Signal Processing Toolbox

For Use with MATLAB®

Computation

Visualization

Programming

The MATH WORKS Inc.

User's Guide

Version 4.2

Meta Platforms, Inc. - Exhibit 1009
Page 1 of 720

How to Contact The MathWorks:

508-647-7000	Phone
508-647-7001	Fax
The MathWorks, Inc. 24 Prime Park Way Natick, MA 01760-1500	Mail
`http://www.mathworks.com`	Web
`ftp.mathworks.com`	Anonymous FTP server
`comp.soft-sys.matlab`	Newsgroup
`support@mathworks.com`	Technical support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`subscribe@mathworks.com`	Subscribing user registration
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

Signal Processing Toolbox User's Guide

© COPYRIGHT 1988 - 1999 by The MathWorks, Inc. All Rights Reserved.

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.

U.S. GOVERNMENT: If Licensee is acquiring the Programs on behalf of any unit or agency of the U.S. Government, the following shall apply: (a) For units of the Department of Defense: the Government shall have only the rights specified in the license under which the commercial computer software or commercial software documentation was obtained, as set forth in subparagraph (a) of the Rights in Commercial Computer Software or Commercial Software Documentation Clause at DFARS 227.7202-3, therefore the rights set forth herein shall apply; and (b) For any other unit or agency: NOTICE: Notwithstanding any other lease or license agreement that may pertain to, or accompany the delivery of, the computer software and accompanying documentation, the rights of the Government regarding its use, reproduction, and disclosure are as set forth in Clause 52.227-19 (c)(2) of the FAR.

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and Target Language Compiler is a trademark of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing History:	December 1996	First printing	New for MATLAB 5.0
	January 1998	Second printing	Revised for MATLAB 5.2
	January 1999	(Online only)	Revised for Version 4.2 (Release 11)

Contents

Before You Begin

What Is the Signal Processing Toolbox? xii

How to Use This Manual xiii

Installation xv

Typographical Conventions xvi
Technical Notations xvii

1 Signal Processing Basics

Signal Processing Toolbox Central Features 1-2
Filtering and FFTs 1-2
Signals and Systems 1-2
Key Areas: Filter Design and Spectral Analysis 1-3
Graphical User Interface (GUI) 1-3
Extensibility 1-3

Representing Signals 1-4
Vector Representation 1-4

Waveform Generation: Time Vectors and Sinusoids 1-6
Common Sequences: Unit Impulse, Unit Step, and Unit Ramp 1-7
Multichannel Signals 1-7
Common Periodic Waveforms 1-7
Common Aperiodic Waveforms 1-8
The pulstran Function 1-9
The Sinc Function 1-10
The Dirichlet Function 1-11

Meta Platforms, Inc. - Exhibit 1009
Page 3 of 720

Working with Data 1-13

Filter Implementation and Analysis 1-14
Convolution and Filtering 1-14
Filters and Transfer Functions 1-15
Filter Coefficients and Filter Names 1-15
Filtering with the filter Function 1-16

filter Function Implementation and Initial Conditions . . . 1-17

Other Functions for Filtering 1-19
Multirate Filter Bank Implementation 1-19
Anti-Causal, Zero-Phase Filter Implementation 1-20
Frequency Domain Filter Implementation 1-22

Impulse Response 1-23

Frequency Response 1-24
Digital Domain 1-24
Analog Domain 1-26
Magnitude and Phase 1-26
Delay 1-28

Zero-Pole Analysis 1-30

Linear System Models 1-32
Discrete-Time System Models 1-32
Transfer Function 1-32
Zero-Pole-Gain 1-33
State-Space 1-34
Partial Fraction Expansion (Residue Form) 1-35
Second-Order Sections (SOS) 1-36
Lattice Structure 1-37
Convolution Matrix 1-39
Continuous-Time System Models 1-40
Linear System Transformations 1-41

Meta Platforms, Inc. - Exhibit 1009
Page 4 of 720

Discrete Fourier Transform 1-43

References 1-46

2 Filter Design

Filter Requirements and Specification 2-2

IIR Filter Design 2-4
Classical IIR Filter Design Using Analog Prototyping 2-6
Complete Classical IIR Filter Design 2-6
Designing IIR Filters to Frequency Domain Specifications . 2-7
Comparison of Classical IIR Filter Types 2-8
Butterworth Filter 2-8
Chebyshev Type I Filter 2-9
Chebyshev Type II Filter 2-10
Elliptic Filter 2-10
Bessel Filter 2-11
Direct IIR Filter Design 2-13
Generalized Butterworth Filter Design 2-14

FIR Filter Design 2-16
Linear Phase Filters 2-17
Windowing Method 2-18
Standard Band FIR Filter Design: fir1 2-20
Multiband FIR Filter Design: fir2 2-21
Multiband FIR Filter Design with Transition Bands 2-22
Basic Configurations 2-22
The Weight Vector 2-24
Anti-Symmetric Filters / Hilbert Transformers 2-25
Differentiators 2-26
Constrained Least Squares FIR Filter Design 2-27
Basic Lowpass and Highpass CLS Filter Design 2-28
Multiband CLS Filter Design 2-29
Weighted CLS Filter Design 2-30

Meta Platforms, Inc. - Exhibit 1009
Page 5 of 720

Arbitrary-Response Filter Design 2-31
Multiband Filter Design 2-32
Filter Design with Reduced Delay 2-34

Special Topics in IIR Filter Design 2-37
Analog Prototype Design 2-38
Frequency Transformation 2-38
Filter Discretization 2-41
Impulse Invariance 2-42
Bilinear Transformation 2-43

