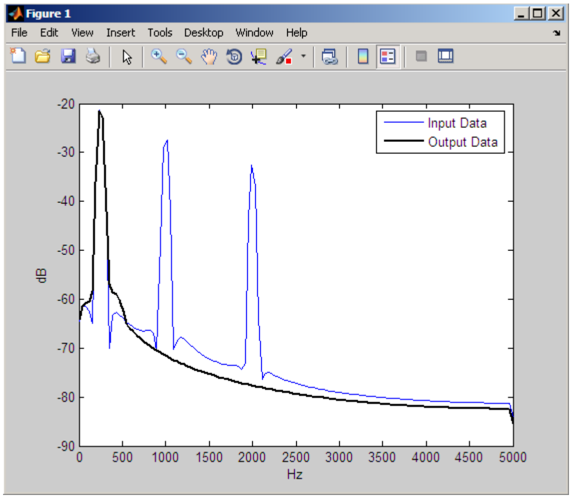
Use a lowpass biquadratic filter to filter out the high frequencies.

```
t = (0:1000)'/8e3;
% Input composed of 1 Khz and 3 Khz sinusoids.
xin = sin(2*pi*1e3*t)+sin(2*pi*3e3*t);
hFromWS = signalblks.SignalReader(xin, 100);
hLog = signalblks.SignalLogger;
% Design a fourth order lowpass filter with a
% normalized cutoff frequency of 0.4.
[z,p,k] = ellip(4,1,60,.4);
[s,g] = zp2sos(z,p,k);
hBqF=signalblks.BiquadFilter('Structure','Direct form I', ...
'SOSMatrix', s, ...
'ScaleValues', g);
while ~isDone(hFromWS)
input = step(hFromWS);
filteredOutput = step(hBqF,input);
step(hLog,filteredOutput);
end
filteredResult = hLog.Buffer;
```

```
swelch = spectrum.welch;
psd(swelch,filteredResult,'Fs',8e3);
```

Design and apply a lowpass filter using the biquad filter object.

```
Fs = 10000; % Sampling frequency 10 kHz
t = 0:(1/Fs):1;
x = cos(2*pi*250*t)+0.5*sin(2*pi*1000*t)+0.25*cos(2*pi*2000*t);
% Design SOS Butterworth filter with fdesign.lowpass
Hd = fdesign.lowpass('Fp,Fst,Ap,Ast',500,550,0.5,60,10000);
D = design(Hd,'butter');
H = signalblks.BiquadFilter('SOSMatrix',D.sosMatrix,...
'ScaleValues',D.ScaleValues);
y = step(H,x');
% Object for Welch spectral analysis
SpecWelch = spectrum.welch('Hamming',256,70);
InputPsd = psd(SpecWelch,x,'Fs',1e4);
OutputPsd = psd(SpecWelch,y,'Fs',1e4);
plot(InputPsd.Frequencies,10*log10(InputPsd.Data),'b');
hold on;
plot(OutputPsd.Frequencies,10*log10(OutputPsd.Data),'k',...
'linewidth',2);
xlabel('Hz'); ylabel('dB');
legend('Input Data','Output Data','Location','best');
```



Algorithm

This object implements the algorithm, inputs, and outputs described on the Biquad Filter block reference page. The object properties correspond to the block parameters, except for:

- **Coefficient source**– the biquad filter object does not accept `dfilt` objects as the `SOSMatrixSource`.
- **Action when the a0 values of the SOS matrix are not one**– the biquad filter objects assumes the zero-th order denominator coefficient is equal to one regardless of the specified value. The biquad filter object does not support the `Error` and `Warn` options found in the corresponding block.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See “What Are Sample- and Frame-Based Processing?” for more information.

See Also

`signalblks.DigitalFilter`

Purpose Create biquad filter object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates a biquad filter object, `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

signalblks.BiquadFilter.getNumInputs

Purpose

Number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.BiquadFilter.getNumOutputs

Purpose Return number of outputs of step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the biquad filter object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.BiquadFilter.reset

Purpose Reset states of biquad filter object

Syntax `reset(H)`

Description `reset(H)` resets the filter states of the biquad filter object,`H`, to the specified initial conditions. After the `step` method applies the biquad filter object to nonzero input data, the states may change. Invoking the `step` method again without first invoking the `reset` method may produce different outputs for an identical input.

Example of resetting filter states:

```
n =0:20;
x = cos(pi/4*n)+sin(pi/2*n);
H = signalblks.BiquadFilter;
y = step(H,x');
% Apply step method without invoking reset
y1 = step(H,x');
isequal(y,y1) % Returns a 0
% Reset filter states to zero
reset(H)
% Now invoke step method
y2 = step(H,x');
isequal(y,y2) % Returns a 1
```

signalblks.BiquadFilter.step

Purpose

Filter input with biquad filter object

Syntax

```
Y = step(H,X)
Y = step(H,X,NUM,DEN)
Y = step(H,X,NUM,DEN,G)
```

Description

`Y = step(H,X)` filters the real or complex input signal `X` , and outputs the filtered values, `Y`. The biquad filter object filters each channel of the input signal independently over time.

`Y = step(H,X,NUM,DEN)` filters the input using `NUM` as the numerator coefficients, and `DEN` as the denominator coefficients of the biquad filter. `NUM` must be a 3–by–N numeric matrix and `DEN` must be a 2–by–N numeric matrix, where N is the number of biquad filter sections. The object assumes the first denominator coefficient of each section to be 1. This configuration is applicable when the `SOSMatrixSource` property is `Input port` and the `ScaleValuesInputPort` property is `false`.

`Y = step(H,X,NUM,DEN,G)` specifies G the scale values of the biquad filter. `G` must be a 1-by-(N+1) numeric vector, where N is the number of biquad filter sections. This configuration is applicable when the `SOSMatrixSource` property is `Input Port` and the `ScaleValuesInputPort` property is true.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.BlockLMSFilter class

Purpose

Compute output, error, and weights using Block LMS adaptive algorithm

Description

The `BlockLMSFilter` object computes output, error, and weights using the Block LMS adaptive algorithm.

Construction

`H=signalblks.BlockLMSFilter` returns an adaptive FIR filter, `H`, that filters the input signal and computes filter weights based on the Block Least Mean Squares (LMS) algorithm.

`H=signalblks.BlockLMSFilter('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an adaptive FIR filter, `H`, with each specified property set to the specified value.

`H=signalblks.BlockLMSFilter(length,blocksize,'`*`PropertyName`*`',`*`PropertyValue`* returns an adaptive FIR filter, `H`, with the `Length` property set to `length`, the `BlockSize` property set to `blocksize`, and other specified properties set to the specified values.

Properties

`Length`

Length of FIR filter weights vector

Specify the length of the FIR filter weights vector as a positive integer scalar. The default value of this property is `32`.

`BlockSize`

Number of samples acquired before weight adaptation

Specify the number of samples of the input signal to acquire before the object updates the filter weights. The input frame length must be an integer multiple of the block size. The default value of this property is `32`.

`StepSizeSource`

Source of adaptation step size

Choose to specify the adaptation step size factor as `Property` or `Input port`. The default value of this property is `Property`.

`StepSize`

Adaptation step size

Specify the adaptation step size factor as a scalar, non-negative numeric value. The default value of this property is `0.1`. This property applies only when you set the `StepSizeSource` property to `'Property'`. This property is tunable.

`LeakageFactor`

Leakage factor used in Leaky LMS algorithm

Specify the leakage factor used in Leaky LMS algorithm as a scalar numeric value between `0` and `1`, both inclusive. When the value is less than `1`, the System object implements a leaky LMS algorithm. The default value of this property is `1`, providing no leakage in the adapting algorithm. This property is tunable.

`InitialWeights`

Initial values of filter weights

Specify the initial values of the filter weights as a scalar or a vector of length equal to the `Length` property value. The default value of this property is `0`.

`AdaptInputPort`

Additional input to enable adaptation of filter weights.

Specify when the object should adapt the filter weights. By default, the value of this property is `false`, and the filter continuously updates the filter weights. When this property is set to `true`, an adaptation control input is provided to the `step` method. If the value of this input is non-zero, the filter continuously updates the filter weights. If the input is zero, the filter weights remain at their current value.

`WeightsResetInputPort`

Additional input to enable weights reset

Specify whether the FIR filter can reset the filter weights. By default, the value of this property is `false`, and the object does not reset the weights. When this property is set to `true`, a reset control input is provided to the `step` method, and the `WeightsResetCondition` property applies. The object resets the filter weights based on the values of the `WeightsResetCondition` property and the reset input to the `step` method.

`WeightsResetCondition`

Condition that triggers the resetting of the filter weights

Specify the event to reset the filter weights as one of `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero`. The object resets the filter weights based on the values of this property and the reset input to the `step` method. This property applies only when you set the `WeightsResetInputPort` property to `true`. The default value of this property is `Non-zero`.

`WeightsOutputPort`

Output the filter weights

Set this property to `true` to output the adapted filter weights. The default value of this property is `true`.

Methods

clone	Create adaptive block LMS filter object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties

reset	Reset internal states of adaptive FIR filter object
step	Filter inputs using Block LMS algorithm

Examples

Remove noise using the Block LMS adaptive algorithm:

```
hblms = signalblks.BlockLMSFilter(10, 5);
hblms.StepSize = 0.01;
hblms.WeightsOutputPort = false;
hfilt = signalblks.DigitalFilter;
hfilt.TransferFunction = 'FIR (all zeros)';
hfilt.Numerator = fir1(10, [.5, .75]);
x = randn(1000,1); % Noise
d = step(hfilt, x) + sin(0:.05:49.95)'; % Noise + Signal
[y, err] = step(hblms, x, d);
subplot(2,1,1), plot(d), title('Noise + Signal');
subplot(2,1,2), plot(err), title('Signal');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Block LMS Filter block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.LMSFilter` | `signalblks.DigitalFilter`

signalblks.BlockLMSFilter.clone

Purpose Create adaptive block LMS filter object with same property values

Syntax

Description `C = clone(H)` creates a `BlockLMSFilter` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

Purpose

Number of expected inputs to step method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.BlockLMSFilter.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `BlockLMSFilter` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.BlockLMSFilter.reset

Purpose Reset internal states of adaptive FIR filter object

Syntax `reset(H)`

Description `reset(H)` sets the internal states of the `BlockLMSFilter` object `H` to their initial values.

Purpose

Filter inputs using Block LMS algorithm

Syntax

```
[Y,ERR,WTS] = step(H,X,D)
[Y,ERR] = step(H,X,D)
[...] = step(H,X,D,MU)
[...] = step(H,X,D,A)
[...] = step(H,X,D,R)
[Y,ERR,WTS] = step(H,X,D,MU,A,R)
```

Description

`[Y,ERR,WTS] = step(H,X,D)` filters the input `X`, using `D` as the desired signal, and returns the filtered output in `Y`, the filter error in `ERR`, and the estimated filter weights in `WTS`. The filter weights update once for every block of data that the object processes.

`[Y,ERR] = step(H,X,D)` filters the input `X`, using `D` as the desired signal, and returns the filtered output in `Y`, and the filter error in `ERR` when the `WeightsOutputPort` property is false.

`[...] = step(H,X,D,MU)` filters the input `X`, using `D` as the desired signal and `MU` as the step size, when you set the `StepSizeSource` property to `Input port`.

`[...] = step(H,X,D,A)` filters the input `X` , using `D` as the desired signal and `A` as the adaptation control, when you set the `AdaptInputPort` property to `true`. When `A` is non-zero, the filter continuously updates the filter weights. When `A` is zero, the filter weights remain constant.

`[...] = step(H,X,D,R)` filters the input `X`, using `D` as the desired signal and `R` as a reset signal, when you set the `WeightsResetInputPort` property to `true`. Use the `WeightsResetCondition` property to set the reset trigger condition. If a reset event occurs, the filter resets the filter weights to their initial values.

`[Y,ERR,WTS] = step(H,X,D,MU,A,R)` filters input `X`, using `D` as the desired signal, `MU` as the step size, `A` as the adaptation control, and `R` as the reset signal, and returns the filtered output in `Y`, the filter error in `ERR`, and the adapted filter weights in `WTS`. Set the properties appropriately to provide all possible inputs.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Buffer input signal

Description

The `Buffer` object buffers input signal. The number of samples per channel in the input to the buffer object must match the condition that the number of samples per channel equal the difference between the output buffer size and buffer overlap, (`Length` - `OverlapLength`). When you set the `FrameBasedProcessing` to `true`, the number of samples per channel equal the number of rows in the input. Otherwise, the number of samples per channel equal `1`.

Construction

`H=signalblks.Buffer` returns a buffer System object, `HBUFF`, used to buffer input signals with overlap.

`H=signalblks.Buffer('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a buffer object, `H`, with each specified property set to the specified value.

`H=signalblks.Buffer(LENGTH,OVERLAPLENGTH,INITIALCONDITIONS,'`*`PropertyNa`* returns a buffer object, `H`, with `Length` property set to `LENGTH`, `OverlapLength` property set to `OVERLAPLENGTH`, `InitialConditions` property set to `INITIALCONDITIONS` and other specified properties set to the specified values.

Properties

`Length`

Number of samples to buffer

Specify the number of consecutive samples from each input channel to buffer. This property can be set to any scalar integer greater than `0` of MATLAB built-in numeric data type. The default value of this property is `64`.

`OverlapLength`

Amount of overlap between outputs

Specify the number of samples by which consecutive output frames overlap. This property can be set to any scalar integer greater than or equal to `0` of MATLAB built-in numeric data type. The default value of this property is `0`.

`InitialConditions`

Initial output

Specify the value of the System object's initial output for cases of nonzero latency as a scalar, vector, or matrix. The default value of this property is `0`.

`FrameBasedProcessing`

Enable frame-based processing

If this property is set to `true`, each column of the input is treated as a separate channel for buffering. If the value of this property is `false`, each element of the input is treated as a separate channel. The default value of this property is `true`.

Methods

clone	Create instance of an object with the same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Returns logical value to indicate whether input attributes and non-tunable properties are locked
reset	Reset the internal states of a System object
step	Buffer input signal based on past values

Examples

Create a buffer of 256 samples with 128 sample overlap

```
hreader=signalblks.SignalReader(randn(1024,1),128);
hbuff=signalblks.Buffer(256,128);
for i=1:8
```

```
y=step(hbuff,step(hreader));
% y is of length 256 with 128 samples from previous input
end
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Buffer block reference page. The object properties correspond to the block properties.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See "What Are Sample- and Frame-Based Processing?" for more information.

See Also

`signalblks.DelayLine` | `signalblks.Delay`

signalblks.Buffer.clone

Purpose Create instance of an object with the same property values

Syntax `C=clone(H)`

Description `C=clone(H)` creates a `Buffer` System object `C`, with the same property values as `H`.

The `clone` method creates a new unlocked object.

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.Buffer.getNumOutputs

Purpose

Number of outputs from step method

Syntax

```
getNumOutputs(H)
```

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Returns logical value to indicate whether input attributes and non-tunable properties are locked

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `Buffer` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.Buffer.reset

Purpose Reset the internal states of a System object

Syntax `reset(H)`

Description `reset(H)` resets the internal states of buffer to their initial values.

For many System objects, this method is a no-op. Objects that have internal states will describe in their help what the reset method does for that object.

The reset method is always a no-op for unlocked System objects, as the states may not be allocated when the object is not locked.

Purpose

Buffer input signal based on past values

Syntax

```
Y=step(H,X)
```

Description

`Y=step(H,X)` creates output `Y` based on current input and stored past values of `X` . Output length is equal to the `Length` property.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.BurgAREstimator class

Purpose

Compute estimate of autoregressive (AR) model parameters using Burg method

Description

The `BurgAREstimator` object computes the estimate of the autoregressive (AR) model parameters using the Burg method.

Construction

`H = signalblks.BurgAREstimator` returns a Burg `BurgAREstimator` System object, `H`, that performs parametric `AR` estimation using the Burg maximum entropy method.

`H = signalblks.BurgAREstimator('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a Burg `AR` estimator object, `H`, with each specified property set to the specified value.

Properties

`AOutputPort`

Enable output of polynomial coefficients

Set this property to `true` to output the polynomial coefficients, `A`, of the AR model the object computes. The default value of this property is `true`. Either the `AOutputPort` property, the `KOutputPort` property, or both must be `true`.

`KOutputPort`

Enable output of reflection coefficients

Set this property to `true` to output the reflection coefficients, `K`, for the AR model that the object computes. The default value of this property is `false`. Either the `AOutputPort` property, the `KOutputPort` property, or both must be `true`.

`EstimationOrderSource`

Source of estimation order

Specify how to determine estimator order as `Auto` or `Property`. When you set this property to `Auto`, the estimation order is assumed to be one less than the length of the input vector. When you set this property to `Property`, the value in EstimationOrder is used. The default value of this property is `Auto`.

`EstimationOrder`

Order of AR model

Set the AR model estimation order to a real positive integer. This property applies when you set the `EstimationOrderSource` to `Property`. The default value of this property is `4`.

Methods

clone	Create Burg AR Estimator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Compute normalized estimate of AR model parameter

Examples

Use the Burg AR Estimator System object to estimate the parameters of an AR model.

```
rs = RandStream.create('mrg32k3a');
noise = randn(rs, 100,1); % Normalized white Gaussian noise
x = filter(1,[1 1/2 1/3 1/4 1/5],noise);
hburgarest = signalblks.BurgAREstimator(...
'EstimationOrderSource', 'Property', ...
'EstimationOrder', 4);
[a, g] = step(hburgarest, x);
x_est = filter(g, a, x);
plot(1:100,[x x_est]);
```

```
title('Original and estimated signals');
legend('Original', 'Estimated');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Burg AR Estimator block reference page. The object properties correspond to the block parameters, except for:

- **Output(s)** block parameter corresponds to the `AOutputPort` and the `KOutputPort` object properties.

See Also

`signalblks.LevinsonSolver`

Purpose Create Burg AR Estimator object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates a `BurgAREstimator` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.BurgAREstimator.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.BurgAREstimator.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `BurgAREstimator` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Compute normalized estimate of AR model parameter

Syntax

```
[A,G] = step(H,X)
[K,G] = step(H,X)
[A,K,G] = step(H,X)
```

Description

`[A,G] = step(H,X)` computes the normalized estimate of the `AR` model parameters to fit the input, `X`, in the least square sense. Output `A` is a column vector that contains the normalized estimate of the `AR` model polynomial coefficients in descending powers of z, and the scalar `G` is the `AR` model gain.

`[K,G] = step(H,X)` returns `K`, a column vector containing the `AR` model reflection coefficients when you set the `KOutputPort` property to `true` and the `AOutputPort` property to `false`.

`[A,K,G] = step(H,X)` returns the `AR` model polynomial coefficients `A`, reflection coefficients `K`, and the scalar gain `G` when the `AOutputPort` and `KOutputPort` properties are both `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.BurgSpectrumEstimator class

Purpose

Compute parametric spectral estimate using Burg method

Description

The `BurgSpectrumEstimator` object computes a parametric spectral estimate of the input using the Burg method. The object fits an autoregressive (AR) model to the signal by minimizing (via least-squares) the forward and backward prediction errors while constraining the AR parameters to satisfy the Levinson-Durbin recursion.

Construction

`H = signalblks.BurgSpectrumEstimator` returns an object, `H`, that estimates the power spectral density (PSD) of the input frame using the Burg method.

`H = signalblks.BurgSpectrumEstimator('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a spectrum estimator, `H`, with each specified property set to the specified value.

Properties

`EstimationOrderSource`

Source of estimation order

Specify the source of the estimation order as `Auto` or `Property`. If this property is set to `Auto`, the estimation order is assumed to be one less than the length of the input vector. The default value of this property is `Property`.

`EstimationOrder`

Order of AR model

Specify the order of AR model as a real positive integer. This property applies only when you set the `EstimationOrderSource` property to `Property`. The default value of this property is `6`.

`FFTLengthSource`

Source of FFT length

Specify the source of the FFT length as `Auto` or `Property`. When you set this property to `Auto`, the FFT length is assumed to be one more than the estimation order. When you set this property to `Property`, the `FFTLength` property value must be an integer power of two.

`FFTLength`

FFT length as power-of-two integer value

Specify the FFT length as a power-of-two numeric scalar. This property applies when you set the `FFTLengthSource` property to `Property`. The default value of this property is `256`.

`SampleTime`

Sample time of input time series

Specify the sample time of the original input time series as a positive numeric scalar. The default value of this property is `1`.

Methods

clone	Create estimator object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Return logical value to indicate whether input attributes and non-tunable properties are locked
step	Compute estimate of power spectral density

Examples

Estimate PSD using the Burg method.

```
x = randn(100,1);
hbEst = signalblks.BurgSpectrumEstimator('EstimationOrder', 4);
y = filter(1,[1 1/2 1/3 1/4 1/5],x); % Fourth order AR filter
p = step(hbEst, y); % Uses default FFT length of 256
```

```
plot([0:255]/256, p);
title('Burg Method Spectral Density Estimate');
xlabel('Normalized frequency'); ylabel('Power/frequency');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Burg Method block reference page. The object properties correspond to the block properties.

See Also

`signalblks.BurgAREstimator` | `signalblks.LevinsonSolver`

Purpose

Create estimator object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates a `BurgSpectrumEstimator` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object.

signalblks.BurgSpectrumEstimator.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.BurgSpectrumEstimator.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.BurgSpectrumEstimator.isLocked

Purpose Return logical value to indicate whether input attributes and non-tunable properties are locked

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `BurgSpectrumEstimator` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Compute estimate of power spectral density

Syntax `Y = step(H,X)`

Description `Y = step(H,X)` outputs `Y`, a spectral estimate of input `X`, using the Burg method.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.CepstralToLPC class

Purpose

Convert cepstral coefficients to linear prediction coefficients

Description

The `CepstralToLPC` object converts cepstral coefficients to linear prediction coefficients (LPC).

Construction

`H=signalblks.CepstralToLPC` returns a System object, `H`, that converts the cepstral coefficients(CCs) to linear prediction coefficients (LPCs).

`H=signalblks.CepstralToLPC('PropertyName',PropertyValue,...)` returns a `Cepstral to LPC` object, `H`, with each specified property set to the specified value.

Properties

`PredictionErrorOutputPort`

Enable prediction error power output

Set this property to `true` to output the prediction error power. The prediction error power is the power of the error output of an FIR analysis filter represented by the LPCs for a given input signal. The default value of this property is `false`.

Methods

clone	Create cepstral to LPC object with same property values
getNumInputs	Returns the number of expected inputs to the method
getNumOutputs	Returns the number of outputs of the method
isLocked	Returns logical value to indicate whether input attributes and non-tunable properties are locked
step	Compute LPC coefficients from column of cepstral coefficients

Examples

Convert Cepstral coefficients to linear prediction coefficients (LPC)

```
cc=[0 0.9920 0.4919 0.3252 0.2418 , ...
0.1917 0.1583 0.1344 0.1165 0.0956]';
hcc2lpc=signalblks.CepstralToLPC;
a=step(hcc2lpc,cc);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the LPC to/from Cepstral Coefficients block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.LPCToCepstral` | `signalblks.LSFToLPC` | `signalblks.RCToLPC`

signalblks.CepstralToLPC.clone

Purpose Create cepstral to LPC object with same property values

Syntax `C=clone(H)`

Description `C=clone(H)` creates a `CepstralToLPC` System object `C`, with the same property values as `H`.

The `clone` method creates a new unlocked object.

Purpose

Returns the number of expected inputs to the `step` method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.CepstralToLPC.getNumOutputs

Purpose Returns the number of outputs of the `step` method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Returns logical value to indicate whether input attributes and non-tunable properties are locked

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `CepstralToLPC` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.CepstralToLPC.step

Purpose

Compute LPC coefficients from column of cepstral coefficients

Syntax

```
A=step(H,CC)
[A,P]=step(H,CC)
```

Description

`A=step(H,CC)` computes the `LPC` coefficients, `A` , from the columns of cepstral coefficients, `CC`.

`[A,P]=step(H,CC)` converts the columns of the cepstral coefficients `CC` to the linear prediction coefficients, and also returns the prediction error power `P` when the `PredictionErrorOutputPort` property is `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose Generate swept-frequency cosine (chirp) signal

Description The `Chirp` object generates a swept-frequency cosine (chirp) signal.

Construction `H = signalblks.Chirp` returns a chirp signal, `H`, with unity amplitude.

`H = signalblks.Chirp('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a chirp signal, `H`, with each specified property set to the specified value.

Properties

`Type`

Frequency sweep type

Specify the frequency sweep type as `Swept cosine`, `Linear`, `Logarithmic`, or `Quadratic`. This property specifies how the output instantaneous frequency sweep varies over time. The default value of this property is `Linear`.

`SweepDirection`

Sweep direction

Specify the sweep direction as either `Unidirectional` or `Bidirectional`. The default value of this property is `Unidirectional`.

`InitialFrequency`

Initial frequency (Hz)

When you set the `Type` property to `Linear`, `Quadratic`, or `Logarithmic`, this property specifies the initial instantaneous frequency in Hertz of the output chirp signal. When you set the `Type` property to `Logarithmic`, the value of this property is one less than the actual initial frequency of the sweep. Also, when the sweep is logarithmic, the initial frequency must be less than the target frequency, specified by the `TargetFrequency` property. This property is tunable. The default value of this property is `1000`.

TargetFrequency

Target frequency (Hz)

When you set the Type property to Linear, Quadratic, or Logarithmic, this property specifies the instantaneous frequency of the output signal in Hertz at the target time. When you set the Type property to Swept Cosine, the target frequency is the instantaneous frequency of the output at half the target time. Also, when the sweep is logarithmic, the target frequency must be greater than the initial frequency, specified by the InitialFrequency property. This property is tunable. The default value of this property is 4000.

TargetTime

Target time (s)

When you set the Type property to Linear, Quadratic, or Logarithmic, this property specifies the target time in seconds at which the target frequency is reached. When you set the Type property to Swept cosine, this property specifies the time at which the sweep reaches $2f_{tgt} - f_{init}$ Hz, where f_{tgt} is the TargetFrequency and f_{init} is the InitialFrequency. The target time should not be greater than the sweep time, specified by the SweepTime property. This property is tunable. The default value of this property is 1.

SweepTime

Sweep time (s)

When you set the SweepDirection property to Unidirectional, the sweep time in seconds is the period of the output frequency sweep. When you set the SweepDirection property to Bidirectional, the sweep time is half the period of the output frequency sweep. The sweep time should be no less than the target time, specified by the TargetTime. This property must be a positive numeric scalar and is tunable. The default value of this property is 1.

`InitialPhase`

Initial phase (rad)

Specify initial phase of the output in radians at time $t = 0$. This property is tunable. The default value of this property is `0`.

`SampleTime`

Sample time

Specify the sample period of the output in seconds as a positive numeric scalar. The output frame period is the sample time multiplied by the number of samples per frame. The default value is `1/8000`.

`SamplesPerFrame`

Samples per output frame

Specify the number of samples to buffer into each output as a positive integer. The default value of this property is `1`.

`OutputDataType`

Output data type

Specify the output data type as `double` or `single`. The default value of this property is `double`.

Methods

clone	Create chirp object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Return logical value to indicate whether input attributes and non-tunable properties are locked

reset	Reset internal states of chirp object
step	Generate chirp signal

Examples

Generate a bidirectional swept chirp signal:

```
hchirp = signalblks.Chirp(...
'SweepDirection', 'Bidirectional', ...
'TargetFrequency', 25, ...
'InitialFrequency', 0,...
'TargetTime', 1, ...
'SweepTime', 1, ...
'SamplesPerFrame', 400, ...
'SampleTime', 1/400);
plot(step(hchirp));
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Chirp block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.SineWave`

Purpose

Create chirp object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates a `Chirp` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

signalblks.Chirp.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Number of outputs from step method

Syntax

```
getNumOutputs(H)
```

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.Chirp.isLocked

Purpose Return logical value to indicate whether input attributes and non-tunable properties are locked

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `Chirp` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset internal states of chirp object

Syntax

```
reset(H)
```

Description `reset(H)` sets the internal states of the `Chirp` object `H` to their initial values. After you reset `H`, the frequency sweep restarts from the beginning.

signalblks.Chirp.step

Purpose Generate chirp signal

Syntax `Y = step(H)`

Description `Y = step(H)` returns a swept-frequency cosine output, `Y`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Decimate input using Cascaded Integrator-Comb filter

Description

The `CICDecimator` object decimates inputs using a Cascaded Integrator-Comb filter. Inputs and outputs to the object have signed fixed-point data types. You must have a Fixed-Point Toolbox license to use the `CICDecimator` System object.

Construction

`H = signalblks.CICDecimator` returns a `CICDecimator` System object, `H`, that you can use to apply a Cascaded Integrator-Comb (CIC) Decimation filter to the input.

`H = signalblks.CICDecimator('PropertyName',PropertyValue,...)` returns a `CICDecimator` object, `H`, with each specified property set to the value you specify.

`H = signalblks.CICDecimator(R,M,N,'PropertyName', PropertyValue, ...)` returns a `CICDecimator` object, `H`, with the `DecimationFactor` property set to *R*, the `DifferentialDelay` property set to *M*, the `NumSections` property set to *N*, and any other specified properties set to the values you specify.

Properties

`DecimationFactor`

Decimation factor of filter

Specify a positive integer amount by which the object decimates the input. The default value of this property is `2`.

`DifferentialDelay`

Differential delay of filter comb sections

Specify a positive integer delay value for the object to use in each comb section of the filter. The default value of this property is `1`.

`NumSections`

Number of integrator and comb sections

Specify the number of integrator and comb sections in the CIC filter as a positive integer value. The default value of this property is 2.

FixedPointDataType

Fixed-point property setting

Specify the fixed-point data type as `Full precision`, `Minimum section word lengths`, `Specify word lengths`, or `Specify word and fraction lengths`. The default value of this property is `Full precision`.

- When you set this property to `Full precision`, the `CICDecimator` object automatically determines the word and fraction lengths of the filter sections and output.
- When you set this property to `Minimum section word length`, the object automatically determines the word and fraction lengths of the filter sections and the fraction length of the output. You must specify the `OutputWordLength`.
- When you set this property to `Specify word lengths`, the object automatically determines the fraction lengths of the filter sections and the output. You must specify values for the `OutputWordLength` and the `SectionWordLengths` properties.
- When you set this property to `Specify word and fraction lengths`, the object does not automatically determine word and fraction lengths for the filter sections or the output. You must specify them using the `OutputFractionLength` , `OutputWordLength`, `SectionFractionLengths`, and `SectionWordLengths` properties.

SectionWordLengths

Word length of each filter section

Specify the fixed-point word length for each section of the CIC filter as a scalar or a vector of length `2*NumSections`. This property applies only when you set the `FixedPointDataType` property to `Specify word lengths`, or `Specify word and fraction lengths`. The default value of this property is `[16 16 16 16]`.

`SectionFractionLengths`

Fraction length of each filter section

Specify the fixed-point fraction length for each section of the CIC filter as a scalar or a vector of length `2*NumSections`. This property applies only when you set the `FixedPointDataType` property to `Specify word and fraction lengths`. The default value of this property is `0`.

`OutputWordLength`

Word length of filter output

Specify the fixed-point word length for the filter output. This property applies only when you set the `FixedPointDataType` property to `Minimum section word lengths`, `Specify word lengths`, or `Specify word and fraction lengths`. The default value of this property is `32`.

`OutputFractionLength`

Fraction length of filter output

Specify the fixed-point fraction length for the output of the CIC filter. This property applies only when you set the `FixedPointDataType` property to `Specify word and fraction lengths`. The default value of this property is `0`.

Methods

clone	Create CIC Decimator object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Number of outputs from step method

isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset filter states of CIC decimator object
step	Decimate input using Cascaded Integrator-Comb filter

Examples

Decimate signal by a factor of 4 (downsample the signal from 44.1 kHz to 11.025 kHz).

```
h = signalblks.CICDecimator(4);
% Specify DecimationFactor = 4, use default NumSections = 2,
% DifferentialDelay = 1
h.FixedPointDataType = 'Minimum section word lengths';
h.OutputWordLength = 16; % Specify the output word length
% Create fixed-point sinusoidal input signal
Fs = 44.1e3; % Original sampling frequency: 44.1 kHz
n = 0:1023; % 1024 samples, 0.0232 second long signal
x = fi(sin(2*pi*1e3/Fs*n),true,16,15);
hsr = signalblks.SignalReader; % Create SignalReader System object
hsr.Signal = x;
hsr.SamplesPerFrame = 64;
% Process the input signal
for ii=1:16
% Decimated output with 16 samples per frame
  y(ii,:) = step(h, step( hsr ));
end
% Plot the first frame of the original and decimated signals.
% The output latency is 2 samples.
gainCIC = ...
(h.DecimationFactor*h.DifferentialDelay)^h.NumSections;
stem(n(1:56)/Fs, double(x(4:59))); hold on;
stem(n(1:14)/(Fs/h.DecimationFactor),...
    double(y(1,3:end))/gainCIC,'r','filled');
```

```
xlabel('Time (sec)');ylabel('Signal Amplitude');
legend('Original signal', 'Decimated signal', ...
    'location', 'north');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the CIC Compensator block reference page. The object properties correspond to the block properties.

See Also

`signalblks.CICInterpolator` | `signalblks.FIRDecimator`

signalblks.CICDecimator.clone

Purpose Create CIC Decimator object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a `CICDecimator` System object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with un-initialized states.

Purpose Return number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.CICDecimator.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax

```
isLocked(H)
```

Description `isLocked(H)` returns the locked state of the `CICDecimator` System object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.CICDecimator.reset

Purpose Reset filter states of CIC decimator object

Syntax `reset(H)`

Description `reset(H)` resets the filter states of the `CICDecimator` System object `H` to zero.

Purpose

Decimate input using Cascaded Integrator-Comb filter

Syntax

`Y = step(H,X)`

Description

`Y = step(H,X)` decimates the fixed-point input `X` to produce a fixed-point output `Y` using the `CICDecimator` System object `H`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.CICInterpolator class

Purpose Interpolate signal using Cascaded Integrator-Comb filter

Description The `CICInterpolator` object interpolates inputs using a Cascaded Integrator-Comb (CIC) filter. Inputs and outputs to the object have signed fixed-point data types. You must have a Fixed-Point Toolbox license to use the `CICInterpolator` System object.

Construction `H = signalblks.CICInterpolator` returns a System object, `H`, that applies a Cascaded Integrator-Comb (CIC) Interpolation filter to the input.

`H = signalblks.CICInterpolator('PropertyName',PropertyValue,...)` returns a `CIC` interpolation object, `H`, with each specified property set to the value you specify.

`H = signalblks.CICInterpolator(R,M,N,'PropertyName',PropertyValue,...)` returns a `CICInterpolator` object, `H`, with the `InterpolationFactor` property set to *R*, the `DifferentialDelay` property set to *M*, the `NumSections` property set to *N*, and other specified properties set to the values you specify.

Properties

`InterpolationFactor`

Interpolation factor of filter

Specify a positive integer amount by which the object interpolates the input signal. The default value of this property is `2`.

`DifferentialDelay`

Differential delay of filter comb sections

Specify a positive integer delay value for the object to use in each comb section of the filter. The default value of this property is `1`.

`NumSections`

Number of integrator and comb sections

Specify the number of integrator and comb sections in the CIC filter as a positive integer value. The default value of this property is 2.

FixedPointDataType

Fixed-point property setting

Specify the fixed-point data type as Full precision, Minimum section word lengths, Specify word lengths, or Specify word and fraction lengths. The default value of this property is Full precision.

- When you set this property to Full precision, the CICInterpolator object automatically determines the word and fraction lengths of the filter sections and output.
- When you set this property to Minimum section word length, the object automatically determines the word and fraction lengths of the filter sections and the fraction length of the output. You must specify the OutputWordLength.
- When you set this property to Specify word lengths, the object automatically determines the fraction lengths of the filter sections and the output. You must specify values for the OutputWordLength and the SectionWordLengths properties.
- When you set this property to Specify word and fraction lengths, the object does not automatically determine word and fraction lengths for the filter sections or the output. You must specify them using the OutputFractionLength , OutputWordLength, SectionFractionLengths, and SectionWordLengths properties.

SectionWordLengths

Word length of each filter section

Specify the fixed-point word length for each section of the CIC filter as a scalar or a vector of length 2*NumSections. This property applies only when you set the FixedPointDataType property to Specify word lengths, or Specify word and fraction lengths. The default value of this property is [16 16 16 16].

SectionFractionLengths

Fraction length of each filter section

Specify the fixed-point fraction length for each section of the CIC filter as a scalar or a vector of length 2*NumSections. This property applies only when you set the FixedPointDataType property to Specify word and fraction lengths. The default value of this property is 0.

OutputWordLength

Word length of filter output

Specify the fixed-point word length for the filter output. This property applies only when you set the FixedPointDataType property to Minimum section word lengths, Specify word lengths, or Specify word and fraction lengths. The default value of this property is 32.

OutputFractionLength

Fraction length of filter output

Specify the fixed-point fraction length for the output of the CIC filter. This property applies only when you set the FixedPointDataType property to Specify word and fraction lengths. The default value of this property is 0.

Methods

clone	Create CIC Interpolator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Return number of outputs of step method

isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset internal states of CIC Interpolator object
step	Interpolate signal using Cascaded Integrator-Comb filter

Examples

Interpolate signal by a factor of 2 (upsample the signal from 22.05 kHz to 44.1 kHz).

```
hcicint = signalblks.CICInterpolator(2);
% Uses default NumSections = 2, DifferentialDelay = 1
% Create fixed-point sinusoidal input signal
Fs = 22.05e3; % Original sampling frequency: 22.05 kHz
n = 0:511; % 512 samples, 0.0113 second long signal
x = fi(sin(2*pi*1e3/Fs*n),true,16,15);
% Create SignalReader System object
hsr = signalblks.SignalReader;
hsr.Signal = x;
hsr.SamplesPerFrame = 32;
% Process the input signal
for ii=1:16
% Interpolated output with 64 samples per frame
  y(ii,:) = step(hcicint,step( hsr ));
end
% Plot the first frame of the original and interpolated signals.
% The output latency is 2 samples.
gainCIC = ...
(hcicint.InterpolationFactor*hcicint.DifferentialDelay)...
^hcicint.NumSections/hcicint.InterpolationFactor;
stem(n(1:31)/Fs, double(x(1:31)),'r','filled'); hold on;
stem(n(1:61)/(Fs*hcicint.InterpolationFactor),...
 double(y(1,4:end))/gainCIC,'b');
xlabel('Time (sec)');ylabel('Signal Amplitude');
```

```
legend('Original signal', 'Interpolated signal', 'location', 'north');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the CIC Interpolation block reference page. The object properties correspond to the block properties, except for:

- The object does not have a property that corresponds to the **Framing** parameter of the CIC Interpolation block. The object always maintains the input frame rate.
- The object does not have a property that allows you to specify the source of the coefficients. The object cannot import coefficients from an `mfilt` object.

See Also

`signalblks.CICDecimator` | `signalblks.FIRInterpolator`

Purpose

Create CIC Interpolator object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates a `CICInterpolator` System object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with un-initialized states.

signalblks.CICInterpolator.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the step method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Return number of outputs of step method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.CICInterpolator.isLocked

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

```
isLocked(H)
```

Description

`isLocked(H)` returns the locked state of the `CICInterpolator` System object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Reset internal states of CIC Interpolator object

Syntax

```
reset(H)
```

Description

`reset(H)` resets the internal states of the `CICInterpolator` System object `H` to zero.

signalblks.CICInterpolator.step

Purpose

Interpolate signal using Cascaded Integrator-Comb filter

Syntax

`Y = step(H,X)`

Description

`Y = step(H,X)` interpolates the fixed-point input `X` to produce a fixed-point output `Y` using the `CICInterpolator` System object `H`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.Convolver class

Purpose

Convolution of two inputs

Description

The `Convolver` computes the convolution of two inputs.

Construction

`H = signalblks.Convolver` returns a convolver object, `H`, that convolves two inputs. If both inputs are vectors, they must both be either row or column vectors. For N-D arrays, the convolver computes the convolution column-wise. For arrays, the number of columns in the inputs must be equal. If one input is a vector and the other is an N-D array, the convolver computes the convolution of the vector with each column of the N-D array. Convolving inputs of length *N* and *M* results in a sequence of length *N+M-1*. Convolving matrices of size *M*–by–*N* and *P*–by–*N*. results in a matrix of size *M+P-1*–by–*N*.

`H = signalblks.Convolver('PropertyName',PropertyValue, ...)` returns a convolver object, `H`, with each property set to the specified value.

Properties

`Method`

Domain for computing convolutions

Specify the domain in which the convolver performs the convolutions as `Time Domain`, `Frequency Domain`, or `Fastest`. Computing convolutions in the time domain minimizes memory use. Computing convolutions in the frequency domain may require fewer computations depending on the input length. If the value of this property is `Fastest`, the object computes convolutions in the domain which minimizes the number of computations. The default value for this property is `Time Domain`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value for this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default value for this property is `Wrap`.

`ProductDataType`

Product word and fraction lengths

Specify the product fixed-point data type as `Internal rule`, `Same as first input`, or `Custom`. The default value for this property is `Internal rule`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `ProductDataType` property is `Custom`. The default value of this property is `numerictype([],32,30)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Internal rule`, `Same as product`, `Same as first input`, or `Custom`. The default value for this property is `Internal rule`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `AccumulatorDataType` property is `Custom`. The default value for this property is `numerictype([],32,30)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as `Same as accumulator`, `Same as product`, `Same as first input`, or `Custom`. The default value for this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `OutputDataType` property is `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create convolver object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs of step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Compute convolution of inputs

Examples

Convolution of two rectangular sequences:

```
hconv = signalblks.Convolver;
x = ones(10,1);
y = step(hconv, x, x);
% Result is a triangular sequence
plot(y);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Convolution block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.Autocorrelator` | `signalblks.Crosscorrelator`

Purpose Create convolver object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a convolver object, `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.Convolver.getNumInputs

Purpose Return number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs of step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs of the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.Convolver.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the convolver object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Compute convolution of inputs

Syntax

`Y = step(H,A,B)`

Description

`Y = step(H,A,B)` convolves the inputs `A` and `B`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.Counter class

Purpose

Count up or down through specified range of numbers

Description

The `Counter` object counts up or down through specified range of numbers.

Construction

`H = signalblks.Counter` returns a counter System object, `H`, that counts up when the input is non-zero.

`H = signalblks.Counter('PropertyName',PropertyValue,...)` returns a counter System object, `H`, with each specified property set to the specified value.

Properties

`Direction`

Counts up or down

Specify the counter direction as `Up` or `Down`. The default value of this property is `Up`. This property is tunable.

`CountEventInputPort`

Add input to specify a count event

Set this property to `true` to enable a count event input for the internal counter. The internal counter increments or decrements whenever the count event input satisfies the condition you specify in the `CountEventCondition` property. When you set this property to `false`, the internal counter is free-running, that is, it increments or decrements on every call to the `step` method. The default value of this property is `true`.

`CountEventCondition`

Condition that increments, decrements, or resets the internal counter

Specify the event at the count event input that increments or decrements the counter as `Rising edge`, `Falling edge`, `Either edge` or `Non-zero`.

When you set the `ResetInputPort` and `CountEventInputPort` properties to `true`, and when the reset input receives the event you specify for the `CountEventCondition`, it resets the counter. This property applies only when you set the `CountEventInputPort` property to `true`.

`CounterSizeSource`

Source of counter size data type

Specify the source of the counter size data type as `Property` or `Input port`. The default value of this property is `Property`.

`CounterSize`

Range of integer values to count through

Specify the range of integer values to count through before recycling to zero as `8 bits`, `16 bits`, `32 bits` or `Maximum`. The default value of this property is `Maximum`.

`MaximumCount`

Counter's maximum value

Specify the counter's maximum value as a numeric scalar value. This property applies only when you set the `CounterSizeSource` property to `Property` and the `CounterSize` property to `Maximum`. The default value of this property is `255`. This property is tunable.

`InitialCount`

Counter initial value

Specify the initial value for the counter. The default value of this property is `0`. This property is tunable.

`CountOutputPort`

Output the count

Set this property to `true` to enable output of the internal count. The default value of this property is `true`. You cannot set both `CountOutputPort` and `HitOutputPort` to `false` at the same time.

HitOutputPort

Output the hit events

Set this property to `true` to enable output of the hit events. You cannot set both `CountOutputPort` and `HitOutputPort` to `false` at the same time. The default value of this property is `true`.

HitValues

Values whose occurrence in count produce a true hit output

Specify an integer scalar or a vector of integers, any of whose occurrences in the count should be flagged by a true at the HIT output. This property applies only when you set the `HitOutputPort` property to `true`.

ResetInputPort

Add input to enable internal counter reset

When you set this property to `true`, specify a reset input to the `step` method. When the reset input receives the event you specify for the `CountEventCondition` property, the counter resets. If you set the `CountEventInputPort` property to `false`, the counter resets whenever the reset input is not zero.

SamplesPerFrame

Number of samples in each output frame

Specify the number of samples in each output frame. This property applies only when you set the `CountEventInputPort` property to `false`, indicating a free-running counter. The default value of this property is `1`.

CountOutputDataType

Data type of count output

Specify the data type of the count output, CNT, as `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32` or `uint32`. This property applies when you set the `CountOutputPort` property to `true`. The default value of this property is `double`.

Methods

clone	Create counter object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset internal states of Counter object
step	Increment or decrement the internal counter

Examples

Use Counter System object for counting from 0 to 5.

```
hcounter = signalblks.Counter('MaximumCount', 5, ...
'CountOutputPort', true, ...
'HitOutputPort', false, ...
'ResetInputPort', false);
 sgnl = [0 1 0 1 0 1 0 1 0 1 0 1 ];
   for ii = 1:length(sgnl)
 %count at every rising edge of the input signal.
 cnt(ii) = step(hcounter, sgnl(ii));
   end
   cnt
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Counter block reference page. The object properties correspond to the block parameters.

- The `CountEventCondition` object property does not have a free-running option. Set the `CountEventInputPort` property to `false` to obtain the free-running option.
- The `CounterSizeSource` and `CounterSize` object properties correspond to the **Counter size** block parameter.
- The `CountOutputPort` and `HitOutputPort` correspond to the **Output** block parameter.
- There is no object property that corresponds to the **Hit data type** block parameter. The output type is logical in MATLAB. (This logical is different from the popup logical in the block. For the object, logical corresponds to Boolean in the block.)

Purpose Create counter object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a `Counter` object `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

signalblks.Counter.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.Counter.isLocked

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

```
isLocked(H)
```

Description

`isLocked(H)` returns the locked state of the `Counter` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset internal states of Counter object

Syntax `reset(H)`

Description `reset(H)` resets the internal states of the counter object `H` to their initial values.

signalblks.Counter.step

Purpose

Increment or decrement the internal counter

Syntax

```
[CNT,HIT] = step(H,EVENT,RESET)
CNT = step(H,EVENT,RESET)
HIT = step(H,EVENT,RESET)
[...] = step(H)
[...] = step(H,EVENT)
```

Description

`[CNT,HIT] = step(H,EVENT,RESET)` increments, decrements, or resets the internal counter as specified by the values of the `EVENT` and `RESET` inputs. The output argument `CNT` denotes the present value of the counter. A trigger event at the `EVENT` input causes the counter to increment or decrement. A trigger event at the `RESET` input resets the counter to its initial state.

`CNT = step(H,EVENT,RESET)` returns the present value of the count when the `CountOutputPort` property is `true`, and the `HitOutputPort` property is `false`.

`HIT = step(H,EVENT,RESET)` returns a boolean value indicating whether the count has reached any of the values specified by the `HitValues` property. This condition applies when you set the `HitOutputPort` property to `true` and the `CountOutputPort` property to `false`.

`[...] = step(H)` increments or decrements the free-running internal counter when you set the `CountEventInputPort` property to `false` and the `ResetInputPort` property to `false`.

`[...] = step(H,EVENT)` increments or decrements the internal counter when the `EVENT` input matches the event you specify for the `CountEventCondition` property and you set the `ResetInputPort` property to `false`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

signalblks.Crosscorrelator class

Purpose

Cross-correlation of two inputs

Description

The `Crosscorrelator` returns the cross-correlation sequence for two discrete-time deterministic inputs, or the cross-correlation sequence estimate for two discrete-time jointly wide-sense stationary (WSS) random processes.

Construction

`H = signalblks.Crosscorrelator` returns a cross-correlator object, `H`, that computes the cross-correlation of two inputs. If both inputs are vectors, they must both be either row or column vectors. For N-D arrays, the cross-correlator computes the correlation column-wise. The number of columns in the inputs must be equal. If one input is a vector and the other is an N-D array, the cross-correlator computes the cross-correlation of the vector with each column of the N-D array. Cross correlating inputs of length N and M results in a cross-correlation sequence of length $N+M-1$. Cross correlating matrices of size M–by–N and P–by–N results in a matrix of cross-correlation sequences of size $M+P-1$–by–N.

`H = signalblks.Crosscorrelator('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns a cross-correlator, `H`, with each property set to the specified value.