References 2-46

3 Statistical Signal Processing

Correlation and Covariance 3-2
Bias and Normalization 3-3
Multiple Channels 3-4

Spectral Analysis 3-5
Welch's Method 3-6
Power Spectral Density Function 3-10
Bias and Normalization in Welch's Method 3-12
Cross-Spectral Density Function 3-14
Confidence Intervals 3-14
Transfer Function Estimate 3-14
Coherence Function 3-15
Multitaper Method 3-16
Yule-Walker AR Method 3-19
Burg Method 3-20
Covariance and Modified Covariance Methods 3-22
MUSIC and Eigenvector Analysis Methods 3-23
Eigenanalysis Overview 3-24
Controlling Subspace Thresholds 3-25

Meta Platforms, Inc. - Exhibit 1009
Page 6 of 720

References 3-27

4 Special Topics

Windows 4-2
Basic Shapes 4-2
Generalized Cosine Windows 4-4
Kaiser Window 4-4
Kaiser Windows in FIR Design 4-7
Chebyshev Window 4-9

Parametric Modeling 4-10
Time-Domain Based Modeling 4-11
Linear Prediction 4-12
Prony's Method (ARMA Modeling) 4-13
Steiglitz-McBride Method (ARMA Modeling) 4-15
Frequency-Domain Based Modeling 4-16

Resampling 4-20

Cepstrum Analysis 4-23
Inverse Complex Cepstrum 4-25

FFT-Based Time-Frequency Analysis 4-27

Median Filtering 4-28

Communications Applications 4-29

Deconvolution 4-33

Specialized Transforms 4-34
Chirp z-Transform 4-34
Discrete Cosine Transform 4-36
Hilbert Transform 4-38

Meta Platforms, Inc. - Exhibit 1009
Page 7 of 720

References **4-40**

5 Interactive Tools

SPTool: An Interactive Signal Processing Environment . . . **5-2**
Overview **5-2**

Using SPTool **5-3**
Opening SPTool **5-3**
Quick Start **5-3**
Example: Importing Signal Data from a MAT-File **5-3**
Basic SPTool Functions **5-5**
File Menu **5-6**
Help Menu **5-6**
Importing Signals, Filters, and Spectra **5-7**
Loading Variables from the MATLAB Workspace **5-7**
Loading Variables from Disk **5-8**
Importing Workspace Contents and File Contents **5-8**
Working with Signals, Filters, and Spectra **5-13**
Component Lists in SPTool **5-14**
Selecting Data Objects in SPTool **5-15**
Editing Data Objects in SPTool **5-15**
Viewing a Signal **5-17**
Viewing a Filter **5-17**
Designing a Filter **5-17**
Applying a Filter **5-18**
Creating a Spectrum **5-19**
Viewing a Spectrum **5-19**
Updating a Spectrum **5-19**

Meta Platforms, Inc. - Exhibit 1009
Page 8 of 720

Customizing Preferences 5-20
Ruler Settings 5-21
Color Settings 5-22
Signal Browser Settings 5-23
Spectrum Viewer Settings 5-24
Filter Viewer Settings 5-25
Filter Viewer Tiling Settings 5-26
Filter Designer Settings 5-27
Default Session Setting 5-28
Exporting Components Setting 5-29
Plug-Ins Setting 5-30
Saving and Discarding Changes to Preferences Settings . . 5-30
Controls for Viewing and Measuring 5-31
Zoom Controls 5-31
Ruler Controls 5-33
Making Signal Measurements 5-37

Using the Signal Browser: Interactive Signal Analysis . . . 5-43
Opening the Signal Browser 5-43
Basic Signal Browser Functions 5-44
Menus 5-45
Zoom Controls 5-46
Ruler and Line Display Controls 5-46
Help Button 5-46
Display Management Controls 5-47
Main Axes Display Area 5-47
Panner 5-48
Making Signal Measurements 5-49
Viewing and Exploring Signals 5-49
Selecting and Displaying a Signal 5-49
Panner Display 5-52
Manipulating Displays 5-53
Working with Signals 5-54
Printing Signal Data 5-54
Saving Signal Data 5-57

Using the Filter Designer: Interactive Filter Design 5-59
Opening the Filter Designer 5-59

Meta Platforms, Inc. - Exhibit 1009
Page 9 of 720

Basic Filter Designer Functions 5-60
Menus 5-60
Filter Pop-Up Menu 5-60
Zoom Controls 5-61
Help Button 5-61
General Controls 5-62
Filter Specifications Panel—Design Methods 5-63
Filter Measurements Panel—Design Methods 5-65
Filter Specifications Panel—Pole/Zero Editor 5-66
Filter Measurements Panel—Pole/Zero Editor 5-68
Magnitude Plot (Display) Area—Design Methods 5-69
Magnitude Plot (Display) Area—Pole/Zero Editor 5-71
Designing Finite Impulse Response (FIR) Filters 5-73
Example: FIR Filter Design, Standard Band Configuration 5-73
Filter Design Options 5-75
Order Selection for FIR Filter Design 5-75
Designing Infinite Impulse Response (IIR) Filters 5-76
Example: Classical IIR Filter Design 5-76
Filter Design Options 5-77
Order Selection for IIR Filter Design 5-78
Redesigning a Filter Using the Magnitude Plot 5-78
Saving Filter Data 5-79
Viewing Frequency Response Plots 5-82

Using the Filter Viewer: Interactive Filter Analysis 5-84
Opening the Filter Viewer 5-84
Basic Filter Viewer Functions 5-84
Menus 5-86
Filter Identification Panel 5-86
Plots Panel 5-86
Frequency Axis Settings 5-87
Zoom Controls 5-87
Help Button 5-87
Main Plots Area 5-88

Meta Platforms, Inc. - Exhibit 1009
Page 10 of 720

Viewing Filter Plots 5-89
Viewing Magnitude Response 5-89
Viewing Phase Response 5-91
Viewing Group Delay 5-93
Viewing a Zero-Pole Plot 5-94
Viewing Impulse Response 5-94
Viewing Step Response 5-95

Using the Spectrum Viewer: Interactive PSD Analysis . . . 5-97
Opening the Spectrum Viewer 5-97
Basic Spectrum Viewer Functions 5-98
Menus 5-99
Signal ID Panel 5-100
Spectrum Management Buttons 5-100
Zoom Controls 5-101
Ruler and Line Display Controls 5-101
Help Button 5-101
Main Axes Display Area 5-101
Making Spectrum Measurements 5-102
Viewing Spectral Density Plots 5-102
Controlling and Manipulating Plots 5-102
Changing Plot Properties 5-102
Choosing Computation Parameters 5-103
Computation Methods and Parameters 5-104
Setting Confidence Intervals 5-107
Printing Spectrum Data 5-107
Saving Spectrum Data 5-110

Example: Generation of Bandlimited Noise 5-113
Create, Import, and Name a Signal 5-113
Design a Filter 5-115
Apply the Filter to a Signal 5-116
View, Play, and Print the Signals 5-117
Compare Spectra of Both Signals 5-120

6 Reference

Meta Platforms, Inc. - Exhibit 1009
Page 11 of 720

Meta Platforms, Inc. - Exhibit 1009
Page 12 of 720

Before You Begin

What Is the Signal Processing Toolbox? xii

How to Use This Manual xiii

Installation xv

Typographical Conventions xvi
Technical Notations xvii

What Is the Signal Processing Toolbox?

This section describes how to begin using the Signal Processing Toolbox. It explains how to use this manual and points you to additional books for toolbox installation information.

The Signal Processing Toolbox is a collection of tools built on the MATLAB® numeric computing environment. The toolbox supports a wide range of signal processing operations, from waveform generation to filter design and implementation, parametric modeling, and spectral analysis. The toolbox provides two categories of tools:

- Signal processing functions
- Graphical, interactive tools

The first category of tools is made up of functions that you can call from the command line or from your own applications. Many of these functions are MATLAB M-files, series of MATLAB statements that implement specialized signal processing algorithms. You can view the MATLAB code for these functions using the statement

```
type function_name
```

or by opening the M-file in the MATLAB Editor/Debugger. You can change the way any toolbox function works by copying and renaming the M-file, then modifying your copy. You can also extend the toolbox by adding your own M-files.

Second, the toolbox provides a number of interactive tools that let you access many of the functions through a *graphical user interface* (GUI). The GUI-based tools provide an integrated environment for filter design, analysis, and implementation, as well as signal exploration and editing. For example, with the graphical user interface tools you can:

- Use the mouse to graphically edit the magnitude response of a filter or measure the slope of a signal with onscreen rulers.
- Play a signal on your system's audio hardware by selecting a menu item or pressing a corresponding keystroke combination.
- Customize the parameters and method of computing the spectrum of a signal.

Meta Platforms, Inc. - Exhibit 1009
Page 14 of 720

How to Use This Manual

If you are a new user. Begin with Chapter 1, “Signal Processing Basics.” This chapter introduces the MATLAB signal processing environment through the toolbox functions. It describes the basic functions of the Signal Processing Toolbox, reviewing its use in basic waveform generation, filter implementation and analysis, impulse and frequency response, zero-pole analysis, linear system models, and the discrete Fourier transform.

When you feel comfortable with the basic functions, move on to Chapter 2 and Chapter 3 for a more in-depth introduction to using the Signal Processing Toolbox:

- Chapter 2, “Filter Design,” for a detailed explanation of using the Signal Processing Toolbox in infinite impulse response (IIR) and finite impulse response (FIR) filter design and implementation, including special topics in IIR filter design.
- Chapter 3, “Statistical Signal Processing,” for how to use the correlation, covariance, and spectral analysis tools to estimate important functions of discrete random signals.

Once you understand the general principles and applications of the toolbox, learn how to use the interactive tools.

- Chapter 5, “Interactive Tools,” for an overview of the interactive GUI environment and examples of how to use it for signal exploration, filter design and implementation, and spectral analysis.

Finally, see the following chapter for a discussion of specialized toolbox functions.

- Chapter 4, “Special Topics,” for specialized functions, including filter windows, parametric modeling, resampling, cepstrum analysis, time-dependent Fourier transforms and spectrograms, median filtering, communications applications, deconvolution, and specialized transforms.

If you are an experienced toolbox user. See Chapter 5, “Interactive Tools,” for an overview of the interactive GUI environment and examples of how to use it for signal viewing, filter design and implementation, and spectral analysis.

Meta Platforms, Inc. - Exhibit 1009
Page 15 of 720

All toolbox users. Use Chapter 6, "Reference," for locating information on specific functions. Reference descriptions include a synopsis of the function's syntax, as well as a complete explanation of options and operations. Many reference descriptions also include helpful examples, a description of the function's algorithm, and references to additional reading material.

Use this manual in conjunction with the software to learn about the powerful features that MATLAB provides. Each chapter provides numerous examples that apply the toolbox to representative signal processing tasks.

Some examples use MATLAB's random number generation function `randn`. In these cases, to duplicate the results in the example, type

```
randn('seed',0)
```

before running the example.