Properties

`Method`

Domain for computing correlations

Specify the domain for computing correlation as `Time Domain`, `Frequency Domain`, or `Fastest`. Computing correlations in the time domain minimizes memory use. Computing correlations in the frequency domain may require fewer computations than computing in the time domain depending on the input length. If the value of this property is `Fastest`, the cross-correlator operates in the domain which minimizes the number of computations. The default value for this property is `Time Domain`.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of Ceiling, Convergent, Floor, Nearest, Round, Simplest, or Zero. The default value for this property is Floor.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of Wrap or Saturate. The default value for this property is Wrap.

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as one of Internal rule, Same as first input, or Custom. The default value for this property is Internal rule.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only when the ProductDataType property is Custom. The default value for this property is numerictype([],32,30).

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as Internal rule, Same as product, Same as first input, or Custom. The default value for this property is Internal rule.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `AccumulatorDataType` property is `Custom`. The default value for this property is `numerictype([],32,30)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as one of `Same as accumulator`, `Same as product`, `Same as first input`, or `Custom`. The default value for this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `OutputDataType` property is `Custom`. The default value for this property is `numerictype([],16,15)`.

Methods

clone	Create cross-correlator object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs of step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Compute cross–correlation sequence

Definitions

The cross-correlation of two length N deterministic inputs, or realizations of jointly WSS random processes, x and y, is:

$$r_{xy}(h) = \begin{cases} \sum_{n=0}^{N-h-1} x(n+h)y^*(n) & 0 \le h \le N-1 \\ r_{yx}^*(h) & -(N-1) \le h \le 0 \end{cases}$$

where h is the lag and * denotes the complex conjugate. If the inputs are realizations of jointly WSS stationary random processes, $r_{xy}(h)$ is an unnormalized estimate of the theoretical cross-correlation:

$$\rho_{xy}(h) = E\{x(n+h)y^*(n)\}$$

where E{ } is the expectation operator.

Examples

Compute correlation between two signals:

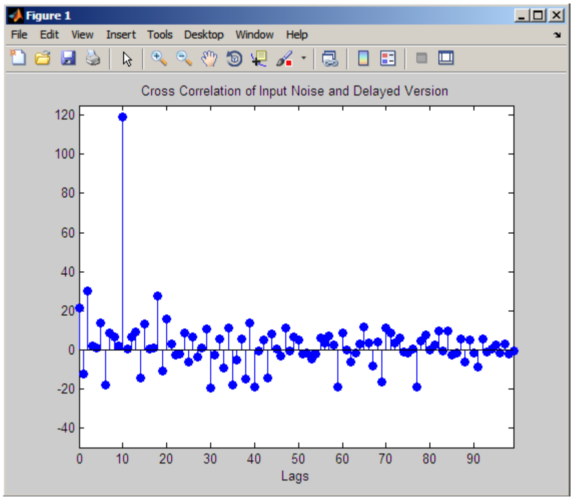
```
hcorr = signalblks.Crosscorrelator;
t=[0:0.001:1];
x1=sin(2*pi*2*t)+0.05*sin(2*pi*50*t);
x2=sin(2*pi*2*t);
y=step(hcorr,x1,x2); %computes cross-correlation of x1 and x2
figure,plot(t,x1,'b',t, x2, 'g');
legend('Input signal 1',' Input signal 2')
figure, plot(y); title('Correlated output')
```

Use cross-correlation to detect delay in jointly stationary white Gaussian noise inputs:

```
S = RandStream('mt19937ar');
RandStream.setDefaultStream(S);
% white Gaussian noise input
x = randn(100,1);
% Create copy delayed by 10 samples
```

signalblks.Crosscorrelator class

```
% x1[n]=x[n-10]
Hdelay = signalblks.Delay(10);
x1= step(Hdelay,x);
Hxcorr = signalblks.Crosscorrelator;
y = step(Hxcorr,x1,x);
lags = 0:99; %Positive lags
stem(lags,y(100:end),'markerfacecolor',[0 0 1]);
axis([0 99 -125 125]);
xlabel('Lags');
title('cross-correlation of Input Noise and Delayed Version');
```



The theoretical cross-correlation sequence is identically zero except at lag 10. Note this is not true in the sample cross-correlation sequence, but the estimate demonstrates a peak at the correct lag.

Algorithm

This object implements the algorithm, inputs, and outputs described on the Correlation block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.Autocorrelator` | `signalblks.Convolver`

Purpose Create cross-correlator object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates a cross-correlator object, `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.Crosscorrelator.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs of step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs of the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.Crosscorrelator.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the cross–correlator.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Compute cross–correlation sequence

Syntax

```
Y = step(H,A,B)
```

Description

`Y = step(H,A,B)` computes the cross–correlation of `A` and `B` and returns the result in `Y`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.CumulativeProduct class

Purpose Compute cumulative product of channel, column, or row elements

Description The `CumulativeProduct` object computes the cumulative product of channel, column, or row elements.

Construction `H = signalblks.CumulativeProduct` returns a cumulative product object, `H`, that computes the cumulative product of input matrix or input vector elements along a specified dimension.

`H = signalblks.CumulativeProduct('PropertyName',PropertyValue,...)` returns a cumulative product object, `H`, with each specified property set to the specified value.

Properties

`Dimension`

Computation dimension for cumulative product

Specify the computation dimension as `Channels (running product)`, `Rows`, or `Columns`. The default value of this property is`Channels (running product)`.

`ResetInputPort`

Enable resetting the cumulative product via an input port

Set this property to `true` to enable resetting the cumulative product. When you set this property to `true`, specify a reset input to the `step` method to reset the cumulative product. `Dimension` property is set to `Channels (running product)`. The default value of this property is `false`.

`ResetCondition`

Reset condition for cumulative product

Specify the event on the reset input port that causes the cumulative product to be reset as one of `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero`. This property applies when you set the `ResetInputPort` property `true` and `Dimension` property to

Channels (running product). The default value of this property is Rising edge.

FrameBasedProcessing

Enable frame-based processing

Set this property to true to enable frame-based processing. Set this property to false to enable sample-based processing. This property is accessible when the Dimension property is set to Channels (running product). The default value of this property is false.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as Ceiling, Convergent, Floor, Nearest, Round, Simplest, or Zero. The default value of this property is Floor.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as Wrap or Saturate. The default value of this property is Wrap.

IntermediateProductDataType

Intermediate product word and fraction lengths

Specify the intermediate product fixed-point data type as Same as input or Custom. The default value of this property is Same as input.

CustomIntermediateProductDataType

Intermediate product word and fraction lengths

Specify the intermediate product fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies when you set the `IntermediateProductDataType` property to `Custom`. The default value of this property is `numerictype([], 16, 15)`.

`ProductDataType`

Product output word and fraction lengths

Specify the product output fixed-point data type as `Same as input` or `Custom`. The default value of this property is `Same as input`.

`CustomProductDataType`

Product output word and fraction lengths

Specify the product output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`. The default value of this property is `numerictype([], 32, 30)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Same as product output`, `Same as input`, or `Custom`. The default value of this property is `Same as input`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([], 32, 30)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as `Same as product output`, `Same as input`, or `Custom`. The default value of this property is `Same as input`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([], 16, 15)`.

Methods

clone	Create cumulative product object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset running cumulative product
step	Compute cumulative product of input along specified dimension for input

Examples

Compute the cumulative product of a matrix.

```
hcumprod = signalblks.CumulativeProduct;
x = magic(2);
y = step(hcumprod,x);
y = step(hcumprod,x)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Cumulative Product block reference page. The object properties correspond to the block parameters, except for:

- The **Reset port** block parameter corresponds to both the `ResetCondition` and the `ResetInputPort` object properties.

See Also

`signalblks.CumulativeSum`

Purpose

Create cumulative product object with same property values

Syntax

`C = clone(H)`

Description

`C = clone(H)` creates a `CumulativeProduct` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

signalblks.CumulativeProduct.getNumInputs

Purpose

Return number of expected inputs to step method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Return number of outputs from step method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.CumulativeProduct.isLocked

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `CumulativeProduct` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Reset running cumulative product

Syntax

```
reset(H)
```

Description

`reset(H)` resets the running cumulative product for object `H` to zero.

signalblks.CumulativeProduct.step

Purpose

Compute cumulative product of input along specified dimension for input

Syntax

```
Y = step(H,X)
Y = step(H,X,R)
```

Description

`Y = step(H,X)` computes the cumulative product along the specified dimension for the input `X`.

`Y = step(H,X,R)` computes the cumulative product of the input elements over time, and optionally resets the cumulative product object's state based on the `ResetCondition` property value, and the value of the reset signal, `R`. This occurs when the `ResetInputPort` property is `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Compute cumulative sum of channel, column, or row elements

Description

The `CumulativeSum` object computes the cumulative sum of channel, column, or row elements.

Construction

`H = signalblks.CumulativeSum` returns a cumulative sum System object, `H`, that computes the running cumulative sum for each channel in the input.

`H = signalblks.CumulativeSum('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a cumulative sum object, `H`, with each specified property set to the specified value.

Properties

`Dimension`

Computation dimension for cumulative sum

Specify the computation dimension as one of `Channels (running sum)`, `Rows`, or `Columns`. The default value of this property is `Channels (running sum)`.

`ResetInputPort`

Enable resetting the cumulative sum via an input port.

Set this property to `true` to enable resetting the cumulative sum. When the property is set to `true`, specify a reset input to the `step` method to reset the cumulative sum. The default value of this property is `false`.

`ResetCondition`

Reset condition for cumulative sum

Specify the event on the reset input port that resets the cumulative sum: `Rising edge`, `Falling edge`, `Either edge`, `Non-zero`. This property applies when you set the `ResetInputPort` property to `true`. The default value of this property is `Rising edge`.

`FrameBasedProcessing`

Enable frame-based processing

Set this property to true to enable frame-based processing. Set this property to false to enable sample-based processing. This property is accessible when the `Dimension` property is set to `Channels (running sum)`. The default value of this property is `false`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Same as input` or `Custom`. The default value of this property is `Same as input`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([], 32, 30)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as one of `Same as accumulator`, `Same as input`, or `Custom`. The default value of this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([], 16, 15)`.

Methods

clone	Create cumulative sum object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset internal states of cumulative sum object to zero
step	Compute cumulative sum along specified dimension for input

Examples

Use a cumulative sum object to compute the cumulative sum of a matrix.

```
hcumsum = signalblks.CumulativeSum;
x = magic(2);
y = step(hcumsum,x);
y = step(hcumsum,x)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Cumulative Sum block reference page. The object properties correspond to the block properties, except for:

- The **Reset port** block parameter corresponds to both the `ResetCondition` and the `ResetInputPort` object properties

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See "What Are Sample- and Frame-Based Processing?" for more information.

See Also

`signalblks.CumulativeProduct`

Purpose Create cumulative sum object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a `CumulativeSum` object `C`, with same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

signalblks.CumulativeSum.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Number of outputs from step method

Syntax

```
getNumOutputs(H)
```

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.CumulativeSum.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax

```
isLocked(H)
```

Description `isLocked(H)` returns the locked state of the `CumulativeSum` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Reset internal states of cumulative sum object to zero

Syntax

`reset(H)`

Description

`reset(H)` sets the states for the running cumulative sum object, `H`, to zero when the `Dimension` property is set to `Channels (running sum)`.

signalblks.CumulativeSum.step

Purpose

Compute cumulative sum along specified dimension for input

Syntax

```
Y = step(H,X)
Y = step(H,X,R)
```

Description

`Y = step(H,X)` computes the cumulative sum along the specified dimension for the input `X`.

`Y = step(H,X,R)` computes the cumulative sum of the input elements over time, and optionally resets the System object's state based on the `ResetCondition` property value, and the value of the reset signal, `R`. Resetting is only possible when the `ResetInputPort` property is `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Discrete cosine transform (DCT)

Description

The `DCT` object computes the discrete cosine transform (DCT) of input.

Construction

`H = signalblks.DCT` returns a discrete cosine transform (DCT) object, `H`, used to compute the DCT of a real or complex input signal.

`H = signalblks.DCT('PropertyName',PropertyValue, ...)` returns a DCT object, `H`, with each property set to the specified value.

Properties

`SineComputation`

Method to compute sines and cosines

Specify how the DCT object computes the trigonometric values as `Trigonometric function` or `Table lookup`. This property must be set to `Table lookup` for fixed-point inputs. The default value for this property is `Table lookup`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling`, `Convergent`, `Floor` , `Nearest`, `Round`, `Simplest`, or `Zero`. This property applies when you set the `SineComputation` property to `Table lookup`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` or `Saturate`. This property applies when you set the `SineComputation` property to `Table lookup`.

`SineTableDataType`

Sine table word-length designation

Specify the sine table fixed-point data type as one of `Same word length as input` or `Custom`. This property applies when you set the `SineComputation` property to `Table lookup`.

`CustomSineTableDataType`

Sine table word length

Specify the sine table fixed-point type as an unscaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `SineComputation` property to `Table lookup` and the `SineTableDataType` property to `Custom`. The default value of this property is `numerictype([],16)`.

`ProductDataType`

Product word and fraction lengths

Specify the product fixed-point data type as one of `Internal rule` , `Same as input`, `Custom`. This property applies when you set the `SineComputation` property to `Table lookup`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `SineComputation` property to `Table lookup` and the `ProductDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Internal rule`, `Same as input`, `Same as product`, or `Custom`. This property applies when you set the `SineComputation` property to `Table lookup`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `SineComputation` property to `Table lookup` and the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as one of `Internal rule` , `Same as input`, `Custom`. This property applies when you set the `SineComputation` property to `Table lookup`. The default value for this property is `Internal rule`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `SineComputation` property to `Table lookup` and the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create discrete cosine transform object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Compute discrete cosine transform (DCT) of input

Examples

Use DCT to analyze the energy content in a sequence:

```
x = (1:128).' + 50*cos((1:128).'*2*pi/40);
hdct = signalblks.DCT;
X = step(hdct, x);
% Set the DCT coefficients which represent less
% than 0.1% of the total energy to 0 and
% reconstruct the sequence using IDCT.
[XX, ind] = sort(abs(X),1,'descend');
ii = 1;
while (norm([XX(1:ii);zeros(128-ii,1)]) <= 0.999*norm(XX))
ii = ii+1;
end
disp(['Number of DCT coefficients that represent 99.9%',...
'of the total energy in the sequence: ',num2str(ii)]);
XXt = zeros(128,1);
XXt(ind(1:ii)) = X(ind(1:ii));
hidct = signalblks.IDCT;
xt = step(hidct, XXt);
plot(1:128,[x xt]);
legend('Original signal','Reconstructed signal',...
 'location','best');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the DCT block reference page. The object properties correspond to the block parameters.

See Also

signalblks.IDCT | signalblks.FFT | signalblks.IFFT

Purpose

Create discrete cosine transform object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates an instance of the current discrete cosine transform object, `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.DCT.getNumInputs

Purpose

Return number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Number of outputs from step method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.DCT.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the DCT object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Compute discrete cosine transform (DCT) of input

Syntax

Description

`Y = step(H,X)` computes the `DCT` of the input `X`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

signalblks.Delay class

Purpose

Delay input by specified number of samples or frames

Description

The `Delay` object delays an input by a specified number of samples or frames.

Construction

`H = signalblks.Delay` returns a delay object, `H`, to delay the input by one sample.

`H = signalblks.Delay('`*`PropertyName`*`',` *`PropertyValue`*`, ...)` returns a delay object, `H`, with each property set to the specified value.

`H = signalblks.Delay(LEN,'`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns a delay object, `H`, with the `Length` property set to `LEN` and other specified properties set to the specified values.

Properties

`Units`

Delay units as samples or frames

Specify the delay units as one of `Samples` or `Frames`. This property applies when you set the `FrameBasedProcessing` property to `true`. The default value for this property is `Samples`.

`Length`

Amount of delay

Specify amount of delay to apply to the input signal. This property can be set to a scalar, a vector, or an array containing nonnegative integers depending on the value of the `FrameBasedProcessing` property.

If the `FrameBasedProcessing` property is `false`, the value can be one of the following:

- A scalar by which to delay all input channels equally, or an N-D array of the same dimensions as the input, whose values specify the number of sample intervals to delay each channel of the input.

If the `FrameBasedProcessing` property is `true`, the value can be one of the following:

- An integer by which to equally delay all channels , a vector whose length is equal to the number of input columns (channels). The default value of this property is 1

`InitialConditionsPerChannel`

Enable different initial conditions per channel

Set this property to `true` to specify different initial conditions per channel. The default value is `false`.

`InitialConditionsPerSample`

Enable different initial conditions per sample

Set this property to `true` to specify different initial conditions per sample. The default value is `false`.

`InitialConditions`

Initial output of delay object

Specify the initial output(s) of the delay object. This property can be set to a scalar, vector, matrix, or a cell array depending on the values of the `FrameBasedProcessing`, `InitialConditionsPerChannel`, `InitialConditionsPerSample`, and `Units` properties. The default value of this property is `0`.

If the `FrameBasedProcessing` property is `false`, and the input is an *N*-D array, the dimensions of this property value must be as follows:

- If the `InitialConditionsPerChannel` and `InitialConditionsPerSample` properties are both `false`, the property value must be a scalar. If the `InitialConditionsPerChannel` property is `true` and the `InitialConditionsPerSample` property is `false`, the value must have the same dimensions as the input. If the

`InitialConditionsPerChannel` property is `false` and the `InitialConditionsPerSample` property is `true`, the value must be a vector of length equal to the `Length` property value.

- If the `InitialConditionsPerChannel` and `InitialConditionsPerSample` properties are both `true`, the property value must be a cell array of the same size as the input signal. Each cell of the cell array represents the delay values for one channel, and must be a vector of length equal to the `Length` property value.

If the `FrameBasedProcessing` property is `true`, and the input is an *M*-by-*N* matrix, the dimensions of this property value must be as follows:

- If the `InitialConditionsPerChannel` and `InitialConditionsPerSample` properties are both `false`, the property value must be a scalar.
- If the `InitialConditionsPerChannel` property is `true` and the `InitialConditionsPerSample` property is `false`, the value must be a vector of length *N*.
- If the `InitialConditionsPerChannel` property is `false`, the `InitialConditionsPerSample` property is `true`, and the `Units` property is `Frames`, the value must be a vector of length equal to the product of *M* and the `Length` property value.
- If the `InitialConditionsPerChannel` property is `false`, the `InitialConditionsPerSample` property is `true`, and the `Units` property is `Samples`, the value must be a vector of length equal to the `Length` property value.
- If the `InitialConditionsPerChannel` and `InitialConditionsPerChannel` properties are both `true`, the property value must be a cell array of size *N*. Each cell of the cell array represents the delay values for one channel.
- If the `Units` property is `Frames`, each cell must be a vector of length equal to the product of *M* and the `Length` property value.

- If the Units property is Samples, each cell must be a vector of length equal to the Length property value.

ResetInputPort

Enable resetting delay states

Specify when the delay object should reset the delay states. By default, the value of this property is false, and the object does not reset the delay states. When this property is set to true, a reset control input is provided to the step method, and the ResetCondition property applies. The object resets the delay states based on the values of the ResetCondition property and the reset input to the step method.

ResetCondition

Reset trigger setting for delay

Specify the event to reset the delay as one of Rising edge, Falling edge, Either edge, or Non-zero. The delay object resets the delay based on the values of this property and the reset input to the step method. This property applies when you set the ResetInputPort property to true. The default value for this property is Non-zero.

FrameBasedProcessing

Enable frame-based processing

Set this property to true to enable frame-based processing. Set this property to false to enable sample-based processing. The default value of this property is true.

Methods

clone	Create delay object with same property values
getNumInputs	Number of expected inputs to step method

getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset delay states
step	Apply delay to input

Examples

Delay input by five samples:

```
hdelay1 = signalblks.Delay(5);
% Output is [0 0 0 0 0 1 2 3 4 5]'
y = step(hdelay1, (1:10)');
```

Delay input by one frame:

```
hdelay2 = signalblks.Delay;
hdelay2.Units = 'Frames';
hdelay2.Length = 1;
% Output is zeros(10,1)
y1 = step(hdelay2, (1:10)');
% Output is (1:10)'
y2 = step(hdelay2, (11:20)');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Delay block reference page. The object properties correspond to the block parameters.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See "What Are Sample- and Frame-Based Processing?" for more information.

See Also

`signalblks.VariableIntegerDelay` | `signalblks.VariableFractionalDelay`.

signalblks.Delay.clone

Purpose Create delay object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a delay object, `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

Purpose

Number of expected inputs to step method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.Delay.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the delay object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.Delay.reset

Purpose Reset delay states

Syntax `reset(H)`

Description `reset(H)` resets the states of the delay object `H` to the values specified in the `InitialConditions` property.

After you invoke the `step` method for a nonzero input, the delay object states may change and invoking the `step` method again without invoking the `reset` method may produce different outputs for identical inputs.

For example:

```
H = signalblks.Delay(5);
% Delay input by 5 samples
y = step(H, (1:10)');
% Output is [0 0 0 0 0 1 2 3 4 5]'
% Invoke step without reset
y1 = step(H,(1:10)');
% Output is [6 7 8 9 10 1 2 3 4 5]'
% Reset states
reset(H);
y2 = step(H,(1:10)');
% Output is [0 0 0 0 0 1 2 3 4 5]'
```

Purpose Apply delay to input

Syntax

```
Y = step(H,X)
Y = step(H,X,R)
```

Description

`Y = step(H,X)` adds delay to input `X` to return output `Y`.

`Y = step(H,X,R)` adds delay to `X` , and selectively resets the delay object's state based on the value of reset input `R` and the value of the `ResetCondition` property. This option is available when the `ResetInputPort` property is `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.DelayLine class

Purpose

Rebuffer sequence of inputs with one-sample shift

Description

The `DelayLine` object rebuffers a sequence of inputs with one-sample shift.

Construction

`H = signalblks.DelayLine` returns a delay line System object, `H`, that buffers the input samples into a sequence of overlapping or underlapping matrix outputs.

`H = signalblks.DelayLine('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a delay line System object, `H`, with each specified property set to the specified value.

`H = signalblks.DelayLine(DELAYSIZE,INITIAL,'`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a delay line System object, `H`, with the `Length` property set to `DELAYSIZE`, `InitialConditions` property set to `INITIAL` and other specified properties set to the specified values.

Properties

`Length`

Number of rows in output matrix

Specify the number of rows in the output matrix as a scalar positive integer. The default value of this property is `64`.

`InitialConditions`

Initial delay line output

Set the value of the object's initial output as `scalar`, `vector`, or `matrix`.

For vector outputs, the following selections apply for the `InitialConditions` property:

- a vector of the same size
- a scalar value to be repeated across all elements of the initial output

For matrix outputs, the following selections apply for the `InitialConditions` property:

- a matrix of the same size
- a vector (equal to the length of the number of matrix rows) that repeats across all columns of the initial output
- a scalar that repeats across all elements of the initial output

The default value of this property is `0`.

`DirectFeedthrough`

Enable passing input data to output without extra frame delay

When you set this property to `true`, there is no input data delay by an extra frame before it is available at the output buffer. Instead, the input data is available immediately at the output. When you set this property to `false`, there is one frame delay on the output. The default value of this property is `false`.

`EnableOutputInputPort`

Enable selective output linearization

The object internally uses a circular buffer, even though the output is linear. To obtain a valid output, the object must linearize the circular buffer. When this property is `true`, the object uses an additional Boolean input to determine if it must calculate a valid output. If Boolean the input value is `1`, the object's output is linearized, and thus valid. If Boolean the input value is `0`, the output is not linearized, and is invalid. This allows the object to be more efficient when each step does not require the tapped Delay Line's output. When you set this property to `false` the output is always linearized and valid. The default value of this property is `false`.

`HoldPreviousValue`

Hold previous valid value for invalid output

If you set this property to `true`, the most recent, valid value is held on the output. If you set this property to `false`, the signal on the output is invalid data. This property applies only when you set the `EnableOutputInputPort` property to `true`. The default value of this property is `false`.

`FrameBasedProcessing`

Enable frame-based processing

Set this property to `true` to enable frame-based processing. Set this property to `false` to enable sample-based processing.

For sample-based processing, the object buffers a sequence of sample-based length-N vector inputs (row, or column) into a sequence of overlapping frame-based Mo-by-N matrix outputs. The `Length` property (Mo>1), specifies Mo. Each input vector becomes a row in the frame-based output matrix. At each `step` call, the object adds the new input vector to the last row of the output; each output overlaps the previous output by Mo-1 samples.

For frame-based processing, the object rebuffers a sequence of frame-based Mi-by-N matrix inputs into a sequence of Mo-by-N matrix outputs. The `Length` property specifies the output frame size. Depending on whether Mo is greater than, less than, or equal to the input frame size, Mi, the output frames are underlapped or overlapped. The object rebuffers each of the N input channels independently.

- When Mo > Mi, the output frame overlap is the difference between the output and input frame size, Mo-Mi.
- When Mo < Mi, the output is underlapped. The object discards the first Mi-Mo samples of each input frame, buffering only the last Mo samples into the corresponding output frame.
- When Mo = Mi, the output data is identical to the input data, except for the object's latency delay.

The default value of this property is `true`.

Methods

clone	Create delay line object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset internal states of delay line object
step	Return delayed version of input X

Examples

Use a delay line object with a delay line size of 4 samples.

```
hdelayline = signalblks.DelayLine( ...
'Length', 4, ...
'DirectFeedthrough', true, ...
'InitialConditions', -2, ...
'FrameBasedProcessing', false, ...
'EnableOutputInputPort', true, ...
'HoldPreviousValue', true);
en = logical([1 1 0 1 0]);
y = zeros(4,5);
for ii = 1:5
y(:,ii) = step(hdelayline, ii, en(ii));
end
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Delay Line block reference page. The object properties correspond to the block parameters.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing`

property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See “What Are Sample- and Frame-Based Processing?” for more information.

See Also

`signalblks.Delay`

signalblks.DelayLine.clone

Purpose Create delay line object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a `DelayLine` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

signalblks.DelayLine.getNumInputs

Purpose

Return number of expected inputs to step method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Number of outputs from step method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.DelayLine.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `DelayLine` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset internal states of delay line object

Syntax

```
reset(H)
```

Description `reset(H)` sets the internal delay buffers of the `DelayLine` object `H` to their initial conditions.

signalblks.DelayLine.step

Purpose Return delayed version of input X

Syntax

```
Y = step(H,X)
Y = step(H,X,EN)
```

Description

`Y = step(H,X)` returns the delayed version of input `X`. `Y` is an output matrix with the same number of rows as the delay line size.

`Y = step(H,X,EN)` selectively outputs the delayed version of input `X` depending on the boolean input `EN` when the `EnableOutputInputPort` property is set to `true`. If `EN` is `false`, use the `HoldPreviousValue` property to specify if the object should hold the previous output value(s).

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Static or time-varying digital filter

Description

The `DigitalFilter` object filters each channel of input using static or time-varying digital filter implementations.

Construction

`H = signalblks.DigitalFilter` returns a IIR digital filter object, `H`, which independently filters each channel of the input over time using a specified digital filter implementation. The numerator coefficients are `[1 2]` and the denominator coefficients are `[1 0.1]`.

`H = signalblks.DigitalFilter('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns a digital filter object, `H`, with each property set to the specified value.

Properties

`TransferFunction`

Type of filter transfer function

Specify the type of transfer function of the digital filter as one of `IIR (poles & zeros)`, `IIR (all poles)`, `FIR (all zeros)`. The default value for this property is `IIR (poles & zeros)`.

`Structure`

Filter structure

Specify the filter structure.

When you set the `TransferFunction` property to `FIR (all zeros)`, you can specify the filter structure as one of `Direct form`,`Direct form symmetric`, `Direct form antisymmetric`, `Direct form transposed`, or `Lattice MA`. The default value for this property is `Direct form`.
When you set the `TransferFunction` property to `IIR (all poles)`, you can specify the filter structure as one of `Direct form`, `Direct form transposed`, or `Lattice AR`. The default value for this property is `Direct form`.
When you set the `TransferFunction` to `IIR (poles & zeros)`, you can specify the filter structure as one of `Direct form I`, `Direct form I transposed`, `Direct form II`, `Direct`

form II transposed, Biquad direct form I (SOS), Biquad direct form I transposed (SOS), Biquad direct form II (SOS), or Biquad direct form II transposed (SOS). The biquad filter structures do not apply when you set the CoefficientsSource property to Input port. The default value for this property is Direct form II transposed.

CoefficientsSource

Source of filter coefficients

Specify the source of the filter coefficients as one of Property or Input port. When the filter coefficients are specified via input port, the digital filter object updates the time-varying filter once every frame if the FrameBasedProcessing property is true, or once every sample otherwise. The default value for this property is Property.

Numerator

Numerator coefficients

Specify the filter numerator coefficients as a real or complex numeric vector. This property applies when you set the TransferFunction property to FIR (all zeros), the CoefficientsSource property to Property and the Structure property is not set to Lattice MA. This property also applies when you set the TransferFunction property to IIR (poles & zeros), the CoefficientsSource to Property, and the TransferFunction property is set to one of Direct form, Direct form symmetric, Direct form antisymmetric or Direct form transposed. The default value of this property is [1 2]. This property is tunable.

Denominator

Denominator coefficients

Specify the filter denominator coefficients as a real or complex numeric vector. This property applies when you set the `TransferFunction` property to `IIR (all poles)`, the `CoefficientsSource` property to `Property` and the `Structure` property is not set to `Lattice AR`. This property also applies when you set the `TransferFunction` property to `IIR (poles & zeros)`, the `CoefficientsSource` to `Property`, and the `TransferFunction` property is set to one of `Direct form`, `Direct form symmetric`, `Direct form antisymmetric`, or `Direct form transposed`. When the `TransferFunction` property is `IIR (poles & zeros)`, the numerator and denominator must have the same complexity. This property is tunable. The default value of this property is `[1 0.1]`.

`ReflectionCoefficients`

Reflection coefficients of lattice filter structure

Specify the reflection coefficients of a lattice filter as a real or complex numeric vector. This property applies when you set the `TransferFunction` property to `FIR (all zeros)`, the `Structure` property to `Lattice MA`, and the `CoefficientsSource` property to `Property`. This property also applies when you set the `TransferFunction` property to `IIR (all poles)`, the `Structure` property to `Lattice AR`, and the `CoefficientsSource` property to `Property`. The default value of this property is `[0.2 0.4]`. This property is tunable.

`SOSMatrix`

SOS matrix of biquad filter structure

Specify the second-order section (SOS) matrix of a biquad filter as an M-by-6 matrix, where M is the number of sections in the filter. Each row of the SOS matrix contains the numerator and denominator coefficients of the corresponding section in the filter. The first three elements of each row are the numerator coefficients and the last three elements are the denominator coefficients. The coefficients can be real or complex. This property applies when you set the `TransferFunction` property to `IIR (poles and zeros)`, the `CoefficientsSource` property to `Property`, and the `Structure` property to one of `Biquad direct form I (SOS)`, Biquad direct form I transposed (SOS), Biquad direct form II (SOS) or Biquad direct form II transposed (SOS). The default value of this property is [1 0.3 0.4 1 0.1 0.2]. This property is tunable.

ScaleValues

Scale values of biquad filter structure

Specify the scale values to be applied before and after each section of a biquadratic filter. The scale values parameter must be a scalar or a vector of length M+1, where M is the number of sections. If this property is set to a scalar, the value is used as the gain value before the first section of the second-order filter. The rest of the gain values are set to 1. If this property is set to a vector of M+1 values, each value is used for a separate section of the filter. This property applies when you set the TransferFunction property to IIR (poles & zeros), the CoefficientsSource property to Property, and the Structure property to one of Biquad direct form I (SOS), Biquad direct form I transposed (SOS), Biquad direct form II (SOS) or Biquad direct form II transposed (SOS). The default value of this property is 1. This property is tunable.

IgnoreFirstDenominatorCoefficient

Assume first denominator coefficient is 1

Setting this Boolean property to true reduces the number of computations the digital filter object must make to produce the output by omitting the first denominator term, *a0*, (poles side) in the filter structure. The object's output is invalid if you set this property to true when the first denominator filter coefficient is not always 1 for your time-varying filter. Note that the object ignores this property for fixed-point inputs, since this object does not support non-unity *a0* coefficients for fixed-point inputs. This property applies when you set the CoefficientsSource property to Input port, the TransferFunction property to IIR (all poles), and the Structure property is not set to Lattice AR. This property also applies when you set the CoefficientsSource property to `Input port`, the `TransferFunction` property to `IIR (poles & zeros)`, and the `Structure` property is set to one of `Direct form`, `Direct form symmetric`, `Direct form antisymmetric`, or `Direct form transposed`. The default value of this property is `true`.

`InitialConditions`

Initial conditions

Specify the initial conditions of the filter states. When the `TransferFunction` property is `FIR (all zeros)` or `IIR (all poles)`, the number of states or delay elements is equal to the number of reflection coefficients for the lattice structure, or the number of filter coefficients-1 for the other direct form structures. When the `TransferFunction` property is `IIR (poles & zeros)`, the number of states for direct form II and direct form II transposed structure is equal to the `max(N,M)-1`, where N and M are the number of poles and zeros respectively. The initial conditions can be specified as a scalar, vector, or matrix. If this property is set to a scalar value, the digital filter object initializes all delay elements in the filter to that value. If this property is set to a vector whose length is equal to the number of delay elements in the filter, each vector element specifies a unique initial condition for a corresponding delay element. The object applies the same vector of initial conditions to each channel of the input signal. If this property is set to a vector whose length is equal to the product of the number of input channels and the number of delay elements in the filter, each element specifies a unique initial condition for a corresponding delay element in a corresponding channel. If this property is set to a matrix with the same number of rows as the number of delay elements in the filter, and one column for each channel of the input signal, each element specifies a unique initial condition for a corresponding delay element in a corresponding channel. This property applies when you do not set the `Structure` property to one of `Direct Form I`, `Direct Form I transposed`, `Biquad direct form`

I (SOS), or Biquad direct form I transposed (SOS). The default value of this property is 0.

NumeratorInitialConditions

Initial conditions on zeros side

Specify the initial conditions of the filter states on the side of the filter structure with the zeros. This property applies when you set the TransferFunction property to IIR (poles & zeros) and the Structure property is one of Direct form I, Direct form I transposed, Biquad direct form I (SOS), or Biquad direct form I transposed (SOS). The default value of this property is 0. To learn how to specify initial conditions, see the InitialConditions property.

DenominatorInitialConditions

Initial conditions on poles side

Specify the initial conditions of the filter states on the side of the direct form I (noncanonic) filter structure with the poles. This property applies when you set the TransferFunction property to IIR (poles & zeros) and the Structure property to Direct form I, Direct form I transposed, Biquad direct form I (SOS), or Biquad direct form I transposed (SOS). The default value of this property is 0. To learn how to specify initial conditions, see the InitialConditions property.

FrameBasedProcessing

Enable frame based processing

Set this property to true to enable frame based processing. Set this property to false to enable sample based processing. The default value of this property is true.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value for this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default value for this property is `Wrap`.

`TapSumDataType`

Tap sum word and fraction lengths

Specify the tap sum fixed-point data type as one of `Same as input` or `Custom`. This property applies when you set the `TransferFunction` property to `FIR (all zeros)` and the `Structure` property to `Direct form symmetric` or `Direct form antisymmetric`. The default value for this property is `Same as input`.

`CustomTapSumDataType`

Tap sum word and fraction lengths

Specify the tap sum fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `TapSumDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`MultiplicandDataType`

Multiplicand word and fraction lengths

Specify the multiplicand fixed-point data type as one of `Same as output` or`Custom`. This property applies when you set the `Structure` property to `Direct form I transposed` or `Biquad direct form I transposed (SOS)`. The default value for this property is `Same as output`.

`CustomMultiplicandDataType`

Multiplicand word and fraction lengths

Specify the multiplicand fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `MultiplicandDataType` property to `Custom`. The default value of this property to `numerictype([],32,30)`.

`SectionInputDataType`

Section input word and fraction lengths

Specify the section input fixed-point data type as one of `Same as input` or `Custom`. Setting this property also sets the `SectionOutputDataType` property to the same value. This property applies when you set the `TransferFunction` property to `IIR (poles & zeros)`, and the `Structure` property corresponds to one of the biquad filter structures. The default value for this property is `Same as input`.

`CustomSectionInputDataType`

Section input word and fraction lengths

Specify the section input fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `SectionInputDataType` property to `Custom`. The word lengths of the `CustomSectionInputDataType` and `CustomSectionOutputDataType` property values must be equal. The default value of this property is `numerictype([],16,15)`.

`SectionOutputDataType`

Section output word and fraction lengths

Specify the section output fixed-point data type as one of `Same as input` or`Custom`. Setting this property also sets the `SectionInputDataType` property to the same value. This property applies when you set the `TransferFunction` property to `IIR (poles & zeros)`, and the `Structure` property corresponds to one of the biquads. The default value for this property is `Same as input`.

`CustomSectionOutputDataType`

Section Output word and fraction lengths

Specify the section output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `SectionOutputDataType` property to `Custom`. The word lengths of the `CustomSectionInputDataType` and `CustomSectionOutputDataType` property values must be equal. The default value of this property is `numerictype([],16,15)`.

`NumeratorCoefficientsDataType`

Numerator coefficients word and fraction lengths

Specify the numerator coefficients fixed-point data type as one of `Same word length as input` or `Custom`. Setting this property also sets the `DenominatorCoefficientsDataType` and the `ScaleValuesDataType` properties, if applicable, to the same value. This property applies when you set the `CoefficientsSource` property to `Property`, and the `TransferFunction` property is not set to `IIR (all poles)`. The default value for this property is `Same word length as input`.

`CustomNumeratorCoefficientsDataType`

Numerator coefficients word and fraction lengths

Specify the numerator coefficients fixed-point type as a `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `NumeratorCoefficientsDataType` property to `Custom`. The word lengths of `CustomNumeratorCoefficientsDataType`, `CustomDenominatorCoefficientsDataType` and the `ScaleValuesDataType` property values, if applicable must be the same. The default value of this property is `numerictype([],16,15)`.

`DenominatorCoefficientsDataType`

Denominator coefficients word and fraction lengths

Specify the denominator coefficients fixed-point data type as one of `Same word length as input` or `Custom`. Setting this property also sets the NumeratorCoefficientsDataType and the ScaleValuesDataType properties, if applicable, to the same value. This property applies when you set the CoefficientsSource property to Property and the TransferFunction property is not set to FIR (all zeros). The default value for this property is Same word length as input.

CustomDenominatorCoefficientsDataType

Denominator coefficients word

and fraction lengths Specify the denominator coefficients fixed-point type as a numerictype object with a Signedness of Auto. This property applies when you set the DenominatorCoefficientsDataType property to Custom. The word lengths of CustomNumeratorCoefficientsDataType, CustomDenominatorCoefficientsDataType and the CustomScaleValuesDataType property values, if they apply, must be the same. The default value of this property is numerictype([],16,15).

ScaleValuesDataType

Scale values word and fraction lengths

Specify the scale values fixed-point data type as one of Same word length as input or Custom. Setting this property also sets the NumeratorCoefficientsDataType and the DenominatorCoefficientsDataType properties, if they apply, to the same value. This property applies when you set the CoefficientsSource property to Property, the TransferFunction property to IIR (poles & zeros), and the Structure property corresponds to one of the biquad filter structures. The default value for this property is Same word length as input.

CustomScaleValuesDataType

Scale values word and fraction lengths

Specify the scale values fixed-point type as a `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `CoefficientsSource` property to `Property` and the `ScaleValuesDataType` property to `Custom`. The word lengths of the `CustomNumeratorCoefficientsDataType`, `CustomDenominatorCoefficientsDataType` and the `CustomScaleValuesDataType` property values, if applicable, must be the same. The default value of this property is `numerictype([],16,15)`.

`ProductDataType`

Product word and fraction lengths

Specify the product fixed-point data type as one of `Same as input` or `Custom`. The default value for this property is `Same as input`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Same as input`, `Same as product`, or `Custom`. The default value for this property is `Same as product`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`StateDataType`

State word and fraction lengths

Specify the state fixed-point data type as one of `Same as input` , `Same as accumulator`, or `Custom`. This property does not apply to any of the direct form or direct form I filter structures. The default value for this property is `Same as accumulator`.

`CustomStateDataType`

State word and fraction lengths

Specify the state fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `StateDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as one of `Same as input` , `Same as accumulator`, or `Custom`. The default value for this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create static or time-varying digital filter with same property values
getNumInputs	Return number of expected inputs to step method

getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset internal states of digital filter
step	Filter input with digital filter object

Examples

Low pass filter waveform with two sinusoidal components using an FIR filter.

```
t = [0:63]./32e3;
xin = (sin(2*pi*4e3*t)+sin(2*pi*12e3*t)) / 2;
hSR = signalblks.SignalReader(xin', 4);
hLog = signalblks.SignalLogger;
hFilt = signalblks.DigitalFilter;
hFilt.TransferFunction = 'FIR (all zeros)';
hFilt.Numerator = fir1(10,0.5);
while ~isDone(hSR)
input = step(hSR);
filteredOutput = step(hFilt,input);
step(hLog,filteredOutput);
end
filteredResult = hLog.Buffer;
sper = spectrum.periodogram;
psd(sper,filteredResult,'Fs',32e3)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Digital Filter block reference page. The object properties correspond to the block parameters.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing`

property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See "What Are Sample- and Frame-Based Processing?" for more information.

See Also

`signalblks.BiquadFilter`

Purpose Create static or time-varying digital filter with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates a digital filter object, `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

signalblks.DigitalFilter.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the step method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.DigitalFilter.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the digital filter.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset internal states of digital filter

Syntax `reset(H)`

Description `reset(H)` resets the filter states of the digital filter object,`H`, to their initial values of 0. The initial filter state values correspond to the initial conditions for the difference equation defining the filter. After the `step` method applies the digital filter object to nonzero input data, the states may be nonzero. Invoking the `step` method again without first invoking the`reset` method may produce different outputs for an identical input.

For example:

```
H = signalblks.DigitalFilter;
H.TransferFunction = 'FIR (all zeros)';
H.Numerator = fir1(20,0.25);
n = 0:100;
x = cos(0.2*pi*n)+sin(0.8*pi*n);
y = step(H,x);
% Filter states are nonzero
% Invoke step method again without resetting states
y1 = step(H,x);
isequal(y,y1) % returns 0
% Now reset filter states to 0
reset(H)
% Invoke step method
y2 = step(H,x);
isequal(y,y2) % returns a 1
```

signalblks.DigitalFilter.step

Purpose Filter input with digital filter object

Syntax

```
Y = step(HFILT,X)
Y = step(HFILT,X,COEFF)
Y = step(HFILT,X,NUM,DEN)
```

Description

`Y = step(HFILT,X)` filters the real or complex input signal `X` using the digital filter, `H`, to produce the output `Y` . The digital filter objects operates on each channel of the input signal independently over time.

`Y = step(HFILT,X,COEFF)` uses the time-varying numerator or denominator coefficients, `COEFF` , to filter the input signal `X` and produce the output `Y` . This option is possible when the `TransferFunction` property is either `FIR (all zeros)` or `IIR (all poles)` and the `CoefficientsSource` property is `Input port`.

`Y = step(HFILT,X,NUM,DEN)` uses the time-varying numerator coefficients `NUM` and the time-varying denominator coefficients `DEN` to filter the input signal `X` and produce the output `Y` . This option is possible when the `TransferFunction` property is`IIR (poles and zeros)` and the `CoefficientsSource` property is `Input port`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Dyadic analysis filter bank

Description

The `DyadicAnalysisFilterBank` object uses a series of highpass and lowpass FIR filters to provide approximate octave band frequency decompositions of the input. Each filter output is downsampled by a factor of two. With the appropriate analysis filters and tree structure, the dyadic analysis filter bank represents a discrete wavelet transform (DWT), or discrete wavelet packet transform (DWPT).

Construction

`H = signalblks.DyadicAnalysisFilterBank` constructs a dyadic analysis filter bank object, `H`, that computes the level-two discrete wavelet transform (DWT) of a column vector input, or the columns of an input 2D matrix, using the Daubechies 3rd-order extremal phase wavelet. The length of the input along the first dimension must be a multiple of 4.

`H = signalblks.DyadicAnalysisFilterBank('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns a dyadic analysis filter bank object, `H`, with each property set to the specified value.

Properties

`Filter`

Type of filter used in the subband decomposition

Specify the type of filter used to determine the high and lowpass FIR filters in the dyadic analysis filter bank as `Custom` , `Haar`, `Daubechies`, `Symlets`, `Coiflets`, `Biorthogonal`, `Reverse Biorthogonal`, or `Discrete Meyer`. All property values except `Custom` require Wavelet Toolbox software. If the value of this property is `Custom`, the filter coefficients are specified by the values of the `CustomLowpassFilter` and `CustomHighpassFilter` properties. Otherwise, the dyadic analysis filter bank object uses the Wavelet Toolbox function `wfilters` to construct the filters. The following table lists supported wavelet filters and example syntax to construct the filters:

Filter	Example setting	Syntax for analysis filters
Haar	N/A	`[Lo_D,Hi_D]=wfilters('haar');`
Daubechies extremal phase	`WaveletOrder=3;`	`[Lo_D,Hi_D]=wfilters('db3');`
Symlets (Daubechies least-asymmetric)	`WaveletOrder=4;`	`[Lo_D,Hi_D]=wfilters('sym4');`
Coiflets	`WaveletOrder=1;`	`[Lo_D,Hi_D]=wfilters('coif1');`
Biorthogonal	`FilterOrder='[3/1]';`	`[Lo_D,Hi_D,Lo_R,Hi_R]=... wfilters('bior3.1');`
Reverse biorthogonal	`FilterOrder='[3/1]';`	`[Lo_D,Hi_D,Lo_R,Hi_R]=... wfilters('rbior3.1');`
Discrete Meyer	N/A	`[Lo_D,Hi_D]=wfilters('dmey');`

`CustomLowpassFilter`

Lowpass FIR filter coefficients

Specify a vector of lowpass FIR filter coefficients, in powers of z^{-1}. The lowpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the `CustomHighpassFilter` property. This property applies when you set the `Filter` property to `Custom`. The default value for this property specifies a Daubechies 3rd-order extremal phase scaling (lowpass) filter.

`CustomHighpassFilter`

Highpass FIR filter coefficients

Specify a vector of highpass FIR filter coefficients, in powers of z^{-1}. The highpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the

CustomLowpassFilter property. This property applies when you set the Filter property to Custom. The default value for this property specifies a Daubechies 3rd-order extremal phase wavelet (highpass) filter.

WaveletOrder

Order for orthogonal wavelets

Specify the order of the wavelet selected in the Filter property. This property applies when you set the Filter property to an orthogonal wavelet: Daubechies (Daubechies extremal phase), Symlets (Daubechies least-asymmetric), or Coiflets. The default value of this property is 2.

FilterOrder

Analysis and synthesis filter orders for biorthogonal filters

Specify the order of the analysis and synthesis filter orders for biorthogonal filter banks as 1 / 1, 1 / 3, 1 / 5, 2 / 2, 2 / 4, 2 / 6, 2 / 8, 3 / 1, 3 / 3, 3 / 5, 3 / 7, 3 / 9, 4 / 4, or 5 / 5, 6 / 8. Unlike orthogonal wavelets, biorthogonal wavelets require different filters for the analysis (decomposition) and synthesis (reconstruction) of an input. The first number indicates the order of the synthesis (reconstruction) filter. The second number indicates the order of the analysis (decomposition) filter. This property applies when you set the Filter property to Biorthogonal or Reverse Biorthogonal. The default value for this property is 1 / 1.

NumLevels

Number of filter bank levels used in the analysis (decomposition)

Specify the number of filter bank analysis levels a positive integer. A level-N asymmetric structure produces $N+1$ output subbands. A level-N symmetric structure produces 2^N output subbands. The default value of this property is 2. The size of the input along the first dimension must be a multiple of 2^N, where N is the number of levels.

`TreeStructure`

Structure of filter bank

Specify the structure of the filter bank as `Asymmetric` or `Symmetric`. The asymmetric structure decomposes only the lowpass filter output from each level, while the symmetric structure decomposes the highpass and lowpass filter outputs from each level. If the analysis filters are scaling (lowpass) and wavelet (highpass) filters, the asymmetric structure is the discrete wavelet transform, while the symmetric structure is the discrete wavelet packet transform. When this property is `Symmetric`, the output consists of 2^N subbands each of size $M/2^N$, where M is the length of the input along the first dimension and N is the value of the `NumLevels` property. When this property is `Asymmetric`, the output consists of $N+1$ subbands. The following equation gives the length of the output in the k–th subband in the asymmetric case:

$$M_k = \begin{cases} \dfrac{M}{2^k} & 1 \le k \le N \\ \dfrac{M}{2^N} & k = N+1 \end{cases}$$

The default value for this property is `Asymmetric`.