Meta Platforms, Inc. - Exhibit 1009
Page 16 of 720

Installation

To install this toolbox on a workstation, see the *MATLAB Installation Guide for UNIX*. To install the toolbox on a PC, see the *MATLAB PC Installation Guide*.

To determine if the Signal Processing Toolbox is already installed on your system, check for a subdirectory named `signal` within the main toolbox directory or folder.

Meta Platforms, Inc. - Exhibit 1009
Page 17 of 720

Typographical Conventions

To Indicate	This Manual Uses	Example
Example code	`Monospace type.`	To assign the value 5 to `A`, enter `A = 5`

MATLAB output	`Monospace type.`	MATLAB responds with
		`A =`
		`5`

Function names	`Monospace type.`	The `cos` function finds the cosine of each array element.
New terms	*Italics.*	An *array* is an ordered collection of information.
Keys	**Boldface** with an initial capital letter.	Press the **Return** key.
Menu names, items, and GUI controls	**Boldface** with an initial capital letter.	Choose the **File** menu.
Mathematical expressions	Variables in *italics*. Functions, operators, and constants in standard type.	This vector represents the polynomial $p = x^2 + 2x + 3$

Meta Platforms, Inc. - Exhibit 1009
Page 18 of 720

Technical Notations

This manual and the Signal Processing Toolbox functions use the following technical notations:

Nyquist frequency	One-half the sampling frequency. Most toolbox functions normalize this value to 1.
`x(1)`	The first element of a data sequence or filter, corresponding to zero lag.
Ω	Analog frequency in radians per second.
ω or `w`	Digital frequency in radians per second.
`f`	Digital frequency in Hertz.
$[x, y)$	The interval from x to y, including x but not including y

Meta Platforms, Inc. - Exhibit 1009
Page 19 of 720

Meta Platforms, Inc. - Exhibit 1009
Page 20 of 720

1

Signal Processing Basics

Signal Processing Toolbox Central Features 1-2

Representing Signals 1-4

Waveform Generation: Time Vectors and Sinusoids 1-6

Working with Data 1-13

Filter Implementation and Analysis 1-14

filter Function Implementation and Initial Conditions . 1-17

Other Functions for Filtering 1-19

Impulse Response 1-23

Frequency Response 1-24

Zero-Pole Analysis 1-30

Linear System Models 1-32

Discrete Fourier Transform 1-43

References 1-46

Signal Processing Toolbox Central Features

This chapter describes how to begin using MATLAB and the Signal Processing Toolbox for your signal processing applications. It assumes a basic knowledge and understanding of signals and systems, including such topics as filter and linear system theory and basic Fourier analysis.

Many examples throughout the chapter demonstrate how to apply toolbox functions. If you are not already familiar with MATLAB's signal processing capabilities, use this chapter in conjunction with the software to try examples and learn about the powerful features available to you.

The Signal Processing Toolbox functions are algorithms, expressed mostly in M-files, that implement a variety of signal processing tasks. These toolbox functions are a specialized extension of the MATLAB computational and graphical environment.

Filtering and FFTs

Two of the most important functions for signal processing are not in the Signal Processing Toolbox at all, but are built-in MATLAB functions:

- `filter` applies a digital filter to a data sequence.
- `fft` calculates the discrete Fourier transform of a sequence.

The operations these functions perform are the main computational workhorses of classical signal processing. Both are described in this chapter. The Signal Processing Toolbox uses many other standard MATLAB functions and language features, including polynomial root finding, complex arithmetic, matrix inversion and manipulation, and graphics tools.

Signals and Systems

The basic entities that toolbox functions work with are signals and systems. The functions emphasize digital, or discrete, signals and filters, as opposed to analog, or continuous, signals. The principal filter type the toolbox supports is the linear, time-invariant digital filter with a single input and a single output. You can represent linear time-invariant systems using one of several models (such as transfer function, state-space, zero-pole-gain, and second-order section) and convert between representations.

Meta Platforms, Inc. - Exhibit 1009
Page 22 of 720

Key Areas: Filter Design and Spectral Analysis

In addition to its core functions, the toolbox provides rich, customizable support for the key areas of filter design and spectral analysis. It is easy to implement a design technique that suits your application, design digital filters directly, or create analog prototypes and discretize them. Toolbox functions also estimate power spectral density and cross spectral density, using either parametric or nonparametric techniques. Chapters 2 and 3, respectively, detail toolbox functions for filter design and spectral analysis.

There are functions for computation and graphical display of frequency response, as well as functions for system identification; generating signals; discrete cosine, chirp-*z*, and Hilbert transforms; lattice filters; resampling; time-frequency analysis; and basic communication systems simulation.

Graphical User Interface (GUI)

The power of the Signal Processing Toolbox is greatly enhanced by its easy-to-use graphical user interface. The GUI provides an integrated set of interactive tools for performing a wide variety of signal processing tasks. These tools enable you to use the mouse and menus to manipulate a rich graphical environment for signal viewing, filter design and implementation, and spectral analysis.

Extensibility

Perhaps the most important feature of the MATLAB environment is that it is extensible: MATLAB lets you create your own M-files to meet numeric computation needs for research, design, or engineering of signal processing systems. Simply copy the M-files provided with the Signal Processing Toolbox and modify them as needed, or create new functions to expand the functionality of the toolbox.

Meta Platforms, Inc. - Exhibit 1009
Page 23 of 720

Representing Signals

The central data construct in MATLAB is the *numeric array*, an ordered collection of real or complex numeric data with two or more dimensions. The basic data objects of signal processing (one-dimensional signals or sequences, multichannel signals, and two-dimensional signals) are all naturally suited to array representation.

Vector Representation

MATLAB represents ordinary one-dimensional sampled data signals, or sequences, as *vectors*. Vectors are 1-by-*n* or *n*-by-1 arrays, where *n* is the number of samples in the sequence. One way to introduce a sequence into MATLAB is to enter it as a list of elements at the command prompt. The statement

```
x = [4 3 7 –9 1]
```

creates a simple five-element real sequence in a row vector. Transposition turns the sequence into a column vector

```
x = x'
```

resulting in

```
x =
     4
     3
     7
    –9
     1
```

Column orientation is preferable for single channel signals because it extends naturally to the multichannel case. For multichannel data, each column of a matrix represents one channel. Each row of such a matrix then corresponds to a sample point. A three-channel signal that consists of `x`, `2x`, and `x/π` is

```
y = [x 2*x x/pi]
```


Meta Platforms, Inc. - Exhibit 1009
Page 24 of 720

This results in

```
y =

    4.0000    8.0000    1.2732
    3.0000    6.0000    0.9549
    7.0000   14.0000    2.2282
   -9.0000  -18.0000   -2.8648
    1.0000    2.0000    0.3183
```


Meta Platforms, Inc. - Exhibit 1009
Page 25 of 720

Waveform Generation: Time Vectors and Sinusoids

A variety of toolbox functions generate waveforms. Most require you to begin with a vector representing a time base. Consider generating data with a 1000 Hz sample frequency, for example. An appropriate time vector is

```
t = (0:0.001:1)';
```

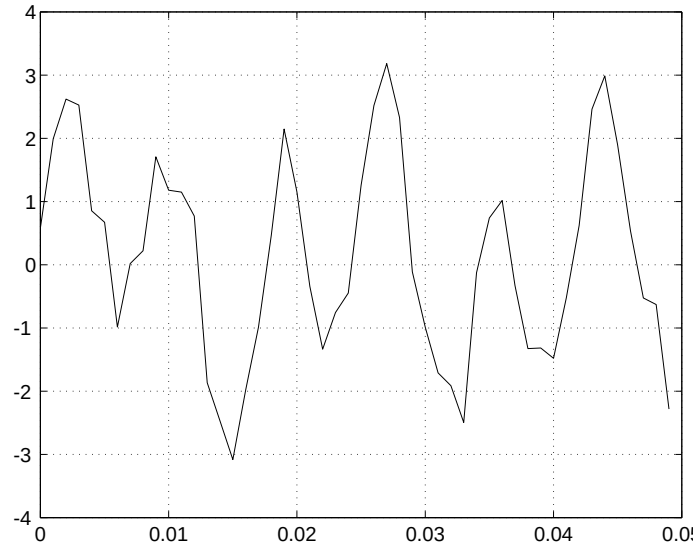
where MATLAB's colon operator creates a 1001-element row vector that represents time running from zero to one second in steps of one millisecond. The transpose operator (') changes the row vector into a column; the semicolon (;) tells MATLAB to compute but not display the result.

Given `t` you can create a sample signal `y` consisting of two sinusoids, one at 50 Hz and one at 120 Hz with twice the amplitude:

```
y = sin(2*pi*50*t) + 2*sin(2*pi*120*t);
```

The new variable `y`, formed from vector `t`, is also 1001 elements long. You can add normally distributed white noise to the signal and graph the first fifty points using

```
yn = y + 0.5*randn(size(t));
plot(t(1:50),yn(1:50))
```



Meta Platforms, Inc. - Exhibit 1009
Page 26 of 720

Common Sequences: Unit Impulse, Unit Step, and Unit Ramp

Since MATLAB is a programming language, an endless variety of different signals is possible. Here are some statements that generate several commonly used sequences, including the unit impulse, unit step, and unit ramp functions:

```
t = (0:0.001:1)';
y = [1; zeros(99,1)];  % impulse
y = ones(100,1);       % step (filter assumes 0 initial cond.)
y = t;                 % ramp
y = t.^2;
y = square(4*t);
```

All of these sequences are column vectors – the last three inherit their shapes from `t`.

Multichannel Signals

Use standard MATLAB array syntax to work with multichannel signals. For example, a multichannel signal consisting of the last three signals generated above is

```
z = [t t.^2 square(4*t)];
```

You can generate a multichannel unit sample function using the outer product operator. For example, a six-element column vector whose first element is one, and whose remaining five elements are zeros, is

```
a = [1 zeros(1,5)]';
```

To duplicate column vector `a` into a matrix without performing any multiplication, use MATLAB's colon operator and the `ones` function.

```
c = a(:,ones(1,3));
```

Common Periodic Waveforms

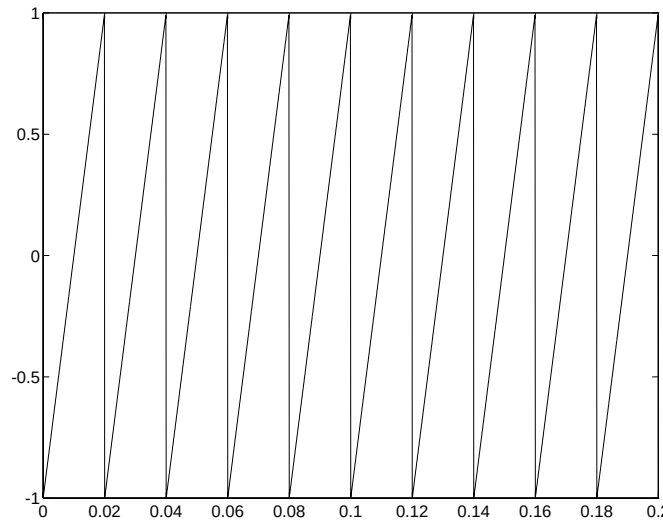
The toolbox provides functions for generating widely used periodic waveforms:

Meta Platforms, Inc. - Exhibit 1009
Page 27 of 720

- `sawtooth` generates a sawtooth wave with peaks at ±1 and a period of 2π. An optional `width` parameter specifies a fractional multiple of 2π at which the signal's maximum occurs.
- `square` generates a square wave with a period of 2π. An optional parameter specifies *duty cycle*, the percent of the period for which the signal is positive.