Methods

clone	Create dyadic analysis filter bank object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs of step method
isLocked	Locked status (logical) for input attributes and non-tunable properties

reset	Reset filter states
step	Decompose input with dyadic filter bank

Examples

Denoise square wave input using dyadic analysis and synthesis filter banks:

```
t = 0:.0001:.0511;
x= square(2*pi*30*t);
xn = x' + 0.08*randn(length(x),1);
hdydanl = signalblks.DyadicAnalysisFilterBank;
% The filter coefficients correspond to a 'haar' wavelet.
hdydanl.CustomLowpassFilter = [1/sqrt(2) 1/sqrt(2)];
hdydanl.CustomHighpassFilter = [-1/sqrt(2) 1/sqrt(2)];
hdydsyn = signalblks.DyadicSynthesisFilterBank;
hdydsyn.CustomLowpassFilter = [1/sqrt(2) 1/sqrt(2)];
hdydsyn.CustomHighpassFilter = [1/sqrt(2) -1/sqrt(2)];
C = step(hdydanl, xn);
% Subband outputs
C1 = C(1:256); C2 = C(257:384); C3 = C(385:512);
% Set higher frequency coefficients to zero
% to remove the noise.
x_den = step(hdydsyn, [zeros(length(C1),1);...
zeros(length(C2),1);C3]);
subplot(2,1,1), plot(xn); title('Original noisy Signal');
subplot(2,1,2), plot(x_den); title('Denoised Signal');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Dyadic Analysis Filter Bank block reference page. The object properties correspond to the block parameters, except for:

- The dyadic analysis filter bank object always concatenates the subbands into a single column vector for a column vector input, or into the columns of a matrix for a matrix input. This behavior corresponds to the block's behavior when you set the **Output** parameter to `Single port`.

See Also

`signalblks.DyadicSynthesisFilterBank` | `signalblks.SubbandAnalysisFilter`

Purpose

Create dyadic analysis filter bank object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates a dyadic analysis filter bank object, `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

signalblks.DyadicAnalysisFilterBank.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Return number of outputs of step method

Syntax

```
getNumOutputs(H)
```

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.DyadicAnalysisFilterBank.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the dyadic analysis filter bank object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset filter states

Syntax `reset(H)`

Description `reset(H)` resets the filter states of the dyadic analysis filter bank object, `H`, to their initial values of zero. After the `step` method applies the dyadic analysis filter bank to nonzero input data, the filter states may change. Invoking the `step` method again without first invoking the `reset` method may produce different outputs for an identical input.

Example of resetting filter states:

```
X = [1 1 7 9 2 8 8 6]';
H = signalblks.DyadicAnalysisFilterBank('NumLevels',1);
% Filter states are zero
y = step(H,X);
% Invoke step method again without resetting states
y1 = step(H,X);
isequal(y,y1) % Returns 0
reset(H); % Reset filter states to zero
y2 = step(H,X);
isequal(y,y2) % Returns 1
```

signalblks.DyadicAnalysisFilterBank.step

Purpose Decompose input with dyadic filter bank

Syntax

```
Y = step(H,X)
```

Description

`Y = step(H,X)` computes the subband decomposition of the input `X` and outputs the dyadic subband decomposition in `Y` as a single concatenated column vector or matrix of coefficients. Each column of `X` is treated as an independent input, and the number of rows of `X` must be a multiple of 2^N, where N is the value of the `NumLevels` property. The elements of `Y` are ordered with the highest frequency subband first followed by subbands in decreasing frequency. The following examples illustrate the subband ordering for both asymmetric and symmetric tree structures.

Subband ordering for level-two asymmetric tree structure:

```
t = 0:.001:1.023;
% Sampling frequency 1 kHz input length 1024
x= square(2*pi*30*t);
xn = x' + 0.08*randn(length(x),1);
% Default asymmetric structure with
% Daubechies order 3 extremal phase wavelet
H = signalblks.DyadicAnalysisFilterBank;
Y = step(H,xn);
% Level 2 yields 3 subbands (two detail-one approximation)
% Nyquist frequency is 500 Hz
D1 =Y(1:512); % subband approx. [250, 500] Hz
D2 = Y(513:768); % subband approx. [125, 250] Hz
Approx = Y(769:1024); % subband approx. [0,125] Hz
```

Subband ordering for symmetric tree structure:

```
t = 0:.001:1.023;
% Sampling frequency 1 kHz input length 1024
x= square(2*pi*30*t);
xn = x' + 0.08*randn(length(x),1); % symmetric structure with
% Daubechies order 3 extremal phase wavelet
```

```
H = signalblks.DyadicAnalysisFilterBank('TreeStructure',...
'Symmetric');
Y = step(H,xn);
D1 = Y(1:256); % subband approx. [375,500] Hz
D2 = Y(257:512); % subband approx. [250,375] Hz
D3 = Y(513:768); % subband approx. [125,250] Hz
Approx = Y(769:1024); % subband approx. [0, 125] Hz
```

signalblks.DyadicSynthesisFilterBank class

Purpose

Reconstruct signals from subbands with smaller bandwidths and smaller sample rates

Description

The `DyadicSynthesisFilterBank` object reconstructs signals from subbands with smaller bandwidths and lower sample rates. The filter bank uses a series of highpass and lowpass FIR filters to repeatedly reconstruct the signal.

Construction

`H = signalblks.DyadicSynthesisFilterBank` returns a synthesis filter bank, `H`, that reconstructs a signal from its subbands with smaller bandwidths and smaller sample rates.

`H = signalblks.DyadicSynthesisFilterBank('`*`Propertyname`*`',`*`PropertyValue`*`,...)` returns a synthesis filter bank, `H`, with each specified property set to the specified value.

Properties

`Filter`

Type of filter used in filter bank

Specify the type of filter used to determine the highpass and lowpass FIR filters in the filter bank as one of `Custom`, `Haar`, `Daubechies`, `Symlets`, `Coiflets`, `Biorthogonal`, `Reverse Biorthogonal`, or `Discrete Meyer`. If you set this property to `Custom`, the `CustomLowpassFilter` and `CustomHighpassFilter` properties specify the filter coefficients. Otherwise, the object uses the Wavelet Toolbox `wfilters` function to construct the filters. Depending on the filter, the `WaveletOrder` or `FilterOrder` property might apply. For a list of the supported wavelets, see the table below.

Filter	Sample Setting for Related Filter Specification Properties	Corresponding Wavelet Toolbox Function Syntax
Haar	None	wfilters('haar')
Daubechies	H.WaveletOrder = 4	wfilters('db4')
Symlets	H.WaveletOrder = 3	wfilters('sym3')
Coiflets	H.WaveletOrder = 1	wfilters('coif1')
Biorthogonal	H.FilterOrder = '[3/1]'	wfilters('bior3.1')
Reverse Biorthogonal	H.FilterOrder = '[3/1]'	wfilters('rbior3.1')
Discrete Meyer	None	wfilters('dmey')

In order to automatically design wavelet-based filters, install the Wavelet Toolbox product. Otherwise, use the `CustomLowpassFilter` and `CustomHighpassFilter` properties to specify lowpass and highpass FIR filters.

`CustomLowpassFilter`

Lowpass FIR filter coefficients

Specify a vector of lowpass FIR filter coefficients, in descending powers of z. The lowpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the `CustomHighpassFilter` property. This property applies only when you set the `Filter` property to `Custom`. To perfectly reconstruct a signal decomposed by the `DyadicAnalysisFilterBank`, design the filters in the synthesis filter bank to perfectly reconstruct the outputs of the analysis filter bank. Otherwise, the reconstruction is imperfect. The default values of this property specify a perfect reconstruction filter for the default settings of the analysis filter bank (based on a third-order Daubechies wavelet).

`CustomHighpassFilter`

Highpass FIR filter coefficients

Specify a vector of highpass FIR filter coefficients, in descending powers of z. The highpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the `CustomLowpassFilter` property. This property applies only when you set the `Filter` property to `Custom`. To perfectly reconstruct a signal decomposed by the `DyadicAnalysisFilterBank`, design the filters in the synthesis filter bank to perfectly reconstruct the outputs of the analysis filter bank. Otherwise, the reconstruction is imperfect. The default values of this property specify a perfect reconstruction filter for the default settings of the analysis filter bank (based on a third-order Daubechies wavelet).

`WaveletOrder`

Wavelet order

Specify the order of the wavelet selected in the `Filter` property. This property applies only when you set the `Filter` property to `Daubechies`, `Symlets` or `Coiflets`. The default value of this property is `2`.

`FilterOrder`

Wavelet order for synthesis filter stage

Specify the order of the wavelet for the synthesis filter stage as:

- First order: `'[1/1]'`, `'[1/3]'`, or `'[1/5]'`.
- Second order: `'[2/2]'`, `'[2/4]'`, `'[2/6]'`, or `'[2/8]'`.
- Third order: `'[3/1]'`, `'[3/3]'`, `'[3/5]'`, `'[3/7]'`, or `'[3/9]'`.
- Fourth order: `'[4/4]'`.
- Fifth order: `'[5/5]'`.

- Sixth order: `'[6/8]'`.

This property applies only when you set the `Filter` property to `Biorthogonal` or `Reverse Biorthogonal`. The default value of this property is `'[1/1]'`.

`NumLevels`

Number of filter bank levels

Specify the number of filter bank levels as a scalar integer. An N-level asymmetric structure has $N + 1$ input subbands, and an N-level symmetric structure has 2^N input subbands. The default value of this property is `2`.

`TreeStructure`

Structure of filter bank

Specify the structure of the filter bank as `Asymmetric` or `Symmetric`. In the asymmetric structure, the low-frequency subband input to each level is the output of the previous level, while the high-frequency subband input to each level is an input to the filter bank. In the symmetric structure, both the low- and high-frequency subband inputs to each level are outputs from the previous level. The default value of this property is `Asymmetric`.

Methods

clone	Create synthesis filter bank object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs from the step method
isLocked	Return logical value to indicate whether input attributes and non-tunable properties are locked

reset	Reset internal states of dyadic synthesis filter bank object
step	Reconstruct signal from high- and low-frequency subbands

Examples

Remove noise from a signal using filter banks:

```
t = 0:.0001:.0511;
x= square(2*pi*30*t);
xn = x' + 0.08*randn(length(x),1);
hdydanl = signalblks.DyadicAnalysisFilterBank;
% The filter coefficients correspond to a 'haar' wavelet.
hdydanl.CustomLowpassFilter = [1/sqrt(2) 1/sqrt(2)];
hdydanl.CustomHighpassFilter = [-1/sqrt(2) 1/sqrt(2)];
hdydsyn = signalblks.DyadicSynthesisFilterBank;
hdydsyn.CustomLowpassFilter = [1/sqrt(2) 1/sqrt(2)];
hdydsyn.CustomHighpassFilter = [1/sqrt(2) -1/sqrt(2)];
C = step(hdydanl, xn);
% Subband outputs
C1 = C(1:256); C2 = C(257:384); C3 = C(385:512);
% Set high-frequency coefficients to zero to remove noise.
x_den = step(hdydsyn, ...
  [zeros(length(C1),1); ...
   zeros(length(C2),1); ...
   C3]);
subplot(2,1,1), plot(xn); title('Original Noisy Signal');
subplot(2,1,2), plot(x_den); title('Denoised Signal');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Dyadic Synthesis Filter Bank block reference page. The object properties correspond to the block parameters, except for:

- The object only receives data as a vector or matrix of concatenated subbands, as specified using the `step` method.

See Also

`signalblks.DyadicAnalysisFilterBank` |
`signalblks.SubbandSynthesisFilter`

signalblks.DyadicSynthesisFilterBank.clone

Purpose Create synthesis filter bank object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a `DyadicSynthesisFilterBank` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.DyadicSynthesisFilterBank.getNumOutputs

Purpose Return number of outputs from the step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Return logical value to indicate whether input attributes and non-tunable properties are locked

Syntax

```
isLocked(H)
```

Description

`isLocked(H)` returns the locked state of the `DyadicSynthesisFilterBank` object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.DyadicSynthesisFilterBank.reset

Purpose Reset internal states of dyadic synthesis filter bank object

Syntax `reset(H)`

Description `reset(H)` sets the internal states of the dyadic synthesis filter bank object `H` to their initial values.

Purpose Reconstruct signal from high- and low-frequency subbands

Syntax

```
X = step(H,S)
```

Description `X = step(H,S)` reconstructs the concatenated subband input `S` to output `X`. Each column of input `S` contains the subbands for an independent signal, with upper rows containing the high-frequency subbands and lower rows containing the low-frequency subbands. The number of rows of `S` must be a multiple of 2^N, where N is the value of the `NumLevels` property.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.FFT class

Purpose

Discrete Fourier transform

Description

The `FFT` object computes the discrete Fourier transform (DFT) of an input.

Construction

`H = signalblks.FFT` returns a `FFT` object, `H`, that computes the DFT of an N-D array. For column vectors or multidimensional arrays, the `FFT` object computes the DFT along the first dimension. If the input is a row vector, the `FFT` object computes a row of single-sample DFTs and issues a warning.

`H = signalblks.FFT('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns a `FFT` object, `H`, with each property set to the specified value.

Properties

`TwiddleFactorComputation`

Computation method of twiddle factor

Specify the computation method of the twiddle factor term as `Trigonometric function` or `Table lookup`. The property must `Table lookup` for fixed-point inputs. The default value for this property is `Table lookup`.

`TableOptimization`

Optimization of trigonometric values table

Select the optimization of table of trigonometric values table as `Speed` or `Memory` . This property applies only when the `TwiddleFactorComputation` property is `Table lookup`. The property must be `Speed` for fixed-point inputs. The default value for this property is `Speed`.

`BitReversedOutput`

Order of output elements relative to input elements

Designate order of output channel elements relative to order of input elements. Set this property to `true` to output the frequency indices in bit-reversed order. The default value of this property is `false`, which corresponds to a linear ordering of frequency indices.

`Normalize`

Divide butterfly outputs by two

Set this property to `true` to divide each butterfly of the FFT by two. The default value of this property is `false` and no scaling occurs.

`FFTLengthSource`

Source of FFT length

Specify how to determine the FFT length as `Auto` or `Property`. When this property is `Auto`, the FFT length is equal to the number of rows of the input signal and must be a power of two. The default value for this property is `Auto`.

`FFTLength`

FFT length as power of two

Specify the FFT length as a power of two. This property applies only when the `FFTLengthSource` property is `Property`. The default value of this property is 64.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, `Zero`. This property applies only when the `TwiddleFactorComputation` property is `Table lookup` and the `TableOptimization` property is `Speed`. The default value for this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. This property applies only when the `TwiddleFactorComputation` property is

`Table lookup` and the `TableOptimization` property is `Speed`. The default value for this property is `Wrap`.

`SineTableDataType`

Sine table word and fraction lengths

Specify the sine table data type as `Same word length as input` or `Custom`. This property applies only when the `TwiddleFactorComputation` property is `Table lookup` and the `TableOptimization` property is `Speed`. The default value for this property is `Same word length as input`.

`CustomSineTableDataType`

Sine table word and fraction lengths

Specify the sine table fixed-point type as an unscaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `TwiddleFactorComputation` property is `Table lookup`, the `TableOptimization` property is `Speed`, and the `SineTableDataType` property is `Custom`. The default value of this property is `numerictype([],16)`.

`ProductDataType`

Product word and fraction lengths

Specify the product data type as `Internal rule`, `Same as input`, or `Custom`. This property applies only when the `TwiddleFactorComputation` property is `Table lookup` and the `TableOptimization` property is `Speed`. The default value for this property is `Internal rule`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the`TwiddleFactorComputation` property is `Table lookup`, the `TableOptimization` property is `Speed`, and the

ProductDataType property is Custom. The default value of this property is numerictype([],32,30).

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator data type as Internal rule, Same as input, Same as product, or Custom. This property applies only when the TwiddleFactorComputation property is Table lookup and the TableOptimization property is Speed. The default value for this property is Internal rule.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only when the TwiddleFactorComputation property is Table lookup, the TableOptimization property is Speed and the AccumulatorDataType property is Custom. The default value of this property is numerictype([],32,30).

OutputDataType

Output word and fraction lengths

Specify the output data type as one of Internal rule, Same as input, Custom. This property applies only when the TwiddleFactorComputation property is Table lookup and the TableOptimization property is Speed. The default value for this property is Internal rule.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only when theTwiddleFactorComputation property is Table lookup, theTableOptimization property is Speed, and the

`OutputDataType` property is `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create FFT object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs of step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Compute discrete Fourier transform of input

Examples

Find frequency components of a signal in additive noise:

```
Fs = 800; L = 1000;
t = (0:L-1)'/Fs;
x = sin(2*pi*250*t) + 0.75*cos(2*pi*340*t);
y = x + .5*randn(size(x)); % noisy signal
hfft = signalblks.FFT('FFTLengthSource', 'Property', ...
'FFTLength', 1024);
Y = step(hfft, y);
% Plot the single-sided amplitude spectrum
plot(Fs/2*linspace(0,1,512), 2*abs(Y(1:512)/1024));
title('Single-sided amplitude spectrum of noisy signal y(t)');
xlabel('Frequency (Hz)'); ylabel('|Y(f)|');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the FFT block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.IFFT` | `signalblks.DCT` | `signalblks.IDCT`

signalblks.FFT.clone

Purpose

Create FFT object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates an `FFT` object with the same property values as `H`. The clone method creates a new unlocked object.

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.FFT.getNumOutputs

Purpose

Return number of outputs of step method

Syntax

```
getNumOutputs(H)
```

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the FFT object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.FFT.step

Purpose

Compute discrete Fourier transform of input

Syntax

`Y = step(H,X)`

Description

`Y = step(H,X)` computes the FFT, `Y` , of the input `X` along the first dimension of `X`. The length of `X` along the first dimension is `N`. When the `FFTLengthSource` property is `Auto`, `N` should a positive integer power of two and the `FFT` length is assumed to be `N` . When the `FFTLengthSource` property is `Property`, `N` can be any positive integer value and the `FFTLength` property must have a positive integer power of two value.

Purpose

FIR polyphase decimator

Description

The `FIRDecimator` object resamples vector or matrix inputs along the first dimension at a rate M times slower than the input sampling rate, where M is the integer-valued downsampling factor. The decimation combines an FIR anti-aliasing filter with downsampling. The FIR decimator object uses a polyphase implementation of the FIR filter.

Construction

`H = signalblks.FIRDecimator` returns an FIR decimator, `H`, which applies an FIR filter with a cutoff frequency of `0.4*pi` radians/sample to the input and downsamples the filter output by factor of 2.

`H = signalblks.FIRDecimator ('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns an FIR decimator, `H`, with each property set to the specified value.

`H = signalblks.FIRDecimator(DECIM, NUM, '`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns an FIR decimator, `H`, with the integer-valued `DecimationFactor` property set to `DECIM`, the `Numerator` property set to `NUM`, and other specified properties set to the specified values.

Properties

`DecimationFactor`

Decimation factor

Specify the downsampling factor as a positive integer. The FIR decimator reduces the sampling rate of the input by this factor. The size of the input along the first dimension must be a multiple of the decimation factor. The default value for this property is `2`.

`Numerator`

FIR filter coefficients

Specify the numerator coefficients of the FIR filter in powers of z^{-1}. The following equation defines the system function for a filter of length L:

$$H(z) = \sum_{l=0}^{L-1} b_l z^{-l}$$

To prevent aliasing as a result of downsampling, the filter transfer function should have a normalized cutoff frequency no greater than 1/`DecimationFactor`. You can specify the filter coefficients as a vector in the supported data types. The FIR decimator does not support `dfilt` or `mfilt` objects as sources of the filter coefficients. The default value of this property is `fir1(35,0.4)`.

`Structure`

Filter structure

Specify the implementation of the FIR filter as `Direct form` or `Direct form transposed`. The default value for this property is `Direct form`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value for this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` or `Saturate`.

`CoefficientsDataType`

Coefficients word and fraction lengths

Specify the coefficients fixed-point data type as `Same word length as input` or `Custom`. The default value for this property is `Same word length as input`.

`CustomCoefficientsDataType`

Coefficients word and fraction lengths

Specify the coefficients fixed-point type as a `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `CoefficientsDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

`ProductDataType`

Product word and fraction lengths

Specify the product fixed-point data type as `Internal rule` , `Same as input`, or `Custom`. The default value for this property is `Internal rule`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Internal rule` , `Same as product`, `Same as input`, or `Custom`. The default value for this property is `Internal rule`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as `Same as accumulator`, `Same as product`, `Same as input`, or `Custom`. The default value for this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([]1,16,15)`.

Methods

clone	Create FIR decimator object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs of step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset filter states of FIR decimator
step	Decimate input by integer factor

Definitions

Polyphase Subfilters

A polyphase implementation of an FIR decimator *splits* the lowpass FIR filter impulse response into M different subfilters, where M is the downsampling, or decimation factor. Let $h(n)$ denote the FIR filter impulse response of length L and $u(n)$ the input signal. Decimating the filter output by a factor of M is equivalent to the downsampled convolution:

$$y(n) = \sum_{l=0}^{L-1} h(l)u(nM - l)$$

The key to the efficiency of polyphase filtering is that specific input values are only multiplied by select values of the impulse response in the downsampled convolution. For example, letting *M=2*, the input values *u(0),u(2),u(4), ...* are only combined with the filter coefficients *h(0),h(2),h(4),...*, and the input values *u(1),u(3),u(5), ...* are only combined with the filter coefficients *h(1),h(3),h(5),...*. By splitting the filter coefficients into two polyphase subfilters, no unnecessary computations are performed in the convolution. The outputs of the convolutions with the polyphase subfilters are interleaved and summed to yield the filter output. The following MATLAB code demonstrates how to construct the two polyphase subfilters for the default order 35 filter in the `Numerator` property and the default `DecimationFactor` property value of two:

```
M = 2;
Num = fir1(35,0.4);
FiltLength = length(Num);
Num = flipud(Num(:));
  if (rem(FiltLength, M) ~= 0)
     nzeros = M - rem(FiltLength, M);
     % Appending zeros if necessary
     Num = [zeros(nzeros,1); Num];
   end
len = length(Num);
nrows = len / M;
PolyphaseFilt = flipud(reshape(Num, M, nrows).');
```

Note that the columns of `PolyphaseFilt` are subfilters containing the two *phases* of the filter in `Num`. For a general downsampling factor of M, there are M phases and therefore M subfilters.

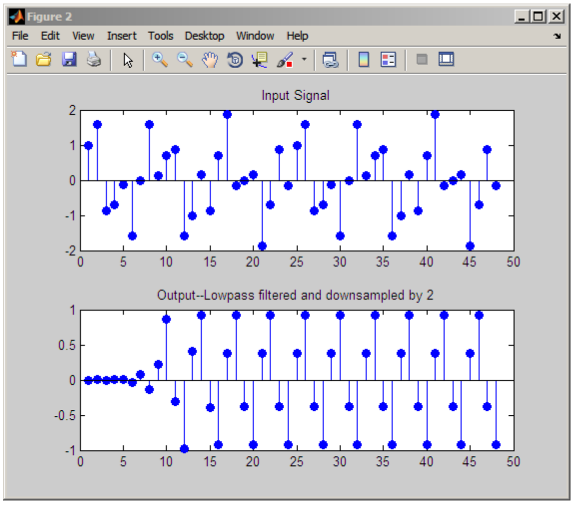
Examples

Decimate a sum of sine waves with angular frequencies of π/4 and 2π/3 radians/sample by a factor of two. To prevent aliasing, the

FIR decimator filters out the 2π/3 radians/sample component before downsampling:

```
x = cos(pi/4*[0:95]')+sin(2*pi/3*[0:95]');
H = signalblks.FIRDecimator;
y = step(H,x);
% View group delay of default FIR filter
fvtool(fir1(35,0.4),1,'analysis','grpdelay');
% Group delay of the default linear-phase FIR filter is 17.5 samples
% Downsampling by a factor of two expect an approx. 8.75
% sample delay in the output y with the initial filter states of zero
subplot(211);
stem(x(1:length(x)/2),'b','markerfacecolor',[0 0 1]);
title('Input Signal');
subplot(212);
stem(y,'b','markerfacecolor',[0 0 1]);
title('Output--Lowpass filtered and downsampled by 2');
```

Note the delay in the decimated output is consistent with the group delay of the filter when the initial filter states are zero.



Reduce the sampling rate of an audio signal by 1/2 and play it:

```
 hmfr = signalblks.MultimediaFileReader('AudioOutputDataType',...
'single');
 hap = signalblks.AudioPlayer(22050/2);
 hfirdec = signalblks.FIRDecimator;
 while ~isDone(hmfr)
   frame = step(hmfr);
   y = step(hfirdec, frame);
   step(hap, y);
 end
 close(hmfr);
 pause(0.5);
 close(hap);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the FIR Decimation block reference page. The object properties correspond to the block parameters, except for:

- **Coefficient source**—The FIR decimator does not support `mfilt` objects .
- **Framing**– The FIR decimator only supports **Maintain input frame rate**
- **Output buffer initial conditions**–The FIR decimator does not support this parameter.

See Also

`signalblks.FIRInterpolator` | `signalblks.FIRRateConverter`

signalblks.FIRDecimator.clone

Purpose Create FIR decimator object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an FIR decimator object, `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

Purpose

Return number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.FIRDecimator.getNumOutputs

Purpose Return number of outputs of step method

Syntax

```
getNumOutputs(H)
```

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax

```
isLocked(H)
```

Description `isLocked(H)` returns the locked state of the FIR decimator.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.FIRDecimator.reset

Purpose Reset filter states of FIR decimator

Syntax `reset(H)`

Description `reset(H)` resets the filter states of the FIR decimator object,`H`, to the initial values of zero. After the `step` method applies the FIR decimator to nonzero input data, the filter states may change. Invoking the `step` method again without first invoking the `reset` method may produce different outputs for an identical input.

Example of resetting filter states:

```
H = signalblks.FIRDecimator;
x = cos(pi/8*[0:1023]')+sin(pi/4*[0:1023]');
y = step(H,x);
% Invoke step method again without reset
y1 = step(H,x);
isequal(y,y1) % Returns 0
% Reset filter states
reset(H);
y2 = step(H,x);
isequal(y,y2) % Returns 1
```

Purpose

Decimate input by integer factor

Syntax

```
Y = step(H,X)
```

Description

`Y = step(H,X)` outputs the filtered and downsampled values, `Y`, of the input signal, `X`. A *Ki*–by–*N* input matrix is treated as *N* independent channels. The length of each column must be a multiple of the `DecimationFactor` property value. The FIR decimator operates on each channel separately and generates a *Ko*–by–*N* output matrix where *Ko*=*Ki*/*M* with *M* the decimation factor. The object supports real and complex floating-point and fixed-point inputs, except for complex unsigned fixed-point inputs.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.FIRInterpolator class

Purpose

Polyphase FIR interpolator

Description

The `FIRInterpolator` object upsamples an input by the integer upsampling factor, *L*, followed by an FIR anti-imaging filter. The filter coefficients are scaled by the interpolation factor. The filter is implemented using a polyphase interpolation structure. The result is a discrete-time signal with a sampling rate *L* times the original sampling rate.

Construction

`H= signalblks.FIRInterpolator` returns an FIR interpolator, `H`, which upsamples an input signal by a factor of 3 and applies an FIR filter to interpolate the output.

`H = signalblks.FIRInterpolator('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns an FIR interpolator, `H`, with each property set to the specified value.

`H = signalblks.FIRInterpolator(INTERP, NUM,'`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns an FIR interpolation object, `H`, with the `InterpolationFactor` property set to `INTERP`, the `Numerator` property set to `NUM`, and other properties set to the specified values.

Properties

`InterpolationFactor`

Interpolation factor

Specify the integer factor, *L*, by which to increase the sampling rate of the input signal. The polyphase implementation uses *L* polyphase subfilters to compute convolutions at the lower sample rate. The FIR interpolator delays and interleaves these lower-rate convolutions to obtain the higher-rate output. The property value defaults to 3.

`Numerator`

FIR filter coefficients

Specify the numerator coefficients of the FIR anti-imaging filter as the coefficients of a polynomial in z^{-1}. Indexing from zero, the filter coefficients are:

$$H(z) = \sum_{n=0}^{N-1} b(n) z^{-n}$$

To act as an effective anti-imaging filter, the coefficients must correspond to a lowpass filter with a normalized cutoff frequency no greater than the reciprocal of the `InterpolationFactor`. The filter coefficients are scaled by the value of the `InterpolationFactor` property before filtering the signal. To form the L polyphase subfilters, `Numerator` is appended with zeros if necessary. The default value of this property is the output of `fir1(15,0.25)`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value for this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` or `Saturate`. The default value for this property is `Wrap`.

`CoefficientsDataType`

Coefficient word and fraction lengths

Specify the coefficients fixed-point data type as one of `Same word length as input` or `Custom`. The default value for this property is `Same word length as input`.

CustomCoefficientsDataType

Coefficient word and fraction lengths

Specify the coefficients fixed-point type as a numerictype object with a Signedness of Auto. This property applies only when the CoefficientsDataType property is Custom. The default value of this property is numerictype([],16,15).

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as one of Internal rule, Same as input, or Custom. The default value for this property is Internal rule.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only when the ProductDataType property is Custom. The default value of this property is numerictype([],32,30).

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of Internal rule, Same as product, Same as input, or Custom. The default value for this property is Internal rule.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled numerictypeobject with a Signedness of Auto. This property applies only when the AccumulatorDataType property is Custom. The default value of this property is numerictype([],32,30).

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as one of `Same as accumulator`, `Same as product`, `Same as input`, or `Custom`. The default value for this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `OutputDataType` property is `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create FIR interpolator object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs of step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset FIR interpolator filter states
step	Upsample and interpolate input

Examples

Double the sampling rate of an audio signal from 22.05 kHz to 44.1 kHz and play the audio:

```
hmfr = signalblks.MultimediaFileReader('AudioOutputDataType',...
'single');
hap = signalblks.AudioPlayer(44100);
hfirint = signalblks.FIRInterpolator(2, ...
```

```
firpm(30, [0 0.45 0.55 1], [1 1 0 0]));
while ~isDone(hmfr)
 frame = step(hmfr);
 y = step(hfirint, frame);
 step(hap, y);
end
pause(1);
close(hmfr);
close(hap);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the FIR Interpolation block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.FIRDecimator` | `signalblks.FIRRateConverter`

Purpose Create FIR interpolator object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an FIR interpolator, `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

signalblks.FIRInterpolator.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs of step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of output from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.FIRInterpolator.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the FIR interpolator.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset FIR interpolator filter states

Syntax `reset(H)`

Description `reset(H)` resets the filter states of the FIR filter in the interpolator object, `H`, to their initial values of 0. The initial filter state values correspond to the initial conditions for the constant coefficient linear difference equation defining the FIR filter. After the `step` method applies the interpolator to nonzero input data, the states may be nonzero. Invoking the `step` method again without first invoking the `reset` method may produce different outputs for an identical input.

Reset filter states to 0 to produce consistent output:

```
H = signalblks.FIRInterpolator(2);
x =[1 -1]'; x = repmat(x,8,1);
y = step(H,x); % Filter states are nonzero
% Use reset method to set states to zero
reset(H);
% Apply FIR interpolator to input x
y1 = step(H,x);
isequal(y,y1)
% Returns a 1
```

signalblks.FIRInterpolator.step

Purpose

Upsample and interpolate input

Syntax

`Y = step(H,X)`

Description

`Y = step(H,X)` outputs the upsampled and interpolated values, `Y` , of the input signal `X` . A K-by-N input matrix is treated as N independent channels, and the FIR interpolator object interpolates each channel over the first dimension and generates a M-by-N output matrix, where M is the product of K and the upsampling factor, L.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Sample rate converter

Description

The `FIRRateConverter` performs sampling rate conversion by a rational factor on a vector or matrix input. The FIR rate convertor cascades an interpolator with a decimator. The interpolator upsamples the input by the upsampling factor, L, followed by a lowpass FIR filter. The FIR filter acts both as an anti-imaging filter and an anti-aliasing filter prior to decimation. The decimator downsamples the output of upsampling and FIR filtering by the downsampling factor M. The upsampling and downsampling factors must be relatively prime, or coprime. The result is a discrete-time signal with a sampling rate L/M times the original sampling rate.

Construction

`H = signalblks.FIRRateConverter` returns a FIR sample rate converter, `H`, that resamples an input signal at a rate 3/2 times the original sampling rate.

`H = signalblks.FIRRateConverter('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns an FIR sample rate converter, `H`, with each property set to the specified value.

`H = signalblks.FIRRateConverter(L,M,NUM,'`*`PropertyName`*`',` *`PropertyValue`*`, ...)` returns an FIR sample rate converter, `H`, with the `InterpolationFactor` property set to `L`, the `DecimationFactor` property set to `M`, the `Numerator` property set to `NUM`, and other specified properties set to the specified values.

Properties

`InterpolationFactor`

Interpolation factor

Specify the integer upsampling factor. The property defaults to 3.

`DecimationFactor`

Decimation factor

Specify the integer downsampling factor. The property defaults to 2.

`Numerator`

FIR filter coefficients

Specify the FIR filter coefficients in powers of z^{-1}. The length of filter coefficients must exceed the interpolation factor. The filter should be lowpass with normalized cutoff frequency no greater than `min(1/InterpolationFactor, 1/DecimationFactor)`. All initial filter states are zero. The default value of this property is `firpm(70,[0 0.28 0.32 1],[1 1 0 0])`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value for this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` or `Saturate`. The default value for this property is `Wrap`.

`CoefficientsDataType`

word and fraction lengths of the filter coefficients

Specify the filter coefficient fixed-point data type as one of `Same word length as input` or `Custom`. The default value for this property is `Same word length as input`.

`CustomCoefficientsDataType`

Word and fraction lengths of the filter coefficients

Specify the filter coefficient fixed-point type as a `numerictype` object with a `Signedness` of `Auto`. This property applies only when the CoefficientsDataType property is Custom. The default value of this property is numerictype([],16,15).

ProductDataType

Product word and fraction lengths

Specify the product fixed-point data type as one of Internal rule, Same as input, or Custom. The default value for this property is Internal rule.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only when the ProductDataType property is Custom. The default value of this property is numerictype([],32,30).

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of Internal rule, Same as product, Same as input, or Custom. The default value for this property is Internal rule.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only when the AccumulatorDataType property is Custom. The default value of this property is numerictype([],32,30).

OutputDataType

Output word and fraction lengths

Specify the output fixed-point data type as one of Same as accumulator, Same as product, Same as input, or Custom. The default value for this property is Same as accumulator.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `OutputDataType` property is `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create FIR sample rate convertor object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset states of FIR sample rate converter
step	Resample input with FIR rate converter

Examples

Resample an audio signal from 48 kHz to 32 kHz and play it

```
hmfr = signalblks.MultimediaFileReader('audio48kHz.wav', ...
'AudioOutputDataType', 'single', ...
'SamplesPerAudioFrame', 300);
hap = signalblks.AudioPlayer(32000);
% Create an FIRRateConverter System object with interpolation
% factor = 2, decimation factor = 3. The default FIR filter
% coefficients correspond to a lowpass filter with normalized
% cutoff frequency of 1/3 which is appropriate for this application.
hfirrc = signalblks.FIRRateConverter(2,3);
```

```
while ~isDone(hmfr)
audio1 = step(hmfr);
audio2 = step(hfirrc, audio1);
step(hap, audio2);
end
close(hmfr);
close(hap);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the FIR Rate Conversion block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.FIRInterpolator` | `signalblks.FIRDecimator`

signalblks.FIRRateConverter.clone

Purpose Create FIR sample rate convertor object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an FIR sample rate converter object, `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

Purpose

Return number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.FIRRateConverter.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the FIR sample rate converter.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.FIRRateConverter.reset

Purpose Reset states of FIR sample rate converter

Syntax `reset(H)`

Description `reset(H)` resets the filter states of the FIR filter in the sample rate converter, `H`, to their initial values of 0. The initial filter state values correspond to the initial conditions for the constant coefficient linear difference equation defining the FIR filter. After the `step` method applies the FIR rate converter to nonzero input data, the states may be nonzero. Invoking the `step` method again without first invoking the `reset` method may produce different outputs for an identical input.

Reset filter states to 0 to produce consistent output:

```
H = signalblks.FIRRateConverter(2,4);
x =[1 -1]'; x = repmat(x,8,1);
y = step(H,x); % Filter states are nonzero
% Use reset method to set states to zero
reset(H);
% Apply sampling rate converter to input x
y1 = step(H,x);
isequal(y,y1)
% Returns a 1
```

Purpose Resample input with FIR rate converter

Syntax

```
Y = step(H,X)
```

Description `Y = step(H,X)` resamples the input `X` and returns the resampled signal `Y`. An M-by-N matrix input is treated as `N` independent channels.

dsp_links

Purpose

Identify whether blocks in model are current, deprecated, or obsolete

Syntax

```
dsp_links
dsp_links('modelname')
```

Description

`dsp_links` returns a structure with three elements that identify whether the Signal Processing Blockset blocks in the current model are current, deprecated, or obsolete. Each element represents one of the three block categories and contains a cell array of strings. Each string is the name of a library block in the current model.

`dsp_links('modelname')` returns the three-element structure for the specified model.

Definitions

Obsolete Blocks

Obsolete blocks are blocks that the blockset no longer supports. In some cases, these blocks no longer function properly.

Deprecated Blocks

Deprecated blocks are blocks that the blockset still supports but are likely to become obsolete in a future release. Refer to the block reference page for suggested replacements.

Current Blocks

Current blocks are blocks that the blockset supports and that represent the latest block functionality.

Examples

Display block support information for the specified model, and then find the name of the first current block:

```
sys = 'dspcochlear';
load_system(sys)     % Load the dspcochlear model
links = dsp_links(sys) % Run dsp_links on the model
links.current{1}     % Find the name of the first current block
```

See Also

`liblinks`

Purpose

Open top-level Signal Processing Blockset library

Syntax

`dsplib`

Description

`dsplib` opens the top-level Signal Processing Blockset block library model.

Examples

View and gain access to the Signal Processing Blockset blocks:

```
dsplib
```

Alternatives

To view and gain access to the Signal Processing Blockset blocks using the Simulink library browser:

- Type `simulink` at the MATLAB command line, and then expand the Signal Processing Blockset node in the library browser.
- Click the Simulink icon  from the MATLAB desktop or from a model.

dspstartup

Purpose

Configure Simulink environment for signal processing systems

Syntax

```
dspstartup
```

Description

`dspstartup` configures Simulink environment parameters with settings appropriate for a typical signal processing project. You can use the `dspstartup` function in the following ways:

- At the MATLAB command line. Doing so configures the Simulink environment in your current session for signal processing projects.
- By adding a call to the `dspstartup` function from your `startup.m` file. When you do so, MATLAB configures your Simulink environment for typical signal processing projects each time you launch MATLAB.

When the function successfully configures your Simulink environment, MATLAB displays the following message in the command window.

```
Changed default Simulink settings for signal processing
systems (dspstartup.m).
```

The `dspstartup.m` file executes the following commands. See "Model and Block Parameters" in the Simulink documentation.

```
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');
```

Examples

Add a call to the `dspstartup` function from your `startup.m` file:

1. To find out if there is a `startup.m` file on your MATLAB path, run the following code at the MATLAB command line:

   ```
   which startup.m
   ```

2. If MATLAB returns `'startup.m' not found.`, see "Startup Options" in the MATLAB documentation to learn more about the `startup.m` file.

   If MATLAB returns a path to your `startup.m` file, open that file for editing.

   ```
   edit startup.m
   ```

3. Add a call to the `dspstartup` function. Your `startup.m` file now resembles the following code sample:

   ```
   %STARTUP   Startup file
   %   This file is executed when MATLAB starts up, if it exists
   %   anywhere on the path.  In this case, the startup.m file
   %   runs the dspstartup.m file to configure the Simulink
   %   environment with settings appropriate for typical
   %   signal processing projects.

   dspstartup;
   ```

See Also

`startup`

How To

- "Configuring the Simulink Environment for Signal Processing Models"

signalblks.Histogram class

Purpose

Generate histogram of input or sequence of inputs

Description

The `histogram` object generates a histogram for an input or a sequence of inputs.

Construction

`H = signalblks.Histogram` returns a histogram object, `H`, that computes the frequency distribution of the elements in each input matrix.

`H = signalblks.Histogram('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns a histogram object, `H`, with each specified property set to the specified value.

`H = signalblks.Histogram(`*`MIN`*`,`*`MAX`*`,`*`NUMBINS`*`,'`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a histogram object, `H`, with the `LowerLimit` property set to *`MIN`*, `UpperLimit` property set to *`MAX`*, `NumBins` property set to *`NUMBINS`* and other specified properties set to the specified values.

Properties

`LowerLimit`

Lower boundary.

Specify the lower boundary of the lowest-valued bin as a real-valued scalar value. `NaN` and `Inf` are not valid values for this property. The default value of this property is `0`. This property is tunable.

`UpperLimit`

Upper boundary

Specify the upper boundary of the highest-valued bin as a real-valued scalar value. `NaN` and `Inf` are not valid values for this property. The default value of this property is `10`. This property is tunable.

`NumBins`

Number of bins in the histogram

Specify the number of bins in the histogram. The default value of this property is `11`.

`Normalize`

Enable output vector normalization

Specify whether the histogram object normalizes the output vector, v, so that $sum(v) = 1$. The object ignores this property for fixed-point signals. The default value of this property is `false`.

`RunningHistogram`

Enable calculation over time

Set this property to `true` to enable running histogram calculations for the input elements over time. Set this property to `false` to compute a histogram for the current input. The default value of this property is `false`.

`ResetInputPort`

Enable resetting in running histogram mode

Set this property to `true` to enable resetting for the running histogram. When you set the property to `true`, specify a reset input for the `step` method that resets the running histogram. This property applies when you set the `RunningHistogram` property to `true`. When this property is `false`, the histogram object never resets. The default value of this property is `false`.

`ResetCondition`

Reset condition for running histogram mode

Specify the event that resets the running histogram: `Rising edge`, `Falling edge`, `Either edge`, `Non-zero`. This property applies when you set the `ResetInputPort` property to `true`. The default value of this property is `Non-zero`

`FrameBasedProcessing`

Enable frame-based processing

Setting this property to `true` enables frame-based processing. Setting this property to `false` enables sample-based processing. The default value of this property is `true`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as: `Ceiling`, `Convergent`, `Floor`, `Nearest`, Round, `Simplest` or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`ProductDataType`

Product word and fraction lengths

Specify the product fixed-point data type as `Same as input` or `Custom`. The default value of this property is `Same as input`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as: `Same as product`, `Same as input` or `Custom`. The default value of this property is `Same as input`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

Methods

clone	Create histogram object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Return number of outputs from method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset histogram bin values to zero
step	Return histogram for input data

Examples

Compute a Histogram with four bins, for possible input values 1 through 4.

```
hhist = signalblks.Histogram(1,4,4);
y = step(hhist, [1 2 2 3 3 3 4 4 4 4]');
% y is equal to [1; 2; 3; 4] - one ones, two twos, etc.
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Histogram block reference page. The object properties correspond to the block parameters, except for:

- **Reset port** block parameter corresponds to both the `ResetCondition` and the `ResetInputPort` object properties.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See "What Are Sample- and Frame-Based Processing?" for more information.

See Also

`signalblks.Maximum` | `signalblks.Minimum` | `signalblks.Mean`

Purpose Create histogram object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates a `Histogram` System object C, with the same property values as H. The clone method creates a new unlocked object with uninitialized states.

signalblks.Histogram.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Return number of outputs from `step` method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.Histogram.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the Histogram System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset histogram bin values to zero

Syntax `reset(H)`

Description `reset(H)` sets the `Histogram` object bin values to zero when you set the `RunningHistogram` property to `true`.

signalblks.Histogram.step

Purpose

Return histogram for input data

Syntax

```
Y = step(H,X)
Y = step(H,X,R)
```

Description

`Y = step(H,X)` returns a histogram `Y` for the input data `X` . When the `RunningHistogram` property is `true`, `Y` corresponds to the histogram of the input elements over time.

`Y = step(H,X,R)` computes the histogram of the input elements over time, and optionally resets the histogram's state based on the value of `R` and the object's `ResetCondition` property. This is possible when the `RunningHistogram` and the `ResetInputPort` properties are `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Inverse discrete cosine transform (IDCT)

Description

The `IDCT` object computes the inverse discrete cosine transform (IDCT) of an input.

Construction

`H = signalblks.IDCT` returns a inverse discrete cosine transform (IDCT) object, `H`, used to compute the IDCT of a real or complex input signal using the `Table lookup` method.

`H = signalblks.IDCT('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an inverse discrete cosine transform (IDCT) object, `H`, with each property set to the specified value.

Properties

`SineComputation`

Method to compute sines and cosines

Specify how the IDCT object computes the trigonometric function values as `Trigonometric function` or `Table lookup`. This property must be set to `Table lookup` for fixed-point inputs. The default value for this property is `Table lookup`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling`, `Convergent`, `Floor` , `Nearest`, `Round`, `Simplest`, or `Zero`. This property applies when you set the `SineComputation` property to `Table lookup`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` or `Saturate`. This property applies when you set the `SineComputation` property to `Table lookup`.

`SineTableDataType`

Sine table word-length designation

Specify the sine table fixed-point data type as one of `Same word length as input` or `Custom`. This property applies when you set the `SineComputation` property to `Table lookup`.

`CustomSineTableDataType`

Sine table word length

Specify the sine table fixed-point type as a signed, unscaled `numerictype` object. This property applies when you set the `SineComputation` property to `Table lookup` and the `SineTableDataType` property to `Custom`. The default value of this property is `numerictype(true,16)`.

`ProductDataType`

Product word and fraction lengths

Specify the product fixed-point data type as one of `Internal rule` , `Same as input`, `Custom`. This property applies when you set the `SineComputation` property to `Table lookup`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a signed, scaled `numerictype` object. This property applies when you set the `SineComputation` property to `Table lookup` and the `ProductDataType` property to `Custom`. The default value of this property is `numerictype(true,32,30)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Internal rule`, `Same as input`, `Same as product`, or `Custom`. This property applies when you set the `SineComputation` property to `Table lookup`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a signed, scaled `numerictype` object. This property applies when you set the `SineComputation` property to `Table lookup` and the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype(true,32,30)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as one of `Internal rule` , `Same as input`, `Custom`. This property applies when you set the `SineComputation` property to `Table lookup`. The default value for this property is `Internal rule`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a signed, scaled `numerictype` object. This property applies when you set the `SineComputation` property to `Table lookup` and the `OutputDataType` property to `Custom`. The default value of this property is`numerictype(true,16,15)`.

Methods

clone	Create inverse discrete cosine transform object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs of step method

isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Compute inverse discrete cosine transform (IDCT) of input

Examples

Use DCT to analyze the energy content in a sequence:

```
x = (1:128).' + 50*cos((1:128).'*2*pi/40);
hdct = signalblks.DCT;
X = step(hdct, x);
% Set the DCT coefficients which represent less
% than 0.1% of the total energy to 0 and
% reconstruct the sequence using IDCT.
[XX, ind] = sort(abs(X),1,'descend');
ii = 1;
while (norm([XX(1:ii);zeros(128-ii,1)]) <= 0.999*norm(XX))
ii = ii+1;
end
disp(['Number of DCT coefficients that represent 99.9%',...
'of the total energy in the sequence: ',num2str(ii)]);
XXt = zeros(128,1);
XXt(ind(1:ii)) = X(ind(1:ii));
hidct = signalblks.IDCT;
xt = step(hidct, XXt);
plot(1:128,[x xt]);
legend('Original signal','Reconstructed signal',...
'location','best');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the IDCT block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.DCT` | `signalblks.FFT` | `signalblks.IFFT`.

Purpose

Create inverse discrete cosine transform object with same property values

Syntax

`C = clone(H)`

Description

`C = clone(H)` creates a copy of the current IDCT object, `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.IDCT.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs of step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.IDCT.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the IDCT object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Compute inverse discrete cosine transform (IDCT) of input

Syntax

`Y = step(H,X)`

Description

`Y = step(H,X)` computes the inverse discrete cosine transform, `Y` , of input `X`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.IFFT class

Purpose

Inverse discrete Fourier transform (IDFFT)

Description

The `IFFT` object computes the inverse discrete Fourier transform (IDFFT) of input.

Construction

`H = signalblks.IFFT` returns an `IFFT` object, `H`, that computes the IDFT of a column vector or N-D array. For column vectors or N-D arrays, the `IFFT` object computes the IDFT along the first dimension of the array. If the input is a row vector, the `IFFT` object computes a row of single-sample IDFTs and issues a warning.

`H = signalblks.IFFT('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns an `IFFT` object, `H`, with each property set to the specified value.

Properties

`TwiddleFactorComputation`

Computation method of twiddle factors

Specify the computation method of the twiddle factors as `Trigonometric function` or `Table lookup`. The property must be set to `Table lookup` for fixed-point inputs. The default value of this property is `Table lookup`.

`TableOptimization`

Optimization of trigonometric values table

Select the optimization of the trigonometric values table as `Speed` or `Memory`. This property applies only when the `TwiddleFactorComputation` property is `Table lookup`. The property must be set to `Speed` for fixed-point inputs. The default value of this property is `Speed`.

`BitReversedInput`

Enable bit-reversed order interpretation of input elements

Set this property to true if the order of Fourier transformed input elements to the `IFFT` object are in bit-reversed order. This property applies only when the `FFTLengthSource` property is

Auto. The default value of this property is false, which denotes linear ordering.