To generate 1.5 seconds of a 50 Hz sawtooth wave with a sample rate of 10 kHz and plot 0.2 seconds of the generated waveform, use

```
Fs = 10000;
t = 0:1/Fs:1.5;
x = sawtooth(2*pi*50*t);
plot(t,x), axis([0 0.2 –1 1])
```



Common Aperiodic Waveforms

The toolbox also provides functions for generating several widely used aperiodic waveforms:

- `gauspuls` generates a Gaussian-modulated sinusoidal pulse with a specified time, center frequency, and fractional bandwidth. Optional parameters

Meta Platforms, Inc. - Exhibit 1009
Page 28 of 720

return in-phase and quadrature pulses, the RF signal envelope, and the cutoff time for the trailing pulse envelope.

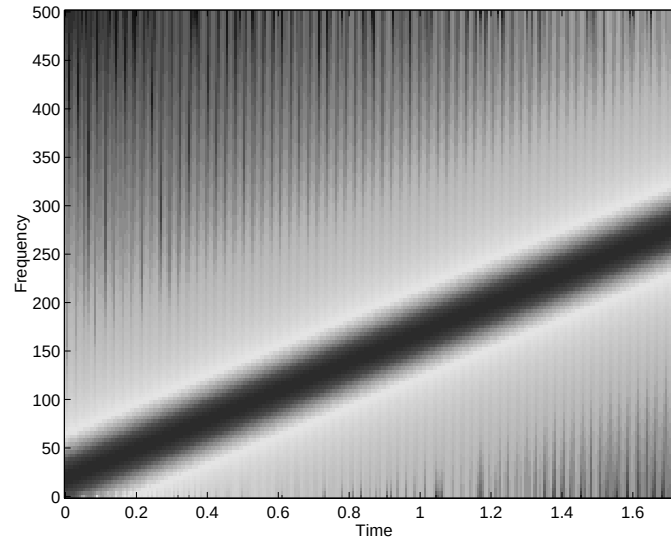
- `chirp` generates a linear swept-frequency cosine signal. An optional parameter specifies alternative sweep methods. An optional parameter `phi` allows initial phase to be specified in degrees.

To compute 2 seconds of a linear chirp signal with a sample rate of 1 kHz, that starts at DC and crosses 150 Hz at 1 second, use

```
t = 0:1/1000:2;
y = chirp(t,0,1,150);
```

To plot the spectrogram, use

```
specgram(y,256,1000,256,250)
```



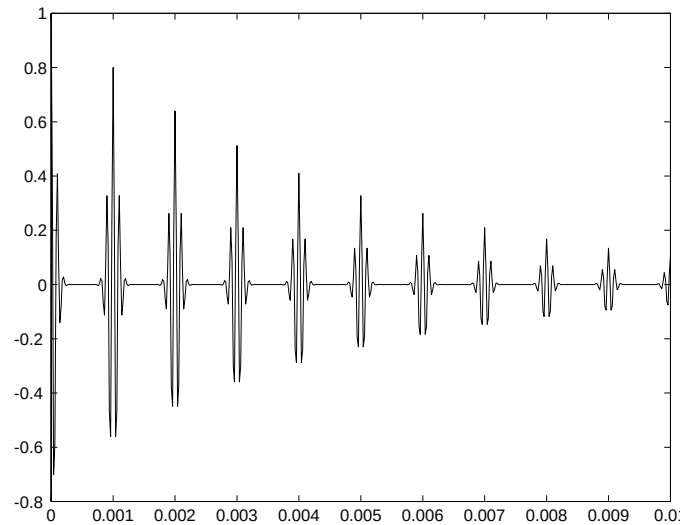
The pulstran Function

The `pulstran` function generates pulse trains from either continuous or sampled prototype pulses. The following example generates a pulse train consisting of the sum of multiple delayed interpolations of a Gaussian pulse. The pulse train is defined to have a sample rate of 50 kHz, a pulse train length

Meta Platforms, Inc. - Exhibit 1009
Page 29 of 720

of 10 ms, and a pulse repetition rate of 1 kHz; D specifies the delay to each pulse repetition in column 1 and an optional attenuation for each repetition in column 2. The pulse train is constructed by passing the name of the gauspuls function to pulstran, along with additional parameters that specify a 10 kHz Gaussian pulse with 50% bandwidth:

```
T = 0:1/50E3:10E-3;
D = [0:1/1E3:10E-3;0.8.^(0:10)]';
Y = pulstran(T,D,'gauspuls',10E3,0.5);
plot(T,Y)
```



The Sinc Function

The sinc function computes the mathematical sinc function for an input vector or matrix x. The sinc function is the continuous inverse Fourier transform of the rectangular pulse of width 2π and height 1:

The sinc function has a value of 1 where x is zero, and a value of

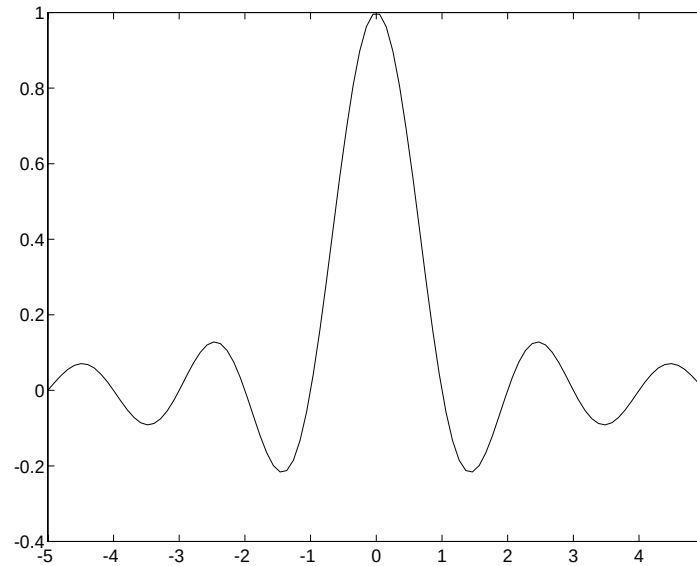
$$\frac{\sin(\pi x)}{\pi x}$$

for all other elements of x.

Meta Platforms, Inc. - Exhibit 1009
Page 30 of 720

To plot the sinc function for a linearly spaced vector with values ranging from –5 to 5,

```
x = linspace(–5,5);
y = sinc(x);
plot(x,y)
```



The Dirichlet Function

The toolbox function `diric` computes the Dirichlet function, sometimes called the *periodic sinc* or *aliased sinc* function, for an input vector or matrix `x`. The Dirichlet function is

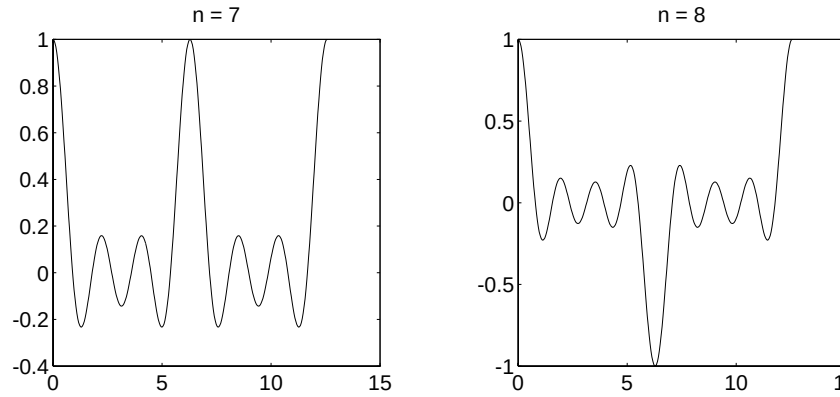
$$\mathrm{diric}(x) = \begin{cases} -1^{k(n-1)} & x = 2\pi k, \quad k = 0, \pm 1, \pm 2, \ldots \\ \dfrac{\sin(nx/2)}{n\sin(x/2)} & \text{otherwise} \end{cases}$$

where n is a user-specified positive integer. For n odd, the Dirichlet function has a period of 2π; for n even, its period is 4π. The magnitude of this function is $(1/n)$ times the magnitude of the discrete-time Fourier transform of the n-point rectangular window.

Meta Platforms, Inc. - Exhibit 1009
Page 31 of 720

To plot the Dirichlet function over the range 0 to 4π for `n = 7` and `n = 8`, use

```
x = linspace(0,4*pi,300);
plot(x,diric(x,7))
plot(x,diric(x,8))
```



Meta Platforms, Inc. - Exhibit 1009
Page 32 of 720

Working with Data

The examples in the preceding sections obtain data in one of two ways:

- By direct input, that is, entering the data manually at the keyboard
- By using a MATLAB or toolbox function, such as `sin`, `cos`, `sawtooth`, `square`, or `sinc`

Some applications, however, may need to import data from outside MATLAB. Depending on your data format, you can do this in the following ways:

- Load data from an ASCII file or MAT-file with MATLAB's `load` command
- Read the data into MATLAB with a low-level file I/O function, such as `fopen`, `fread`, and `fscanf`
- Develop a MEX-file to read the data

Other resources are also useful, such as a high-level language program (in Fortran or C, for example) that converts your data into MAT-file format—see the *MATLAB Application Programming Interface* reference manual for details. MATLAB reads such files using the `load` command.

Similar techniques are available for exporting data generated within MATLAB. See *Using MATLAB* for more details on importing and exporting data, and see the online *MATLAB Function Reference* for descriptions of file loading and I/O routines.

Meta Platforms, Inc. - Exhibit 1009
Page 33 of 720

Filter Implementation and Analysis

This section describes how to filter discrete signals using MATLAB's `filter` function and other functions in the Signal Processing Toolbox. It also discusses how to use the toolbox functions to analyze filter characteristics, including impulse response, magnitude and phase response, group delay, and zero-pole locations.

Convolution and Filtering

The mathematical foundation of filtering is convolution. MATLAB's `conv` function performs standard one-dimensional convolution, convolving one vector with another.

```
conv([1 1 1],[1 1 1])

ans =

     1     2     3     2     1
```

NOTE Convolve rectangular matrices for two-dimensional signal processing using the `conv2` function.