ConjugateSymmetricInput

Enable conjugate symmetric interpretation of input

Set this property to true if the input is conjugate symmetric to yield real-valued outputs. The discrete Fourier transform of a real valued sequence is conjugate symmetric, and setting this property to true optimizes the IDFT computation method. Setting this property to false for conjugate symmetric inputs may result in complex output values with nonzero imaginary parts due to rounding errors. Setting this property to true for non-conjugate symmetric inputs results in invalid outputs. This property applies only when theFFTLengthSource property is Auto. The default value of this property is false.

Normalize

Enable dividing output by FFT length

Specify if the IFFT output should be divided by the FFT length. The default value of this property is true and each element of the output is divided by the FFT length.

FFTLengthSource

Source of FFT length

Specify how to determine the FFT length as Auto or Property. When this property is set to Auto, the FFT length is equal to the number of rows of the input signal and must be a power of two. This property applies only when both the BitReversedInput and ConjugateSymmetricInput properties are false. The default value of this property is Auto.

FFTLength

FFT length as power of two

Specify the FFT length as a power of two. This property applies only when the BitReversedInput and ConjugateSymmetricInput properties are false, and the FFTLengthSource property is Property. The default value of this property is 64.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as Ceiling, Convergent, Floor, Nearest, Round, Simplest, or Zero. This property applies only when the TwiddleFactorComputation property is Table lookup and the TableOptimization property is Speed. The default value of this property is Floor.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as Wrap or Saturate. This property applies only when the TwiddleFactorComputation property is Table lookup and the TableOptimization property is Speed. The default value of this property is Wrap.

SineTableDataType

Sine table word and fraction lengths

Specify the sine table data type as Same word length as input or Custom. This property applies only when the TwiddleFactorComputation property is Table lookup and the TableOptimization property is Speed. The default value of this property is Same word length as input.

CustomSineTableDataType

Sine table word and fraction lengths

Specify the sine table fixed-point type as an unscaled numerictype object with a Signedness of Auto. This property applies only when the TwiddleFactorComputation property is Table lookup, the TableOptimization property is Speed, and the

SineTableDataType property is Custom. The default value of this property is numerictype([],16).

ProductDataType

Product word and fraction lengths

Specify the product data type as Internal rule, Same as input, or Custom. This property applies only when the TwiddleFactorComputation property is Table lookup and the TableOptimization property is Speed. The default value of this property is Internal rule.

CustomProductDataType

Product word and fraction lengths

Specify the product fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only when the TwiddleFactorComputation property is Table lookup, the TableOptimization property is Speed, and the ProductDataType property is Custom. The default value of this property is numerictype([],32,30).

AccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator data type as Internal rule, Same as input, Same as product, or Custom. This property applies only when the TwiddleFactorComputation property is Table lookup and the TableOptimization property is Speed. The default value of this property is Internal rule.

CustomAccumulatorDataType

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only when the TwiddleFactorComputation property is Table lookup, the TableOptimization property is Speed, and the

AccumulatorDataType property is Custom. The default value of this property is numerictype([],32,30).

OutputDataType

Output word and fraction lengths

Specify the output data type as Internal rule, Same as input, or Custom. This property applies only when the TwiddleFactorComputation property is Table lookup and the TableOptimization property is Speed. The default value of this property is Internal rule.

CustomOutputDataType

Output word and fraction lengths

Specify the output fixed-point type as a scaled numerictype object with a Signedness of Auto. This property applies only when the TwiddleFactorComputation property is Table lookup, the TableOptimization property is Speed, and the OutputDataType property is Custom. The default value of this property is numerictype([],16,15).

Methods

clone	Create IFFT object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Compute inverse discrete Fourier transform of input

Examples

Use DFT and IDFT to analyze the energy content in a sequence:

```
Fs = 40; L = 128;
t = (0:L-1)'/Fs;
x = sin(2*pi*10*t) + 0.75*cos(2*pi*15*t);
y = x + .5*randn(size(x)); % noisy signal
hfft = signalblks.FFT;
hifft = signalblks.IFFT('ConjugateSymmetricInput', true);
X = step(hfft, x);
[XX, ind] = sort(abs(X),1,'descend');
XXn = sqrt(cumsum(XX.^2))/norm(XX);
ii = find(XXn > 0.999, 1);
disp(['Number of FFT coefficients that represent 99.9% of ...
the', 'total energy in the sequence: ', num2str(ii)]);
XXt = zeros(128,1);
XXt(ind(1:ii)) = X(ind(1:ii));
xt = step(hifft, XXt);
% Verify the reconstructed signal matches the original
norm(x-xt)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the IFFT block reference page. The object properties correspond to the block parameters, except for:

- **Output sampling mode** parameter is not supported by `signalblks.IFFT`.

See Also

`signalblks.FFT` | `signalblks.DCT` | `signalblks.IDCT`

signalblks.IFFT.clone

Purpose Create IFFT object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an IFFT object, `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.IFFT.getNumInputs

Purpose

Number of expected inputs to step method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.IFFT.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.IFFT.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the IFFT object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.IFFT.step

Purpose

Compute inverse discrete Fourier transform of input

Syntax

```
Y = step(H,X)
```

Description

`Y = step(H,X)` computes the inverse discrete Fourier transform (IDFT) , `Y` , of the input `X` along the first dimension of `X`. When the `FFTLengthSource` property is `Auto`, the length of `X` along the first dimension must be a positive integer power of two. When the `FFTLengthSource` property is `'Property'`, the length of `X` along the first dimension can be any positive integer and the `FFTLength` property must be a positive integer power of two.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Linear or FIR interpolation

Description

The Interpolator object interpolates real-valued inputs using linear or polyphase FIR interpolation. Use the Interpolator object to upsample a frame-based or sample-based input.

Construction

H = signalblks.Interpolator returns a handle to an interpolation object, H. The interpolation method defaults to Linear with frame-based input processing and a 5x1 vector of interpolation points in H.InterpolationPoints.

H = signalblks.Interpolator('*PropertyName*',*PropertyValue*,...) returns an interpolation object, H, with the specified property name and value pairs.

Properties

InterpolationPointsSource

Source of interpolation points

This property can be set to Property or Input port. If you set this property to Input port, you must provide the interpolation points as an input to the step method when you interpolate the input. The default value for this property is Property.

InterpolationPoints

Interpolation points

This property only applies when you set the InterpolationPointsSource property to Property. Specify the interpolation points as a row or column vector. The valid range of the values in the interpolation vector is from 1 to the number of samples in each channel of the input. If you specify interpolation points outside of the valid range, the interpolator object *clips* the point to the nearest point in the valid range. For example, if the input is [2 3.1 -2.1], the valid range of interpolation points is from 1 to 3. If you specify the following vector of interpolation points [-1 1.5 2 2.5 3 3.5], the interpolator object clips -1 to 1 and 3.5 to 3, resulting in the interpolation points [1 1.5 2 2.5 3 3]. The default value of this property is [1.1;4.8;2.67;1.6;3.2]. This property is tunable.

Method

Interpolation method

Specify the interpolation method as Linear or FIR. When this property is Linear, the interpolator object interpolates data values by assuming that the data varies linearly between adjacent samples. When this property is FIR, the interpolator object uses polyphase interpolation to replace filtering (convolution) at the upsampled rate with a series of convolutions at the lower rate. The interpolator object always performs linear interpolation if there are insufficient low-rate samples to perform FIR interpolation as explained in the description of the FilterHalfLength property. The default value for this property is Linear.

FilterHalfLength

Interpolation filter half length

This property applies only when you set the Method property to FIR. For a filter half length of *P*, the polyphase FIR subfilters have length *2P*. FIR interpolation always requires *2P* low-rate samples for every interpolation point. If the interpolation point does not correspond to a low-rate sample, FIR interpolation requires *P* low-rate samples below and *P* low-rate samples above the interpolation point. If the interpolation point corresponds to a low-rate sample, the *2P*-sample requirement includes the low-rate sample. If there are less than *2P* neighboring low-rate samples, the interpolator object uses linear interpolation. For example, for an input [1 4 1 4 1 4 1 4], upsampling by a factor of four results in the equally spaced interpolation points InterP = 1:0.25:8;. The points InterP(9:12) are [3.0 3.25 3.5 3.75]. Assuming a FilterHalfLength property equal to 2, interpolating at these points uses the four low-rate samples from the input with indices (2,3,4,5). If you set the FilterHalfLength property to 4 in this case, the interpolator object uses linear interpolation because there are not enough low-rate samples to perform FIR interpolation.

The longer the `FilterHalfLength` property, the better the quality of the interpolation, but this improvement comes at the cost of increased computation and requires more low-rate samples below and above the interpolation point. In general, setting the `FilterHalfLength` property between 4 and 6 provides reasonably accurate interpolation.

`InterpolationPointsPerSample`

Interpolation points per input sample

This property only applies when you set the `Method` property to `FIR`. This property is the upsampling factor, *L*. An upsampling factor of *L* inserts *L-1* zeros between low-rate samples. Interpolation results from filtering the upsampled sequence with a lowpass anti-imaging filter. The interpolator object uses a polyphase FIR implementation with `InterpolationPointsPerSample` subfilters of length *2P*, where *P* is the `FilterHalfLength` property. Denoting the low-rate samples in the upsampled input by *nL*, *n=1,2,...*, the interpolator object uses exactly one of the `InterpolationPointsPerSample` subfilters, or filter arms, to interpolate at the points *nL+i/L*, where *i=0,1,2,...,L-1*. If you specify interpolation points which do not correspond to a polyphase subfilter, the interpolator object rounds the point down to the nearest interpolation point associated with a polyphase subfilter. For example, if you set the `InterpolationPointsPerSample` property to `4` and interpolate at the points `[3 3.2 3.4 3.6 3.8]`, the interpolator object uses the first polyphase subfilter for the points `[3.0 3.2]`, the second subfilter for the point `3.4`, the third subfilter for the point `3.6`, and the fourth subfilter for the point `3.8`.

`Bandwidth`

Normalized input bandwidth

This property applies only when you set the `Method` property to `FIR`. The normalized bandwidth is a real–valued scalar between 0 and 1, where 1 is equal to the Nyquist frequency, or 1/2 the sampling frequency (*Fs*). If you know that your input does not have frequency content above some cutoff frequency less than the Nyquist frequency, you can use that information to improve the FIR interpolation filters by relaxing the stopband requirements in frequency regions where the signal has no energy. For example, if the input signal does not have frequency content above *Fs/8*, you can specify a value of `0.25` for the `Bandwidth` property. The default value is `0.5`.

`FrameBasedProcessing`

Enable frame-based processing

Set this property to `true` to use frame-based processing. Set this property to `false` to use sample-based processing. The default value is `true`. The dimension of the input and the value of the `FrameBasedProcessing` property determine how the interpolator object operates on inputs to produce the interpolated output. See "How the Block Applies Interpolation Arrays to Inputs" on page 2-703 in the Signal Processing Blockset documentation for details.

Methods

clone	Create interpolator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Compute linear or FIR interpolation of input

Definitions

A polyphase implementation of an FIR interpolation filter *splits* the lowpass FIR filter impulse response into a number of different subfilters. Let L be the number of interpolation points per sample, or the upsampling factor, and P be the half length of the polyphase subfilters. Indexing from zero, if $h(n)$ is the impulse response of the FIR filter, the k-th subfilter is:

$$h_k(n) = h(k + nL) \quad k = 0,1,\ldots,L-1 \quad n = 0,1,\ldots,2P-1$$

The following table describes the decomposition of an 18–coefficient FIR filter into 3 polyphase subfilters of length 6, the defaults for the FIR interpolator object:

Coefficients	Polyphase Subfilter
$h(0),h(3),h(6), \ldots, h(15)$	$h_0(n)$
$h(1),h(4),h(7), \ldots, h(16)$	$h_1(n)$
$h(2),h(5),h(8), \ldots, h(17)$	$h_2(n)$

The following code shows how to find the polyphase subfilters for the default FIR interpolator object:

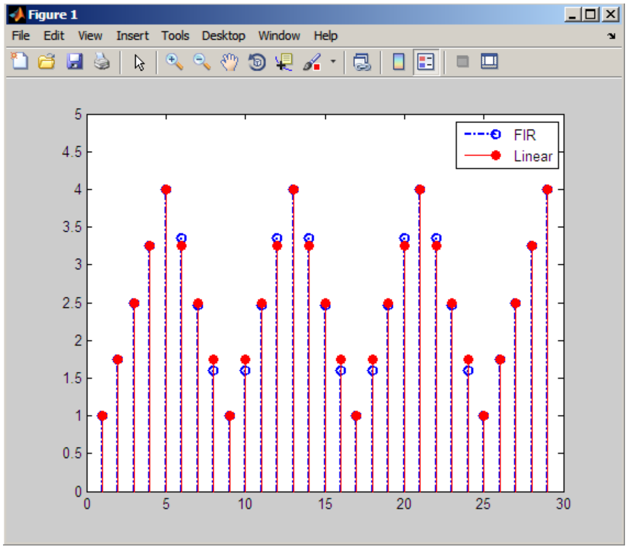
```
H = signalblks.Interpolator('Method','FIR');
L = H.InterpolationPointsPerSample;
P = H.FilterHalfLength;
FiltCoeffs = intfilt(L,P,H.Bandwidth);
% Returns filter of length 2*P*L-1
FiltLen=length(FiltCoeffs);
FiltCols = ceil(FiltLen/2/L);
% We need 2*P*L coefficients
% Prepending a zero does not affect the filter magnitude
FiltCoeffs = [zeros(FiltCols*2*L-FiltLen,1); FiltCoeffs(:)];
% Each column of PolyPhaseCoeffs is a polyphase subfilter
PolyPhaseCoeffs  = reshape(FiltCoeffs,FiltCols,2*L)';
```

signalblks.Interpolator class

Examples

Compare linear interpolation with FIR interpolation:

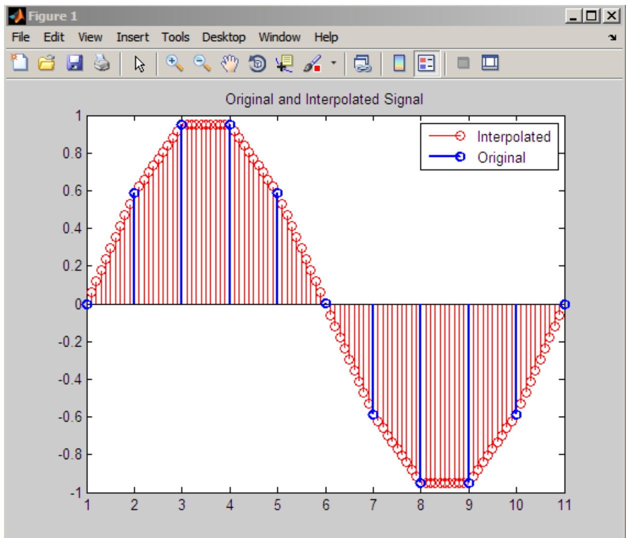
```
x =[1 4];
x = repmat(x,1,4);
x1 = 1:0.25:8;
hFIR =signalblks.Interpolator('Method','FIR','FilterHalfLength',2,...
'InterpolationPoints',x1,'InterpolationPointsPerSample',4);
hLin =signalblks.Interpolator('InterpolationPoints',x1);
OutFIR = step(hFIR,x');
OutLin = step(hLin,x');
stem(OutFIR,'b-.','linewidth',2); hold on;
stem(OutLin,'r','markerfacecolor',[1 0 0]);
axis([0 30 0 5]); legend('FIR','Linear','Location','Northeast');
```



Note that for the indices 1 to 5 and 25 to 29, the interpolator object uses linear interpolation in both cases. This is because there are not enough low-rate samples surrounding the interpolation points at those indices to use FIR interpolation with the specified filter length.

Interpolate a sinusoid with linear interpolation:

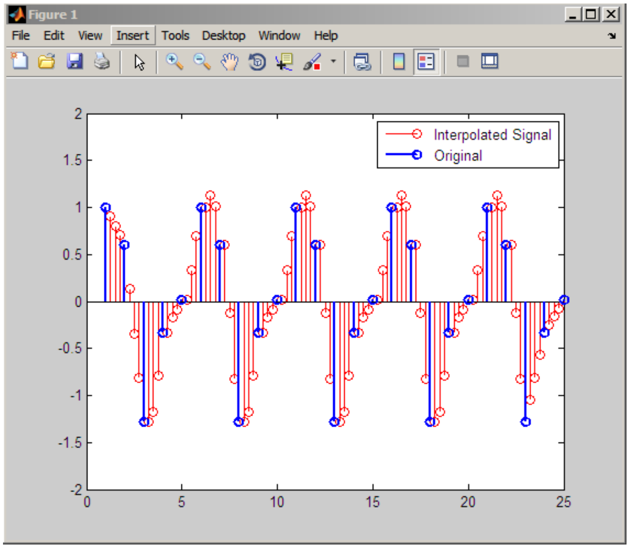
```
t = 0:.0001:.0511;
x = sin(2*pi*20*t);
x1 = x(1:50:end);
I = 1:0.1:length(x1);
H = signalblks.Interpolator('InterpolationPointsSource',...
'Input port');
y = H.step(x1',I');
stem(I',y, 'r');
title('Original and Interpolated Signal');
hold on; stem(x1, 'Linewidth', 2);
legend('Interpolated','Original');
```



Interpolate a sum of sinusoids with FIR interpolation and `Input Port` as the source of interpolation points:

```
Fs = 1000;
t = 0:(1/Fs):0.1-(1/Fs);
```

```
I = 1:(1/4):length(x1);
x = cos(2*pi*50*t)+0.5*sin(2*pi*100*t);
% Decimate without aliasing
x1 = x(1:4:end);
H = signalblks.Interpolator('Method','FIR',...
'FilterHalfLength',3,'InterpolationPointsSource','Input Port');
y = H.step(x1',I');
stem(I,y,'r'); hold on;
axis([0 25 -2 2]);
stem(x1,'b','linewidth',2);
legend('Interpolated Signal','Original',...
'Location','Northeast');
```



Algorithm

This object implements the algorithm, inputs, and outputs described on the Interpolation block reference page. The object properties correspond to the Simulink block parameters, except for:

- **Out of range interpolation points** — The interpolator object only has the `Clip` option. The Simulink block has the additional `Clip and warn` and `Error` options.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See "What Are Sample- and Frame-Based Processing?" for more information.

See Also

`intfilt` | `signalblks.FIRInterpolator` | `signalblks.VariableFractionalDelay`

signalblks.Interpolator.clone

Purpose Create interpolator object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an interpolator object, `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

Purpose Number of expected inputs to step method

Syntax `getNumInputs(h)`

Description `getNumInputs(h)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.Interpolator.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the interpolator.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.Interpolator.step

Purpose

Compute linear or FIR interpolation of input

Syntax

```
Y = step(H,X)
Y = step(H,X,IPTS)
```

Description

`Y = step(H,X)` outputs the interpolated sequence, `Y` , of the input vector or matrix `X` as specified in the `InterpolationPoints` property. Depending on the dimension and frame status of the input and the dimension of interpolation points vector, the interpolator usually applies the interpolation points vector either to each channel of a matrix input, resulting in a matrix output or to each input vector (as if the input vector were a single channel), resulting in a vector output with the same orientation as the input (row or column).

`Y = step(H,X,IPTS)` outputs the interpolated sequence as specified by the input argument `IPTS` when the `InterpolationPointsSource` property is set to 'Input port'. `IPTS` is a row or column vector of interpolation points.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Factor square Hermitian positive definite matrices into lower, upper, and diagonal components

Description

The `LDLFactor` object factors square Hermitian positive definite matrices into lower, upper, and diagonal components. The object uses only the lower triangle of S.

Construction

`H = signalblks.LDLFactor` returns an LDL factor System object, `H`, that computes unit lower triangular `L` and diagonal `D` such that `S` = LDL' for square, symmetric/Hermitian, positive definite input matrix `S`.

`H = signalblks.LDLFactor('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an `LDL` factor System object, `H`, with each specified property set to the specified value.

Properties

Fixed-Point Properties

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Internal rule`, `Same as input`, `Same as product` or `Custom`. The default value of this property is `Internal rule`

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`CustomIntermediateProductDataType`

Intermediate product word and fraction lengths

Specify the intermediate product fixed-point type as a signed, scaled `numerictype` object. This property applies when you set the `IntermediateProductDataType` property to `Custom`. The default value of this property is `numerictype(true,16,15)`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`IntermediateProductDataType`

Intermediate product word and fraction lengths

Specify the intermediate product fixed-point data type as `Same as input` or `Custom`. The default value of this property is `Same as input`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as `Same as input` or `Custom`. The default value of this property is `Same as input`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`ProductDataType`

Product word and fraction lengths

Specify the product fixed-point data type as `Internal rule`, `Same as input` or `Custom`. The default value of this property is `Internal rule`.

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as: `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest` or `Zero`. The default value of this property is `Floor`.

Methods

clone	Create an LDL Factor object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs for step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Decomposes matrix into lower, upper, and diagonal components

Examples

Decompose a square Hermitian positive definite matrix using LDL factor.

```
A = gallery('randcorr',5);
hldl = signalblks.LDLFactor;
y = step(hldl, A);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the LDL Factorization block reference page. The object properties correspond to the block parameters, except for:

- There is no object property that corresponds to the **Non-positive definite input** block parameter. The object does not issue any alerts for Non-positive definite inputs. The output is not a valid factorization. A partial factorization is present in the upper left corner of the output.

See Also

`signalblks.LUFactor`

Purpose Create an LDL Factor object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an `LDLFactor` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.LDLFactor.getNumInputs

Purpose

Return number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs for step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.LDLFactor.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `LDLFactor` object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Decomposes matrix into lower, upper, and diagonal components

Syntax

`Y = step(H,S)`

Description

`Y = step(H,S)` decomposes the matrix `S` into lower, upper, and diagonal components. The output `Y` is a composite matrix with the `L` as its lower triangular part and `D` as the diagonal and `L'`as its upper triangular part. If `S` is not positive definite the output `Y` is not a valid factorization.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.LevinsonSolver class

Purpose Solve linear system of equations using Levinson-Durbin recursion

Description The `LevinsonSolver` object solves linear systems of equations using Levinson-Durbin recursion.

Construction `H = signalblks.LevinsonSolver` returns a System object, `H`, that solves a Hermitian Toeplitz system of equations using the Levinson-Durbin recursion.

`H = signalblks.LevinsonSolver('PropertyName',PropertyValue,...)` returns a Levinson-Durbin object, `H`, with each specified property set to the specified value.

Properties

`AOutputPort`

Enable polynomial coefficients output

Set this property to `true` to output the polynomial coefficients A. Both `AOutputPort` and `KOutputPort` properties cannot be `false` at the same time. For scalar inputs, set the `AOutputPort` property to `true`. The default value of this property is `false`.

`KOutputPort`

Enable reflection coefficients output

Set this property to `true` to output the reflection coefficients K. You cannot set both the `AOutputPort` and `KOutputPort` properties to `false` at the same time. For scalar inputs, the `KOutputPort` property must be `false`. The default value of this property is `true`.

`PredictionErrorOutputPort`

Enable prediction error output

Set this property to `true` to output the prediction error. The default value of this property is `false`.

`ZerothLagZeroAction`

Action when value of lag zero is zero

Specify the output for an input with the first coefficient as zero. Select `Ignore` or `Use zeros`. The default value of this property is `Use zeros`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap`, `Saturate`. The default value of this property is `Wrap`.

`ACoefficientDataType`

A coefficient word and fraction lengths

This is a constant property with value `Custom`.

`CustomACoefficientDataType`

A coefficient word and fraction lengths

Specify the *A* coefficient fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. The default value of this property is `numerictype([],16,15)`.

`KCoefficientDataType`

K coefficient word and fraction lengths

This is a constant property with value `Custom`.

`CustomKCoefficientDataType`

K coefficient word and fraction lengths

Specify the K coefficient fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. The default value of this property is `numerictype([],16,15)`.

`PredictionErrorDataType`

Prediction error power word and fraction lengths

Specify the prediction error power fixed-point data type as `Same as input` or `Custom`. The default value of this property is `Same as input`.

`CustomPredictionErrorDataType`

Prediction error power word and fraction lengths

Specify the prediction error power fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `PredictionErrorDataType` property is `Custom`. The default value of this property is `numerictype([],16,15)`.

`ProductDataType`

Product word and fraction lengths

Specify the product fixed-point data type as `Same as input` or `Custom`. The default value of this property is `Custom`

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `ProductDataType` property is `Custom`. The default value of this property is `numerictype([],32,30)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the Accumulator fixed-point data type as `Same as input`, `Same as product`, or `Custom`. The default value of this property is `Custom`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `AccumulatorDataType` property is `Custom`. The default value of this property is `numerictype([],32,30)`.

Methods

clone	Create Levinson solver object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Returns reflection coefficients corresponding to columns of input

Examples

Use the Levinson solver to compute polynomial coefficients from autocorrelation coefficients.

```
hlevinson = signalblks.LevinsonSolver;
hlevinson.AOutputPort = true;
hlevinson.KOutputPort = false;
x = (1:100)';
hac = signalblks.Autocorrelator(...
'MaximumLagSource', 'Property', ...
'MaximumLag', 10);
```

```
a = step(hac, x);
c = step(hlevinson, a); % Compute polynomial coefficients
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Levinson-Durbin block reference page. The object properties correspond to the block parameters, except for:

- **Output(s)** block parameter corresponds to the `AOutputPort` and the `KOutputPort` object properties.

See Also

`signalblks.Autocorrelator`

Purpose Create Levinson solver object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a `LevinsonSolver` object System object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.LevinsonSolver.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.LevinsonSolver.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `LevinsonSolver` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Returns reflection coefficients corresponding to columns of input

Syntax

```
K = step(H,X)
A = step(H,X)
[A, K] = step(H,X)
[..., P] = step(H,X)
```

Description

`K = step(H,X)` returns reflection coefficients `K` corresponding to the columns of input `X`. `X` is typically a column or matrix of autocorrelation coefficients with lag `0` as the first element.

`A = step(H,X)` returns polynomial coefficients `A` when the `AOutputPort` property is `true` and the `KOutputPort` property is `false`.

`[A, K] = step(H,X)` returns polynomial coefficients `A` and reflection coefficients `K` when both the `AOutputPort` and `KOutputPort` properties are `true`.

`[..., P] = step(H,X)` also returns the error power `P` when the `PredictionErrorOutputPort` property is `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

liblinks

Purpose Check model for blocks from specific Signal Processing Blockset libraries

Syntax

```
liblinks(lib)
liblinks(lib,sys)
liblinks(lib,sys,c)
```

Description `liblinks(lib)` returns a cell array of strings that lists the blocks in the current model that are linked to the specified libraries. The input `lib` provides a cell array of strings with the library names. Use the library name visible in the title bar when you open a library model.

`liblinks(lib,sys)` acts on the named model `sys`.

`liblinks(lib,sys,c)` changes the foreground color of the returned blocks to the color `c`. Possible values of `c` are `'blue'`, `'green'`, `'red'`, `'cyan`', `'magenta'`, `'yellow'`, or `'black'`.

Examples Check for blocks from the Signal Processing Sources library in the specified model:

```
rlsdemo
liblinks('dspsrcs4',gcs)
```

See Also `dsp_links`

Purpose

Number of samples of delay introduced by buffering and unbuffering operations

Syntax

```
d = rebuffer_delay(f,n,v)
d = rebuffer_delay(f,n,v,'mode')
```

Description

`d = rebuffer_delay(f,n,v)` returns the delay, in samples, introduced by the Buffer or Unbuffer block in multitasking operations.

`d = rebuffer_delay(f,n,v,'mode')` returns the delay, in samples, introduced by the Buffer or Unbuffer block in the specified tasking mode.

Input Arguments

`f`

Frame size of the input to the Buffer or Unbuffer block.

`n`

Size of the output buffer. Specify one of the following:

- The value of the **Output buffer size** parameter, if you are computing the delay introduced by a Buffer block.
- `1`, if you are computing the delay introduced by an Unbuffer block.

`v`

Amount of buffer overlap. Specify one of the following:

- The value of the **Buffer overlap** parameter, if you are computing the delay introduced by a Buffer block.
- `0`, if you are computing the delay introduced by an Unbuffer block.

`'mode'`

The tasking mode of the model. Specify one of the following options:

- `'singletasking'`
- `'multitasking'`

Default: `'multitasking'`

Definitions

Multitasking

When you run a model in `MultiTasking` mode, Simulink processes groups of blocks with the same execution priority through each stage of simulation based on task priority. Multitasking mode helps to create valid models of real-world multitasking systems, where sections of your model represent concurrent tasks. The **Tasking mode for periodic sample times** parameter on the Solver pane of the Configuration Parameters dialog box controls this setting.

Singletasking

When you run a model in `SingleTasking` mode, Simulink processes all blocks through each stage of simulation together. The **Tasking mode for periodic sample times** parameter on the Solver pane of the Configuration Parameters dialog box controls this setting.

Examples

Compute the delay introduced by a Buffer block in a multitasking model:

1. Open a model containing a Buffer block. For this example, open the `doc_buffer_tut4` model by typing doc_buffer_tut4 at the MATLAB command line.
2. Double-click the Buffer block to open the block mask. Verify that you have the following settings:
   - **Output buffer size** = 3
   - **Buffer overlap** = 1
   - **Initial conditions** = 0

Based on these settings, two of the required inputs to the `rebuffer_delay` function are as follows:

- `n = 3`
- `v = 1`

3 To determine the frame size of the input signal to the Buffer block, open the Signal From Workspace block mask. Verify that you have the following settings:

- **Signal** = `sp_examples_src`
- **Sample time** = `1`
- **Samples per frame** = `4`

Because **Samples per frame** = `4`, you know the `f` input to the `rebuffer_delay` function is `4`.

4 After you verify the values of all the inputs to the `rebuffer_delay` function, determine the delay that the Buffer block introduces in this multitasking model. To do so, type the following at the MATLAB command line:

```
d = rebuffer_delay(4,3,1)

d =
     8
```

Compute the delay introduced by an Unbuffer block in a multitasking model:

1 Open a model containing an Unbuffer block. For this example, open the `doc_unbuffer_ref1` model by typing doc_unbuffer_ref1 at the MATLAB command line.

2 To determine the frame size of the input to the Buffer block, open the Signal From Workspace block mask by double-clicking the block in your model. Verify that you have the following settings:

- **Signal** = `sp_examples_src`
- **Sample time** = `1`
- **Samples per frame** = `3`

Because **Samples per frame** = `3`, you know the `f` input to the `rebuffer_delay` function is `3`.

3 Use the `rebuffer_delay` function to determine the amount of delay that the Unbuffer block introduces in this multitasking model. To compute the delay introduced by the Unbuffer block, use `f = 3`, `n = 1` and `v = 0`.

```
d = rebuffer_delay(3,1,0)

d =
     3
```

See Also

Buffer | Unbuffer

How To

- "Buffering Delay and Initial Conditions"

signalblks.LMSFilter class

Purpose

LMS adaptive filter

Description

The `LMSFilter` implements an adaptive FIR filter object that returns the filtered output, the error vector, and filter weights. The LMS filter uses one of five different LMS algorithms.

Construction

`H = signalblks.LMSFilter` returns an adaptive FIR filter object, `H`, that computes the filtered output, filter error and the filter weights for a given input and desired signal using the Least Mean Squares (LMS) algorithm.

`HLMS = signalblks.LMSFilter('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns an LMS filter object, `H`, with each property set to the specified value.

`H = signalblks.LMSFilter(LEN, '`*`PropertyName`*`',` *`PropertyValue`*`, ...)` returns an LMS filter object, `H`, with the `Length` property set to `LEN`, and other specified properties set to the specified values.

Properties

`Method`

Method to calculate filter weights

Specify the method used to calculate filter weights as `LMS`, `Normalized LMS`, `Sign-Error LMS`, `Sign-Data LMS`, or `Sign-Sign LMS`. The default value of this property is `LMS`.

`Length`

Length of FIR filter weights vector

Specify the length of the FIR filter weights vector as a positive integer. The default value of this property is `32`.

`StepSizeSource`

How to specify adaptation step size

Choose how to specify the adaptation step size factor as `Property` or `Input port`. The default value of this property is `Property`.

`StepSize`

Adaptation step size

Specify the adaptation step size factor as a nonnegative real number. For convergence of the normalized LMS method, the step size should be greater than `0` and less than `2`. This property only applies when the `StepSizeSource` property is `Property`. The default value of this property is `0.1`. This property is tunable.

`LeakageFactor`

Leakage factor used in the LMS filter

Specify the leakage factor as a real number between `0` and `1` inclusive. A leakage factor of `1` corresponds to no leakage in the adapting method. The default value of this property is `1`. This property is tunable.

`InitialConditions`

Initial conditions of filter weights

Specify the initial values of the FIR filter weights as a scalar or vector of length equal to the `Length` property value. The default value of this property is `0`.

`AdaptInputPort`

Enable weight adaptation

Specify when the LMS filter should adapt the filter weights. By default, the value of this property is `false`, and the object continuously updates the filter weights. When this property is set to `true`, an adaptation control input is provided to the `step` method. If the value of this input is nonzero, the object continuously updates the filter weights. If the input is zero, the filter weights remain at their current value.

`WeightsResetInputPort`

Enable weight reset

Specify when the LMS filter should reset the filter weights. By default, the value of this property is `false`, and the object does not reset the weights. When this property is set to `true`, a reset control input is provided to the `step` method, and the `WeightsResetCondition` property applies. The object resets the filter weights based on the values of the `WeightsResetCondition` property and the `reset` input to the `step` method.

`WeightsResetCondition`

Reset trigger setting for filter weights

Specify the event to reset the filter weights as `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero`. The LMS filter resets the filter weights based on the values of this property and the `reset` input to the `step` method. This property only applies when the `WeightsResetInputPort` property is `true`. The default value of this property is `Non-zero`.

`WeightsOutputPort`

Enable returning filter weights

Set this property to `true` to output the adapted filter weights. The default value of this property is `true`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

StepSizeDataType

Step size word and fraction lengths

Specify the step size fixed-point data type as Same word length as first input or Custom. Setting this property also sets the LeakageFactorDataType property to the same value. This property only applies when the StepSizeSource property is Property. The default value of this property is Same word length as first input.

CustomStepSizeDataType

Step size word and fraction lengths

Specify the step size fixed-point type as a numerictype object with a Signedness of Auto. This property only applies when the StepSizeSource property is Property and the StepSizeDataType property is Custom. The word lengths of this property and the CustomLeakageFactorDataType property must be the same. The default value of this property is numerictype([],16,15).

LeakageFactorDataType

Leakage factor word and fraction lengths

Specify the leakage factor fixed-point data type as Same word length as first input or Custom. Setting this property also sets the StepSizeDataType property to the same value. This property only applies when the StepSizeSource property is Property. The default value of this property is Same word length as first input.

CustomLeakageFactorDataType

Leakage factor word and fraction lengths

Specify the leakage factor fixed-point type as a numerictype object with a Signedness of Auto. This property only applies when the StepSizeSource property is Property and the LeakageFactorDataType property is Custom. The word lengths of this property and the CustomStepSizeDataType property

must be the same. The default value of this property is `numerictype([],16,15)`.

`WeightsDataType`

Weights word and fraction lengths

Specify the filter weights fixed-point data type as `Same as first input` or `Custom`. The default value of this property is `Same as first input`.

`CustomWeightsDataType`

Weights word and fraction lengths

Specify the filter weights fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property only applies when the `WeightsDataType` property is `Custom`. The default value of this property is `numerictype([],16,15)`.

`EnergyProductDataType`

Energy product word and fraction lengths

Specify the energy product fixed-point data type as `Same as first input` or `Custom`. This property only applies when the `Method` property is `Normalized LMS`. Setting this property also sets the `ConvolutionProductDataType`, `StepSizeErrorProductDataType`, `WeightsUpdateProductDataType`, and `QuotientDataType` properties to the same value. The default value of this property is `Same as first input`.

`CustomEnergyProductDataType`

Energy product word and fraction lengths

Specify the energy product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property only applies when the `Method` property is `Normalized LMS` and the `EnergyProductDataType` property is `Custom`. The word lengths of the `CustomEnergyProductDataType`, `CustomConvolutionProductDataType`, CustomStepSizeErrorProductDataType, CustomWeightsUpdateProductDataType, and CustomQuotientDataType properties must be the same. The default value of this property is numerictype([],32,20).

EnergyAccumulatorDataType

Energy accumulator word and fraction lengths

Specify the energy accumulator fixed-point data type as Same as first input or Custom. This property only applies when the Method property is Normalized LMS. Setting this property also sets the ConvolutionAccumulatorDataType property to the same value. The default value of this property is Same as first input.

CustomEnergyAccumulatorDataType

Energy accumulator word and fraction lengths

Specify the energy accumulator fixed-point type as a scaled numerictype object with a Signedness of Auto. This property only applies when the Method property is Normalized LMS and the EnergyAccumulatorDataType property is Custom. The word lengths of the CustomEnergyProductDataType, CustomConvolutionProductDataType, CustomStepSizeErrorProductDataType, CustomWeightsUpdateProductDataType, and CustomQuotientDataType properties must be the same. The default value of this property is numerictype([],32,20).

ConvolutionProductDataType

Convolution product word and fraction lengths

Specify the convolution product fixed-point data type as Same as first input or Custom. Setting this property also sets the EnergyProductDataType, StepSizeErrorProductDataType, WeightsUpdateProductDataType and QuotientDataType properties to the same value. The default value of this property is Same as first input.

CustomConvolutionProductDataType

Convolution product word and fraction lengths

Specify the convolution product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property only applies when the `ConvolutionProductDataType` property is `Custom`. The word lengths of the `CustomEnergyProductDataType`, `CustomConvolutionProductDataType`, `CustomStepSizeErrorProductDataType`, `CustomWeightsUpdateProductDataType`, and `CustomQuotientDataType` properties must be the same. The default value of this property is `numerictype([],32,20)`.

`ConvolutionAccumulatorDataType`

Convolution accumulator word and fraction lengths

Specify the convolution accumulator fixed-point data type as `Same as first input` or `Custom`. Setting this property also sets the `EnergyAccumulatorDataType` property to the same value. The default value of this property is `Same as first input`.

`CustomConvolutionAccumulatorDataType`

Convolution accumulator word and fraction lengths

Specify the convolution accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property only applies when the `ConvolutionAccumulatorDataType` property is `Custom`. The word lengths of the `CustomEnergyAccumulatorDataType` and `CustomConvolutionAccumulatorDataType` properties must be the same. The default value of this property is`numerictype([],32,20)`.

`StepSizeErrorProductDataType`

Step size error product word and fraction lengths

Specify the step size error product fixed-point data type as `Same as first input` or `Custom`. Setting this property also sets the `ConvolutionProductDataType`, `EnergyProductDataType`, `WeightsUpdateProductDataType`, and `QuotientDataType` properties to the same value. The default value of this property is `Same as first input`.

`CustomStepSizeErrorProductDataType`

Step size error product word and fraction lengths

Specify the step size error product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property only applies when the `StepSizeErrorProductDataType` property is `Custom`. The word lengths of the `CustomEnergyProductDataType`, `CustomConvolutionProductDataType`, `CustomStepSizeErrorProductDataType`, `CustomWeightsUpdateProductDataType`, and `CustomQuotientDataType` properties must be the same. The default value of this property is `numerictype([],32,20)`.

`WeightsUpdateProductDataType`

Weight update product word and fraction lengths

Specify the weight update product fixed-point data type as `Same as first input` or`Custom`. Setting this property also sets the `ConvolutionProductDataType`, `EnergyProductDataType`, `StepSizeErrorProductDataType`, and `QuotientDataType` properties to the same value. The default value of this property is `Same as first input`.

`CustomWeightsUpdateProductDataType`

Weight update product word and fraction lengths

Specify the weight update product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property only applies when the `WeightsUpdateProductDataType` property is `Custom`. The word lengths of the `CustomEnergyProductDataType`, `CustomConvolutionProductDataType`, `CustomStepSizeErrorProductDataType`, `CustomWeightsUpdateProductDataType`, and `CustomQuotientDataType` properties must be the same. The default value of this property is `numerictype([],32,20)`.

`QuotientDataType`

Quotient word and fraction lengths

Specify the quotient fixed-point data type as `Same as first input`, `Custom`. This property only applies when the `Method` property is `Normalized LMS`. Setting this property also sets the `ConvolutionProductDataType`, `EnergyProductDataType`, `StepSizeErrorProductDataType`, and `WeightsUpdateProductDataType` properties to the same value. The default value of this property is `Same as first input`.

`CustomQuotientDataType`

Quotient word and fraction lengths

Specify the quotient fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property only applies when the `Method` property is `Normalized LMS` and the `QuotientDataType` property is `Custom`. The word lengths of the `CustomEnergyProductDataType`, `CustomConvolutionProductDataType`, `CustomStepSizeErrorProductDataType`, `CustomWeightsUpdateProductDataType`, and `CustomQuotientDataType` properties must be the same. The default value of this property is `numerictype([],32,20)`.

Methods

clone	Create LMS filter object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs of step method
isLocked	Locked status (logical) for input attributes and non-tunable properties

reset	Reset filter states for LMS filter
step	Apply LMS adaptive filter to input

Examples

System identification of an FIR filter:

```
hlms1 = signalblks.LMSFilter(11, 'StepSize', 0.01);
hfilt = signalblks.DigitalFilter; % System to be identified
hfilt.TransferFunction = 'FIR (all zeros)';
hfilt.Numerator = fir1(10, .25);
x = randn(1000,1); % input signal
d = step(hfilt, x) + 0.01*randn(1000,1); % desired signal
[y,e,w] = step(hlms1, x, d);
subplot(2,1,1), plot(1:1000, [d,y,e]);
title('System Identification of an FIR filter');
legend('Desired', 'Output', 'Error');
xlabel('time index'); ylabel('signal value');
subplot(2,1,2); stem([hfilt.Numerator.', w]);
legend('Actual','Estimated');
xlabel('coefficient #'); ylabel('coefficient value');
```

Noise cancellation:

```
hlms2 = signalblks.LMSFilter('Length', 11, ...
'Method', 'Normalized LMS',...
'AdaptInputPort', true, ...
'StepSizeSource', 'Input port', ...
'WeightsOutputPort', false);
hfilt2 = signalblks.DigitalFilter(...
'TransferFunction', 'FIR (all zeros)', ...
'Numerator', fir1(10, [.5, .75]));
x = randn(1000,1); % Noise
d = step(hfilt2, x) + sin(0:.05:49.95)'; % Noise + Signal
a = 1; % adaptation control
mu = 0.05; % step size
```

```
[y, err] = step(hlms2, x, d, mu, a);
subplot(2,1,1), plot(d), title('Noise + Signal');
subplot(2,1,2),plot(err), title('Signal');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the LMS Filter block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.BlockLMSFilter` | `signalblks.DigitalFilter`

Purpose Create LMS filter object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an LMS filter object, `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

Purpose Return number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.LMSFilter.getNumOutputs

Purpose Return number of outputs of step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the LMS filter.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.LMSFilter.reset

Purpose Reset filter states for LMS filter

Syntax `reset(H)`

Description `reset(H)` resets the filter states of the LMS FIR filter,`H`, to their initial values specified in the `InitialConditions` property. The initial filter state values correspond to the initial conditions for difference equation defining the adaptive filter. After the `step` method applies the LMS filter to nonzero input data, the states may be different. Invoking the `step` method again without first invoking the`reset` method may produce different outputs for an identical input.

Example of resetting filter states:

```
hlms1 = signalblks.LMSFilter(11, 'StepSize', 0.01);
hfilt = signalblks.DigitalFilter; % System to be identified
hfilt.TransferFunction = 'FIR (all zeros)';
hfilt.Numerator = fir1(10, .25);
x = randn(1000,1); % input signal
d = step(hfilt, x) + 0.01*randn(1000,1); % desired signal
[y,e,w] = step(hlms1, x, d);
% Call step method again without resetting filter states
[y1,e1,w1] = step(hlms1,x,d);
isequal(y,y1) % Returns 0
% Now reset the filter states to zero
reset(hlms1)
% invoke step method
[y2,e2,w2] = step(hlms1,x,d);
isequal(y,y2) % Returns 1
```

Purpose

Apply LMS adaptive filter to input

Syntax

```
[Y,ERR,WTS] = step(H,X,D)
[Y, ERR] = step(H,X,D)
[...] = step(H,X,D,MU)
[...] = step(H,X,D,A)
[...] = step(H,X,D,R)
[Y,ERR,WTS] = step(H,X,D,MU,A,R)
```

Description

`[Y,ERR,WTS] = step(H,X,D)` applies the LMS filter object, `H` to the input `X` , using `D` as the desired signal and returns the filtered output in `Y` , the filter error in `ERR` , and the estimated filter weights in `WTS` . The LMS filter estimates the filter weights needed to minimize the mean square error between the output and the desired signal.

`[Y, ERR] = step(H,X,D)` filters the input `X` , using `D` as the desired signal, and returns the filtered output in `Y` and the filter error in `ERR` when the `WeightsOutputPort` property is `false`.

`[...] = step(H,X,D,MU)` filters the input `X` , using `D` as the desired signal and `MU` as the step size, when the `StepSizeSource` property is `'Input port'`.

`[...] = step(H,X,D,A)` filters the input `X` , using `D` as the desired signal and `A` as the adaptation control, when the `AdaptInputPort` property is `true`. When `A` is nonzero, the LMS filter continuously updates the filter weights. When `A` is zero, the filter weights remain constant.

`[...] = step(H,X,D,R)` filters the input `X` , using `D` as the desired signal and `R` as a reset signal when the `WeightsResetInputPort` property is `true`. The `WeightsResetCondition` property can be used to set the reset trigger condition. If a reset event occurs, the LMS filter resets the filter weights to their initial values.

`[Y,ERR,WTS] = step(H,X,D,MU,A,R)` filters the input `X` using `D` as the desired signal, `MU` as the step size, `A` as the adaptation control, and `R` as the reset signal, and returns the filtered output in `Y` , the filter error in `ERR` , and the adapted filter weights in `WTS`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Solve lower-triangular matrix equation

Description

The `LowerTriangularSolver` object solves $\mathbf{LX} = \mathbf{B}$ for $\mathbf{X}$ when $\mathbf{L}$ is a square, lower-triangular matrix with the same number of rows as $\mathbf{B}$.

Construction

`H = signalblks.LowerTriangularSolver` returns a linear system solver, `H`, used to solve the linear system $\mathbf{LX} = \mathbf{B}$, where $\mathbf{L}$ is a lower (or unit-lower) triangular matrix.

`H = signalblks.LowerTriangularSolver('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a linear system solver, `H`, with each specified property set to the specified value.

Properties

`OverwriteDiagonal`

Replace diagonal elements of input with ones

When you set this property to `true`, the linear system solver replaces the elements on the diagonal of the input, $\mathbf{L}$, with ones. Set this property to either `true` or `false`. The default value of this property is `false`.

`RealDiagonalElements`

Indicate that diagonal of complex input is real

When you set this property to `true`, the linear system solver optimizes computation speed if the diagonal elements of complex input, $\mathbf{L}$, are real. This property applies only when you set the `OverwriteDiagonal` property to `false`. Set this property to either `true` or `false`. The default value of this property is `false`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`ProductDataType`

Data type of product

Specify the product data type as `Internal rule`, `Same as input`, or `Custom`. The default value of this property is `Internal rule`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `ProductDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`AccumulatorDataType`

Data type of accumulator

Specify the accumulator data type as `Internal rule`, `Same as first input`, `Same as product`, or `Custom`. The default value of this property is `Internal rule`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`OutputDataType`

Data type of output

Specify the output data type as `Same as first input` or `Custom`. The default value of this property is `Same as first input`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create lower triangular solver object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Return number of outputs from step method
isLocked	Return logical value to indicate whether input attributes and non-tunable properties are locked
step	Solve matrix equation for specified inputs

Examples

Solve a lower-triangular matrix equation.

```
hlowtriang = signalblks.LowerTriangularSolver;
u = tril(rand(4, 4));
b = rand(4, 1);
% Check that result is the solution to the linear
% equations.
x1 = inv(u)*b
x = step(hlowtriang, u, b)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Forward Substitution block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.UpperTriangularSolver`

Purpose Create lower triangular solver object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a `LowerTriangularSolver` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object.

signalblks.LowerTriangularSolver.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.LowerTriangularSolver.isLocked

Purpose

Return logical value to indicate whether input attributes and non-tunable properties are locked

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `LowerTriangularSolver` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Solve matrix equation for specified inputs

Syntax

```
X = step(H,L,B)
```

Description

`X = step(H,L,B)` computes the solution, $\mathbf{X}$, of the matrix equation $\mathbf{LX} = \mathbf{B}$, where $\mathbf{L}$ is a square, lower-triangular matrix with the same number of rows as the matrix $\mathbf{B}$.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.LPCToAutocorrelation class

Purpose

Convert linear prediction coefficients to autocorrelation coefficients

Description

The `LPCToAutocorrelation` System object converts linear prediction coefficients to autocorrelation coefficients.

Construction

`H = signalblks.LPCToAutocorrelation` returns an LPC to autocorrelation System object, `H`, that converts linear prediction coefficients (LPC) to autocorrelation coefficients.

`H = signalblks.LPCToAutocorrelation('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an `LPC` to autocorrelation conversion object, `H`, with each specified property set to the specified value.

Properties

`PredictionErrorInputPort`

Enable prediction error power input

Choose how to select the prediction error power. When you set this property to `true`, you must specify the prediction error power as a second input to the `step` method. When you set this property to `false`, the object assumes that the prediction error power is `1`. The default value of this property is `false`.

`NonUnityFirstCoefficientAction`

Action to take when first LPC coefficient is not `1`

Specify the action that the object takes when the first coefficient of each channel of the LPC input is not `1`. Select `Replace with 1` or `Normalize`. The default value of this property is `Replace with 1`.

Methods

clone	Create LPC to autocorrelation object with same property values
getNumInputs	Return number of expected inputs to step method

getNumOutputs	Return number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Compute autocorrelation coefficients from LPC coefficents

Examples

Convert the linear prediction coefficients to autocorrelation coefficients.

```
a = [1.0 -1.4978 1.4282 -1.3930 0.9076 -0.3855 0.0711].';
hlpc2ac = signalblks.LPCToAutocorrelation;
ac = step(hlpc2ac, a);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the LPC/RC to Autocorrelation block reference page. The object properties correspond to the block parameters, except for:

- The object does not have a property that corresponds to the **Type of Conversion** block parameter. The object's behavior corresponds to the block's behavior when you set the **Type of Conversion** parameter to `LPC to autocorrelation`.

See Also

`signalblks.LPCToLSF` | `signalblks.LPCToRC` | `signalblks.LPCToCepstral` | `signalblks.RCToAutocorrelation`

signalblks.LPCToAutocorrelation.clone

Purpose Create LPC to autocorrelation object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an `LPCToAutocorrelation` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.LPCToAutocorrelation.getNumInputs

Purpose

Return number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.LPCToAutocorrelation.getNumOutputs

Purpose Return number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `LPCToAutocorrelation` System object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.LPCToAutocorrelation.step

Purpose Compute autocorrelation coefficients from LPC coefficents

Syntax

```
AC = step(H,A)
AC = step(H,A,P)
```

Description

`AC = step(H,A)` converts the columns of the linear prediction coefficients, `A`, to autocorrelation coefficients, `AC`. The object assumes a prediction error power of `1`.

`AC = step(H,A,P)` converts the columns of the linear prediction coefficients, `A`, to autocorrelation coefficients, `AC`, using `P` as the prediction error power, when you set the `PredictionErrorInputPort` property to `true`. `P` must be a row vector with same number of columns as `A`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Convert linear prediction coefficients to cepstral coefficients

Description

The `LPCToCepstral` object converts linear prediction coefficients to cepstral coefficients.

Construction

`H = signalblks.LPCToCepstral` returns an LPC to cepstral converter object, `H`, that converts linear prediction coefficients (LPCs) to cepstral coefficients (CCs).

`H = signalblks.LPCToCepstral('PropertyName',PropertyValue,...)` returns an LPC to cepstral converter object, `H`, with each specified property set to the specified value.

Properties

`PredictionErrorInputPort`

Enable prediction error power input

Choose how to set the prediction error power. When you set this property to `true`, you must specify the prediction error as a second input to the `step` method. When you set this property to `false`, the prediction error power is assumed to be 1. The default value for this property is `false`.

`CepstrumLengthSource`

Source of cepstrum length

Select how to specify the length of cepstral coefficients: `Auto` or `Property`. The default value of this property is `Auto`. When this property is set to `Auto`, the length of each channel of the cepstral coefficients output is the same as the length of each channel of the input LPC coefficients. The default value for this property is `Property`.

`CepstrumLength`

Number of output cepstral coefficients

Set the length of the output cepstral coefficients vector as a scalar numeric integer. This property applies when you set the

CepstrumLengthSource property to Property. The default value of this property is 10.

NonUnityFirstCoefficientAction

LPC coefficent non unity action

Specify the action that the object takes when the first coefficient of each channel of the LPC input is not 1. Select Replace with 1 or Normalize. The default value for this property is Replace with 1.

Methods

clone	Create LPC to cepstral object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Compute cepstral coefficients from columns of input LPC coefficients

Examples

Convert the linear prediction coefficients (LPC) to cepstral coefficients.

```
hlevinson = signalblks.LevinsonSolver;
hlevinson.AOutputPort = true; % Output polynomial coefficients
hac = signalblks.Autocorrelator;
hac.MaximumLagSource = 'Property';
hac.MaximumLag = 9; % Compute autocorrelation lags between [0:9]
hlpc2cc = signalblks.LPCToCepstral;
x = [1:100]';
a = step(hac, x);
```

```
A = step(hlevinson, a); % Compute LPC coefficients
CC = step(hlpc2cc, A); % Convert LPC to CC.
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the LPC to/from Cepstral Coefficients block reference page. The object properties correspond to the block parameters, except for:

- The object does not have a property that corresponds to the **Type of Conversion** block parameter. The object's behavior corresponds to the block's behavior when you set the **Type of Conversion** parameter to `LPCs to cepstral coefficients`.

See Also

`signalblks.CepstralToLPC` | `signalblks.LPCToLSF` | `signalblks.LPCToRC`

signalblks.LPCToCepstral.clone

Purpose Create LPC to cepstral object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an `LPCToCepstral` System object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object.

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.LPCToCepstral.getNumOutputs

Purpose

Number of outputs from step method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `LPCToCepstral` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.LPCToCepstral.step

Purpose

Compute cepstral coefficients from columns of input LPC coefficients

Syntax

```
CC = step(H,A)
CC = step(H,A,P)
```

Description

`CC = step(H,A)` computes the cepstral coefficients, `CC`, from the columns of input linear prediction coefficients, `A`. The object assumes the prediction error power is `1`.

`CC = step(H,A,P)` computes the cepstral coefficients, `CC`, from the columns of input linear prediction coefficients, `A`, using `P` as the prediction error power, when you set the `PredictionErrorInputPort` property to `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.LPCToLSF class

Purpose

Convert linear prediction coefficients to line spectral frequencies

Description

The LPCToLSF object converts linear prediction coefficients to line spectral frequencies.

Construction

H = signalblks.LPCToLSF returns a System object, H, that converts linear prediction coefficients (LPCs) to line spectral frequencies (LSFs).

H = signalblks.LPCToLSF('*PropertyName*',*PropertyValue*,...) returns an LPC to LSF System object, H, with each specified property set to the specified value.

Properties

NumCoarseGridPoints

Number of coarse subintervals used for finding roots (LSP values)

Specify the number of coarse subintervals, n, used for finding line spectral pairs (LSP) values as a positive scalar integer. LSPs, which are the roots of two particular polynomials related to the input LPC polynomial, always lie in the range (-1, 1). The System object finds these roots using the Chebyshev polynomial root finding method. To compute LSF outputs, the object computes the arc cosine of the LSPs, outputting values ranging from 0 to pi radians. The object divides the interval (-1, 1) into n subintervals and looks for roots in each subinterval. If you set n too small in relation to the LPC polynomial order, the object can fail to find some of the roots. The default value of this property is 64. This property is tunable.

NumBisects

Value of bisection refinement used for finding roots

Specify the root bisection refinement value, k, used in the Chebyshev polynomial root finding method, where each line spectral pair (LSP) output is within $\frac{1}{(n \cdot 2^k)}$ of the actual LSP value. Here n represents the value of the NumCoarseGridPoints property, and the object searches a maximum of $k \cdot (n-1)$ points for finding the roots. You must set the NumBisects property value k, to a positive scalar integer. The default value of this property is 4. This property is tunable.

ExceptionOutputPort

Produces an output with the validity status of the LSF output

Set this property to true to return a second output that indicates whether the computed LSF values are valid. The output is a vector with a length equal to the number of channels. A logical value of 1 indicates valid output. A logical value of 0 indicates invalid output. The LSF outputs are invalid when the object fails to find all the LSF values or when the input LPCs are unstable. The default value of this property is false.

OverwriteInvalidOutput

Enable overwriting invalid output with previous output

Specify the action that the System object should take for invalid LSF outputs. When you set this property to true, the object overwrites the invalid output with the previous output. When you set this property to false, the object does not take any action on invalid outputs and ignores the outputs.

FirstOutputValuesSource

Source of values for first output when output is invalid

Specify the source of values for the first output when the output is invalid as Auto or Property. This property applies when you set the OverwriteInvalidOutput property to true. The default value of this property is Auto. When you set this property to Auto, the object uses a default value for the first output. This default value corresponds to the LSF representation of an allpass filter.

FirstOutputValues

Value of the first output

Specify a numeric vector of LSF values for overwriting an invalid first output. The length of this vector must be one less than the length of the input LPC vector. For multichannel inputs, you can set this property to a matrix with the same number of channels as the input, or one vector that will be applied to every channel. The default value of this property is an empty vector. This property applies when you set the `OverwriteInvalidOutput` property to `true` and the `FirstOutputValuesSource` property to `Property`.

`NonUnityFirstCoefficientAction`

Action to take when first LPC coefficient is not `1`

Specify the action the object takes when the first coefficient of each channel of the LPC input is not `1` as `Replace with 1` or `Normalize`. The default value of this property is `Replace with 1`.

Methods

clone	Create LPC To LSF object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset values for overwriting invalid outputs to their initial values
step	converts LPC coefficients to line spectral frequencies

Examples

Convert to linear prediction coefficients to line spectral frequencies.

```
a = [1.0000 0.6149 0.9899 0.0000 0.0031 -0.0082]';
```

```
hlpc2lsf = signalblks.LPCToLSF;
y = step(hlpc2lsf, a); % Convert to LSF coefficients
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the LPC to LSF/LSP Conversion block reference page. The object properties correspond to the block parameters, except for:

- There is no object property that corresponds to the **Output** block parameter. The object only supports LSF outputs in the range $(0, \prod)$

See Also

signalblks.LSFToLPC | signalblks.LPCToLSP

Purpose Create LPC To LSF object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an `LPCToLSF` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.LPCToLSF.getNumInputs

Purpose

Number of expected inputs to step method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Number of outputs from step method

Syntax

```
getNumOutputs(H)
```

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.LPCToLSF.isLocked

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `LPCToLSF` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset values for overwriting invalid outputs to their initial values

Syntax `reset(H)`

Description `reset(H)` resets the values for overwriting the invalid outputs to their initial values.

signalblks.LPCToLSF.step

Purpose converts LPC coefficients to line spectral frequencies

Syntax

```
LSF = step(H,A)
[..., STATUS] = step(H,A)
```

Description

`LSF = step(H,A)` converts the `LPC` coefficients, `A` , to line spectral frequencies, `LSF`, in the range (0 pi). The System object operates along the columns of the input `A`.

`[..., STATUS] = step(H,A)` also returns the status flag, `STATUS`, indicating if the current output is valid when the `ExceptionOutputPort` property is `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Convert linear prediction coefficients to line spectral pairs

Description

The `LPCToLSP` object converts linear prediction coefficients to line spectral pairs.

Construction

`H = signalblks.LPCToLSP` returns a System object, `H`, that converts linear prediction coefficients (LPCs) to line spectral pairs (LSPs).

`H = signalblks.LPCToLSP('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an `LPC` to `LSF` System object, `H`, with each specified property set to the specified value.

Properties

`NumCoarseGridPoints`

Number of coarse subintervals used for finding roots (LSP values)

Specify the number of coarse subintervals, n, used for finding line spectral pairs (LSP) values, as a positive scalar integer. LSPs, which are the roots of two particular polynomials related to the input LPC polynomial, always lie in the range (-1, 1). The System object finds these roots using the Chebyshev polynomial root finding method. The object divides the interval (-1, 1) into n subintervals and looks for roots in each subinterval. If n is set to too small a number in relation to the LPC polynomial order, the object can fail to find some of the roots. The default value of this property is `64`. This property is tunable.

`NumBisects`

Value of bisection refinement used for finding roots

Specify the root bisection refinement value, k, that the Chebyshev polynomial uses in the root finding method. For each line spectral pair (LSP) the output is within

$$\frac{1}{\mathrm{n} \cdot 2^{\mathrm{k}}}$$

of the actual LSP value. Here n represents the value of the `NumCoarseGridPoints` property and the object searches a maximum of

$$k \cdot (n\text{-}1)$$

points for finding the roots. The `NumBisects` property value k, must be a positive scalar integer. The default value of this property is `4`. This property is tunable.

`ExceptionOutputPort`

Produces an output with the validity status of the LSP output

Set this property to `true` to return a second output that indicates whether the computed LSP values are valid. The object outputs a vector length equal to the number of channels. A logical value of `1` indicates the output is valid. A logical value of `0` indicates the output is invalid. The LSP outputs are invalid when the object fails to find all the LSP values or when the input LPCs are unstable. The default value of this property is `false`.

`OverwriteInvalidOutput`

Enable overwriting invalid output with previous output

Specify the action that the object takes for invalid LSP outputs. When you set this property to `true`, the object overwrites the invalid output with the previous output. When you set this property to `false`, the object takes no action on invalid outputs and ignores the outputs.

`FirstOutputValuesSource`

Source of values for first output when output is invalid

Specify the source of values for the first output when the output is invalid as `Auto` or `Property`. This property applies only when you set the `OverwriteInvalidOutput` property to `true`. The default value for this property is `Auto`. When this property is `Auto`, the object uses a default value for the first output. This default value corresponds to the LSP representation of an allpass filter.

`FirstOutputValues`

Value of the first output

Specify a numeric vector of LSP values for overwriting an invalid first output. The length of this vector must be one less than the length of the input LPC vector. For multichannel inputs, set this property can to a matrix with the same number of channels as the input, or one vector that you're applying to every channel. The default value of this property is an empty vector. This property applies only when you set the `OverwriteInvalidOutput` property to `true` and the `FirstOutputValuesSource` property to `Property`.

`NonUnityFirstCoefficientAction`

First coefficient non unity action

Specify the action that the object takes when the first coefficient of each channel of the LPC input is not equal to `1`. Specify as one of `Replace with 1` or `Normalize`. The default value for this property is `Replace with 1`.

Methods

clone	Create LPC To LSP object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Convert the linear prediction coefficients to line spectral pairs

Examples

Convert the LPC coefficients to LSP coefficients.

```
a = [1.0000 0.6149 0.9899 0.0000 0.0031 -0.0082]';
hlpc2lsp = signalblks.LPCToLSP;
y = step(hlpc2lsp, a); % Convert to LSP coefficients
```

Algorithms

This object implements the algorithm, inputs, and outputs described on the LPC to LSF/LSP Conversion block reference page. The object properties correspond to the block parameters, except for:

- No object property corresponds to the **Output** block parameter. The object only supports LSP outputs.

See Also

`signalblks.LSPToLPC` | `signalblks.LPCToLSF`

Purpose Create LPC To LSP object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a `LPCToLSP` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.LPCToLSP.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.LPCToLSP.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `LPCToLSP` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Convert the linear prediction coefficients to line spectral pairs

Syntax

```
LSF = step(H,A)
[..., STATUS] = step(H,A)
```

Description

`LSF = step(H,A)` converts the `LPC` coefficients, `A` , to line spectral pairs normalized in the range (-1 1), `LSP`. The object operates along the columns of the input `A`.

`[..., STATUS] = step(H,A)` also returns the status flag, `STATUS`, indicating if the current output is valid when the `ExceptionOutputPort` property is `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.LPCToRC class

Purpose

Convert linear prediction coefficients to reflection coefficients

Description

The `LPCToRC` object converts linear prediction coefficients to reflection coefficients.

Construction

`H = signalblks.LPCToRC` returns an LPC to RC System object, `H`, that converts linear prediction coefficients (LPC) to reflection coefficients (RC).

`H = signalblks.LPCToRC('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an LPC to RC conversion object, `H`, with each specified property set to the specified value.

Properties

`PredictionErrorOutputPort`

Enable normalized prediction error power output

Set this property to `true` to return the normalized error power as a vector with one element per input channel. Each element varies between zero and one. The default value of this property is `true`.

`ExceptionOutputPort`

Produces an output with the stability status of a filter represented by LPC coefficients

Set this property to `true` to return the stability status of the filter. A logical value of `1` indicate a stable filter. A logical value of `0` indicate an unstable filter. The default value of this property is `false`.

`NonUnityFirstCoefficientAction`

Action to take when first LPC coefficient is not 1

Specify the action that the object takes when the first coefficient of each channel of the LPC input is not 1. Select `Replace with 1` or `Normalize`. The default value of this property is `Replace with 1`.

Methods

clone	Create LPC to RC object with same property values
getNumInputs	Return the number of expected inputs to step method
getNumOutputs	Return number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Convert columns of linear prediction coefficients to reflection coefficients

Examples

Convert the linear prediction coefficients to reflection coefficients.

```
load mtlb
hlevinson = signalblks.LevinsonSolver;
hlevinson.AOutputPort = true;
hlevinson.KOutputPort = false;
hac = signalblks.Autocorrelator;
hlpc2rc = signalblks.LPCToRC;
hac.MaximumLagSource = 'Property';
% Compute autocorrelation for lags between [0:10]
hac.MaximumLag = 10;
a = step(hac, mtlb);
A = step(hlevinson, a); % Compute LPC coefficients
[K, P] = step(hlpc2rc, A); % Convert to RC
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the LPC to/from RC block reference page. The object properties correspond to the block parameters, except for:

- There is no object property that corresponds to the **Type of conversion** block parameter. The object always converts LPC to RC.

- The `NonUnityFirstCoefficientAction` object property corresponds to the **If first input value is not 1** block parameter. There is neither a `Normalize and warn` nor an `Error` option for the object.

See Also

`signalblks.RCToLPC` | `signalblks.LPCToAutocorrelation`

Purpose

Create LPC to RC object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates a `LPCToRC` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.LPCToRC.getNumInputs

Purpose Return the number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.LPCToRC.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `LPCToRC` System object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Convert columns of linear prediction coefficients to reflection coefficients

Syntax

```
[K,P] = step(H,A)
K = step(H,A)
[..., S] = step(H,A)
```

Description

`[K,P] = step(H,A)` converts the columns of linear prediction coefficients, `A`, to reflection coefficients `K` and outputs the normalized prediction error power, `P`.

`K = step(H,A)` converts the columns of linear prediction coefficients, `A`, to reflection coefficients `K`, when you set the `PredictionErrorOutputPort` property to `false`.

`[..., S] = step(H,A)` also outputs the `LPC` filter stability, `S`, when you set the `ExceptionOutputPort` property to `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.LSFToLPC class

Purpose

Convert line spectral frequencies to linear prediction coefficients

Description

The `LSFToLPC` object converts line spectral frequencies to linear prediction coefficients.

Construction

`H = signalblks.LSFToLPC` returns an LSF to LPC System object, `H`, which converts line spectral frequencies (LSFs) to linear prediction coefficients (LPCs).

Methods

clone	Create LSF to LPC object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Return number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Convert input line spectral frequencies to linear prediction coefficients

Examples

Convert line spectral frequencies to linear prediction coefficients.

```
a = [1.0000 0.6149 0.9899 0.0000 0.0031 -0.0082]'
hlpc2lsf = signalblks.LPCToLSF;
ylsf = step(hlpc2lsf, a);
hlsf2lpc = signalblks.LSFToLPC;
ylpc = step(hlsf2lpc, ylsf) % Check values are same as a.
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the LSF/LSP to LPC Conversion block reference page. The object properties correspond to the block parameters, except for:

- The object does not have a property that corresponds to the **Input** block parameter. The object's behavior corresponds to the block's behavior when you set the **Input** parameter to `LSF in range (0 pi)`.

See Also

`signalblks.LPCToLSF` | `signalblks.LSPToLPC`

signalblks.LSFToLPC.clone

Purpose

Create LSF to LPC object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates an `LSFToLPC` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.LSFToLPC.getNumOutputs

Purpose Return number of outputs from step method

Syntax

```
getNumOutputs(H)
```

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `LSFToLPC` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.LSFToLPC.step

Purpose

Convert input line spectral frequencies to linear prediction coefficients

Syntax

```
A = step(H,LSF)
```

Description

`A = step(H,LSF)` converts the input line spectral frequencies, (LSF), in the range `(0,pi)`, `LSF` , to linear prediction coefficients, `A`. The input can be a vector or a matrix, where each column of the matrix is treated as a separate channel.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Convert line spectral pairs to linear prediction coefficients

Description

The `LSPToLPC` object converts line spectral pairs to linear prediction coefficients.

Construction

`H = signalblks.LSPToLPC` returns an LSP to LPC System object, `H`, which converts line spectral pairs (LSPs) to linear prediction coefficients (LPCs).

Methods

clone	Create LSP to LPC object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Return number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Convert input line spectral pairs to linear prediction coefficients

Examples

Convert line spectral pairs to linear prediction coefficients.

```
ylsp = [0.7080 0.0103 -0.3021 -0.3218 -0.7093]';
hlsp2lpc = signalblks.LSPToLPC;
ylpc = step(hlsp2lpc, ylsp)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the LSF/LSP to LPC Conversion block reference page. The object properties correspond to the block parameters, except for:

- No object property corresponds to the **Input** block parameter. The object converts LSP in the range (-1, 1) to LPC.

See Also

signalblks.LPCToLSP | signalblks.LSFToLPC

signalblks.LSPToLPC.clone

Purpose

Create LSP to LPC object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates an `LSPToLPC` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.LSPToLPC.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.LSPToLPC.isLocked

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `LSPToLPC` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Convert input line spectral pairs to linear prediction coefficients

Syntax

```
A = step(H,LSP)
```

Description

`A = step(H,LSP)` converts the input line spectral pairs in the range (-1,1), `LSP`, to linear prediction coefficients, `A`. The input can be a vector or a matrix, where each column of the matrix is treated as a separate channel.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.LUFactor class

Purpose

Factor square matrix into lower and upper triangular matrices

Description

The `LUFactor` object factors a square matrix into lower and upper triangular matrices.

Construction

`H = signalblks.LUFactor` returns an `LUFactor` System object, `H`, which factors a row permutation of a square input matrix `A` as

$\mathrm{A_p} = \mathrm{L} \cdot U$, where `L` is the unit-lower triangular matrix, and `U` is the upper triangular matrix. The row-pivoted matrix A_p contains the rows of A permuted as indicated by the permutation index vector P. The equivalent MATLAB code is `Ap = A(P,:)`.

`H = signalblks.LUFactor('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an `LUFactor` object, `H`, with each specified property set to the specified value.

Properties

`ExceptionOutputPort`

Set to `true` to output the singularity of input

Set this property to `true` to output the singularity of the input as logical data type values of true or false. An output of true indicates that the current input is singular, and an output of false indicates the current input is nonsingular.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Zero`, `Nearest`, `Ceiling` or `Floor`. The default value of this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`ProductDataType`

Product word and fraction lengths

Specify the product fixed-point data type as `Internal rule`, `Same as input` or `Custom`. The default value of this property is `Internal rule`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Internal rule`, `Same as input`, `Same as product` or `Custom`. The default value of this property is `Internal rule`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as `Same as input` or `Custom`. The default value of this property is `Same as input`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create LU Factor object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Decompose matrix into lower and uppper triangular matrices

Examples

Decompose a square matrix into the lower and upper components.

```
hlu = signalblks.LUFactor;
x = rand(4)
[LU, P] = step(hlu, x);
L = tril(LU,-1)+diag(ones(size(LU,1),1));
U = triu(LU);
y = L*U
% Check back whether this is equal to the permuted x
xp = x(P,:)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the LU Factorization block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.LDLFactor`

Purpose Create LU Factor object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an `LUFactor` object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.LUFactor.getNumInputs

Purpose

Number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description

`getNumInputs(H)` returns the number of expected inputs from the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Number of outputs from step method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.LUFactor.isLocked

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `LUFactor` object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Decompose matrix into lower and uppper triangular matrices

Syntax

```
[LU,P] = step(H,A)
[LU,P,S] = step(H,A)
```

Description

`[LU,P] = step(H,A)` decomposes the matrix `A` into lower and uppper triangular matrices. The output `LU` is a composite matrix with lower triangle elements from `L` and upper triangle elements from `U` . The permutation vector `P` is the second output.

`[LU,P,S] = step(H,A)` returns an additional output `S` indicating if the input is singular when the `ExceptionOutputPort` property is set to `true`

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

.

signalblks.Maximum class

Purpose Find maximum value of input or sequence of inputs

Description The `Maximum` object finds the maximum values of an input or sequence of inputs.

Construction `H = signalblks.Maximum` returns an object, `H`, that computes the value and index of the maximum elements in an input or a sequence of inputs.

`H = signalblks.Maximum('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a maximum-finding object, `H`, with each specified property set to the specified value.

Properties

`ValueOutputPort`

Output maximum value

Set this property to `true` to output the maximum (when `RunningMaximum` is `false`) or the running maximum (when `RunningMaximum` is `true`). The default value of this property is `true`.

`RunningMaximum`

Calculate over single input or multiple inputs

When you set this property to `true`, the object computes the maximum value over a sequence of inputs. When you set this property to `false`, the object computes the maximum value over the current input. The default value of this property is `false`.

`IndexOutputPort`

Output the index of the maximum value

Set this property to `true` to output the index of the maximum value of the input. This property applies only when you set the `RunningMaximum` property to `false`. The default value of this property is `true`.

`ResetInputPort`

Additional input to enable resetting of running maximum

Set this property to `true` to enable resetting of the running maximum. When you set this property to `true`, a reset input must be specified to the `step` method to reset the running maximum. This property applies only when you set the `RunningMaximum` property to `true`. The default value of this property is `false`.

`ResetCondition`

Condition that triggers resetting of running maximum

Specify the event that resets the running maximum as `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero`. This property applies only when you set the `ResetInputPort` property to `true`. The default value of this property is `Non-zero`.

`IndexBase`

Numbering base for index of maximum value

Specify the numbering used when computing the index of the maximum value as starting from either `One` or `Zero`. This property applies only when you set the `IndexOutputPort` property to true. The default value of this property is `One`.

`Dimension`

Dimension to operate along

Specify how the maximum calculation is performed over the data as one of `All`, `Row`, `Column`, or `Custom`. This property applies when you set the `RunningMaximum` property to `false`. The default value for this property is `Column`.

`CustomDimension`

Numerical dimension to calculate over

Specify the integer dimension of the input signal over which the object finds the maximum. The value of this property cannot exceed the number of dimensions in the input signal. This property only applies when you set the `Dimension` property to `Custom`. The default value of this property is `1`.

ROIProcessing

Enable region-of-interest processing

Set this property to true to enable calculation of the maximum value within a particular region of an image. This property applies when you set the Dimension property to All and the RunningMaximum property to false. The default value of this property is false.

For full ROI processing support, install the Video and Image Processing Blockset product. If you only have the Signal Processing Blockset product installed, you can only specify the value of the ROIForm property as Rectangles.

ROIForm

Type of region of interest

Specify the type of region of interest as Rectangles, Lines, Label matrix, or Binary mask. This property applies only when you set the ROIProcessing property to true. The default value of this property is Rectangles.

For full ROI processing support, install the Video and Image Processing Blockset product. If you have only the Signal Processing Blockset product installed, you can only specify the value of this property as Rectangles.

ROIPortion

Calculate over entire ROI or just perimeter

Specify whether to calculate the maximum over the Entire ROI or the ROI perimeter. This property applies only when you set the ROIForm property to Rectangles. The default value of this property is Entire ROI.

ROIStatistics

Calculate statistics for each ROI or one for all ROIs

Specify whether to calculate `Individual statistics for each ROI` or a `Single statistic for all ROIs`. This property applies only when you set the `ROIForm` property to `Rectangles`, `Lines`, or `Label matrix`.

`ValidityOutputPort`

Output flag indicating if any part of ROI is outside input image

When you set the `ROIForm` property to `Lines` or `Rectangles`, set this property to `true` to return the validity of the specified ROI being completely inside of the image. When you set the `ROIForm` property to `Label Matrix`, set this property to `true` to return the validity of the specified label numbers. The default value of this property is `false`.

`FrameBasedProcessing`

Process input as frames or samples

Set this property to `true` to enable frame-based processing for 2-D inputs. Set this property to `false` to enable sample-based processing. The object always performs sample-based processing for *N*-dimensional inputs where *N* is greater than `2`. This property applies when you set the `RunningMaximum` to `true`. The default value of this property is `true`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Action to take when integer input is out-of-range

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`ProductDataType`

Data type of product

Specify the product fixed-point data type as `Same as input` or `Custom`. The default value of this property is `Same as input`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`AccumulatorDataType`

Data type of accumulator

Specify the accumulator fixed-point data type as `Same as product`, `Same as input`, or `Custom`. The default value of this property is `Same as product`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

Methods

clone	Create maximum-finding object with same property values
getNumInputs	Return number of expected inputs to step method

getNumOutputs	Number of outputs from step method
isLocked	Return logical value to indicate whether input attributes and non-tunable properties are locked
reset	Reset computation of running maximum
step	Compute maximum value

Examples

Find a maximum value and its index.

```
hmax1 = signalblks.Maximum;
x = randn(100,1);
[y, I] = step(hmax1, x);
```

Compute a running maximum.

```
hmax2 = signalblks.Maximum;
hmax2.RunningMaximum = true;
x = randn(100,1);
y = step(hmax2, x); % Find running maximum
% y(i) is the maximum of all values in the vector x(1:i)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Maximum block reference page. The object properties correspond to the block parameters, except for:

- **Treat sample-based row input as a column** block parameter is not supported by the `signalblks.Maximum` object.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See “What Are Sample- and Frame-Based Processing?” for more information.

See Also

`signalblks.Minimum` | `signalblks.Mean`

Purpose Create maximum-finding object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates a `Maximum` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

signalblks.Maximum.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Number of outputs from step method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.Maximum.isLocked

Purpose

Return logical value to indicate whether input attributes and non-tunable properties are locked

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `Maximum` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset computation of running maximum

Syntax `reset(H)`

Description `reset(H)` resets the computation of the running maximum for the `Maximum` object `H`.

signalblks.Maximum.step

Purpose Compute maximum value

Syntax

```
[VAL,IND] = step(H,X)
VAL = step(H,X)
IND = step(H,X)
VAL = step(H,X,R)
[...] = step(H,I,ROI)
[...] = step(H,I,LABEL,LABELNUMBERS)
[...,FLAG] = step(H,I,ROI)
[...,FLAG] = step(H,I,LABEL,LABELNUMBERS)
```

Description

`[VAL,IND] = step(H,X)` returns the maximum value, `VAL`, and the index or position of the maximum value, `IND`, along a dimension of `X` specified by the value of the `Dimension` property.

`VAL = step(H,X)` returns the maximum value, `VAL`, of the input `X`. When the `RunningMaximum` property is `true`, `VAL` corresponds to the maximum value over a sequence of inputs.

`IND = step(H,X)` returns the zero- or one-based index `IND` of the maximum value. To enable this type of processing, set the `IndexOutputPort` property to `true` and the `ValueOutputPort` and `RunningMaximum` properties to `false`.

`VAL = step(H,X,R)` computes the maximum value, `VAL`, over a sequence of inputs, and resets the state of `H` based on the value of reset signal, `R`, and the `ResetCondition` property. To enable this type of processing, set the `RunningMaximum` property to `true` and the `ResetInputPort` property to `true`.

`[...] = step(H,I,ROI)` computes the maximum of an input image, `I`, within the given region of interest, `ROI`. To enable this type of processing, set the `ROIProcessing` property to `true` and the `ROIForm` property to `Lines`, `Rectangles` or `Binary mask`.

`[...] = step(H,I,LABEL,LABELNUMBERS)` computes the maximum of an input image, `I`, for a region whose labels are specified in the vector `LABELNUMBERS`. To enable this type of processing, set the `ROIProcessing` property to `true` and the `ROIForm` property to `Label matrix`.

`[...,FLAG] = step(H,I,ROI)` also returns `FLAG`, indicating whether the given region of interest is within the image bounds. To enable this type of processing, set the `ROIProcessing` and `ValidityOutputPort` properties to `true` and the `ROIForm` property to `Lines`, `Rectangles` or `Binary mask`.

`[...,FLAG] = step(H,I,LABEL,LABELNUMBERS)` also returns `FLAG`, indicating whether the input label numbers are valid. To enable this type of processing, set the `ROIProcessing` and `ValidityOutputPort` properties to `true` and the `ROIForm` property to `Label matrix`.

For full ROI processing support, install the Video and Image Processing Blockset product. If you have only the Signal Processing Blockset product installed, you can only specify the value of the `ROIForm` property as `Rectangles`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.Mean class

Purpose

Find mean value of input or sequence of inputs

Description

The `Mean` object finds the mean of an input or sequence of inputs.

Construction

`H = signalblks.Mean` returns an object, `H`, that computes the mean of an input or a sequence of inputs.

`H = signalblks.Mean('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a mean-finding object, `H`, with each specified property set to the specified value.

Properties

`RunningMean`

Calculate over single input or multiple inputs

When you set this property to `true`, the object calculates the mean over a sequence of inputs. When you set this property to `false`, the object computes the mean over the current input. The default value of this property is `false`.

`ResetInputPort`

Additional input to enable resetting of running mean

Set this property to `true` to enable resetting of the running mean. When you set this property to `true`, a reset input must be specified to the `step` method to reset the running mean. This property applies only when you set the `RunningMean` property to `true`. The default value of this property is `false`.

`ResetCondition`

Condition that triggers resetting of running mean

Specify the event that resets the running maximum as `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero`. This property applies only when you set the `ResetInputPort` property to `true`. The default value of this property is `Non-zero`.

`Dimension`

Dimension to operate along

Specify how the mean calculation is performed over the data as `All`, `Row`, `Column`, or `Custom`. This property applies when you set the `RunningMean` property to `false`. The default value of this property is `Column`.

`CustomDimension`

Numerical dimension to calculate over

Specify the integer dimension, indexed from one, of the input signal over which the object calculates the mean. The value of this property cannot exceed the number of dimensions in the input signal. This property only applies when you set the `Dimension` property to `Custom`. The default value of this property is `1`.

`ROIProcessing`

Enable region-of-interest processing

Set this property to `true` to enable calculation of the mean within a particular region of an image. This property applies when you set the `Dimension` property to `All` and the `RunningMean` property to `false`. The default value of this property is `false`.

For full ROI processing support, install the Video and Image Processing Blockset product. If you only have the Signal Processing Blockset product installed, you can only specify the value of the `ROIForm` property as `Rectangles`.

`ROIForm`

Type of region of interest

Specify the type of region of interest as `Rectangles`, `Lines` or `Label matrix`. This property applies only when you set the `ROIProcessing` property to `true`. The default value of this property is `Rectangles`.

For full ROI processing support, install the Video and Image Processing Blockset product. If you have only the Signal Processing Blockset product installed, you can only specify the value of this property as `Rectangles`.

`ROIPortion`

Calculate over entire ROI or just perimeter

Specify whether to calculate the mean over the `Entire ROI` or the `ROI perimeter`. This property applies only when you set the `ROIForm` property to `Rectangles`. The default value of this property is `Entire ROI`.

`ROIStatistics`

Calculate statistics for each ROI or one for all ROIs

Specify whether to calculate `Individual statistics for each ROI` or a `Single statistic for all ROIs`. This property applies only when you set the `ROIForm` property to `Rectangles`, `Lines`, or `Label matrix`. The default value of this property is `Individual statistics for each ROI`.

`ValidityOutputPort`

Output flag indicating if any part of ROI is outside input image

When you set the `ROIForm` property to `Lines`, `Rectangles`, or `Binary mask`, set this property to `true` to return the validity of the specified ROI being completely inside of the image. When you set the `ROIForm` property to `Label Matrix`, set this property to `true` to return the validity of the specified label numbers. The default value of this property is `false`.

`FrameBasedProcessing`

Process input as frames or samples

Set this property to `true` to enable frame-based processing for 2-D inputs. Set this property to `false` to enable sample-based processing. The object always performs sample-based processing for N-dimensional inputs where N is greater than `2`. This property applies when you set the `RunningMean` to `true`. The default value of this property is `true`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Action to take when integer input is out-of-range

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`AccumulatorDataType`

Data type of accumulator

Specify the accumulator fixed-point data type as `Same as input`, or `Custom`. The default value of this property is `Same as input`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`OutputDataType`

Data type of output

Specify the output fixed-point data type as `Same as accumulator`, `Same as input`, or `Custom`. The default value of this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

Methods

clone	Create mean object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs from step method
isLocked	Return logical value to indicate whether input attributes and non-tunable properties are locked
reset	Reset internal states of mean-finding object
step	Compute mean

Examples

Compute the mean of a signal:

```
hmean1 = signalblks.Mean;
x = randn(100,1);
y = step(hmean1, x);
```

Compute the running mean of a signal:

```
 hmean2 = signalblks.Mean;
 hmean2.RunningMean = true;
 x = randn(100,1);
 y = step(hmean2, x);    % Find running mean
% y(i) is the mean of all values in the vector x(1:i)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Mean block reference page. The object properties correspond to the block parameters, except for:

- **Treat sample-based row input as a column** block parameter is not supported by the `signalblks.Mean` object.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See "What Are Sample- and Frame-Based Processing?" for more information.

See Also

`signalblks.Maximum` | `signalblks.Minimum`

signalblks.Mean.clone

Purpose Create mean object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a `Mean` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

Purpose

Return number of expected inputs to step method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.Mean.getNumOutputs

Purpose Return number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Return logical value to indicate whether input attributes and non-tunable properties are locked

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `Mean` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.Mean.reset

Purpose Reset internal states of mean-finding object

Syntax `reset(H)`

Description `reset(H)` sets the internal states of the `Mean` object `H` to their initial values.

Purpose

Compute mean

Syntax

```
Y = step(H,X)
Y = step(H,X,R)
Y = step(H,X,ROI)
Y = step(H,X,LABEL,LABELNUMBERS)
[Y,FLAG] = step(H,X,ROI)
[Y,FLAG] = step(H,X,LABEL,LABELNUMBERS)
```

Description

`Y = step(H,X)` computes the mean of `X`. When you set the `RunningMean` property to `true`, `Y` corresponds to the mean over a sequence of inputs.

`Y = step(H,X,R)` computes the mean value, `Y`, of the input elements over time, and optionally resets the computation of the running mean based on the value of the reset signal, `R`, and the `ResetCondition` property. To enable this type of processing, set the `RunningMean` property to `true` and the `ResetInputPort` property to `true`.

`Y = step(H,X,ROI)` computes the mean of input image `X` within the given region of interest `ROI`. To enable this type of processing, set the `ROIProcessing` property to `true` and the `ROIForm` property to `Lines`, `Rectangles` or `Binary mask`.

`Y = step(H,X,LABEL,LABELNUMBERS)` computes the mean of the input image, `X`, for the region whose labels are specified in the vector `LABELNUMBERS`. The regions are defined and labeled in the matrix `LABEL`. To enable this type of processing, set the `ROIProcessing` property to `true` and the `ROIForm` property to `Label matrix`.

`[Y,FLAG] = step(H,X,ROI)` also returns `FLAG`, indicating whether the given region of interest `ROI`, is within the image bounds. To enable this type of processing, set the `ROIProcessing` and `ValidityOutputPort` properties to `true` and the `ROIForm` property to `Lines`, `Rectangles` or `Binary mask`.

`[Y,FLAG] = step(H,X,LABEL,LABELNUMBERS)` also returns `FLAG` which indicates whether the input label numbers are valid. To enable this type of processing, set the `ROIProcessing` and `ValidityOutputPort` properties to `true` and the `ROIForm` property to `Label matrix`.

For full ROI processing support, install the Video and Image Processing Blockset product. If you have only the Signal Processing Blockset product installed, you can only specify the value of this property as `Rectangles`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Median value of input

Description

The `Median` object computes the median value of the input. The object can compute the median along each row or column of the input, along a specified dimension, or of the entire input.

Construction

`H = signalblks.Median` returns a median System object, `H`, that computes the median along the columns of the input using the `Quick sort` sorting method.

`H = signalblks.Median('PropertyName',PropertyValue,...)` returns a median System object, `H`, with each property set to the value you specify.

Properties

`SortMethod`

Sort method

Specify the method the object should use to sort the data before computing the median. You can specify `Quick sort` or `Insertion sort`. The quick sort algorithm uses a recursive sort method and is faster at sorting more than 32 elements. The insertion sort algorithm uses a non-recursive method and is faster at sorting less than 32 elements. If you are using the `Median` object to generate code, you should use the insertion sort algorithm to prevent recursive function calls in your generated code. The default value of this property is `Quick sort`.

`Dimension`

Dimension to operate along

Specify the dimension along which the object computes the median values. You can specify `All`, `Row`, `Column`, or `Custom`. The default value of this property is `Column`.

`CustomDimension`

Numerical dimension to operate along

Specify the dimension of the input signal (as a one-based value), over which the object computes the median. The value of this property cannot exceed the number of dimensions in the input signal. This property applies only when you set the `Dimension` property to `Custom`. The default value of this property is `1`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap`, or `Saturate`. The default value of this property is `Wrap`.

`ProductDataType`

Product word and fraction lengths

Specify the product data type as `Same as input`, or `Custom`. The default value of this property is `Same as input`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product data type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `ProductDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator data type as `Same as product`, `Same as input`, or `Custom`. The default value of this property is `Same as product`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the fixed-point accumulator data type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`OutputDataType`

Output word and fraction lengths

Specify the output data type as `Same as accumulator`, `Same as product`, `Same as input`, or `Custom`. The default value of this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the data type of the output as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create median object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Number of outputs from step method

isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Compute median value of input

Examples

Compute the median value of the input column.

```
h = signalblks.Median;
x = [7 -9 0 -1 2 0 3 5 -9]';
y = step(h, x)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Median block reference page. The object properties correspond to the block properties, except for:

- **Treat sample-based row input as a column** block parameter is not supported by the `signalblks.Median` System object.

See Also

`signalblks.Mean` | `signalblks.Minimum` | `signalblks.Maximum` | `signalblks.Variance`

Purpose Create median object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates a median object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object.

signalblks.Median.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.Median.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `Median` System object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Compute median value of input

Syntax

`Y = step(H,X)`

Description

`Y = step(H,X)` computes the median value of the input `X` and returns the result in `Y`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.Minimum class

Purpose Find minimum values of input or sequence of inputs

Description The `Minimum` object finds the minimum value of an input or sequence of inputs.

Construction `H = signalblks.Minimum` returns an object, `H`, that computes the value and/or index of the minimum elements in an input or a sequence of inputs.

`H = signalblks.Minimum('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a minimum-finding object, `H`, with each specified property set to the specified value.

Properties

`ValueOutputPort`

Output minimum value

Set this property to `true` to output the minimum (when `RunningMinimum` is `false`) or the running minimum (when `RunningMinimum` is `true`). The default value of this property is `true`.

`RunningMinimum`

Calculate over single input or multiple inputs

When you set this property to `true`, the object computes the minimum value over a sequence of inputs. When you set this property to `false`, the object computes the minimum value over the current input. The default value of this property is `false`.

`IndexOutputPort`

Output the index of the minimum value

Set this property to `true` to output the index of the minimum value of the input. This property applies only when you set the `RunningMinimum` property to `false`. The default value of this property is `true`.

`ResetInputPort`

Additional input to enable resetting of running minimum

Set this property to `true` to enable resetting of the running minimum. When you set this property to `true`, a reset input must be specified to the `step` method to reset the running minimum. This property applies only when you set the `RunningMinimum` property to `true`. The default value of this property is `false`.

`ResetCondition`

Condition that triggers resetting of running minimum

Specify the event that resets the running minimum as `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero`. This property applies only when you set the `ResetInputPort` property to `true`. The default value of this property is `Non-zero`.

`IndexBase`

Numbering base for index of minimum value

Specify the numbering used when computing the index of the minimum value as starting from either `One` or `Zero`. This property applies only when you set the `IndexOutputPort` property to `true`. The default value of this property is `One`.

`Dimension`

Dimension to operate along

Specify how the minimum calculation is performed over the data as one of `All`, `Row`, `Column`, or `Custom`. This property applies when you set the `RunningMinimum` property to `false`. The default value for this property is `Column`.

`CustomDimension`

Numerical dimension to calculate over

Specify the integer dimension of the input signal over which the object finds the minimum. The value of this property cannot exceed the number of dimensions in the input signal. This property only applies when you set the `Dimension` property to `Custom`. The default value of this property is `1`.

`ROIProcessing`

Enable region-of-interest processing

Set this property to `true` to enable calculation of the minimum value within a particular region of an image. This property applies when you set the `Dimension` property to `All` and the `RunningMinimum` property to `false`. The default value of this property is `false`.

For full ROI processing support, install the Video and Image Processing Blockset product. If you only have the Signal Processing Blockset product installed, you can only specify the value of the `ROIForm` property as `Rectangles`.

`ROIForm`

Type of region of interest

Specify the type of region of interest as `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`. This property applies only when you set the `ROIProcessing` property to `true`. The default value of this property is `Rectangles`.

For full ROI processing support, install the Video and Image Processing Blockset product. If you have only the Signal Processing Blockset product installed, you can only specify the value of this property as `Rectangles`.

`ROIPortion`

Calculate over entire ROI or just perimeter

Specify whether to calculate the minimum over the `Entire ROI` or the `ROI perimeter`. This property applies only when you set the `ROIForm` property to `Rectangles`. The default value of this property is `Entire ROI`.

`ROIStatistics`

Calculate statistics for each ROI or one for all ROIs

Specify whether to calculate `Individual statistics for each ROI` or a `Single statistic for all ROIs`. This property applies only when you set the `ROIForm` property to `Rectangles`, `Lines`, or `Label matrix`.

`ValidityOutputPort`

Output flag indicating if any part of ROI is outside input image

When you set the `ROIForm` property to `Lines` or `Rectangles`, set this property to `true` to return the validity of the specified ROI being completely inside of the image. When you set the `ROIForm` property to `Label Matrix`, set this property to `true` to return the validity of the specified label numbers. The default value of this property is `false`.

`FrameBasedProcessing`

Process input as frames or samples

Set this property to `true` to enable frame-based processing for 2-D inputs. Set this property to `false` to enable sample-based processing. The object always performs sample-based processing for *N*-dimensional inputs where *N* is greater than `2`. This property applies when you set the `RunningMinimum` to `true`. The default value of this property is `true`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Action to take when integer input is out-of-range

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`ProductDataType`

Data type of product

Specify the product fixed-point data type as `Same as input` or `Custom`. The default value of this property is `Same as input`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`AccumulatorDataType`

Data type of accumulator

Specify the accumulator fixed-point data type as `Same as product`, `Same as input`, or `Custom`. The default value of this property is `Same as product`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

Methods

clone	Create minimum-finding object with same property values
getNumInputs	Number of expected inputs to step method

getNumOutputs	Return number of outputs from method
isLocked	Return logical value to indicate whether input attributes and non-tunable properties are locked
reset	Reset internal states of minimum-finding object
step	Operate on inputs to calculate outputs

Examples

Find a minimum value and its index:

```
hmin1 = signalblks.Minimum;
x = randn(100,1);
[y, I] = step(hmin1, x);
```

Compute a running minimum:

```
hmin2 = signalblks.Minimum;
hmin2.RunningMinimum = true;
x = randn(100,1);
y = step(hmin2, x); % Find running minimum
% y(i) is the minimum of all values in the vector x(1:i)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Minimum block reference page. The object properties correspond to the block parameters, except for:

- **Treat sample-based row input as a column** block parameter is not supported by the `signalblks.Minimum` object.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing`

property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See "What Are Sample- and Frame-Based Processing?" for more information.

See Also

`signalblks.Maximum` | `signalblks.Mean`

Purpose

Create minimum-finding object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates a `Minimum` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

signalblks.Minimum.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs from `step` method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.Minimum.isLocked

Purpose Return logical value to indicate whether input attributes and non-tunable properties are locked

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `Minimum` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset internal states of minimum-finding object

Syntax `reset(H)`

Description `reset(H)` sets the internal states of the `Minimum` object `H` to their initial values.

signalblks.Minimum.step

Purpose

Operate on inputs to calculate outputs

Syntax

```
[VAL,IND] = step(H,X)
VAL = step(H,X)
IND = step(H,X)
VAL = step(H,X,R)
[...] = step(H,I,ROI)
[...] = step(H,I,LABEL,LABELNUMBERS)
[...,FLAG] = step(H,I,ROI)
[...,FLAG] = step(H,I,LABEL,LABELNUMBERS)
```

Description

`[VAL,IND] = step(H,X)` returns the minimum value, `VAL`, and the index or position of the minimum value, `IND`, along a dimension of `X` specified by the value of the `Dimension` property.

`VAL = step(H,X)` returns the minimum value, `VAL`, of the input `X`. When the `RunningMinimum` property is `true`, `VAL` corresponds to the minimum value over a sequence of inputs.

`IND = step(H,X)` returns the zero- or one-based index `IND` of the minimum value when the `IndexOutputPort` property is `true` and the `ValueOutputPort` property is `false`. The `RunningMinimum` property must be false.

`VAL = step(H,X,R)` computes the minimum value, `VAL`, over a sequence of inputs, and resets the state of `H` based on the value of reset signal, `R`, and the `ResetCondition` property. To enable this type of processing, set the `RunningMinimum` property to `true` and the `ResetInputPort` property to `true`.

`[...]  = step(H,I,ROI)` computes the minimum of an input image, `I`, within the given region of interest, `ROI`. To enable this type of processing, set the `ROIProcessing` property to `true` and the `ROIForm` property to `Lines`, `Rectangles` or `Binary mask`.

`[...]  = step(H,I,LABEL,LABELNUMBERS)` computes the minimum of an input image, `I`, for a region whose labels are specified in the vector `LABELNUMBERS`. This property applies only when you set the

`ROIProcessing` property to `true` and the `ROIForm` property to `Label matrix`.

`[...,FLAG] = step(H,I,ROI)` also returns `FLAG`, indicating whether the given region of interest is within the image bounds. To enable this type of processing, set the `ROIProcessing` and `ValidityOutputPort` properties to `true` and the `ROIForm` property to `Lines`, `Rectangles` or `Binary mask`.

`[...,FLAG] = step(H,I,LABEL,LABELNUMBERS)` also returns `FLAG`, indicating whether the input label numbers are valid. To enable this type of processing, set the `ROIProcessing` and `ValidityOutputPort` properties to `true` and the `ROIForm` property to `Label matrix`.

For full ROI processing support, install the Video and Image Processing Blockset product. If you have only the Signal Processing Blockset product installed, you can only specify the value of the `ROIForm` property as `Rectangles`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.MultimediaFileReader class

Purpose Read video frames, audio samples, or both from multimedia file

Description The `MultimediaFileReader` object reads video frames, audio samples, or both from a multimedia file.

Construction `H = signalblks.MultimediaFileReader` returns a multimedia file reader System object, `H`, to read video and/or audio from a multimedia file.

`H = signalblks.MultimediaFileReader('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a multimedia file reader System object, `H`, with each specified property set to the specified value.

`H = signalblks.MultimediaFileReader(FILENAME,'`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a multimedia file reader System object, `H`, with `Filename` property set to *`FILENAME`* and other specified properties set to the specified values.

Properties

`Filename`

Name of multimedia file from which to read

Specify the name of a multimedia file as a string. Specify the full path for the file only if the file is not on the MATLAB path. On UNIX platforms, the multimedia file reader only supports uncompressed AVI files. The default value of this property is `speech_dft.avi`.

`PlayCount`

Number of times to play the file

Specify a positive integer or `inf` to represent the number of times to play the file. The default value of this property is `inf`.

`AudioOutputPort`

Choose to output audio data

Use this property to control the audio output from the multimedia file reader. This property applies only when a multimedia file contains both audio and video streams. The default value of this property is `true`.

`VideoOutputPort`

Choose to output video data

Use this property to control the video output from the multimedia file reader. This property only applies when a multimedia file contains both audio and video streams. The default value of this property is `false`.

`SamplesPerAudioFrame`

Number of samples per audio frame

Specify the number of samples per audio frame as a positive scalar integer value. This property applies when the multimedia file contains only audio data. The default value of this property is `1024`.

`ImageColorSpace`

Choose whether output is RGB, intensity, or YCbCr

Specify whether you want the multimedia file reader to output `RGB`, `Intensity`, or `YCbCr 4:2:2` video frames. This property applies only when the multimedia file contains video. The default value for this property is `RGB`.

`VideoOutputDataType`

Data type of video data output

Set the data type of the video data output from the multimedia file reader. The multimedia file reader supports the following output data types: `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32` or `Inherit`. This property applies if the multimedia file contains video. The default value for this property is `single`.

`AudioOutputDataType`

Data type of audio samples output

Set the data type of the audio data output from the multimedia file reader. The multimedia file reader supports the following output data types: `double`, `single`, `int16`, `uint8`. This property applies only if the multimedia file contains audio. The default value of this property is `int16`.

Methods

clone	Create multimedia file reader object with same property values
close	Release resources
getNumInputs	Number of expected inputs to step method
getNumOutputs	Returns number of outputs from method
info	Return information about specific multimedia file
isDone	End-of-file status (logical)
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset internal states of multimedia file reader to read from beginning of file
step	Output one frame of video frames, audio samples, or both to multimedia signal

Examples

Read an audio file and play it back using the standard audio output device.

```
hmfr = signalblks.MultimediaFileReader('speech_dft.avi');
```

```
hap = signalblks.AudioPlayer('SampleRate', 22050);
while ~isDone(hmfr)
audio = step(hmfr);
step(hap, audio);
end
close(hmfr); % close the input file
close(hap); % close the audio output device
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the From Multimedia File block reference page. The object properties correspond to the block parameters, except for:

- The object has no corresponding property for the **Inherit sample time from file** block parameter. The object always inherits the sample time from the file.
- The **Multimedia outputs** block parameter corresponds to both the `AudioOutputPort` and the `VideoOutputPort` object properties.
- The object has no corresponding property for the **Image signal** block parameter. The object always outputs an *M*-by-*N*-by-*P* color video signal.

See Also

`signalblks.MultimediaFileWriter`

signalblks.MultimediaFileReader.clone

Purpose Create multimedia file reader object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a `MultimediaFileReader` System object `C`, with the same property values as `H`.

The clone method creates a new unlocked object with uninitialized states.

Purpose Release resources

Syntax `close(H)`

Description `close(H)` releases system resources (such as memory, file handles or hardware connections).

signalblks.MultimediaFileReader.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of outputs to the `step` method

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Returns number of outputs from `step` method

Syntax

```
getNumOutputs(H)
```

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.MultimediaFileReader.info

Purpose Return information about specific multimedia file

Syntax

```
S = info(H)
```

Description `S = info(H)` returns a `MATLAB` structure, `S`, with information about the multimedia file specified in the `Filename` property. The number of fields for `S` varies depending on the audio or video content of the file. The possible fields and values for the structure `S` are:

Field	Value
Audio	Logical value indicating if the file has audio content.
Video	Logical value indicating if the file has video content.
AudioSampleRate	Audio sampling rate of the multimedia file in Hz. This field is available when the file has audio content.
AudioNumBits	Number of bits used to encode the audio stream. This field is available when the file has audio content.
AudioNumChannels	Number of audio channels. This field is available when the file has audio content.
FrameRate	Frame rate of the video stream in frames per second. The value may vary from the actual frame rate of the recorded video, and takes into consideration any synchronization issues between audio and video streams when the file contains both audio and video content. This implies that video frames may be dropped if the audio stream leads the video stream by more than 1/(actual video frames per second). This field is available when the file has video content.
VideoSize	Video size as a two-element numeric vector of the form [VideoWidthInPixels, VideoHeightInPixels]. This field is available when the file has video content.
VideoFormat	Video signal format. This field is available when the file has video content.

Purpose End-of-file status (logical)

Syntax `STATUS = isDone(H)`

Description `STATUS = isDone(H)` returns a logical value, `STATUS` , when the `MultimediaFileReader` object, `H` , reaches the end of the multimedia file. If you set the `PlayCount` property to a value greater than `1` , `STATUS` is `true` each time the object reaches the end of the file. `STATUS` is the same as the `EOF` output value in the process method syntax.

signalblks.MultimediaFileReader.isLocked

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `MultimediaFileReader` object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset internal states of multimedia file reader to read from beginning of file

Syntax `reset(H)`

Description `reset(H)` resets the `MultimediaFileReader` object to read from the beginning of the file.

signalblks.MultimediaFileReader.step

Purpose

Output one frame of video frames, audio samples, or both to multimedia signal

Syntax

```
AUDIO = step(H)
I = step(H)
[I,AUDIO] = step(H)
[...,EOF] = step(H)
[Y,CB,CR] = step(H)
[Y,CB,CR,AUDIO] = step(H)
```

Description

`AUDIO = step(H)` outputs one frame of audio samples, `AUDIO`. This behavior requires the `AudioOutputPort` property to be `true` and an input file which contains audio data. After the object plays the file the number of times `PlayCount` specifies, `AUDIO` contains silence.

`I = step(H)` outputs one frame of multidimensional video signal, `I`. To obtain this behavior, set the `VideoOutputPort` property to `true` and use an input file that contains video data.

`[I,AUDIO] = step(H)` outputs one frame of multidimensional video signal, `I` , and one frame of audio samples, `AUDIO` . This behavior requires the `AudioOutputPort` and `VideoOutputPort` properties to be `true` and an input file which contains audio and video data.

`[...,EOF] = step(H)` gives the end-of-file indicator in `EOF`. `EOF` is true each time the output contains the last audio sample and/or video frame in the file.

`[Y,CB,CR] = step(H)` outputs one frame of `YCbCr 4:2:2` video data in the color components `Y`, `CB`, and `CR`. This behavior applies when you set the `VideoOutputPort` property to `true`, the `ImageColorSpace` property to `YCbCr 4:2:2`, and an input file which contains video data.

`[Y,CB,CR,AUDIO] = step(H)` outputs one frame of `YCbCr 4:2:2` video data in the color components `Y`, `CB`, and `CR`, and one frame of audio samples in AUDIO. This applies when you set the `AudioOutputPort` and `VideoOutputPort` properties to `true`, the `ImageColorSpace` property to `YCbCr 4:2:2`, and an input file which contains audio and video data.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.MultimediaFileWriter class

Purpose

Write video frames, audio samples, or both to multimedia file

Description

The `MultimediaFileWriter` object writes video frames, audio samples, or both to a multimedia file.

Construction

`H = signalblks.MultimediaFileWriter` returns a multimedia file writer System object, `H`, that writes video frames, audio samples, or both to a multimedia file (such as an `AVI` file).

`H = signalblks.MultimediaFileWriter('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a multimedia file writer System object, `H`, with each specified property set to the specified value.

`H = signalblks.MultimediaFileWriter(`*`FILENAME`*`,'`*`PropertyName`*`',`*`PropertyValue`*`,...` returns a multimedia file writer System object, `H`, with `Filename` property set to *`FILENAME`* and other specified properties set to the specified values.

Properties

`Filename`

Name of multimedia file to which to write

Specify the name of the multimedia file as a string. The default value of this property is `output.avi`.

`FileFormat`

Multimedia file format

Specify which multimedia file format the object writes. On Microsoft platforms, select `AVI`, `WAV`, `WMV`, or `WMA`. These abbreviations correspond to the following file formats:

- WAV: Microsoft WAVE Files
- WMV: Windows Media Video
- WMA: Windows Media Audio

- AVI: Audio-Video Interleave

The default setting for this property is AVI.

On all other platforms, the only selection is AVI.

`AudioInputPort`

Choose to write audio data

Use this property to specify whether the multimedia file writer object writes audio samples to a multimedia file. When you enable both this property and `VideoInputPort`, ensure that the video and audio input signals have the same frame period. Adjust the frame size (or number of rows) of the audio signal so that the frame period of the video signal is the same as the frame period of the audio signal. Calculate frame size by dividing the audio signal frequency by the frame rate of the video signal. The audio signal frequency is in samples per second (specified by `SampleRate`). The video signal frame rate is in frames per second (specified by `FrameRate`). The default selection for this property is `true`.

The multimedia file object takes a column vector as an input. Every column is a separate channel and each row corresponds to a single audio sample.

`VideoInputPort`

Choose to write video data

Use this property to specify whether the multimedia file writer object writes video frames to a multimedia file. When you enable both this property and the `AudioInputPort` property, ensure that the video and audio input signals have the same frame period. Adjust frame size (or number of rows) for the audio signal so the frame period is equal of both the video signal and the audio signal. Calculate frame size by dividing the audio signal frequency by the frame rate of the video signal. The audio signal frequency is in samples per second (specified by `SampleRate`). The video signal frame rate is in frames per second (specified by `FrameRate`). The default selection for this property is `false`

`AudioCompressor`

Algorithm that compresses audio data

Specify the type of compression algorithm the multimedia file writer uses to compress the audio data. Compression reduces the size of the multimedia file. Select `None (uncompressed)` to save uncompressed audio data to a multimedia file. The other options available reflect the audio compression algorithms installed on your system. This property is only available when writing WAV or AVI files on Windows platforms.

`VideoCompressor`

Algorithm that compresses video data

Specify the type of compression algorithm to multimedia file writer uses to compress the video data. This compression reduces the size of the multimedia file. Choose `None (uncompressed)` to save uncompressed video data to a multimedia file. The other options reflect the available video compression algorithms installed on your system. This property is only available when writing AVI files on Windows platforms.

`SampleRate`

Sampling rate of audio data stream

Specify the sampling rate of the input audio data as a positive numeric scalar. The default value of this property is `44100`. This property only applies when you set the `AudioInputPort` property to `true`.

`FrameRate`

Frame rate of video data stream

Specify the frame rate of the video data in frames per second as a positive numeric scalar. The default value of this property is `30`. This property only applies when you set the `VideoInputPort` property to `true`.

`AudioDataType`

Data type of the uncompressed audio

Specify the type of uncompressed audio data written to the file. This property only applies when writing uncompressed WAV files.

`FileColorSpace`

Color space to use when creating a file

Specify the color space of AVI files as `RGB` or `YCbCr 4:2:2`. This property only applies on Windows platforms when you set the FileFormat property to `AVI`. The default value for this property is `RGB`.

Methods

clone	Create multimedia file writer object with the same property values
close	Release resources
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Write one frame of audio output samples

Examples

Decimate an audio signal and write it to disk as an AVI file.

```
hmfr = signalblks.MultimediaFileReader('AudioOutputDataType',...
'double');
hfirdec = signalblks.FIRDecimator; % decimate by 2
hmfw = signalblks.MultimediaFileWriter...
('speech_dft_11025.avi', ...
```

```
'SampleRate', 22050/2);
while ~isDone(hmfr)
audio = step(hmfr);
audiod = step(hfirdec, audio);
step(hmfw, audiod);
end
close(hmfr);
close(hmfw);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the To Multimedia File block reference page. The object properties correspond to the block parameters, except for:

- The **Write** block parameter corresponds to both the `AudioOutputPort` and the `VideoOutputPort` object properties
- The `FrameRate` and `SampleRate` properties are not available on the block. The block inherits these signals from the input line connected to the ports.

See Also

`signalblks.MultimediaFileReader`

Purpose Create multimedia file writer object with the same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates a `MultimediaFile Writer` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.MultimediaFileWriter.close

Purpose Release resources

Syntax `close(H)`

Description `close(H)` releases system resources (such as memory, file handles or hardware connections).

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs from the `step` method

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `MultimediaFileWriter` System object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.MultimediaFileWriter.step

Purpose Write one frame of audio output samples

Syntax

```
step(H,AUDIO)
step(H,I)
step(H,I,AUDIO)
step(H,Y,Cb,Cr)
step(H,Y,CB,CR)
step(H,Y,CB,CR,AUDIO)
```

Description

`step(H,AUDIO)` writes one frame of audio samples, `AUDIO` , to the output file when you enable the `AudioInputPort` property. `AUDIO` represents either a vector or an M-by-2 matrix for mono or stereo inputs, respectively.

`step(H,I)` writes one frame of video, `I` , to the output file when you enable the `VideoInputPort` property. `I` is an M-by-N-by-3 color video signal.

`step(H,I,AUDIO)` writes one frame of video, `I` , and one frame of audio samples, `AUDIO` , to the output file when you enable both the `AudioInputPort` and `VideoInputPort` properties.

`step(H,Y,Cb,Cr)` writes one frame of video with `Y`, `Cb`, `Cr` components to the file. This applies only when you set the `FileColorSpace` property to `YCbCr 4:2:2`

`step(H,Y,CB,CR)` writes one frame of `YCbCr 4:2:2` data in the color components `Y`, `CB`, and `CR`, to the output file when you set the `VideoInputPort` property to `true`. The width of `CB` and `CR` must be half of the width of `Y`, and the value of the `FileColorSpace` property must be set to `YCbCr 4:2:2`.

`step(H,Y,CB,CR,AUDIO)` writes one frame of `YCbCr 4:2:2` data in the color components `Y`, `CB`, and `CR`, and one frame of audio samples, `AUDIO`, to the output file when you set both the `AudioInputPort` and `VideoInputPort` properties to `true`. The width of `CB` and `CR` must be half of the width of `Y`, and the value of the `FileColorSpace` property must be set to `YCbCr 4:2:2`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

signalblks.NCO class

Purpose

Generate real or complex sinusoidal signals

Description

The numerically controlled oscillator, or `NCO`, object generates real or complex sinusoidal signals. The amplitude of the generated signal is always 1.

Construction

`H = signalblks.NCO` returns an NCO System object, `H`, that generates a multichannel real or complex sinusoidal signal, with independent frequency and phase in each output channel.

`H = signalblks.NCO('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an `NCO` System object, `H`, with each specified property set to the specified value.

Properties

`PhaseIncrementSource`

Source of phase increment

Indicate how to specify the phase increment: `Property` or `Input port`. The default value of this property is `Input port`.

`PhaseIncrement`

Phase increment

Specify the phase increment as an integer scalar. This property applies only when you set the `PhaseIncrementSource` property to `Property`. The default value of this property is `100`.

`PhaseOffsetSource`

Source of phase offset

Specify the phase offset as `Property` or `Input port`. The default value of this property is `Property`.

`PhaseOffset`

Phase offset

Specify the phase offset as an integer scalar. This property applies only when you set the `PhaseOffsetSource` property to `Property`. The default value of this property is `0`.

`Dither`

Enable adding internal dithering to NCO algorithm

Set this property to `true` to add internal dithering to the NCO algorithm. Dithering is added using the PN Sequence Generator from the Communications Blockset product. The default value of this property is `true`.

`NumDitherBits`

Number of dither bits

Specify the number of dither bits as a positive integer. This property applies only when you set the `Dither` property to `true`. The default value of this property is `4`.

`PhaseQuantization`

Enable quantization of accumulated phase

Set this property to `true` to enable quantization of the accumulated phase. The default value of this property is `true`.

`NumQuantizerAccumulatorBits`

Number of quantizer accumulator bits

Specify the number of quantizer accumulator bits as an integer scalar greater than 1 and less than the accumulator word length. This property determines the number of entries in the lookup table of sine values. This property applies only when you set the `PhaseQuantization` property to `true`. The default value of this property is `12`.

`PhaseQuantizationErrorOutputPort`

Enable output of phase quantization error

Set this property to `true` to output the phase quantization error. This property applies only when you set the `PhaseQuantization` property to `true`. The default value of this property is `false`.

`Waveform`

Type of output signal

Specify the type of the output signal: `Sine`, `Cosine`, `Complex exponential` or `Sine and cosine`. The default value of this property is `Sine`.

`SamplesPerFrame`

Number of output samples per frame

Specify the number of samples per frame of the output signal. This property applies only when you set the `PhaseOffsetSource` property to `Property`. The default value of this property is `1`. When the `PhaseOffsetSource` property is `Input port`, and the `PhaseIncrementSource` property is `Property`, the number of rows or frame size of the phase offset input determines the number of samples per frame of the output signal. When you set both the `PhaseOffsetSource` and `PhaseIncrementSource` properties to `Input port`, the number of rows in the inputs must be 1, and the samples per frame of the output signal is 1.

`OutputDataType`

Output data type

Specify the output data type as one of `double`, `single` or `Custom`. The default value of this property is `Custom`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

This is a constant property with value `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

This is a constant property with value `Wrap`.

`AccumulatorDataType`

Accumulator word and fraction lengths

This is a constant property with value `Custom`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as an unscaled `numerictype` object with a `Signedness` of `Auto`. The default value of this property is `numerictype([],16)`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,14)`.

Methods

clone	Create NCO object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs from step method
info	Return characteristic information about generated signal
isLocked	Locked status (logical) for input attributes and non-tunable properties

reset	Reset accumulator of NCO object
step	Return sinusoidal signal with specified phase increment

Examples

Design an NCO source according to given specifications.

```
F0 = 510; % Output frequency = 510 Hz
df = 0.05; % Frequency resolution = 0.05 Hz
minSFDR = 96; % Spurious free dynamic range >= 96 dB
Ts = 1/8000; % Sample period = 1/8000 sec
dphi = pi/2; % Desired phase offset = pi/2;
```

Calculate number of accumulator bits required for the given frequency resolution

```
Nacc = ceil(log2(1/(df*Ts)));
% Actual frequency resolution achieved
actdf = 1/(Ts*2^Nacc);
% Calculate number of quantized accumulator bits
% required from the SFDR requirement
Nqacc = ceil((minSFDR-12)/6);
% Calculate the phase increment
phIncr = round(F0*Ts*2^Nacc);
% Calculate the phase offset
phOffset = 2^Nacc*dphi/(2*pi);

hnco = signalblks.NCO('PhaseIncrementSource', 'Property', ...
'PhaseIncrement', phIncr,...
'PhaseOffset', phOffset,...
'NumDitherBits', 4, ...
'NumQuantizerAccumulatorBits', Nqacc,...
'SamplesPerFrame', 1/Ts, ...
'CustomAccumulatorDataType', numerictype(true,Nacc));

y = step(hnco);
% Plot the mean-square spectrum of the 510 Hz sinewave
```

```
% generated by the NCO
sperd = spectrum.periodogram('hann'); ...
sperd.SamplingFlag = 'periodic';
msspectrum(sperd,double(y),'Fs',1/Ts);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the NCO block reference page. The object properties correspond to the block properties, except for:

- There is no object property that corresponds to the **Sample time** block parameter. The objects assumes a sample time of one second.

See Also

`signalblks.SineWave`

signalblks.NCO.clone

Purpose Create NCO object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an `NCO` system object `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states

signalblks.NCO.getNumInputs

Purpose Return number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.NCO.getNumOutputs

Purpose Return number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Return characteristic information about generated signal

Syntax

`S = info(H)`

Description

`S = info(H)` returns a structure containing characteristic information, `S`, about the signal being generated by the `NCO` System object, `H`. The number of fields of `S`, and their values vary depending on the property value settings of `H`. For description of the possible fields and their values, see the following table.

Field	Value
`NumPointsLUT`	Number of data points for lookup table. The lookup table is implemented as a quarter-wave sine table.
`SineLUTSize`	Quarter-wave sine lookup table size in bytes.
`TheoreticalSFDR`	Theoretical spurious free dynamic range (SFDR) in dBc. This field applies when you set the `PhaseQuantization` property to `true`.
`FrequencyResolution`	Frequency resolution of the `NCO` in Hz. The sample time of the output signal is assumed to be `1` sec.

signalblks.NCO.isLocked

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `NCO` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Reset accumulator of NCO object

Syntax

```
reset(H)
```

Description

`reset(H)` resets the accumulator of the `NCO` System object, `H`, to zero.

signalblks.NCO.step

Purpose

Return sinusoidal signal with specified phase increment

Syntax

```
Y = step(H,INC)
Y = step(H)
Y = step(H,OFFSET)
Y = step(H,INC,OFFSET)
[Y, QERR] = step(H,...)
```

Description

`Y = step(H,INC)` returns a sinusoidal signal, `Y` , generated by the `NCO`, with the specified phase increment, `INC`. `INC` must be a built-in integer or a fi object, and can either be a scalar or a row vector, where each row element corresponds to a separate channel.

`Y = step(H)` returns a sinusoidal signal, `Y`, when the `PhaseIncrementSource` and `PhaseOffsetSource` properties are both `Property`.

`Y = step(H,OFFSET)` returns a sinusoidal signal, `Y`, with phase offset, `OFFSET`, when the `PhaseOffsetSource` property is `Input port`. `OFFSET` must be a built-in integer or a fi object. The number of rows of `OFFSET` determines the number of samples per frame of the output signal, and the number of columns of the `OFFSET` determines the number of channels of the output signal.

`Y = step(H,INC,OFFSET)` returns a sinusoidal signal, `Y`, with phase increment, `INC`, and phase offset, `OFFSET`, when the `PhaseIncrementSource` and the `PhaseOffsetSource` properties are both `Input port`. `INC` and `OFFSET` must both be row vectors of the same length, where the length determines the number of channels in the output signal.

`[Y, QERR] = step(H,...)` returns a sinusoidal signal, `Y`, and output quantization error, `QERR` , when the `PhaseQuantization` and the `PhaseQuantizationErrorOutputPort` properties are both `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

signalblks.Normalizer class

Purpose

Perform vector normalization along the specified dimension

Description

The `Normalizer` object performs vector normalization along rows, columns, or specified dimension.

Construction

`H = signalblks.Normalizer` returns a normalization object, `H`, that normalizes the input over each column by the squared 2–norm of the column plus a bias term of 1e-10 used to protect against divide-by-zero.

`H = signalblks.Normalizer('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns a normalization object, `H`, with each property set to the specified value.

Properties

`Method`

Type of normalization to perform

Specify the type of normalization to perform as `2-norm` or `Squared 2-norm`. The `2-norm` mode supports floating-point signals only. The `Squared 2-norm` supports both fixed-point and floating-point signals. The default value for this property is `Squared 2-norm`.

`Bias`

Real number added in denominator to avoid division by zero

Specify the real number to be added in the denominator to avoid division by zero. The default value of this property is 1e-10. This property is tunable.

`Dimension`

Dimension to operate along

Specify whether to normalize along `Column` , `Row`, or `Custom`. The default value for this property is `Column`.

`CustomDimension`

Numerical dimension to operate along

Specify the one-based value of the dimension over which to normalize. The value of this parameter cannot exceed the number of dimensions in the input signal. This property applies when Dimension property is Custom. The default value of this property is 1.

FrameBasedProcessing

Enable frame-based processing

Set this property to true to enable frame-based processing. Set this property to false to enable sample-based processing. The default value for this property is true.

If the FrameBasedProcessing property is true:

- The normalization object treats a column vector or the columns of a matrix as single, independent channels.
- The normalization object treats a length N row vector as N independent channels.

If the FrameBasedProcessing property is false:

- The normalization object treats each entry of a vector or matrix as an independent channel.

Fixed-Point Properties

RoundingMethod

Rounding method for fixed-point operations

Specify the rounding method as one of Ceiling, Convergent, Floor , Nearest, Round, Simplest, or Zero. The default value for this property is Floor.

OverflowAction

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` or `Saturate`. The default value for this property is `Wrap`.

`ProductDataType`

Product word and fraction lengths

Specify the product fixed-point data type as one of `Same as input` or `Custom`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`. The default value of this property to `numerictype([],32,32)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as one of `Same as product`, `Same as input`, `Custom`. The default value for this property is `Same as product`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as one of `Same as accumulator`, `Same as product` , `Same as input`, `Custom`. The default value for this property is `Same as product`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],32,32)`.

Methods

clone	Create normalization object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Return number of outputs of step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Normalize input along specified dimension

Examples

Normalize a matrix:

```
hnorm = signalblks.Normalizer;
x = magic(3);
y = step(hnorm, x);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Normalization block reference page. The object properties correspond to the block parameters, except for:

- **Treat sample-based row input as column**– The block allows you to input a row vector and normalize the row vector as a column vector. The normalization object always normalizes along the value of the `Dimension` property.

- The normalization object does not support the **Minimum** and **Maximum** options for data output.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See "What Are Sample- and Frame-Based Processing?" for more information.

See Also

`signalblks.ArrayVectorMultiplier`

Purpose Create normalization object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates a normalizer object, `C`, with the same property values. The clone method creates a new unlocked object.

signalblks.Normalizer.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs of step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.Normalizer.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the normalization object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Normalize input along specified dimension

Syntax

```
Y = step(H,X)
```

Description

`Y = step(H,X)` returns a normalized output`Y` . The input `X` must be floating-point for the `2-norm` mode, and either fixed-point and floating-point for the `Squared 2-norm` mode.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.PeakFinder class

Purpose

Determine extrema (maxima or minima) in input signal

Description

The `PeakFinder` object determines the extrema (maxima or minima) in the input signal.

Construction

`H = signalblks.PeakFinder` returns a peak finder System object, `H`, that compares the current signal value to the previous and next values to determine if the current value is an extremum.

`H = signalblks.PeakFinder('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a peak finder System object, `H`, with each specified property set to the specified value.

Properties

`PeakType`

Looking for maxima, minima, or both

Specify whether the object is looking for maxima, minima, or both. This property can be set to: `Maxima`, `Minima`, `Maxima and Minima`. The default value of this property is `Maxima and Minima`.

`PeakIndicesOutputPort`

Enable output of the extrema indices

Set this property to `true` to output the extrema indices. The default value of this property is `false`.

`PeakValuesOutputPort`

Enable output of the extrema values

Set this property to `true` to output the extrema values. The default value of this property is `false`.

`MaximumPeakCount`

Number of extrema to look for in each input signal

The object stops searching the input signal for extrema once the maximum number of extrema has been found. The value of this property must be an integer greater than or equal to one. The default value of this property is `10`.

`IgnoreSmallPeaks`

Enable ignoring peaks below a threshold

Set this property to `true` if you want to eliminate the detection of peaks whose amplitudes are within a specified threshold of neighboring values. The default value of this property is `false`.

`PeakThreshold`

Threshold below which peaks are ignored

Specify the noise threshold value. This property defines the current input value to be a maximum if:

- The current input value minus the previous input value is greater than the threshold, and
- The current input value minus next input value is greater than the threshold.

This property applies when you set the `IgnoreSmallPeaks` property to `true`. The default value of this property is `0`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

The rounding method is a constant property set to `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

signalblks.PeakFinder class

Methods

clone	Create peak finder object with same property values
getNumInputs	Return number of expected inputs to the step method
getNumOutputs	Return number of outputs from step method
isLocked	Return logical value to indicate whether input attributes and non-tunable properties are locked
step	Number of extrema in input signal

Examples

Determine whether each value of an input signal is local maximum or minimum.

```
hpeaks1 = signalblks.PeakFinder;
hpeaks1.PeakIndicesOutputPort = true;
hpeaks1.PeakValuesOutputPort = true;

x1 = [9 6 10 3 4 5 0 12]';
% Find the peaks of each input
% [prev;cur;next]: {[9;6;10],[6;10;3],...}
[cnt1, idx1, val1, pol1] = step(hpeaks1, x1);
```

Determine peak values for a fixed-point input signal.

```
hpeaks2 = signalblks.PeakFinder('PeakType', 'Maxima', ...
'PeakValuesOutputPort', true, ...
'MaximumPeakCount', 2, ...
'IgnoreSmallPeaks', true, ...
'PeakThreshold', 0.25, ...
'OverflowAction', 'Saturate');
x2 = fi([-1;0.5;0],true,16,15);
[cnt2, val2] = step(hpeaks2, x2);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Peak Finder block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.Maximum` | `signalblks.Minimum`

signalblks.PeakFinder.clone

Purpose Create peak finder object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a `PeakFinder` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

Purpose Return number of expected inputs to the step method

Syntax

```
getNumInputs(H)
```

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.PeakFinder.getNumOutputs

Purpose Return number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Return logical value to indicate whether input attributes and non-tunable properties are locked

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `PeakFinder` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.PeakFinder.step

Purpose

Number of extrema in input signal

Syntax

```
YCNT = step(H,X)
[YCNT,IDX] = step(H,X)
[..., VAL] = step(H,X)
[..., POL] = step(H,X)
```

Description

`YCNT = step(H,X)` returns the number of extrema (minima, maxima or both) `YCNT` in input signal `X`.

`[YCNT,IDX] = step(H,X)` returns the number of extrema `YCNT` and peak indices `IDX` in input signal `X` when the `PeakIndicesOutputPort` property is `true`.

`[..., VAL] = step(H,X)` also returns the peak values `VAL` in input signal `X` when the `PeakValuesOutputPort` property is `true`.

`[..., POL] = step(H,X)` also returns the extrema polarity `POL` in input signal `X` when the `PeakType` property is `Maxima and Minima` and the `PeakIndicesOutputPort` property is `true`. The polarity is `1` for maxima and `0` for minima.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Unwrap signal phase

Description

Unwraps the signal phase input specified in radians.

Construction

`H = signalblks.PhaseUnwrapper` returns a System object, `H`, that adds or subtracts appropriate multiples of 2π to each input element to remove phase discontinuities (unwrap).

`H = signalblks.PhaseUnwrapper('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an unwrapped System object, `H`, with each specified property set to the specified value.

Properties

`InterFrameUnwrap`

Enable unwrapping of phase discontinuities between successive frames.

Set this property to `false` to unwrap phase discontinuities only within the frame. Set this property to `true` to also unwrap phase discontinuities between successive frames. This property applies when you set the `FrameBasedProcessing` to `true`. The default value of this property is `true`.

`Tolerance`

Jump size as a true phase discontinuity

Specify the jump size that the phase unwrapper recognizes as a true phase discontinuity. The default value of this property is set to π (rather than a smaller value) to avoid altering legitimate signal features. To increase the phase wrapper sensitivity, set the `Tolerance` property to a value slightly less than π.

`FrameBasedProcessing`

Enable frame-based processing

Set this property to `true` to enable frame-based processing. Set this property to `false` to enable sample-based processing. The default value of this property is `true`.

signalblks.PhaseUnwrapper class

Methods

clone	Create phase unwrapper object with same property values
getNumInputs	Returns the number of expected inputs to the method
getNumOutputs	Returns the number of outputs of the method
isLocked	RLocked status (logical) for input attributes and non-tunable properties
reset	Reset the internal states of the Phase Unwrapper System object
step	Unwraps the input phase signal input.

Examples

Unwrap input phase data

```
hunwrap = signalblks.PhaseUnwrapper;
p = [0 2/5 4/5 -4/5 -2/5 0 2/5 4/5 -4/5 -2/5 0 2/5 ...
 4/5 -4/5, -2/5]*pi;
y = step(hunwrap, p');
figure,stem(p); hold
stem(y, 'r');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Unwrap block reference page. The object properties correspond to the Simulink block parameters.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See “What Are Sample- and Frame-Based Processing?” for more information.

See Also

`unwrap`

Purpose Create phase unwrapper object with same property values

Syntax `clone(H)`

Description `clone(H)` clones the current instance of the phase unwrapper object `H`.

signalblks.PhaseUnwrapper.getNumInputs

Purpose

Returns the number of expected inputs to the `step` method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Returns the number of outputs of the `step` method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.PhaseUnwrapper.isLocked

Purpose RLocked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the phase unwrapper object `H`.

Purpose Reset the internal states of the Phase Unwrapper System object

Syntax `reset(H)`

Description `reset(H)` resets the 2π multiplier `k` to `0` for the phase unwrapper object `H`. See "The Two Unwrap Modes" on page 2-1393 for details.

signalblks.PhaseUnwrapper.step

Purpose Unwraps the input phase signal input.

Syntax `Y = step(H,X)`

Description `Y = step(H,X)` unwraps the input signal `X` in radians for the Root-Mean-Square (RMS) object `H`.

Note The object performs an initialization, the first time the `step` method is executed. This initialization locks non-tunable properties, and input specifications, such as, dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.RCToAutocorrelation class

Purpose

Convert reflection coefficients to autocorrelation coefficients

Description

The `RCToAutocorrelation` object converts reflection coefficients to autocorrelation coefficients.

Construction

`H = signalblks.RCToAutocorrelation` returns an `RCToAutocorrelation` System object, `H`, that converts reflection coefficients to autocorrelation coefficients, assuming an error power of `1`.

`H = signalblks.RCToAutocorrelation('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an object, `H`, that converts reflection coefficients into autocorrelation coefficients, with each specified property set to the specified value.

Properties

`PredictionErrorInputPort`

Enable prediction error power input

Choose how to select the prediction error power. When you set this property to `true`, you must specify the prediction error power as a second input to the `step` method. When you set this property to `false`, the object assumes a prediction error power of `1`. The default value of this property is `false`.

Methods

clone	Create RC to autocorrelation object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Return number of outputs from step method

isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Converts columns of reflection coefficients to autocorrelation coefficients

Examples

Convert reflection coefficients to autocorrelation coefficients.

```
k = [-0.8091 0.2525 -0.5044 0.4295 -0.2804 0.0711].';
hrc2ac = signalblks.RCToAutocorrelation;
ac = step(hrc2ac, k);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the LPC/RC to Autocorrelation block reference page. The object properties correspond to the block parameters, except for:

- There is no object property that corresponds to the **Type of conversion** block parameter.
- **PredictionErrorInputPort** is a pop-up menu choice on the block. Setting the `PredictionErrorInputPort` object property to `false` corresponds to selecting `Assume P = 1` in the pop-up menu. Setting `PredictionErrorInputPort` to `true` corresponds to selecting `Via input port` from the pop-up menu.

See Also

`signalblks.LPCToLSF` | `signalblks.LPCToRC` | `signalblks.LPCToCepstral` | `signalblks.LPCToAutocorrelation`

Purpose

Create RC to autocorrelation object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates a `RCToAutocorrelation` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.RCToAutocorrelation.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.RCToAutocorrelation.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `RCToAutocorrelation` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Converts columns of reflection coefficients to autocorrelation coefficients

Syntax

```
AC = step(H,K)
AC = step(H,K,P)
```

Description

`AC = step(H,K)` converts the columns of the reflection coefficients, `K`, to autocorrelation coefficients, `AC`.

`AC = step(H,K,P)` converts the columns of the reflection coefficients, `K` , to autocorrelation coefficients, `AC`, using `P` as the prediction error power, when you set the `PredictionErrorInputPort` property to `true`. `P` must be a row vector with same number of columns as in `K`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.RCToLPC class

Purpose

Convert reflection coefficients to linear prediction coefficients

Description

The `RCToLPC` object converts reflection coefficients to linear prediction coefficients.

Construction

`H = signalblks.RCToLPC` returns an RC to LPC System object, `H`, that converts reflection coefficients (RC) to linear prediction coefficients (LPC).

`H = signalblks.RCToLPC('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an RC to LPC conversion object, `H`, with each specified property set to the specified value.

Properties

`PredictionErrorOutputPort`

Enable normalized prediction error power output

Set this property to `true` to return the normalized error power as a vector with one element per input channel. Each element varies between `0` and `1`. The default value of this property is `true`.

`ExceptionOutputPort`

Produces an output with the stability status of filter represented by LPC coefficients

Set this property to `true` to return the stability of the filter. The output is a vector of a length equal to the number of channels. A logical value of `1` indicates a stable filter. A logical value of `0` indicates an unstable filter. The default value of this property is `false`.

Methods

clone	Creates an instance of an object with the same property values
getNumInputs	Number of expected inputs to step method

getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Convert columns of reflection coefficients to linear prediction coefficients

Examples

Convert reflection coefficients to linear prediction coefficients.

```
hlevinson = signalblks.LevinsonSolver;
hac = signalblks.Autocorrelator;
hac.MaximumLagSource = 'Property';
hac.MaximumLag = 10; % Compute autocorrelation lags between [0:10]
hrc2lpc = signalblks.RCToLPC;
x = (1:100)';
a = step(hac, x);
k = step(hlevinson, a); % Compute reflection coefficients
[A, P] = step(hrc2lpc, k);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the LPC to/from RC block reference page. The object properties correspond to the block parameters.

- There is no object property that corresponds to the **Type of conversion** block parameter. The object always converts LPC to RC.

See Also

`signalblks.LPCToRC` | `signalblks.LPCToAutocorrelation`

Purpose Creates an instance of an object with the same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an `RCToLPC` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

Purpose

Number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.RCToLPC.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `RCToLPC` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.RCToLPC.step

Purpose

Convert columns of reflection coefficients to linear prediction coefficients

Syntax

```
[A,P] = step(H,K)
A = step(H,K)
[..., S] = step(H,K)
```

Description

`[A,P] = step(H,K)` converts the columns of the reflection coefficients, `K`, to linear prediction coefficients, `A`, and outputs the normalized prediction error power, `P`.

`A = step(H,K)` converts the columns of the reflection coefficients, `K`, to linear prediction coefficients, `A`, when the `PredictionErrorOutputPort` property is `false`.

`[..., S] = step(H,K)` also outputs the `LPC` filter stability, `S`, when the `ExceptionOutputPort` property is `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.RMS class

Purpose

Compute root-mean-square of the vector elements

Description

The `RMS` object computes the root-mean-square of the vector elements.

Construction

`H = signalblks.RMS` returns a System object, `H`, that computes the root-mean-square (RMS) in an input or a sequence of inputs.

`H = signalblks.RMS('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an `RMS` System object, `H`, with each specified property set to the specified value.

Properties

`RunningRMS`

Enable calculating RMS over time

Set this property to `true` to enable calculating the RMS over time. The default value of this property is `false`.

`ResetInputPort`

Enable resetting in running RMS mode

Set this property to `true` to enable resetting the running RMS. When the property is set to `true`, a reset input must be specified to the `step` method to reset the running RMS. This property applies when you set the `RunningRMS` property to `true`. The default value of this property is `false`.

`ResetCondition`

Reset condition for running RMS mode

Specify the event to reset the running RMS as one of `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero`. This property applies when you set the `ResetInputPort` property to `true`. The default value for this property is `Non-zero`.

`Dimension`

Dimension to operate along

Specify the dimension along which to calculate the RMS as one of `All` , `Row`, `Column`, or `Custom`. This property applies only when you set the `RunningRMS` property to `false`. The default value for this property is `Column`.

`CustomDimension`

Numerical dimension to operate along

Specify the dimension (one-based scalar integer value) of the input signal, along which the RMS is computed. The value of this property cannot exceed the number of dimensions in the input signal. This property applies when you set the `Dimension` property to `Custom`. The default value of this property is `1`.

`FrameBasedProcessing`

Enable frame-based processing

Set this property to `true` to enable frame-based processing for 2- dimensional inputs. Set this property to `false` to enable sample-based processing. The object always performs sample-based processing for *N*- dimensional inputs where *N* is greater than `2`. This property applies when you set the `RunningRMS` property to `true`. The default value of this property is `true`.

Methods

clone	Clones the current instance of the Root–Mean–Square object
getNumInputs	Returns the number of expected inputs to the method
getNumOutputs	Returns the number of outputs of the method
isLocked	Locked status (logical) for input attributes and non-tunable properties

reset	Reset the running Root–Mean–Square object
step	Computes the Root–Mean–Square of the input

Examples

Determine the RMS of a vector input

```
in1 = [1:10]';
hrms = signalblks.RMS;
y_rms1 = step(hrms, in1)
```

Determine the RMS of a matrix input

```
in2 = magic(4);
 hrms2d = signalblks.RMS;
 hrms2d.Dimension = 'All';
 y_rms2 = step(hrms2d, in2)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the RMS block reference page. The object properties correspond to the Simulink block parameters, except for:

- **Treat sample-based row input as a column** block parameter is not supported by the `signalblks.RMS` object.
- **Reset Port** block parameter corresponds to both the `ResetCondition` and the `ResetInputPort` object properties.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See "What Are Sample- and Frame-Based Processing?" for more information.

See Also

`signalblks.Mean` | `signalblks.StandardDeviation` | `signalblks.Variance`.

signalblks.RMS.clone

Purpose Clones the current instance of the Root–Mean–Square object

Syntax `clone(H)`

Description `clone(H)` clones the current instance of the Root–Mean–Square (RMS) object `H`

Purpose

Returns the number of expected inputs to the `step` method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method for the Root–Mean–Square (RMS) object `H`.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.RMS.getNumOutputs

Purpose

Returns the number of outputs of the `step` method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the Root–Mean–Square (RMS) object `H`.

signalblks.RMS.reset

Purpose Reset the running Root–Mean–Square object

Syntax `reset(H)`

Description `reset(H)` resets the running Root–Mean–Square (RMS) for the object `H`.

Purpose

Computes the Root–Mean–Square of the input

Syntax

```
Y = step(H,X)
Y = step(H,X,R)
```

Description

`Y = step(H,X)` computes the Root–Mean–Square (RMS) output, `Y` , of input vector `X` of the phase unwrapper object, `H`. When the `RunningRMS` property is true, `Y` corresponds to the `RMS` of the input elements over time.

`Y = step(H,X,R)` computes RMS of the input elements over time of input vector `X`, of the phase unwrapper object, `H`. The `step` method will optionally reset the running RMS state based on the value of `R`, the reset signal, and the `ResetCondition` property for the phase unwrapper, `H`. This is possible when both the `RunningRMS` and the `ResetInputPort` properties are set to `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.ScalarQuantizerDecoder class

Purpose

Convert each index value into quantized output value

Description

The `ScalarQuantizerDecoder` object converts each index value into a quantized output value. The set of all possible quantized output values or codewords is defined by the specified codebook. Input index values less than 0 are set to 0 and index values greater N-1 are set to N-1. N is the length of the codebook vector.

Construction

`H = signalblks.ScalarQuantizerDecoder` returns a scalar quantizer decoder System object, `H`, that transforms zero-based input index values into quantized output values.

`H = signalblks.ScalarQuantizerDecoder('PropertyName',PropertyValue,...)` returns a scalar quantizer decoder object, `H`, with each specified property set to the specified value.

Properties

`CodebookSource`

How to specify code book values

Specify how to determine the code book values as `Property` or `Input port`. The default value of this property is `Property`.

`Codebook`

Code book

Specify the code book as a vector of quantized output values that correspond to each index value. The default value of this property is `1:10`. This property is tunable.

`OutputDataType`

Data type of codebook and quantized output

Specify the data type of the codebook and quantized output values as `Same as input`, `double`, `single` or `Custom`. The default value of this property is `double`.

Fixed-Point Properties

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a signed or unsigned `numerictype` object. This property applies only when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype(true,16)`.

Methods

clone	Create scalar quantizer decoder object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Operate on inputs to calculate outputs

Examples

Given a codebook and index values as inputs, determine the corresponding output quantized values.

```
codebook = single([-2.1655 -1.3238 -0.7365 -0.2249 0.2726, ...
0.7844 1.3610 2.1599]);
indices = uint8([1 3 5 7 6 4 2 0]);
hsqdec = signalblks.ScalarQuantizerDecoder;
hsqdec.CodebookSource = 'Input port';
qout = step(hsqdec, indices, codebook);
```

signalblks.ScalarQuantizerDecoder class

Algorithm

This object implements the algorithm, inputs, and outputs described on the Scalar Quantizer Decoder block reference page. The object properties correspond to the block parameters, except for:

- There is no object property that directly corresponds to the **Action for out of range index value** block parameter. The object sets any index values less than `0` to `0` and any index values greater than or equal to *N* to *N*-1.

See Also

`signalblks.ScalarQuantizerEncoder` | `signalblks.VectorQuantizerDecoder`

Purpose

Create scalar quantizer decoder object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates a `ScalarQuantizerDecoder` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

signalblks.ScalarQuantizerDecoder.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.ScalarQuantizerDecoder.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `ScalarQuantizerDecoder` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Operate on inputs to calculate outputs

Syntax

```
Q = step(H, I)
Q = step(H,I,C)
```

Description

`Q = step(H, I)` returns the quantized output values `Q` corresponding to the input indices `I`. The data type of `I` can be `uint8`, `uint16`, `uint32`, `int8`, `int16` or `int32`. The `OutputDataType` property determines the data type for `Q`.

`Q = step(H,I,C)` uses input `C` as the codebook values when you set the `CodebookSource` property to `Input port`. The data type of `C` can be `double`, `single` or `fixed-point`. The output `Q` has the same data type as the codebook input `C`.

signalblks.ScalarQuantizerEncoder class

Purpose

Encode each input value by associating it with index value of quantization region

Description

The `ScalarQuantizerEncoder` object encodes each input value by associating it with the index value of the quantization region. Then, the object outputs the index of the associated region.

Construction

`H = signalblks.ScalarQuantizerEncoder` returns a scalar quantizer encoder System object, `H`, that maps each input value to a quantization region by comparing the input value to the user-specified boundary points.

`H = signalblks.ScalarQuantizerEncoder('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a scalar quantizer encoder object, `H`, with each specified property set to the specified value.

Properties

`BoundaryPointsSource`

Source of boundary points

Specify how to determine the boundary points and codebook values as `Property` or `Input port`. The default value of this property is `Property`.

`Partitioning`

Quantizer is bounded or unbounded

Specify the quantizer as `Bounded` or `Unbounded`. The default value of this property is `Bounded`.

`BoundaryPoints`

Boundary points of quantizer regions

Specify the boundary points of quantizer regions as a vector. The values of the vector must be in an ascending order. Let [p0 p1 p2 p3 ... pN] denote the boundary points property in quantizer. If the quantizer is bounded, the object uses this property to specify [p0 p1 p2 p3 ... pN]. If the quantizer is unbounded, the object uses this property to specify [p1 p2 p3 ... p(N-1)] and sets p0=-Inf and pN = +Inf. This property applies when you set the `BoundaryPointsSource` property to `Property`. The default value of this property is `1:10`. This property is tunable.

`SearchMethod`

Find quantizer index by linear or binary search

Specify whether to find the appropriate quantizer index using a linear search or a binary search as one of `Linear` or `Binary`. The computational cost of the linear search method is of the order P and the computational cost of the binary search method is of the order

$$\log_2(\mathrm{P})$$

where P is the number of boundary points. The default value of this property is `Linear`.

`TiebreakerRule`

Behavior when input is equal to boundary point

Specify whether the input value is assigned to the lower indexed region or higher indexed region when the input value is equal to boundary point by selecting `Choose the lower index` or `Choose the higher index`. The default value of this property is `Choose the lower index`.

`CodewordOutputPort`

Enable output of codeword value

Set this property to `true` to output the codeword values that correspond to each index value. The default value of this property is `false`.

`QuantizationErrorOutputPort`

Enable output of quantization error

Set this property to `true` to output the quantization error for each input value. The quantization error is the difference between the input value and the quantized output value. The default value of this property is `false`.

`Codebook`

Code book

Specify the code book as a vector of quantized output values that correspond to each region. If the `Partitioning` property is `Bounded` and the boundary points vector has length N, this property must be set to a vector of length N-1. If the `Partitioning` property is `Unbounded` and the boundary points vector has length N, this property must be set to a vector of length N+1. This property applies when you set the `BoundaryPointsSource` property to `Property` and either the `CodewordOutputPort` property or the `QuantizationErrorOutputPort` property is `true`. The default value of this property is `1.5:9.5`. This property is tunable.

`ClippingStatusOutputPort`

Enable output of clipping status

Set this property to `true` to output the clipping status. The output is a 1 when an input value is outside the range defined by the `BoundaryPoints` property. When the value is inside the range, the exception output is a 0. This property applies when you set the `Partitioning` property to `Bounded`. The default value of this property is `false`.

`OutputIndexDataType`

Data type of the index output

Specify the data type of the index output from the object as: `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`. The default value of this property is `int32`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest` or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

Methods

clone	Create an instance of an object with the same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Operate on inputs to calculate outputs

Examples

Quantize the varying fractional inputs between zero and five to the closest integers, and then plot the results.

```
hsqe = signalblks.ScalarQuantizerEncoder;
hsqe.BoundaryPoints = [-.001 .499 1.499 2.499 3.499 4.499 5.001];
hsqe.CodewordOutputPort = true;
hsqe.Codebook = [0 1 2 3 4 5];
input = (0:0.02:5)';
[index, quantizedValue] = step(hsqe, input);
plot(1:length(input), [input quantizedValue]);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Scalar Quantizer Encoder block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.ScalarQuantizerDecoder` | `signalblks.VectorQuantizerEncoder`

Purpose

Create an instance of an object with the same property values

Syntax

Description

`C = clone(H)` creates a `ScalarQuantizerEncoder` object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.ScalarQuantizerEncoder.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Return number of outputs from step method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.ScalarQuantizerEncoder.isLocked

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `ScalarQuantizerEncoder` object H.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Operate on inputs to calculate outputs

Syntax

```
INDEX= step(HSQE,INPUT)
[...] = step(HSQE,INPUT,BPOINTS)
[...] = step(HSQE,INPUT,BPOINTS,CODEBOOK)
[...,CODEWORD] = step(HSQE, ...)
[..., QERR] = step(HSQE, ...)
[...,CLIPSTATUS] = step(HSQE, ...)
```

Description

`INDEX= step(HSQE,INPUT)` returns the `INDEX` of the quantization region to which the `INPUT` belongs. The input data, boundary points, codebook values, quantized output values, and the quantization error must have the same data type whenever they are present.

`[...]  = step(HSQE,INPUT,BPOINTS)` uses input `BPOINTS` as the boundary points when the `BoundaryPointsSource` property is `Input port`.

`[...]  = step(HSQE,INPUT,BPOINTS,CODEBOOK)` uses input `BPOINTS` as the boundary points and input `CODEBOOK` as the code book when the `BoundaryPointsSource` property is `Input port` and either the `CodewordOutputPort` property or the `QuantizationErrorOutputPort` property is `true`.

`[...,CODEWORD] = step(HSQE, ...)` outputs the `CODEWORD` values that corresponds to each index value when the `CodewordOutputPort` property is `true`.

`[..., QERR] = step(HSQE, ...)` outputs the quantization error `QERR` for each input value when the `QuantizationErrorOutputPort` property is `true`.

`[...,CLIPSTATUS] = step(HSQE, ...)` also returns output `CLIPSTATUS` as the clipping status output port for each input value when the `Partitioning` property is `Bounded` and the `ClippingStatusOutputPort` property is `true`. If an input value is outside the range defined by the `BoundaryPoints` property, `CLIPSTATUS` will be `true`. If an input value is inside the range, `CLIPSTATUS` will be `false`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Log simulation data in buffer

Description

The `SignalLogger` object logs MATLAB simulation data. This object accepts any numeric data type.

Construction

`H = signalblks.SignalLogger` returns a signal logger, `H`, that logs 2-D input data in the object.

`H =`
`signalblks.SignalLogger('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a signal logger, `H`, with each specified property set to the specified value.

Properties

`FrameBasedProcessing`

Process input as frames or samples

Set this property to `true` to enable frame-based processing for 2-D inputs. Set this property to `false` to enable sample-based processing. The signal logger always performs sample-based processing for N-dimensional inputs, where N is greater than 2. The default value of this property is `true`.

`BufferLength`

Maximum number of input samples or frames to log

Specify maximum number of most recent samples of data to log when the input is sample based, or the maximum number of most recent frames of data to log when the input is frame based. To capture all data, set this property to `inf`. The default value of this property is `inf`.

`Decimation`

Decimation factor

Setting this property to any positive integer d causes the signal logger to write data at every dth sample. The default value of this property is `1`.

`FrameHandlingMode`

Output dimensionality for frame-based inputs

Set the dimension of the output array for frame-based inputs as `2-D array (concatenate)` or `3-D array (separate)`. Concatenation occurs along the first dimension for `2-D array (concatenate)`. This property applies only when you set the `FrameBasedProcessing` property to `true`. The default value of this property is `2-D array (concatenate)`.

`Buffer`

Logged Data (read only)

The signal logger writes simulation data into a buffer. Specify the maximum length of the buffer with the `BufferLength` property.

Methods

clone	Create signal logger object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Return logical value to indicate whether input attributes and non-tunable properties are locked
reset	Reset internal states of signal logger object
step	Store signal in buffer

Examples

Log input data:

```
hlog = signalblks.SignalLogger;
for i = 1:10
y = sin(i);
```

```
step(hlog, y);
end
log = hlog.Buffer; % log = sin([1;2;3;4;5;6;7;8;9;10]);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Signal To Workspace block reference page. The object properties correspond to the block properties, except for:

- The object always generates fixed-point output for fixed-point input.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See "What Are Sample- and Frame-Based Processing?" for more information.

See Also

`signalblks.SignalReader`

signalblks.SignalLogger.clone

Purpose Create signal logger object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates a `SignalLogger` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.SignalLogger.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Return logical value to indicate whether input attributes and non-tunable properties are locked

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `SignalLogger` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.SignalLogger.reset

Purpose Reset internal states of signal logger object

Syntax `reset(H)`

Description `reset(H)` sets the internal states of the `SignalLogger` object `H` to their initial values.

Purpose

Store signal in buffer

Syntax

`step(H,Y)`

Description

`step(H,Y)` buffers the signal `Y`. The buffer may be accessed at any time from the `Buffer` property of `H`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.SignalReader class

Purpose

Import variable from workspace

Description

The `SignalReader` object imports a variable from the MATLAB workspace.

Construction

`H = signalblks.SignalReader` returns a signal reader System object, `H`, that outputs the variable one sample or frame at a time.

`H = signalblks.SignalReader('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a signal reader object, `H`, with each specified property set to the specified value.

`H = signalblks.SignalReader(signal,spf,'`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a signal reader object, `H`, with the `Signal` property set to `signal`, the `SamplesPerFrame` property set to `spf`, and other specified properties set to the specified values.

Properties

`Signal`

Variable or expression containing the signal

Specify the name of the workspace variable from which to import the signal, or a valid expression specifying the signal. The default value of this property is `[1:10]`.

`SamplesPerFrame`

Number of samples per output frame

Specify the number of samples to buffer into each output frame. This property must be `1` when you specify a 3-D array in the `Signal` property. The default value of this property is `1`.

`SignalEndAction`

Action after final signal values are generated

Specify the output after all of the specified signal samples have been generated as one of `Set to zero`, `Hold final value`, or

`Cyclic repetition`. The default value of this property is `Set to zero`.

Methods

clone	Create signal reader object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isDone	Return end-of-file status for signal reader object
isLocked	Return logical value to indicate whether input attributes and non-tunable properties are locked
reset	Reset internal states of signal reader object
step	Read one sample or frame of signal

Examples

Create a signal reader to output one sample at a time.

```
hsr1 = signalblks.SignalReader;
hsr1.Signal = randn(1024, 1);
y1 = zeros(1024,1);
idx = 1;
while(~isDone(hsr1))
  y1(idx) = step(hsr1);
  idx = idx+1;
end
```

Create a signal reader to output vectors.

```
hsr2 = signalblks.SignalReader(randn(1024, 1), 128);
y2 = step(hsr2); % y2 is a 128-by-1 frame of samples
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Signal From Workspace block reference page. The object properties correspond to the block parameters, except for:

- The object does not have properties that correspond to the **Sample time** or **Warn when frame size does not evenly divide input length** block parameters.

See Also

`signalblks.SignalLogger`

Purpose
Create signal reader object with same property values

Syntax
```
C = clone(H)
```

Description
`C = clone(H)` creates a `SignalReader` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with un-initialized states.

signalblks.SignalReader.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Number of outputs from step method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.SignalReader.isDone

Purpose Return end-of-file status for signal reader object

Syntax `isDone(H)`

Description `isDone(H)` returns a logical value indicating whether or not the `SignalReader` object, `H`, has reached the end of the imported signal. If the `SignalEndAction` property is set to `Cyclic repetition`, this method will return `true` every time the reader reaches the end.

signalblks.SignalReader.isLocked

Purpose

Return logical value to indicate whether input attributes and non-tunable properties are locked

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `SignalReader` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.SignalReader.reset

Purpose

Reset internal states of signal reader object

Syntax

```
reset(H)
```

Description

`reset(H)` resets the signal reader object `H`, to start reading from the beginning of the imported signal.

Purpose

Read one sample or frame of signal

Syntax

`Y = step(H)`

Description

`Y = step(H)` outputs one sample or frame of data, `Y`, from the imported signal. The imported signal is the variable or expression you specify for the `Signal` property of the `SignalReader` System object `H`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.SineWave class

Purpose

Discrete-time sinusoid

Description

The `SineWave` object generates a discrete–time sinusoid. The sine wave object generates a real–valued sinusoid or a complex exponential. A real-valued discrete-time sinusoid is defined as:

$$y(n) = A\sin(2\pi fn + \phi)$$

where A is the amplitude, f is the frequency in Hz, and φ is the initial phase, or phase offset, in radians. A complex exponential is defined as:

$$y(n) = Ae^{j(2\pi fn+\phi)}$$

For both real and complex sinusoids, the amplitude, frequency, and phase offsets can be scalars or length-N vectors, where N is the desired number of channels in the output. When you specify at least one of these properties as a length-N vector, scalar values specified for the other properties are applied to each of the N channels.

Construction

`H = signalblks.SineWave` returns a sine wave object, `H`, that generates a real-valued sinusoid with an amplitude of 1, a frequency of 100 Hz, and a phase offset of 0. By default, the sine wave object only generates one sample.

`H = signalblks.SineWave('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns a sine wave object, `H`, with each property set to the specified value.

`H = signalblks.SineWave(AMP,FREQ,PHASE, '`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns a sine wave object, `H`, with the `Amplitude` property set to `AMP`, the `Frequency` property set to `FREQ`, the `PhaseOffset` property set to `PHASE`, and the other specified properties set to the specified values.

Properties

`Amplitude`

Amplitude of the sine wave

Specify the amplitude as a length-N vector containing the amplitudes of the sine waves in each of N output channels, or a scalar to be applied to all N channels. The vector length must be the same as that specified for the `Frequency` and `PhaseOffset` properties. It is tunable when the `Method` property is `Trigonometric function` or `Differential`. The default value of this property is 1.

`Frequency`

Frequency of the sine wave in Hz

Specify a length-N vector containing frequencies, in Hertz, of the sine waves in each of N output channels, or a scalar to be applied to all N channels. The vector length must be the same as that specified for the `Amplitude` and `PhaseOffset` properties. You can specify positive, zero, or negative frequencies. It is tunable when the `Method` property is `Trigonometric function`. The default value of this property is 100.

`PhaseOffset`

Phase offset of the sine wave in radians

A length-N vector containing the phase offsets, in radians, of the sine waves in each of N output channels, or a scalar to be applied to all N channels. The vector length must be the same as that specified for the `Amplitude` and `Frequency` properties. It is tunable when the `Method` property is `Trigonometric function`. The default value of this property is 0.

`ComplexOutput`

Indicates whether the sine wave is complex or real

Set to `true` to output a complex exponential. The default value of this property is `false`.

`Method`

Method used to generate sinusoids

The sinusoids are generated by either the `Trigonometric function` , `Table lookup`, or `Differential` methods . The `trigonometric function` method computes the sinusoid by sampling the continuous–time sinusoid. The `table lookup` method precomputes the unique samples of every output sinusoid at the start of the simulation, and recalls the samples from memory as needed. The `differential` method uses an incremental algorithm. This algorithm computes the output samples based on the output values computed at the previous sample time and precomputed update terms. The default value for this property is `Trigonometric function`.

`TableOptimization`

Optimizes the table of sine values for speed or memory

Optimizes the table of sine values for `Speed` or `Memory`. When optimized for speed, the table contains k elements, and when optimized for memory, the table contains $k/4$ elements, where k is the number of input samples in one full period of the sine wave. This property applies only when the `Method` property is `Table lookup`. The default value for this property is `Speed`.

`SampleTime`

The sampling period for the sine wave

The default value of this property is 1/1000.

`SamplesPerFrame`

Number of samples per frame

Specify the number of consecutive samples from each sinusoid to buffer into the output frame. The default value of this property is 1.

`OutputDataType`

Output data type

Specify the output data type as `double`, `single`, or `Custom`. The default value for this property is `double`.

Fixed-Point Properties

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a `numerictype` object with a `Signedness` of `Auto`. This property applies only when the `OutputDataType` property is `Custom`. The default value of this property is `numerictype([]1,16)`.

Methods

clone	Create sine wave object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Return number of outputs of step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset sine wave to the beginning
step	Compute discrete-time sine wave

Examples

Generate a sine wave with an amplitude of 2, frequency of 10 Hz, and initial phase of 0.

```
hsin1 = signalblks.SineWave(2, 10);
hsin1.SamplesPerFrame = 1000;
y = step(hsin1);
plot(y);
```

Generate two sine waves offset by a phase of π/2 radians:

```
hsin2 = signalblks.SineWave;
hsin2.Frequency = 10;
hsin2.PhaseOffset = [0 pi/2];
hsin2.SamplesPerFrame = 1000;
y = step(hsin2);
plot(y);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Sine Wave block reference page. The object properties correspond to the block parameters.

See Also

signalblks.Chirp | signalblks.NCO

Purpose Create sine wave object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates a sine wave object, `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

signalblks.SineWave.getNumInputs

Purpose

Number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.SineWave.getNumOutputs

Purpose Return number of outputs of step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.SineWave.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the sine wave object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset sine wave to the beginning

Syntax `reset(H)`

Description `reset(H)` resets the sine wave object to the beginning (*n=0*). If you invoke the `step` method without first invoking the `reset` method, the sine wave object generates additional samples of the original sinusoid. As an example:

```
H = signalblks.SineWave('Amplitude',1,'Frequency',1/4,...,
'SampleTime',1,'SamplesPerFrame',7);
% 7 samples of a sine wave with frequency of 1/4 Hz
y = step(H);
% Call step method without reset
y1 = step(H);
stem([y; y1])
% y1 is a continuation of y
% Now reset
reset(H);
y2 = step(H);
% y2 starts at n=0 and is equal to y
isequal(y,y2)
```

signalblks.SineWave.step

Purpose Compute discrete-time sine wave

Syntax `Y = step(H)`

Description `Y = step(H)` produces the sine wave `Y` using the specifications in the object `H`.

Purpose

Compute standard deviation of input or sequence of inputs

Description

The `StandardDeviation` object computes the standard deviation for an input or sequence of inputs.

Construction

`H = signalblks.StandardDeviation` returns a standard deviation System object, `H`, that computes the standard deviation for the columns of input.

`H = signalblks.StandardDeviation('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a standard deviation System object, `H`, with each specified property set to the specified value.

Properties

`RunningStandardDeviation`

Enable calculation over time

Set this property to `true` to enable the calculation of standard deviation over time. The default value of this property is `false`.

`ResetInputPort`

Enable resetting in running standard deviation mode

Set this property to `true` to enable resetting for the running standard deviation. When the property is set to `true`, you must specify a reset input to the `step` method to reset the running standard deviation. This property applies only when you set the `RunningStandardDeviation` property to `true`. The default value of this property is `false`.

`ResetCondition`

Reset condition for running standard deviation mode

Specify event to reset the running standard deviation as `Rising edge`, `Falling edge`, `Either edge`, `Non-zero`. This property applies only when you set the `ResetInputPort` property to `true`. The default value of this property is `Non-zero`.

`Dimension`

Dimension to operate along

Specify how the standard deviation calculation is performed over the data: `All`, `Row`, `Column` or `Custom`. This property applies only when you set the `RunningStandardDeviation` property to `false`. The default value of this property is `Column`.

`CustomDimension`

Numerical dimension to operate along

Specify the dimension (one-based value) of the input signal, over which the object computes the standard deviation. The value of this property cannot exceed the number of dimensions for the input signal. This property applies when you set the `Dimension` property to `Custom`. The default value of this property is `1`.

`ROIProcessing`

ROIProcessing

Enable region of interest processing

Set this property to `true` to enable calculating the standard deviation within a particular region for each image. This property applies only when you set the `RunningStandardDeviation` property to `false` and the `Dimension` property is `All`. The default value of this property is `false`.

Full ROI processing support requires a Video and Image Processing Blockset license. With only the Signal Processing Blockset license, `Rectangles` is the only selection that applies for the `ROIForm` property.

`ROIForm`

Type of region of interest

Specify the type of region of interest as `Rectangles`, `Lines`, `Label matrix` or `Binary mask`. This property applies only when you set the `ROIProcessing` property to `true`. The default value of this property is `Rectangles`.

Full ROI processing support requires a Video and Image Processing Blockset license. With only the Signal Processing Blockset license, `Rectangles` is the only selection that applies for the `ROIForm` property.

`ROIPortion`

Calculate over entire ROI or just perimeter

Specify the region over which to calculate the standard deviation as Entire ROI or ROI perimeter. This property applies if the `ROIForm` property is `Rectangles`. The default value of this property is `Entire ROI`.

`ROIStatistics`

Statistics for each ROI, or one for all ROIs

Specify what statistics to calculate: `Individual statistics for each ROI` or `Single statistic for all ROIs`. This property applies when you set the `ROIForm` property to `Rectangles`, `Lines`, or `Label matrix`. The default value of this property is `Individual statistics for each ROI`.

`ValidityOutputPort`

Enable output of validity check of ROI or label numbers

Indicate whether to return the validity of the specified ROI being completely inside image when the `ROIForm` property is `Lines`, `Rectangles` or `Binary mask`. Indicate whether to return the validity of the specified label numbers when the `ROIForm` property is `Label Matrix`. This property applies when you set the `ROIForm` property to anything except `Binary Mask`. The default value of this property is `false`.

`FrameBasedProcessing`

Enable frame-based processing

Set this property to `true` to enable frame-based processing. Set this property to `false` to enable sample-based processing. This property applies only when you set the

`RunningStandardDeviation` property to `true`. The default value of this property is `true`.

Methods

clone	Create standard deviation object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset states for running standard deviation computation
step	Calculate standard deviation of input

Examples

Compute running standard deviation of a signal.

```
hstd2 = signalblks.StandardDeviation;
hstd2.RunningStandardDeviation = true;
x = randn(100,1);
y = step(hstd2,x); % Find running standard deviation
% y(i) is the standard deviation of all values in the vector x(1:i)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Standard Deviation block reference page. The object properties correspond to the block parameters, except for:

- The **Reset port** block parameter corresponds to the `ResetInputPort` and `ResetCondition` object properties.
- **Treat sample-based row input as a column** block parameter is not supported by the `signalblks.StandardDeviation` object.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See "What Are Sample- and Frame-Based Processing?" for more information.

See Also

`signalblks.Variance` | `signalblks.Maximum` | `signalblks.Minimum` | `signalblks.Mean`

signalblks.StandardDeviation.clone

Purpose Create standard deviation object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a `StandardDeviation` object `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

Purpose Return number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.StandardDeviation.getNumOutputs

Purpose Return number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `StandardDeviation` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.StandardDeviation.reset

Purpose Reset states for running standard deviation computation

Syntax `reset(H)`

Description `reset(H)` sets the internal states of the `StandardDeviation` object `H` to their initial values when computing the running standard deviation.

Purpose

Calculate standard deviation of input

Syntax

```
Y = step(H,X)
Y = step(H,X,R)
Y = step(H,X,ROI)
Y = step(H,X,LABEL,LABELNUMBERS)
[Y, FLAG] = step(H,X,ROI)
[Y, FLAG] = step(H,X,LABEL,LABELNUMBERS)
```

Description

`Y = step(H,X)` computes the standard deviation of input `X` . It computes the standard deviation of the input elements over time, `Y` , when the `RunningStandardDeviation` property is `true`.

`Y = step(H,X,R)` computes the standard deviation of the input elements over time, `Y` , and optionally resets its state based on the value of reset signal `R` and the `ResetCondition` property. This option is available when the `RunningStandardDeviation` property is `true`.

`Y = step(H,X,ROI)` uses additional input `ROI` as the region of interest when the `ROIProcessing` property is set to `true` and the `ROIForm` property is set to `Lines`, `Rectangles` or `Binary mask`.

`Y = step(H,X,LABEL,LABELNUMBERS)` computes the standard deviation of input image `X` for region labels contained in vector `LABELNUMBERS` , with matrix `LABEL` marking pixels of different regions. This option is available when the `ROIProcessing` property is set to `true` and the `ROIForm` property is set to `Label matrix`.

`[Y, FLAG] = step(H,X,ROI)` also returns `FLAG` which indicates whether the given region of interest is within the image bounds when the `ValidityOutputPort` property is `true`.

`[Y, FLAG] = step(H,X,LABEL,LABELNUMBERS)` also returns `FLAG` which indicates whether the input label numbers are valid when the `ValidityOutputPort` property is `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

signalblks.SubbandAnalysisFilter class

Purpose

Decompose signal into high-frequency and low-frequency subbands

Description

The `SubbandAnalysisFilter` object decomposes signal into high-frequency and low-frequency subbands.

Construction

`H = signalblks.SubbandAnalysisFilter` returns a two-channel subband analysis filter, `H`, that decomposes the input signal into a high-frequency subband and a low-frequency subband, each with half the bandwidth of the input.

`H = signalblks.SubbandAnalysisFilter('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a two-channel subband analysis filter, `H`, with each specified property set to the specified value.

`H = signalblks.SubbandAnalysisFilter(lpc,hpc,'`*`PropertyName`*`',`*`PropertyValue`*`,` returns a two-channel subband analysis filter, `H`, with the `LowpassCoefficients` property set to `lpc`, the `HighpassCoefficients` property set to `hpc`, and other specified properties set to the specified values.

Properties

`LowpassCoefficients`

Lowpass FIR filter coefficients

Specify a vector of lowpass FIR filter coefficients, in descending powers of z. The lowpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the `HighpassCoefficients` property. The default values of this property specify a filter based on a third-order Daubechies wavelet.

`HighpassCoefficients`

Highpass FIR filter coefficient

Specify a vector of highpass FIR filter coefficients, in descending powers of z. The highpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the `LowpassCoefficients` property. The default values of this property specify a filter based on a third-order Daubechies wavelet.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Action to take when integer input is out of range

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`CoefficientsDataType`

Data type of the coefficients

Specify the FIR filter coefficients fixed-point data type as `Same word length as input` or `Custom`. The default value of this property is `Same word length as input`.

`CustomCoefficientsDataType`

Coefficients word and fraction lengths

Specify the FIR filter coefficients fixed-point type as a `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `CoefficientsDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

`ProductDataType`

Data type of product

Specify the product data type as `Internal rule`, `Same as input`, or `Custom`. The default value of this property is `Internal rule`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `ProductDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`AccumulatorDataType`

Data type of accumulator

Specify the accumulator data type as `Internal rule`, `Same as input`, `Same as product`, or `Custom`. The default value of this property is `Internal rule`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`OutputDataType`

Data type of output

Specify the output data type as `Same as accumulator`, `Same as product`, `Same as input`, or `Custom`. The default value of this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,14)`.

Methods

clone	Create subband analysis filter object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Return logical value to indicate whether input attributes and non-tunable properties are locked
reset	Reset filter states of subband analysis filter object
step	Decompose signal into high- and low-frequency subbands

Examples

Perfectly reconstruct a signal using subband analysis and synthesis filters:

```
load dspwlets; % load the filter coefficients lod, hid, lor and hir
ha = signalblks.SubbandAnalysisFilter(lod, hid);
hs = signalblks.SubbandSynthesisFilter(lor, hir);
u = randn(128,1);
[hi, lo] = step(ha, u); % Two channel analysis
y = step(hs, hi, lo); % Two channel synthesis
% Plot difference between original and reconstructed signals with
% filter latency compensated
plot(u(1:end-7)-y(8:end));
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Two-Channel Analysis Subband Filter block reference page. The object properties correspond to the block parameters, except for:

- The `SubbandAnalysisFilter` object does not have a property that corresponds to the **Framing** parameter of the Two-Channel Analysis Subband Filter block. The object assumes the input is frame based and always maintains the input frame rate.

See Also

`signalblks.SubbandSynthesisFilter` | `signalblks.DyadicAnalysisFilterBank`

signalblks.SubbandAnalysisFilter.clone

Purpose Create subband analysis filter object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` create a `SubbandAnalysisFilter` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

Purpose

Return number of expected inputs to step method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.SubbandAnalysisFilter.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outpust from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Return logical value to indicate whether input attributes and non-tunable properties are locked

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `SubbandAnalysisFilter` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.SubbandAnalysisFilter.reset

Purpose Reset filter states of subband analysis filter object

Syntax

```
reset(H)
```

Description `reset(H)` sets the filter states of the `SubbandAnalysisFilter` object `H` to `0`.

Purpose

Decompose signal into high- and low-frequency subbands

Syntax

`[HI,LO] = step(H,SIG)`

Description

`[HI,LO] = step(H,SIG)` decomposes the input signal, `SIG`, into a high-frequency subband, `HI`, and a low-frequency subband, `LO`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.SubbandSynthesisFilter class

Purpose Reconstruct signal from high-frequency and low-frequency subbands

Description The `SubbandSynthesisFilter` object reconstructs a signal from high-frequency and low-frequency subbands.

Construction `H = signalblks.SubbandSynthesisFilter` returns a two-channel subband synthesis filter, `H`, that reconstructs a signal from its high-frequency subband and low-frequency subband, each with half the bandwidth of the original signal.

`H = signalblks.SubbandSynthesisFilter('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a two-channel subband synthesis filter, `H`, with each specified property set to the specified value.

`H = signalblks.SubbandSynthesisFilter(lpc,hpc,'`*`PropertyName`*`',`*`PropertyValue`*`,..` returns a two-channel subband synthesis filter, `H`, with the `LowpassCoefficients` property set to `lpc`, the `HighpassCoefficients` property set to `hpc`, and other specified properties set to the specified values.

Properties `LowpassCoefficients`

Lowpass FIR filter coefficients

Specify a vector of lowpass FIR filter coefficients, in descending powers of z. The lowpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the `HighpassCoefficients` property. The default values of this property specify a filter based on a third-order Daubechies wavelet.

`HighpassCoefficients`

Highpass FIR filter coefficients

Specify a vector of highpass FIR filter coefficients, in descending powers of z. The highpass filter should be a half-band filter that passes the frequency band stopped by the filter specified in the `LowpassCoefficients` property. The default values of this property specify a filter based on a third-order Daubechies wavelet.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Action to take when integer input is out of range

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`CoefficientsDataType`

Data type of the coefficients

Specify the FIR filter coefficients fixed-point data type as `Same word length as input` or `Custom`. The default value of this property is `Same word length as input`.

`CustomCoefficientsDataType`

Coefficient word and fraction lengths

Specify the FIR filter coefficients fixed-point type as a `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `CoefficientsDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

`ProductDataType`

Data type of product

Specify the product data type as `Internal rule`, `Same as input`, or `Custom`. The default value of this property is `Internal rule`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `ProductDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`AccumulatorDataType`

Data type of accumulator

Specify the accumulator data type as `Internal rule`, `Same as input`, `Same as product`, or `Custom`. The default value of this property is `Internal rule`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`OutputDataType`

Data type of output

Specify the output data type as `Same as accumulator` , `Same as product`, `Same as input`, or `Custom`. The default value of this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,14)`.

Methods

clone	Create subband synthesis filter object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset internal states of subband synthesis filter object
step	Reconstruct signal from high- and low-frequency subbands

Examples

Perfectly reconstruct a signal using subband analysis and synthesis filters:

```
% load the filter coefficients lod, hid, lor and hir
load dspwlets;
ha = signalblks.SubbandAnalysisFilter(lod, hid);
hs = signalblks.SubbandSynthesisFilter(lor, hir);
u = randn(128,1);
[hi, lo] = step(ha, u); % Two channel analysis
y = step(hs, hi, lo); % Two channel synthesis
% Plot difference between original and reconstructed signals with
% filter latency compensated
plot(u(1:end-7)-y(8:end));
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Two-Channel Synthesis Subband Filter block reference page. The object properties correspond to the block parameters, except for:

- The object supports only sample-based processing.

See Also

`signalblks.SubbandAnalysisFilter` | `signalblks.DyadicSynthesisFilterBank`

Purpose
Create subband synthesis filter object with same property values

Syntax

```
C = clone(H)
```

Description
`C = clone(H)` creates a `SubbandSynthesisFilter` object `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

signalblks.SubbandSynthesisFilter.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.SubbandSynthesisFilter.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `SubbandSynthesisFilter` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset internal states of subband synthesis filter object

Syntax

```
reset(H)
```

Description `reset(H)` sets the internal states of the `SubbandSynthesisFilter` object `H` to their initial values.

signalblks.SubbandSynthesisFilter.step

Purpose Reconstruct signal from high- and low-frequency subbands

Syntax `Y = step(H,HI,LO)`

Description `Y = step(H,HI,LO)` reconstructs a signal from a high-frequency subband, `HI`, and a low-frequency subband, `LO`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose Decode integer input into floating-point output

Description The `UniformDecoder` object decodes integer input into floating-point output. The decoder adheres to the definition for uniform decoding specified in ITU-T Recommendation G.701.

Construction `H = signalblks.UniformDecoder` returns a uniform decoder, `H`, that performs the inverse operation of the `UniformEncoder` object, reconstructing quantized floating-point values from encoded integer input.

`H = signalblks.UniformDecoder('PropertyName',PropertyValue,...)` returns a uniform decoder, `H`, with each specified property set to the specified value.

`H = signalblks.UniformDecoder(peakvalue,numbits,'PropertyName',PropertyVal` returns a uniform decoder, `H`, with the `PeakValue` property set to `peakvalue`, the `NumBits` property set to `numbits`, and other specified properties set to the specified values.

Properties `PeakValue`

Largest amplitude represented in encoded input

Specify the largest amplitude represented in the encoded input as a non-negative numeric scalar. To correctly decode values encoded with the `UniformEncoder` object, set the `PeakValue` property in both objects to the same value. For more information on setting this property, see the `PeakValue` property description on the `signalblks.UniformEncoder` reference page. The default value of this property is `1`.

`NumBits`

Number of input bits used to encode data

Specify the number of bits used to encode the input data as an integer value between `2` and `32`. The value of this property can be less than the total number of bits supplied by the input data type. To correctly decode values encoded with the `UniformEncoder` object, set the `NumBits` property in both objects to the same value. For more information on setting this property, see the `NumBits` property description on the `signalblks.UniformEncoder` reference page. The default value of this property is `3`.

`OverflowAction`

Action to take when integer input is out of range

Specify the behavior of the uniform decoder when the integer input is out of range as `Saturate` or `Wrap`. The value of the `NumBits` property specifies the representable range of the input. The default value of this property is `Saturate`.

`OutputDataType`

Output data type as single or double

Specify the data type of the output as `single` or `double`. The default value of this property is `double`.

Methods

clone	Create uniform decoder object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Return number of outputs from step method
isLocked	Return logical value to indicate whether input attributes and non-tunable properties are locked
step	Decode integer input into quantized floating-point output

Examples

Decode an encoded sequence:

```
hue = signalblks.UniformEncoder;
hue.PeakValue = 2;
hue.NumBits = 4;
hue.OutputDataType = 'Signed integer';
x = (0:0.25:2)'; % Create an input sequence
hud = signalblks.UniformDecoder;
hud.PeakValue = 2;
hud.NumBits = 4;
x_encoded = step(hue, x);
% Check that the last element has been saturated.
x_decoded = step(hud, x_encoded);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Uniform Decoder block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.UniformEncoder`

signalblks.UniformDecoder.clone

Purpose Create uniform decoder object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a `UniformDecoder` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object.

The clone method creates an instance of an object. The property values, but not internal states, are copied into the new instance of the object.

Purpose

Return number of expected inputs to step method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.UniformDecoder.getNumOutputs

Purpose Return number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Return logical value to indicate whether input attributes and non-tunable properties are locked

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `UniformDecoder` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.UniformDecoder.step

Purpose Decode integer input into quantized floating-point output

Syntax `Y = step(H,X)`

Description `Y = step(H,X)` reconstructs quantized floating-point output `Y` from the encoded integer input `X`. Input `X` can be real or complex values of the following six integer data types: `uint8`, `uint16`, `uint32`, `int8`, `int16`, or `int32`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Quantize and encode floating-point input into integer output

Description

The `UniformEncoder` object quantizes floating-point input, using the precision you specify in the `NumBits` property, and encodes the quantized input into integer output. The operations of the uniform encoder adhere to the definition for uniform encoding specified in ITU-T Recommendation G.701.

Construction

`H = signalblks.UniformEncoder` returns a uniform encoder, `H`, that quantizes floating-point input samples and encodes them as integers using $2N$-level quantization, where N is an integer.

`H = signalblks.UniformEncoder('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns an uniform encoder, `H`, with each specified property set to the specified value.

`H = signalblks.UniformEncoder(peakvalue,numbits,'`*`PropertyName`*`',`*`PropertyVal`* returns a uniform encoder, `H`, with the `PeakValue` property set to `peakvalue`, the `NumBits` property set to `numbits`, and other specified properties set to the specified values.

Properties

`PeakValue`

Largest input amplitude to be encoded

Specify the largest input amplitude to be encoded, as a non-negative numeric scalar. If the real or imaginary input are outside of the interval $[-P,(1-2^{(1-B)})P]$, where P is the peak value and B is the value of the `NumBits` property, the uniform encoder saturates (independently for complex inputs) at those limits. The default value of this property is `1`.

`NumBits`

Number of bits needed to represent output

Specify the number of bits needed to represent the integer output as an integer value between `2` and `32`. The number of levels at which the uniform encoder quantizes the floating-point input is 2^B, where B is the number of bits. The default value of this property is `8`.

`OutputDataType`

Data type of output

Specify the data type of the output as `Unsigned integer` or `Signed integer`. Unsigned outputs are `uint8`, `uint16`, or `uint32`, and signed outputs are `int8`, `int16`, or `int32`. The quantized inputs are linearly (uniformly) mapped to the intermediate integers in the interval $[0,2^{(B-1)}]$ when you set this property to `Unsigned integer`, and in the interval $[-2^{(B-1)},2^{(B-1)}-1]$ when you set this property to `Signed integer`. The variable B in both expressions corresponds to the value of the `NumBits` property.

Methods

clone	Create uniform encoder object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Quantize and encode input

Examples

Encode a sequence:

```
hue = signalblks.UniformEncoder;
hue.PeakValue = 2;
hue.NumBits = 4;
hue.OutputDataType = 'Signed integer';
x = [-1:0.01:1]'; % Create an input sequence
```

```
x_encoded = step(hue, x);
plot(x, x_encoded,'.');
xlabel('Input'); ylabel('Encoded Output'); grid
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Uniform Encoder block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.UniformDecoder`

signalblks.UniformEncoder.clone

Purpose Create uniform encoder object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a `UniformEncoder` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object.

signalblks.UniformEncoder.getNumInputs

Purpose

Number of expected inputs to step method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.UniformEncoder.getNumOutputs

Purpose

Number of outputs from step method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `UniformEncoder` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.UniformEncoder.step

Purpose

Quantize and encode input

Syntax

`Y = step(H,X)`

Description

`Y = step(H,X)` quantizes and encodes the input `X` to `Y`. Input `X` can be real or complex, double- or single-precision. The uniform encoder chooses the output data type according to the following table.

Number of Bits	Unsigned Integer	Signed Integer
2 to 8	`uint8`	`int8`
9 to 16	`uint16`	`int16`
17 to 32	`uint32`	`int32`

The row corresponds to the value of the `NumBits` property, and the column corresponds to the value of the `OutputDataType` property.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Purpose

Solve upper-triangular matrix equation

Description

The `UpperTriangularSolver` object solves $\mathbf{UX} = \mathbf{B}$ for $\mathbf{X}$ when $\mathbf{U}$ is a square, upper-triangular matrix with the same number of rows as $\mathbf{B}$.

Construction

`H = signalblks.UpperTriangularSolver` returns a linear system solver, `H`, used to solve $\mathbf{UX} = \mathbf{B}$ where $\mathbf{U}$ is an upper (or unit-upper) triangular matrix.

`H = signalblks.UpperTriangularSolver('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a linear system solver, `H`, with each specified property set to the specified value.

Properties

`OverwriteDiagonal`

Replace diagonal elements of input with ones

When you set this property to `true`, the linear system solver replaces the elements on the diagonal of the input, $\mathbf{U}$, with ones. Set this property to either `true` or `false`. The default value of this property is `false`.

`RealDiagonalElements`

Indicate that diagonal of complex input is real

When you set this property to `true`, the linear system solver optimizes computation speed if the diagonal elements of complex input, $\mathbf{U}$, are real. This property applies only when you set the `OverwriteDiagonal` property to `false`. Set this property to either `true` or `false`. The default value of this property is `false`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`ProductDataType`

Data type of product

Specify the product data type as `Internal rule`, `Same as input`, or `Custom`. The default value of this property is `Internal rule`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `ProductDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`AccumulatorDataType`

Data type of accumulator

Specify the accumulator data type as `Internal rule`, `Same as first input`, `Same as product`, or `Custom`. The default value of this property is `Internal rule`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,30)`.

`OutputDataType`

Data type of output

Specify the output data type as `Same as first input` or `Custom`. The default value of this property is `Same as first input`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create upper triangular solver object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Return logical value to indicate whether input attributes and non-tunable properties are locked
step	Solve matrix equation for specified inputs

Examples

Solve an upper-triangular matrix equation:

```
huptriang = signalblks.UpperTriangularSolver;
u = triu(rand(4, 4));
b = rand(4, 1);
% Check that result is the solution to the linear
% equations.
x1 = inv(u)*b
x = step(huptriang, u, b)
```

signalblks.UpperTriangularSolver class

Algorithm

This object implements the algorithm, inputs, and outputs described on the Backward Substitution block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.LowerTriangularSolver`

Purpose Create upper triangular solver object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an `UpperTriangularSolver` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object.

signalblks.UpperTriangularSolver.getNumInputs

Purpose

Number of expected inputs to step method

Syntax

`getNumInputs(H)`

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Number of outputs from step method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.UpperTriangularSolver.isLocked

Purpose

Return logical value to indicate whether input attributes and non-tunable properties are locked

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `UpperTriangularSolver` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Solve matrix equation for specified inputs

Syntax

`X = step(H,U,B)`

Description

`X = step(H,U,B)` computes the solution, $\mathbf{X}$, of the matrix equation $\mathbf{UX} = \mathbf{B}$, where $\mathbf{U}$ is a square, upper-triangular matrix with the same number of rows as the matrix $\mathbf{B}$.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.VariableFractionalDelay class

Purpose Delay input by time-varying fractional number of sample periods

Description The `VariableFractionalDelay` object delays the input by a time-varying fractional number of sample periods.

Construction `H = signalblks.VariableFractionalDelay` returns a variable fractional delay System object, `H`, that delays a discrete-time input by a time-varying fractional number of sample periods.

`H = signalblks.VariableFractionalDelay('PropertyName',PropertyValue,...)` returns a variable fractional delay System object, `H`, with each property set to the specified value.

Properties

`InterpolationMethod`

Interpolation method

Specify the method by which the block interpolates between adjacent stored samples to obtain a value for the sample indexed by the input. You can set this property to `Linear`, `FIR`, or `Farrow`. When you set this property to `FIR`, the object uses the Signal Processing Toolbox `intfilt` function to compute an FIR filter for interpolation. The default value of this property is `Linear`.

`FilterHalfLength`

FIR interpolation filter half-length

Specify the half-length of the FIR interpolation filter as a positive scalar integer. This property applies only when you set the `InterpolationMethod` property to `FIR`. For periodic signals, a larger value of this property (that is, a higher order filter) yields a better estimate of the delayed output sample. Setting this property to a value between `4` and `6` (that is, a 7th to 11th order filter) is usually adequate. The default value of this property is `4`.

`FilterLength`

Length of Farrow filter

Specify the length of the FIR filter implemented using the Farrow structure, as a positive scalar integer. This property applies only when you set the `InterpolationMethod` property to `Farrow`. The default value of this property is `4`.

`InterpolationPointsPerSample`

Number of interpolation points per input sample

Specify the number of interpolation points per input sample at which a unique FIR interpolation filter is computed. You must specify the number of interpolation points per input sample as a positive scalar integer. This property applies only when you set the `InterpolationMethod` property to `FIR`. The default value of this property is `10`.

`Bandwidth`

Normalized input bandwidth

Specify the bandwidth to which the interpolated output samples should be constrained. You must enter the bandwidth as a scalar value between `0` and `1`. You can use this property to take advantage of the bandlimited frequency content of the input. For example, if the input signal does not have frequency content above `Fs/4` (where `Fs` is the sampling frequency), you can specify a value of `0.5` for the `Bandwidth` property. A value of `1` for the `Bandwidth` property corresponds to half the sampling frequency (`Fs`). This property applies only when you set the `InterpolationMethod` property to `FIR`. The default value of this property is `1`.

`InitialConditions`

Initial values in memory

Specify the values with which the object's memory is initialized. The dimensions of this property can vary depending on the setting of the `FrameBasedProcessing` property, and whether you want fixed or time-varying initial conditions. The default value of this property is `0`.

When you set the FrameBasedProcessing property to false, the object supports N-dimensional input arrays. For an M-by-N-by-P sample-based input array U, you can set the InitialConditions property as follows:

- To specify fixed initial conditions, set the InitialConditions property to a scalar value. The object initializes every sample of every channel in memory using the value you specify.
- The dimensions you specify for time-varying initial conditions depend on the value of the InterpolationMethod property:
  - If you set the InterpolationMethod property to Linear, set the InitialConditions property to an array of dimension M-by-N-by-P-by-D. The object uses the values in this array to initialize memory samples U(2:D+1), where D is the value of the MaximumDelay property.
  - If you set the InterpolationMethod property to FIR or Farrow, set the InitialConditions property to an array of dimension M-by-N-by-P-by-(D+L). The object uses the values in this array to initialize memory samples U(2:D+1), where D is the value of the MaximumDelay property. For FIR interpolation, L is the value of the FilterHalfLength property. For Farrow interpolation, L is equal to floor of half the value of the FilterLength property (floor(FilterLength/2)).

When you set the FrameBasedProcessing property to true, the object treats each of the N input columns as a frame containing M sequential time samples from an independent channel. For an M-by-N frame-based input matrix U, you can set the InitialConditions property as follows:

- To specify fixed initial conditions, set the InitialConditions property to a scalar value. The object initializes every sample of every channel in memory using the value you specify.

- The dimensions you specify for time-varying initial conditions depend on the value of the `InterpolationMethod` property. To specify different time-varying initial conditions for each channel, set the `InitialConditions` property as follows:
  - If you set the `InterpolationMethod` property to `Linear`, set the `InitialConditions` property to an array of size `1`-by-`N`-by-`D`, where `D` is the value of the `MaximumDelay` property.
  - If you set the `InterpolationMethod` property to `FIR` or `Farrow`, set the `InitialConditions` property to an array of size `1`-by-`N`-by-(`D`+`L`), where `D` is the value of the `MaximumDelay` property. For FIR interpolation, `L` is the value of the `FilterHalfLength` property. For Farrow interpolation, `L` is equal to `floor` of half the value of the `FilterLength` property (`floor(FilterLength/2)`).

`MaximumDelay`

Maximum delay

Specify the maximum delay the object can produce for any sample. The maximum delay must be a positive scalar integer value. The object clips input delay values greater than the `MaximumDelay` to the `MaximumDelay`. The default value of this property is `100`.

`DirectFeedthrough`

Allow direct feedthrough

When you set this property to `true`, the object allows direct feedthrough. When you set this property to `false`, the object increases the minimum possible delay by one. The default value of this property is `true`.

`FIRSmallDelayAction`

Action for small input delay values in FIR interpolation mode

Specify the action the object should take for small input delay values when using the FIR interpolation method. You can set this property to `Clip to the minimum value necessary for centered kernel`, or `Switch to linear interpolation if kernel cannot be centered`. This property applies only when you set the `InterpolationMethod` property to `FIR`. The default value of this property is `Clip to the minimum value necessary for centered kernel`.

`FarrowSmallDelayAction`

Action for small input delay values in Farrow interpolation mode

Specify the action the object should take for small input delay values when using the Farrow interpolation method. You can set this property to `Clip to the minimum value necessary for centered kernel`, or `Use off-centered kernel`. This property applies only when you set the `InterpolationMethod` property to `Farrow`. The default value of this property is `Clip to the minimum value necessary for centered kernel`.

`FrameBasedProcessing`

Treat input as frame based or sample based

Set this property to `true` to enable frame-based processing. When you do so, the object accepts `M`-by-`N` input matrices and treats each of the `N` input columns as a frame containing `M` sequential time samples from an independent channel. Set this property to `false` to enable sample-based processing. When you do so, the object supports `N`-dimensional inputs and treats each element of the input as a separate channel. The default value of this property is `true`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Zero`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap`, or `Saturate`. The default value of this property is `Wrap`.

`CoefficientsDataType`

Coefficient word and fraction lengths

Specify the coefficients data type as `Same word length as input`, or `Custom`. The default value of this property is `Same word length as input`.

`CustomCoefficientsDataType`

Coefficients word length

Specify the coefficients word length as an unscaled `numerictype` object with a `Signedness` of `Auto`. If the `InterpolationMethod` is `Linear`, the `numerictype` object should be unsigned, otherwise it should be signed. This property applies only when you set the `CoefficientsDataType` property to `Custom`. The default value of this property is `numerictype([],32)`.

`ProductPolynomialValueDataType`

Product polynomial values word and fraction lengths

Specify the product polynomial value data type as `Same as first input`, or `Custom`. This property applies only when you set the `InterpolationMethod` property to `Farrow`. The default value of this property is `Same as first input`.

`CustomProductPolynomialValueDataType`

Product polynomial values word and

fraction lengths Specify the product polynomial values fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `InterpolationMethod` property to `Farrow` and the `ProductPolynomialValueDataType` property to `Custom`. The default value of this property is `numerictype([],32,10)`.

`AccumulatorPolynomialValueDataType`

Accumulator polynomial value word and fraction lengths

Specify the accumulator polynomial value data type as `Same as first input`, or `Custom`. This property applies only when you set the `InterpolationMethod` property to `Farrow`. The default value of this property is `Same as first input`.

`CustomAccumulatorPolynomialValueDataType`

Accumulator polynomial value word and fraction lengths

Specify the data type of the accumulator polynomial values as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `InterpolationMethod` property to `Farrow` and the `AccumulatorPolynomialValueDataType` property to `Custom`. The default value of this property is `numerictype([],32,10)`.

`MultiplicandPolynomialValueDataType`

Multiplicand polynomial value word and fraction lengths

Specify the multiplicand polynomial values data type as `Same as first input`, or `Custom`. This property applies only when you set the `InterpolationMethod` property to `Farrow`. The default value of this property is `Same as first input`.

`CustomMultiplicandPolynomialValueDataType`

Multiplicand polynomial value word and fraction lengths

Specify the fixed-point data type of the multiplicand polynomial values as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `InterpolationMethod` property to `Farrow` and the `MultiplicandPolynomialValueDataType` property to `Custom`. The default value of this property is `numerictype([],32,10)`.

`ProductDataType`

Product word and fraction lengths

Specify the product data type as `Same as first input`, or `Custom`. The default value of this property is `Same as first input`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product data type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `ProductDataType` property to `Custom`. The default value of this property is `numerictype([],32,10)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator data type as `Same as product`, `Same as first input`, or `Custom`. The default value of this property is `Same as product`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the fixed-point accumulator data type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,10)`.

`OutputDataType`

Output word and fraction lengths

Specify the output data type as `Same as accumulator`, `Same as first input`, or `Custom`. The default value of this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the data type of the output as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies only when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],32,10)`.

Methods

clone	Create variable fractional delay object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Number of outputs from step method
info	Return characteristic information about valid delay range
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset internal states of variable fractional delay object
step	Delay input by time-varying fractional number of sample periods

Examples

Delay a signal by a varying fractional number of sample periods.

```
hsr = signalblks.SignalReader; % Default signal of 1:10
hvfd = signalblks.VariableFractionalDelay;
hLog = signalblks.SignalLogger;
for ii = 1:10
  delayedsig = step(hvfd, step(hsr), ii/10);
  step(hLog, delayedsig);
end
```

```
sigd = hLog.Buffer;

% The output sigd corresponds to the values of the delayed signal
% that are sampled at fixed-time intervals. For visualization
% purposes, we can instead plot the time instants at which the
% amplitudes of signal samples are constant by treating the
% signals as the sampling instants.

stem(hsr.Signal, 1:10, 'b')
hold on;
stem(sigd.', 1:10, 'r');
legend('Original signal','Variable fractional delayed signal', ...
'Location','best')
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Variable Fractional Delay block reference page. The object properties correspond to the block properties, except for.

- When you set the `DirectFeedthrough` property of the System object to `true`, the object allows direct feedthrough. This behavior is different from the way the block behaves when you select the corresponding **Disable direct feedthrough by increasing minimum possible delay by one** check box on the block dialog. When you enable this block parameter, the block does not allow direct feedthrough.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See "What Are Sample- and Frame-Based Processing?" for more information.

See Also

`signalblks.VariableIntegerDelay` | `signalblks.Delay` | `signalblks.DelayLine`

signalblks.VariableFractionalDelay.clone

Purpose Create variable fractional delay object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates a `VariableFractionalDelay` System object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.VariableFractionalDelay.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose Return characteristic information about valid delay range

Syntax

```
S = info(H)
```

Description `S = info(H)` returns a structure, `S` , containing characteristic information about the `VariableFractionalDelay` System object, `H` . `S` has one field, `ValidDelayRange` which is a string containing the possible range of delay values based on the current property values of the `VariableFractionalDelay` object. The `ValidDelayRange` is in the format `[MinValidDelay, MaxValidDelay]`. The object clips all input delay values to be within this `ValidDelayRange`.

signalblks.VariableFractionalDelay.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `VariableFractionalDelay` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset internal states of variable fractional delay object

Syntax `reset(H)`

Description `reset(H)` resets the internal states of the `VariableFractionalDelay` System object `H` to their initial values.

signalblks.VariableFractionalDelay.step

Purpose

Delay input by time-varying fractional number of sample periods

Syntax

```
Y = step(H,X,D)
```

Description

`Y = step(H,X,D)` delays the input `X` by `D` samples, where `D` should be less than or equal to the value specified in the `MaximumDelay` property.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.VariableIntegerDelay class

Purpose

Delay input by time-varying integer number of sample periods

Description

The `VariableIntegerDelay` object delays input by time-varying integer number of sample periods.

Construction

`H = signalblks.VariableIntegerDelay` returns a variable integer delay System object, `H`, that delays discrete-time input by a time-varying integer number of sample periods.

`H = signalblks.VariableIntegerDelay('PropertyName',PropertyValue,...)` returns a variable integer delay System object, `H`, with each property set to the specified value.

Properties

`MaximumDelay`

Maximum delay

Specify the maximum delay the object can produce for any sample. The maximum delay must be a positive scalar integer value. The object clips input delay values greater than the `MaximumDelay` to the `MaximumDelay`. The default value of this property is `100`.

`InitialConditions`

Initial values in memory

Specify the values with which the object's memory is initialized. The dimensions of this property can vary depending on the setting of the `FrameBasedProcessing` property, and whether you want fixed or time-varying initial conditions. The default value of this property is `0`.

When you set the `FrameBasedProcessing` property to `false`, the object supports `N`-dimensional input arrays. For an `M`-by-`N`-by-`P` sample-based input array `U`, you can set the `InitialConditions` property as follows:

- To specify fixed initial conditions, set the InitialConditions property to a scalar value. The object initializes every sample of every channel in memory using the value you specify.
- To specify time-varying initial conditions, set the InitialConditions property to an array of dimension M-by-N-by-P-by-D. The object uses the values in this array to initialize memory samples U(2:D+1), where D is the value of the MaximumDelay property.

When you set the FrameBasedProcessing property to true, the object treats each of the N input columns as a frame containing M sequential time samples from an independent channel. For an M-by-N frame-based input matrix U, you can set the InitialConditions property as follows:

- To specify fixed initial conditions, set the InitialConditions property to a scalar value. The object initializes every sample of every channel in memory using the value you specify.
- To specify different time-varying initial conditions for each channel, set the InitialConditions property to an array of size 1-by-N-by-D, where D is the value of the MaximumDelay property.

DirectFeedthrough

Allow direct feedthrough

When you set this property to true, the object allows direct feedthrough. When you set this property to false, the object increases the minimum possible delay by one. The default value of this property is true.

FrameBasedProcessing

Treat input as frame based or sample based

Set this property to true to enable frame-based processing. When you do so, the object accepts M-by-N input matrices and treats each of the N input columns as a frame containing M sequential time samples from an independent channel. Set this property to false to enable sample-based processing. When you do so, the object supports N-dimensional inputs and treats each element of the input as a separate channel. The default value of this property is true.

Methods

clone	Create variable integer delay object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset internal states of variable integer delay object
step	Delay input by time-varying integer number of sample periods

Examples

Delay a signal by a varying integer number of sample periods.

```
h = signalblks.VariableIntegerDelay;
x = 1:100;
ii = 0;
k = 0;
yout = [];
while(ii+10 <= 100)
   y = step(h, x(ii+1:ii+10)',k*ones(10,1));
   yout = [yout;y];
   ii = ii+10;
   k = k+1;
```

```
end
stem(x,'b');
hold on; stem(yout,'r');
legend('Original Signal', 'Variable Integer Delayed Signal')
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Variable Integer Delay block reference page. The object properties correspond to the block properties, except for:

- When you set the `DirectFeedthrough` property of the System object to `true`, the object allows direct feedthrough. This behavior is different from the way the block behaves when you select the corresponding **Disable direct feedthrough by increasing minimum possible delay by one** check box on the block dialog. When you enable this block parameter, the block does not allow direct feedthrough.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See "What Are Sample- and Frame-Based Processing?" for more information.

See Also

`signalblks.VariableFractionalDelay` | `signalblks.Delay` | `signalblks.DelayLine`

Purpose Create variable integer delay object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates a `VariableIntegerDelay` System object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

signalblks.VariableIntegerDelay.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.VariableIntegerDelay.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.VariableIntegerDelay.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `VariableIntegerDelay` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset internal states of variable integer delay object

Syntax `reset(H)`

Description `reset(H)` resets the internal states of the `VariableIntegerDelay` System object `H` to their initial values.

Purpose Delay input by time-varying integer number of sample periods

Syntax

```
Y = step(H,X,D)
```

Description `Y = step(H,X,D)` delays the input `X` by `D` samples, where `D` should be less than or equal to the value specified in the `MaximumDelay` property and greater than or equal to `0`. The object clips delay values greater than the `MaximumDelay` to the `MaximumDelay`, and clips values less than zero to zero. If you enter a non-integer delay value, the object rounds it to the nearest integer value.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.Variance class

Purpose

Variance of input or sequence of inputs

Description

The `Variance` object computes variance for an input or sequence of inputs.

Construction

`H = signalblks.Variance` returns a variance System object, `H`, that computes the variance of an input or a sequence of inputs.

`H = signalblks.Variance('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a Vvariance System object, `H`, with each specified property set to the specified value.

Properties

`RunningVariance`

Enable calculation over time

Set this property to `true` to enable variance calculation over time. The default value of this property is `false`.

`ResetInputPort`

Enable reset input port

Set this property to `true` to enable reset input port. When you set the property to `true`, specify a reset input for the step method. The running variance resets anytime the variance object achieves the condition you specify for the `ResetCondition` property. This property applies when you set the `RunningVariance` property to `true`. The default value of this property is `false`.

`ResetCondition`

Reset condition for running variance mode

Specify which event resets the running variance: `Rising edge`, `Falling edge`, `Either edge`, or `Non-zero`. This property applies when you set the `ResetInputPort` property to `true`.

`Dimension`

Dimension to operate along

Specify how the object performs the variance calculation over the data as `All`, `Row`, `Column`, or `Custom`. This property applies when you set the `RunningVariance` property to `false`. The default value of this property is `Column`.

`CustomDimension`

Numerical dimension to operate along

Specify the input signal dimension (one-based value) the object uses to compute variance. The value of this property cannot exceed the number of dimensions in the input signal. This property applies when you set the `Dimension` property to `Custom`. The default value of this property is `1`.

`ROIProcessing`

Enable region-of-interest processing

Set this property to `true` to enable calculating the variance within a particular region of each image. This property applies when you set the `RunningVariance` property to `false` and the `Dimension` property to `All`. The default value of this property is `false`. Full ROI processing support requires a Video and Image Processing Blockset license. With only the Signal Processing Blockset license, `Rectangles` is the only selection for the `ROIForm` property.

`ROIForm`

Define the type of region of interest.

Specify the type of region of interest as `Rectangles`, `Lines`, `Label matrix`, or `Binary mask`. This property applies when you set the `ROIProcessing` property to `true`. The default value for this property is `Rectangle`.

`ROIPortion`

Calculate over entire ROI or just perimeter

Specify the region over which to calculate variance as `Entire ROI` or `ROI perimeter`. This property applies when you set the

`ROIForm` property to `Rectangles`. The default value for this property is `Entire ROI`.

`ROIStatistics`

Statistics for each ROI or one for all ROIs

Specify if statistics calculations are `Individual statistics for each ROI` or `Single statistic for all ROIs`. This property applies when `ROIForm` property is not `Binary mask`. The default value for this property is `Individual statistics for each ROI`.

`ValidityOutputPort`

Enable output of validity check of ROI or label numbers

Indicate whether to return the validity of the specified ROI being completely inside image when the ROIForm property is `Lines` or `Rectangles`. Indicate whether to return the validity of the specified label numbers when the `ROIForm` property is `Label Matrix`. This property applies when you set the `ROIForm` property to anything except `Binary Mask`. The default value of this property is `false`.

`FrameBasedProcessing`

Enable frame-based processing

Set this property to `true` to enable frame-based processing. Set this property to `false` to enable sample-based processing. This property applies when you set the `RunningVariance` property to `true`. The default value of this property is `true`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value of this property is `Floor`

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`InputSquaredProductDataType`

Input-squared product word and fraction lengths

Specify the input-squared product fixed-point data type as `Same as input` or `Custom`. The default value of this property is `Same as input`.

`CustomInputSquaredProductDataType`

Input-squared product word and fraction lengths

Specify the input-squared product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `InputSquaredProductDataType` property to `Custom`. The default value of this property is `numerictype([],32,15)`.

`InputSumSquaredProductDataType`

Input-sum-squared product word and fraction lengths

Specify the input-sum-squared product fixed-point data type as `Same as input-squared product` or `Custom`. The default value of this property is `Same as input-squared product`.

`CustomInputSumSquaredProductDataType`

Input-sum-squared product word and fraction lengths

Specify the input-sum-squared product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `InputSumSquaredProductDataType` property to `Custom`. The default value of this property is `numerictype([],32,23)`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Same as input-squared product`, `Same as input` or `Custom`. The default value of this property is `Same as input-squared product`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to `Custom`. The default value of this property is `numerictype([],32,0)`.

`OutputDataType`

Output word and fraction lengths

Specify the output fixed-point data type as `Same as accumulator`, `Same as input` or `Custom`. The default value of this property is `Same as accumulator`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property only applies when the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,0)`.

Methods

clone	Create variance object with same property values
getNumInputs	Number of expected inputs to step method

getNumOutputs	Return number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
reset	Reset variance to zero
step	Compute variance of input

Examples

Compute the running variance for a signal.

```
hvar = signalblks.Variance;
hvar.RunningVariance = true;
x = randn(100,1);
y = step(hvar, x);
% y(i) is the running variance of all values in the vector x(1:i)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Variance block reference page. The object properties correspond to the block parameters, except for:

- **Treat sample-based row input as a column** block parameter is not supported by the `signalblks.Variance` object.
- **Reset port** block parameter corresponds to both the `ResetCondition` and the `ResetInputPort` object properties.

See Also

`signalblks.Mean` | `signalblks.RMS` | `signalblks.StandardDeviation`

Purpose Create variance object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` create a `Variance` System object `C`, with the same property values as `H`. The clone method creates a new unlocked object with uninitialized states.

signalblks.Variance.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Return number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.Variance.isLocked

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

```
isLocked(H)
```

Description

`isLocked(H)` returns the locked state of the `Variance` System object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose Reset variance to zero

Syntax

```
reset(H)
```

Description `reset(H)` sets the variance to zero when the `RunningVariance` property is `true`.

signalblks.Variance.step

Purpose Compute variance of input

Syntax

```
Y = step(H,X)
Y = step(H,X,R)
VAR2D = step(H,X,ROI)
VAR2D = step(H,X,LABEL,LABELNUMBERS)
[VAR2D,FLAG] = step(H,X,ROI)
[VAR2D,FLAG] = step(H,X,LABEL,LABELNUMBERS)
```

Description

`Y = step(H,X)` computes the variance of input `X`. The object computes the variance of the input elements over time, `Y`, when the `RunningVariance` property is `true`.

`Y = step(H,X,R)` computes the variance of the input elements over time, `Y`, and optionally resets its state based on the value of reset signal `R` , the `ResetInputPort` property and the `ResetCondition` property. This option applies when the `RunningVariance` property is `true` and the `ResetInputPort` property is set to `true`.

`VAR2D = step(H,X,ROI)` computes the variance of input image, `X`, within the given region of interest, `ROI` , when the `ROIProcessing` property is `true` and the `ROIForm` property is `Lines`, `Rectangles` or `Binary mask`. Note that full `ROI` processing support requires a Video and Image Processing Blockset license. With only the Signal Processing Blockset license, the `ROIForm` property only supports `Rectangles`.

`VAR2D = step(H,X,LABEL,LABELNUMBERS)` computes the variance of input image, `X`, for region labels contained in vector `LABELNUMBERS` , with matrix `LABEL` marking pixels of different regions. This option is available when the `ROIProcessing` property is `true` and the `ROIForm` property is `Label matrix`.

`[VAR2D,FLAG] = step(H,X,ROI)` also returns `FLAG` which indicates whether the given region of interest is within the image bounds when both the `ROIProcessing` and `ValidityOutputPort` properties are `true` and the `ROIForm` property is set to `Lines`, `Rectangles` or `Binary mask`.

`[VAR2D,FLAG] = step(H,X,LABEL,LABELNUMBERS)` also returns `FLAG` which indicates whether the input label numbers are valid when both the `ROIProcessing` and `ValidityOutputPort` properties are `true` and the `ROIForm` property is set to `Label matrix`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.VectorQuantizerDecoder class

Purpose

Return vector quantizer codeword for given index value

Description

The `VectorQuantizerDecoder` object returns the vector quantizer codeword for a given index value. Each column of the `Codebook` property represents a codeword.

Construction

`H = signalblks.VectorQuantizerDecoder` creates a vector quantizer decoder System object, `H`, that returns a vector quantizer codeword corresponding to a given, zero-based index value.

`H = signalblks.VectorQuantizerDecoder('`*`PropertyName`*`',`*`PropertyValue`*`,...)` returns a vector quantizer decoder, `H`, with each specified property set to the specified value.

Properties

`CodebookSource`

Source of code book values

Specify the code book source as `Property` or `Input port`. When you select `Property`, the object reads the codebook from the `codebook` property. When you select `Input port`, the object reads the codebook from the input of the `step` method.

The default value of this property is `Property`.

`Codebook`

Code book

Specify quantized output values as a k-by-N matrix, where $\mathrm{k} \geq 1$ and $N \geq 1$. Each column of the codebook matrix is a codeword, and each codeword corresponds to an index value. This property applies when you set the `CodebookSource` property to `Property`. The default value of this property is:

$$\begin{bmatrix} 1.5 & 13.3 & 136.4 & 6.8 \\ 2.5 & 14.3 & 137.4 & 7.8 \\ 3.5 & 15.3 & 138.4 & 8.8 \end{bmatrix}$$

This property is tunable.

`OutputDataType`

Data type of codebook and quantized output

Specify the data type of the codebook and quantized output values as: `Same as input`, `double`, `single` or `Custom`. This property applies only when you set `CodebookSource` to `Property`. The default value of this property is `double`.

Fixed-Point Properties

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a signed or unsigned `numerictype` object. This property applies only when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype(true,16)`.

Methods

clone	Create vector quantizer decoder object with same property values
getNumInputs	Return number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Return quantized output values corresponding to input indices

Examples

Given index values as an input, determine the corresponding vector quantized codewords for a specified codebook.

```
hvqdec = signalblks.VectorQuantizerDecoder;
hvqdec.Codebook = [1 10 100;2 20 200;3 30 300];
indices = uint8([1 0 2 0]);
qout = step(hvqdec, indices)
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Vector Quantizer Decoder block reference page. The object properties correspond to the block parameters, except for:

- There is no object property that directly corresponds to the **Action for out of range index value** block parameter. The object sets any index values less than `0` to `0` and any index values greater than or equal to N to N-1.

See Also

`signalblks.VectorQuantizerEncoder` | `signalblks.ScalarQuantizerDecoder`

Purpose Create vector quantizer decoder object with same property values

Syntax

Description `C = clone(H)` creates a `VectorQuantizerDecoder` object `C`, with the same property values as `H`. The `clone` method creates a new unlocked object with uninitialized states.

signalblks.VectorQuantizerDecoder.getNumInputs

Purpose Return number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.VectorQuantizerDecoder.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the `VectorQuantizerDecoder` object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Return quantized output values corresponding to input indices

Syntax

```
Q = step(H,I)
Q = step(H,I,C)
```

Description

`Q = step(H,I)` returns the quantized output values `Q` corresponding to the input indices `I`. The data type of `I` can be `uint8`, `uint16`, `uint32`, `int8`, `int16` or `int32`. The `OutputDataType` property determines the data type of `Q`.

`Q = step(H,I,C)` uses input `C` as the `codebook` values when the `CodebookSource` property is `Input port`. The data type of `C` can be `double`, `single` or `fixed-point`. The output `Q` has the same data type as the `codebook` input `C`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

---

signalblks.VectorQuantizerEncoder class

Purpose Perform vector quantization encoding

Description The `VectorQuantizerEncoder` object performs vector quantization encoding. The object finds the nearest codeword by computing a distortion based on Euclidean or weighted Euclidean distance.

Construction `H = signalblks.VectorQuantizerEncoder` returns a vector quantizer encoder System object, `H`, that finds a zero-based index of the nearest codeword for each given input column vector.

`H = signalblks.VectorQuantizerEncoder('PropertyName',PropertyValue,...)` returns a vector quantizer encoder System object, `H`, with each specified property set to the specified value.

Properties

`CodebookSource`

Source of codebook values

Specify how to determine the codebook values as `Property` or `Input port`. The default value of this property is `Property`.

`Codebook`

Codebook

Specify the codebook to which the input column vector or matrix is compared, as a k-by-N matrix. Each column of the codebook matrix is a codeword, and each codeword corresponds to an index value. The codeword vectors must have the same number of rows as the input. The first codeword vector corresponds to an index value of `0`, the second codeword vector corresponds to an index value of `1`, and so on. This property applies when you set the `CodebookSource` property to `Property`. The default value of this property is:

$$\begin{bmatrix} 1.5\ 13.3\ 136.4\ 6.8 \\ 2.5\ 14.3\ 137.4\ 7.8 \\ 3.5\ 15.3\ 138.4\ 8.8 \end{bmatrix}$$

This property is tunable.

`DistortionMeasure`

Distortion calculation method

Specify how to calculate the distortion as `Squared error` or `Weighted squared error`. If you set this property to `Squared error`, the object calculates the distortion by evaluating the Euclidean distance between the input column vector and each codeword in the codebook. If you set this property to `Weighted squared error`, the object calculates the distortion by evaluating a weighted Euclidean distance using a weighting factor to emphasize or deemphasize certain input values. The default value of this property is `Squared error`.

`WeightsSource`

Source of weighting factor

Specify how to determine weighting factor as `Property` or `Input port`. This property applies when you set the `DistortionMeasure` property to `Weighted squared error`. The default value of this property is `Property`.

`Weights`

Weighting factor

Specify the weighting factor as a vector of length equal to the number of rows of the input. This property applies when you set the `DistortionMeasure` property to `Weighted squared error` and `WeightsSource` property is `Property`. The default value of this property is [1 1 1]. This property is tunable.

`TiebreakerRule`

Behavior when input column vector is equidistant from two codewords.

Specify whether to represent the input column vector by the lower index valued codeword or higher indexed valued codeword when an input column vector is equidistant from two codewords. This property can be set to `Choose the lower index` or `Choose the higher index`. The default value of this property is `Choose the lower index`.

`CodewordOutputPort`

Enable output of codeword value

Set this property to `true` to output the codeword vectors nearest to the input column vectors. The default value of this property is `false`.

`QuantizationErrorOutputPort`

Enable output of quantization error

Set this property to `true` to output the quantization error value that results when the object represents the input column vector by the nearest codeword. The default value of this property is `false`.

`OutputIndexDataType`

Data type of index output

Specify the data type of the index output as: `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`. The default value of this property is `int32`.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest` or `Zero`. The default value of this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as `Wrap` or `Saturate`. The default value of this property is `Wrap`.

`ProductDataType`

Product word and fraction lengths

Specify the product fixed-point data type as `Same as input` or `Custom`. The default value of this property is `Same as input`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`.

`AccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point data type as `Same as input`, or `Custom`. The default value of this property is `Same as product`.

`CustomAccumulatorDataType`

Accumulator word and fraction lengths

Specify the accumulator fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `AccumulatorDataType` property to`Custom`.

Methods

clone	Create vector quantizer encoder object with same property values
getNumInputs	Number of expected inputs to step method

getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Quantization region(s) to which input belongs

Examples

Find the indices of nearest codewords based on Euclidean distances.

```
hvqenc = signalblks.VectorQuantizerEncoder(...
'Codebook', [-1 -1 1 1;1 -1 -1 1], ...
'CodewordOutputPort', true, ...
'QuantizationErrorOutputPort', true, ...
'OutputIndexDataType', 'uint8');
% Generate an input signal with some additive noise
x = sign(rand(2,40)-0.5) + 0.1*randn(2,40);
[ind, cw, err] = step(hvqenc, x);
plot(cw(1,:), cw(2,:), 'rO', x(1,:), x(2,:), 'g.');
legend('Quantized', 'Inputs', 'location', 'best');
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Vector Quantizer Encoder block reference page. The object properties correspond to the block parameters.

See Also

`signalblks.VectorQuantizerDecoder` | `signalblks.ScalarQuantizerEncoder`

Purpose Create vector quantizer encoder object with same property values

Syntax

Description `C = clone(H)` creates a `VectorQuantizerEncoder` object `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.VectorQuantizerEncoder.getNumInputs

Purpose Number of expected inputs to step method

Syntax `getNumInputs(H)`

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number from the `step` method

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.VectorQuantizerEncoder.isLocked

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the `VectorQuantizerEncoder` System object `H`.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Quantization region(s) to which input belongs

Syntax

```
INDEX = step(H,INPUT)
INDEX = step(H,...,CODEBOOK)
INDEX = step(H,...,WEIGHTS)
[..., CODEWORD] = step(H, ...)
[..., QERR] = step(H, ...)
```

Description

`INDEX = step(H,INPUT)` returns `INDEX`, a scalar or column vector representing the quantization region(s) to which `INPUT` belongs. `INPUT` can be a column vector of size k-by-1 or an `M` multichannel matrix of dimensions k-by-M, where k is the length of each codeword in the codebook. All inputs to the object can be real floating-point or fixed-point values and must be of the same data type. The output index values can be signed or unsigned integers.

`INDEX = step(H,...,CODEBOOK)` uses the codebook given in input `CODEBOOK`, a k-by-N matrix with `N` codewords each of length k. This option is available when the `CodebookSource` property is `Input port`.

`INDEX = step(H,...,WEIGHTS)` uses the input vector `WEIGHTS` to emphasize or de-emphasize certain input values when calculating the distortion measure. `WEIGHTS` must be a vector of length equal to the number of rows of `INPUT`. This option is available when the `DistortionMeasure` property is `Weighted squared error` and the `WeightsSource` property is `Input port`.

`[..., CODEWORD] = step(H, ...)` outputs the `CODEWORD` values that correspond to each index value when the `CodewordOutputPort` property is `true`.

`[..., QERR] = step(H, ...)` outputs the quantization error `QERR` for each input value when the `QuantizationErrorOutputPort` property is `true`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

.

Purpose Window object

Description The `Window` object applies a window to an input signal.

Construction `H = signalblks.Window` returns a window object, `H`, that applies a Hamming window with symmetric sampling.

`H = signalblks.Window('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns a window object, `H`, with each property set to the specified value.

`H = signalblks.Window(WINDOW,'`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns a window object, `H`, with the `WindowFunction` property set to `WINDOW` and other properties set to the specified values.

Properties

`WindowFunction`

Type of window

Specify the type of window to apply as `Bartlett`, `Blackman`, `Boxcar`, `Chebyshev`, `Hamming` , `Hann`, `Hanning`, `Kaiser`, `Taylor`, `Triang`. This property is tunable. The default value for this property is `Hamming`.

`WeightsOutputPort`

Enable the output of window weights

Set this property to `true` to output the window weights. The weights are an M–by-1 vector with M equal to the first dimension of the input. The default value of this property is `false`.

`StopbandAttenuation`

Level of stopband attenuation in decibels

Specify the level of stopband attenuation in decibels. This property only applies when the `WindowFunction` property is `Chebyshev`. The default value of this property is 50. This property is tunable.

`Beta`

Kaiser window parameter

Specify the Kaiser window parameter as a real number. Increasing the absolute value of `Beta` widens the mainlobe and decreases the amplitude of the window sidelobes in the window's frequency magnitude response. This property only applies when `WindowFunction` property is `Kaiser`. The default value of this property is 10. This property is tunable.

`NumConstantSidelobes`

Number of constant sidelobes

Specify the number of constant sidelobes as an integer greater than zero. This property only applies when `WindowFunction` property is `Taylor`. The default value of this property is 4. This property is tunable.

`MaximumSidelobeLevel`

Maximum sidelobe level relative to mainlobe

Specify, in decibels, the maximum sidelobe level relative to the mainlobe as a real number less than or equal to zero. The default value of this property is -30 producing sidelobes with peaks 30 dB down from the mainlobe peak. This property only applies when `WindowFunction` property is `Taylor`. This property is tunable.

`Sampling`

Window sampling for generalized-cosine windows

Specify the window sampling for generalized-cosine windows as `Symmetric` or `Periodic`. This property only applies when `WindowFunction` property is `Blackman`, `Hamming`, `Hann`, or `Hanning`. This property is tunable.

Fixed-Point Properties

`RoundingMethod`

Rounding method for fixed-point operations

Specify the rounding method as one of `Ceiling`, `Convergent`, `Floor`, `Nearest`, `Round`, `Simplest`, or `Zero`. The default value for this property is `Floor`.

`OverflowAction`

Overflow action for fixed-point operations

Specify the overflow action as one of `Wrap` or `Saturate`. The default value for this property is `Wrap`.

`WindowDataType`

Window word and fraction lengths

Specify the window fixed-point data type as one of `Same word length as input` or `Custom`. The default value for this property is `Same word length as input`.

`CustomWindowDataType`

Window word and fraction lengths

Specify the window fixed-point type as a `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `WindowDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

`ProductDataType`

Product word and fraction lengths

Specify the product fixed-point data type as one of `Internal rule`, `Same as input`, or `Custom`. The default value for this property is `Internal rule`.

`CustomProductDataType`

Product word and fraction lengths

Specify the product fixed-point type as a scaled `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `ProductDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

`OutputDataType`

Output data type

Specify the output fixed-point data type as one of `Same as product`, `Same as input`, `Custom`. The default value for this property is `Same as product`.

`CustomOutputDataType`

Output word and fraction lengths

Specify the output fixed-point type as a `numerictype` object with a `Signedness` of `Auto`. This property applies when you set the `OutputDataType` property to `Custom`. The default value of this property is `numerictype([],16,15)`.

Methods

clone	Create window object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Multiply input by window

Examples

Apply Hamming window to input signal:

```
 hwin = signalblks.Window( ...
'WindowFunction', 'Hamming', ...
'WeightsOutputPort',true);
 x = rand(64,1);
[y, w] = step(hwin, x);
% View the window's time and frequency domain responses
```

```
wvtool(w);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Window Function block reference page. The object properties correspond to the block parameters, except for:

- **Operation**– The window object does not support the `Generate window` option.
- **Operation**–The `Generate and apply window` option on the block corresponds to the `WeightsOutputPort` property set to `true` on the window object.
- The window object only supports frame-based processing.

See Also

`signalblks.FFT` | `sigwin.bartlett` | `sigwin.blackman` | `sigwin.chebwin` | `sigwin.hamming` | `sigwin.hann` | `sigwin.kaiser` | `sigwin.taylorwin` | `sigwin.triang` | `wvtool`

signalblks.Window.clone

Purpose Create window object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates a window object, `C`, with the same property values as `H`. The clone method creates a new unlocked object.

Purpose

Number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description

`getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

signalblks.Window.getNumOutputs

Purpose Number of outputs from step method

Syntax `getNumOutputs(H)`

Description `getNumOutputs(H)` returns the number of outputs from the `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

Purpose

Locked status (logical) for input attributes and non-tunable properties

Syntax

`isLocked(H)`

Description

`isLocked(H)` returns the locked state of the window object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

signalblks.Window.step

Purpose

Multiply input by window

Syntax

```
Y = step(H,X)
[Y, W] = step(H,X)
```

Description

`Y = step(H,X)` generates the windowed output, `Y` , of the input, `X`, using the specified window.

`[Y, W] = step(H,X)` returns the window values `W` when the `WeightsOutputPort` property is `true`.

signalblks.ZeroCrossingDetector class

Purpose Zero crossing detector

Description The `ZeroCrossingDetector` object counts the number of times the signal crosses zero, or changes sign. The zero crossing detector supports both floating-point and fixed-point data types.

Construction `H = signalblks.ZeroCrossingDetector` returns a zero crossing detector object, `H`, that counts the number of zero crossings in the real-valued floating-point or fixed-point frame-based vector or matrix.

`H = signalblks.ZeroCrossingDetector('`*`PropertyName`*`',`*`PropertyValue`*`, ...)` returns a zero crossing detector object, `H`, with each property set to the specified value.

Properties `FrameBasedProcessing`

Enable frame-based processing

Set this property to `true` to enable frame-based processing. Set this property to `false` to enable sample-based processing. The default value for this property is `true`.

If the `FrameBasedProcessing` property is `true`:

- The zero crossing detector treats a column vector or the columns of a matrix as single, independent channels.
- The zero crossing detector treats a length N row vector as N independent channels.

If the `FrameBasedProcessing` property is `false`:

- The zero crossing detector treats each entry of a vector or matrix as an independent channel.

Methods

clone	Create zero crossing detection object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Return number of outputs of step method
isLocked	Locked status (logical) for input attributes and non-tunable properties
step	Count zero crossings in input

Examples

Find number of zero crossing in electrocardiogram data:

```
EcgData = ecg(500)';
Hzerocross = signalblks.ZeroCrossingDetector;
NumZeroCross = step(Hzerocross,EcgData);
% NumZeroCross is equal to 4
plot(1:500,EcgData,'b',[0 500],[0 0],'r','linewidth',2);
```

Algorithm

This object implements the algorithm, inputs, and outputs described on the Zero Crossing block reference page. The object properties correspond to the block parameters.

Objects and blocks interpret frames differently. Objects process inputs as frames or as samples by setting the `FrameBasedProcessing` property. Blocks process inputs as frames or as samples by inheriting the frame information from the input ports. See “What Are Sample- and Frame-Based Processing?” for more information.

Purpose Create zero crossing detection object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates a zero crossing detector object, `C`, with the same property values as `H`. The clone method creates a new unlocked object.

signalblks.ZeroCrossingDetector.getNumInputs

Purpose Number of expected inputs to step method

Syntax

```
getNumInputs(H)
```

Description `getNumInputs(H)` returns the number of expected inputs to the `step` method.

The `getNumInputs` method returns a positive integer representing the number of expected inputs to the `step` method. This value will change if any properties that turn inputs on or off are changed. The `step` method must be called with a number of input arguments equal to the result of `getNumInputs(H)`.

Purpose

Return number of outputs of step method

Syntax

`getNumOutputs(H)`

Description

`getNumOutputs(H)` returns the number of outputs from `step` method.

The `getNumOutputs` method returns a positive integer representing the number of outputs from the `step` method. This value will change if any properties that turn inputs on or off are changed.

signalblks.ZeroCrossingDetector.isLocked

Purpose Locked status (logical) for input attributes and non-tunable properties

Syntax `isLocked(H)`

Description `isLocked(H)` returns the locked state of the zero crossing detector object.

The `isLocked` method returns a logical value to indicate whether input attributes and non-tunable properties are locked for the object. The object performs an internal initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. Once this occurs, the `isLocked` method returns a `true` value.

Purpose

Count zero crossings in input

Syntax

`Y = step(H,X)`

Description

`Y = step(H,X)` counts the number of zero crossings, `Y`, in the vector or matrix input `X`.

Note The object performs an initialization the first time the `step` method is executed. This initialization locks non-tunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a non-tunable property or an input specification, the System object issues a warning and re-initializes.

Function Reference

Signal Processing Functions

dsplib	Open top-level Signal Processing Blockset library
dspstartup	Configure Simulink environment for signal processing systems
dsp_links	Identify whether blocks in model are current, deprecated, or obsolete
liblinks	Check model for blocks from specific Signal Processing Blockset libraries
rebuffer_delay	Number of samples of delay introduced by buffering and unbuffering operations

Glossary

This glossary defines terms related to fixed-point data types and numbers. These terms may appear in some or all of the documents that describe products from The MathWorks™ that have fixed-point support.

arithmetic shift

Shift of the bits of a binary word for which the sign bit is recycled for each bit shift to the right. A zero is incorporated into the least significant bit of the word for each bit shift to the left. In the absence of overflows, each arithmetic shift to the right is equivalent to a division by 2, and each arithmetic shift to the left is equivalent to a multiplication by 2.

See also binary point, binary word, bit, logical shift, most significant bit

bias

Part of the numerical representation used to interpret a fixed-point number. Along with the slope, the bias forms the scaling of the number. Fixed-point numbers can be represented as

$$\mathit{real\text{-}world\ value} = (\mathit{slope} \times \mathit{stored\ integer}) + \mathit{bias}$$

where the slope can be expressed as

$$\mathit{slope} = \mathit{fractional\ slope} \times 2^{\mathit{exponent}}$$

See also fixed-point representation, fractional slope, integer, scaling, slope, [Slope Bias]

binary number

Value represented in a system of numbers that has two as its base and that uses 1's and 0's (bits) for its notation.

See also bit

binary point

Symbol in the shape of a period that separates the integer and fractional parts of a binary number. Bits to the left of the binary point are integer bits and/or sign bits, and bits to the right of the binary point are fractional bits.

See also binary number, bit, fraction, integer, radix point

binary point-only scaling

Scaling of a binary number that results from shifting the binary point of the number right or left, and which therefore can only occur by powers of two.

See also binary number, binary point, scaling

binary word

Fixed-length sequence of bits (1's and 0's). In digital hardware, numbers are stored in binary words. The way in which hardware components or software functions interpret this sequence of 1's and 0's is described by a data type.

See also bit, data type, word

bit

Smallest unit of information in computer software or hardware. A bit can have the value 0 or 1.

ceiling (round toward)

Rounding mode that rounds to the closest representable number in the direction of positive infinity. This is equivalent to the `ceil` mode in Fixed-Point Toolbox software.

See also convergent rounding, floor (round toward), nearest (round toward), rounding, truncation, zero (round toward)

contiguous binary point

Binary point that occurs within the word length of a data type. For example, if a data type has four bits, its contiguous binary point must be understood to occur at one of the following five positions:

.0000
0.000
00.00
000.0
0000.

See also data type, noncontiguous binary point, word length

convergent rounding

Rounding mode that rounds to the nearest allowable quantized value. Numbers that are exactly halfway between the two nearest allowable quantized values are rounded up only if the least significant bit (after rounding) would be set to `0`.

See also ceiling (round toward), floor (round toward), nearest (round toward), rounding, truncation, zero (round toward)

data type

Set of characteristics that define a group of values. A fixed-point data type is defined by its word length, its fraction length, and whether it is signed or unsigned. A floating-point data type is defined by its word length and whether it is signed or unsigned.

See also fixed-point representation, floating-point representation, fraction length, signedness, word length

data type override

Parameter in the Fixed-Point Tool that allows you to set the output data type and scaling of fixed-point blocks on a system or subsystem level.

See also data type, scaling

exponent

Part of the numerical representation used to express a floating-point or fixed-point number.

1. Floating-point numbers are typically represented as

$$real\text{ - }world\ value = mantissa \times 2^{exponent}$$

2. Fixed-point numbers can be represented as

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

where the slope can be expressed as

$$slope = fractional\ slope \times 2^{exponent}$$

The exponent of a fixed-point number is equal to the negative of the fraction length:

$$exponent = -1 \times fraction\ length$$

See also bias, fixed-point representation, floating-point representation, fraction length, fractional slope, integer, mantissa, slope

fixed-point representation

Method for representing numerical values and data types that have a set range and precision.

1. Fixed-point numbers can be represented as

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

where the slope can be expressed as

$$slope = fractional\ slope \times 2^{exponent}$$

The slope and the bias together represent the scaling of the fixed-point number.

2. Fixed-point data types can be defined by their word length, their fraction length, and whether they are signed or unsigned.

See also bias, data type, exponent, fraction length, fractional slope, integer, precision, range, scaling, slope, word length

floating-point representation

Method for representing numerical values and data types that can have changing range and precision.

1. Floating-point numbers can be represented as

$$real\text{-}world\ value = mantissa \times 2^{exponent}$$

2. Floating-point data types are defined by their word length.

See also data type, exponent, mantissa, precision, range, word length

floor (round toward)

Rounding mode that rounds to the closest representable number in the direction of negative infinity.

See also ceiling (round toward), convergent rounding, nearest (round toward), rounding, truncation, zero (round toward)

fraction

Part of a fixed-point number represented by the bits to the right of the binary point. The fraction represents numbers that are less than one.

See also binary point, bit, fixed-point representation

fraction length

Number of bits to the right of the binary point in a fixed-point representation of a number.

See also binary point, bit, fixed-point representation, fraction

fractional slope

Part of the numerical representation used to express a fixed-point number. Fixed-point numbers can be represented as

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

where the slope can be expressed as

$$slope = fractional\ slope \times 2^{exponent}$$

The term *slope adjustment* is sometimes used as a synonym for fractional slope.

See also bias, exponent, fixed-point representation, integer, slope

guard bits

Extra bits in either a hardware register or software simulation that are added to the high end of a binary word to ensure that no information is lost in case of overflow.

See also binary word, bit, overflow

integer

1. Part of a fixed-point number represented by the bits to the left of the binary point. The integer represents numbers that are greater than or equal to one.

2. Also called the "stored integer." The raw binary number, in which the binary point is assumed to be at the far right of the word. The integer is part of the numerical representation used to express a fixed-point number. Fixed-point numbers can be represented as

$$real\text{ - }world\ value = 2^{-fraction\ length} \times stored\ integer$$

or

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

where the slope can be expressed as

$$slope = fractional\ slope \times 2^{exponent}$$

See also bias, fixed-point representation, fractional slope, integer, real-world value, slope

integer length

Number of bits to the left of the binary point in a fixed-point representation of a number.

See also binary point, bit, fixed-point representation, fraction length, integer

least significant bit (LSB)

Bit in a binary word that can represent the smallest value. The LSB is the rightmost bit in a big-endian-ordered binary word. The weight of the LSB is related to the fraction length according to

$$weight\ of\ LSB = 2^{-fraction\ length}$$

See also big-endian, binary word, bit, most significant bit

logical shift

Shift of the bits of a binary word, for which a zero is incorporated into the most significant bit for each bit shift to the right and into the least significant bit for each bit shift to the left.

See also arithmetic shift, binary point, binary word, bit, most significant bit

mantissa

Part of the numerical representation used to express a floating-point number. Floating-point numbers are typically represented as

$$real\text{-}world\ value = mantissa \times 2^{exponent}$$

See also exponent, floating-point representation

most significant bit (MSB)

Bit in a binary word that can represent the largest value. The MSB is the leftmost bit in a big-endian-ordered binary word.

See also binary word, bit, least significant bit

nearest (round toward)

Rounding mode that rounds to the closest representable number, with the exact midpoint rounded to the closest representable number in the direction of positive infinity. This is equivalent to the `nearest` mode in Fixed-Point Toolbox software.

See also ceiling (round toward), convergent rounding, floor (round toward), rounding, truncation, zero (round toward)

noncontiguous binary point

Binary point that is understood to fall outside the word length of a data type. For example, the binary point for the following 4-bit word is understood to occur two bits to the right of the word length,

0000__.

thereby giving the bits of the word the following potential values:

$2^5 2^4 2^3 2^2$__.

See also binary point, data type, word length

one's complement representation

Representation of signed fixed-point numbers. Negating a binary number in one's complement requires a bitwise complement. That is, all 0's are flipped to 1's and all 1's are flipped to 0's. In one's complement notation there are two ways to represent zero. A binary word of all 0's represents "positive" zero, while a binary word of all 1's represents "negative" zero.

See also binary number, binary word, sign/magnitude representation, signed fixed-point, two's complement representation

overflow

Situation that occurs when the magnitude of a calculation result is too large for the range of the data type being used. In many cases you can choose to either saturate or wrap overflows.

See also saturation, wrapping

padding

Extending the least significant bit of a binary word with one or more zeros.

See also least significant bit

precision

1. Measure of the smallest numerical interval that a fixed-point data type and scaling can represent, determined by the value of the number's least significant bit. The precision is given by the slope, or the number of fractional bits. The term *resolution* is sometimes used as a synonym for this definition.

2. Measure of the difference between a real-world numerical value and the value of its quantized representation. This is sometimes called quantization error or quantization noise.

See also data type, fraction, least significant bit, quantization, quantization error, range, slope

Q format

Representation used by Texas Instruments™ to encode signed two's complement fixed-point data types. This fixed-point notation takes the form

$$Qm.n$$

where

- Q indicates that the number is in Q format.
- m is the number of bits used to designate the two's complement integer part of the number.

- *n* is the number of bits used to designate the two's complement fractional part of the number, or the number of bits to the right of the binary point.

In Q format notation, the most significant bit is assumed to be the sign bit.

See also binary point, bit, data type, fixed-point representation, fraction, integer, two's complement

quantization

Representation of a value by a data type that has too few bits to represent it exactly.

See also bit, data type, quantization error

quantization error

Error introduced when a value is represented by a data type that has too few bits to represent it exactly, or when a value is converted from one data type to a shorter data type. Quantization error is also called quantization noise.

See also bit, data type, quantization

radix point

Symbol in the shape of a period that separates the integer and fractional parts of a number in any base system. Bits to the left of the radix point are integer and/or sign bits, and bits to the right of the radix point are fraction bits.

See also binary point, bit, fraction, integer, sign bit

range

Span of numbers that a certain data type can represent.

See also data type, precision

real-world value

Stored integer value with fixed-point scaling applied. Fixed-point numbers can be represented as

$$real\text{ - }world\ value = 2^{-fraction\ length} \times stored\ integer$$

or

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

where the slope can be expressed as

$$slope = fractional\ slope \times 2^{exponent}$$

See also integer

resolution

See **precision**

rounding

Limiting the number of bits required to express a number. One or more least significant bits are dropped, resulting in a loss of precision. Rounding is necessary when a value cannot be expressed exactly by the number of bits designated to represent it.

See also bit, ceiling (round toward), convergent rounding, floor (round toward), least significant bit, nearest (round toward), precision, truncation, zero (round toward)

saturation

Method of handling numeric overflow that represents positive overflows as the largest positive number in the range of the data type being used, and negative overflows as the largest negative number in the range.

See also overflow, wrapping

scaled double

A double data type that retains fixed-point scaling information. For example, in Simulink and Fixed-Point Toolbox software you can use data type override to convert your fixed-point data types to scaled doubles. You can then simulate to determine the ideal floating-point behavior of your system. After you gather that information you can turn data type override off to return to fixed-point data types, and your quantities still have their original scaling information because it was held in the scaled double data types.

scaling

1. Format used for a fixed-point number of a given word length and signedness. The slope and bias together form the scaling of a fixed-point number.

2. Changing the slope and/or bias of a fixed-point number without changing the stored integer.

See also bias, fixed-point representation, integer, slope

shift

Movement of the bits of a binary word either toward the most significant bit ("to the left") or toward the least significant bit ("to the right"). Shifts to the right can be either logical, where the spaces emptied at the front of the word with each shift are filled in with zeros, or arithmetic, where the word is sign extended as it is shifted to the right.

See also arithmetic shift, logical shift, sign extension

sign bit

Bit (or bits) in a signed binary number that indicates whether the number is positive or negative.

See also binary number, bit

sign extension

Addition of bits that have the value of the most significant bit to the high end of a two's complement number. Sign extension does not change the value of the binary number.

See also binary number, guard bits, most significant bit, two's complement representation, word

sign/magnitude representation

Representation of signed fixed-point or floating-point numbers. In sign/magnitude representation, one bit of a binary word is always the dedicated sign bit, while the remaining bits of the word encode the magnitude of the number. Negation using sign/magnitude representation consists of flipping the sign bit from 0 (positive) to 1 (negative), or from 1 to 0.

See also binary word, bit, fixed-point representation, floating-point representation, one's complement representation, sign bit, signed fixed-point, signedness, two's complement representation

signed fixed-point

Fixed-point number or data type that can represent both positive and negative numbers.

See also data type, fixed-point representation, signedness, unsigned fixed-point

signedness

The signedness of a number or data type can be signed or unsigned. Signed numbers and data types can represent both positive and negative values, whereas unsigned numbers and data types can only represent values that are greater than or equal to zero.

See also data type, sign bit, sign/magnitude representation, signed fixed-point, unsigned fixed-point

slope

Part of the numerical representation used to express a fixed-point number. Along with the bias, the slope forms the scaling of a fixed-point number. Fixed-point numbers can be represented as

$$\textit{real-world value} = (\textit{slope} \times \textit{stored integer}) + \textit{bias}$$

where the slope can be expressed as

$$\textit{slope} = \textit{fractional slope} \times 2^{\textit{exponent}}$$

See also bias, fixed-point representation, fractional slope, integer, scaling, [Slope Bias]

slope adjustment

See **fractional slope**

[Slope Bias]

Representation used to define the scaling of a fixed-point number.

See also bias, scaling, slope

stored integer

See **integer**

trivial scaling

Scaling that results in the real-world value of a number being simply equal to its stored integer value:

$$real\text{-}world\ value = stored\ integer$$

In [Slope Bias] representation, fixed-point numbers can be represented as

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

In the trivial case, slope = 1 and bias = 0.

In terms of binary point-only scaling, the binary point is to the right of the least significant bit for trivial scaling, meaning that the fraction length is zero:

$$real\text{-}world\ value = stored\ integer \times 2^{-fraction\ length} = stored\ integer \times 2^{0}$$

Scaling is always trivial for pure integers, such as `int8`, and also for the true floating-point types `single` and `double`.

See also bias, binary point, binary point-only scaling, fixed-point representation, fraction length, integer, least significant bit, scaling, slope, [Slope Bias]

truncation

Rounding mode that drops one or more least significant bits from a number.

See also ceiling (round toward), convergent rounding, floor (round toward), nearest (round toward), rounding, zero (round toward)

two's complement representation

Common representation of signed fixed-point numbers. Negation using signed two's complement representation consists of a translation into one's complement followed by the binary addition of a one.

See also binary word, one's complement representation, sign/magnitude representation, signed fixed-point

unsigned fixed-point

Fixed-point number or data type that can only represent numbers greater than or equal to zero.

See also data type, fixed-point representation, signed fixed-point, signedness

word

Fixed-length sequence of binary digits (1's and 0's). In digital hardware, numbers are stored in words. The way hardware components or software functions interpret this sequence of 1's and 0's is described by a data type.

See also binary word, data type

word length

Number of bits in a binary word or data type.

See also binary word, bit, data type

wrapping

Method of handling overflow. Wrapping uses modulo arithmetic to cast a number that falls outside of the representable range the data type being used back into the representable range.

See also data type, overflow, range, saturation

zero (round toward)

Rounding mode that rounds to the closest representable number in the direction of zero. This is equivalent to the `fix` mode in Fixed-Point Toolbox software.

See also ceiling (round toward), convergent rounding, floor (round toward), nearest (round toward), rounding, truncation

Index

C

D

E

F

N

O

P

Q

R

S

T

U