A digital filter's output $y(n)$ is related to its input $x(n)$ by convolution with its impulse response $h(n)$.

$$y(n) = h(n) * x(n) = \sum_{m=-\infty}^{\infty} h(n-m)x(m)$$

If a digital filter's impulse response $h(n)$ is finite length, and the input $x(n)$ is also finite length, you can implement the filter using `conv`. Store $x(n)$ in a vector `x`, $h(n)$ in a vector `h`, and convolve the two.

```
x = randn(5,1);   % a random vector of length 5
h = [1 1 1 1]/4;  % length 4 averaging filter
y = conv(h,x);
```


Meta Platforms, Inc. - Exhibit 1009
Page 34 of 720

Filters and Transfer Functions

In general, the z-transform $Y(z)$ of a digital filter's output $y(n)$ is related to the z-transform $X(z)$ of the input by

$$Y(z) = H(z)X(z) = \frac{b(1) + b(2)z^{-1} + \cdots + b(nb+1)z^{-nb}}{a(1) + a(2)z^{-1} + \cdots + a(na+1)z^{-na}}X(z)$$

where $H(z)$ is the filter's *transfer function*. Here, the constants $b(i)$ and $a(i)$ are the filter coefficients and the order of the filter is the maximum of na and nb.

NOTE The filter coefficients start with subscript 1, rather than 0. This reflects MATLAB's standard indexing scheme for vectors.

MATLAB stores the coefficients in two vectors, one for the numerator and one for the denominator. By convention, MATLAB uses row vectors for filter coefficients.

Filter Coefficients and Filter Names

Many standard names for filters reflect the number of a and b coefficients present:

- When nb = 0 (that is, b is a scalar), the filter is an Infinite Impulse Response (IIR), all-pole, recursive, or autoregressive (AR) filter.
- When na = 0 (that is, a is a scalar), the filter is a Finite Impulse Response (FIR), all-zero, nonrecursive, or moving average (MA) filter.
- If both na and nb are greater than zero, the filter is an IIR, pole-zero, recursive, or autoregressive moving average (ARMA) filter.

The acronyms AR, MA, and ARMA are usually applied to filters associated with filtered stochastic processes.

Meta Platforms, Inc. - Exhibit 1009
Page 35 of 720

Filtering with the filter Function

It is simple to work back to a difference equation from the z-transform relation shown earlier. Assume that $a(1) = 1$. Move the denominator to the left-hand side and take the inverse z-transform.

$$y(n) + a(2)y(n-1) + \cdots + a(na+1)y(n-na) = b(1)x(n) + b(2)x(n-1) + \cdots + b(nb+1)x(n-nb)$$

In terms of current and past inputs, and past outputs, $y(n)$ is

$$y(n) = b(1)x(n) + b(2)x(n-1) + \cdots + b(nb+1)x(n-nb) - a(2)y(n-1) - \cdots - a(na+1)y(n-na)$$

This is the standard time-domain representation of a digital filter, computed starting with $y(1)$ and assuming zero initial conditions. This representation's progression is

$$\begin{aligned} y(1) &= b(1)x(1) \\ y(2) &= b(1)x(2) + b(2)x(1) - a(2)y(1) \\ y(3) &= b(1)x(3) + b(2)x(2) + b(3)x(1) - a(2)y(2) - a(3)y(1) \\ \vdots &= \vdots \end{aligned}$$

A filter in this form is easy to implement with the `filter` function. For example, a simple single-pole filter (lowpass) is

```
b = 1;            % numerator
a = [1 -0.9];     % denominator
```

where the vectors `b` and `a` represent the coefficients of a filter in transfer function form. To apply this filter to your data

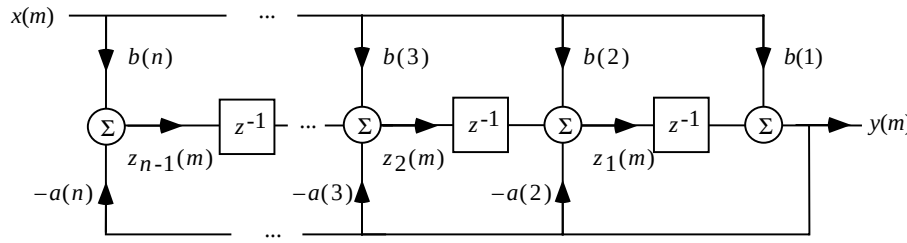
```
y = filter(b,a,x);
```

`filter` gives you as many output samples as there are input samples, that is, the length of `y` is the same as the length of `x`. If the first element of `a` is not 1, `filter` divides the coefficients by `a(1)` before implementing the difference equation.

Meta Platforms, Inc. - Exhibit 1009
Page 36 of 720

filter Function Implementation and Initial Conditions

`filter` is implemented as a transposed direct form II structure



where n-1 is the filter order. This is a canonical form that has the minimum number of delay elements.

At sample m, `filter` computes the difference equations

$$
\begin{aligned}
y(m) &= b(1)x(m) + z_1(m-1) \\
z_1(m) &= b(2)x(m) + z_2(m-1) - a(2)y(m) \\
\vdots &= \vdots \\
z_{n-2}(m) &= b(n-1)x(m) + z_{n-1}(m-1) - a(n-1)y(m) \\
z_{n-1}(m) &= b(n)x(m) - a(n)y(m)
\end{aligned}
$$

In its most basic form, `filter` initializes the delay outputs $z_i(1)$, $i = 1, \ldots, n$-1 to 0. This is equivalent to assuming both past inputs and outputs are zero. Set the initial delay outputs using a fourth input parameter to `filter`, or access the final delay outputs using a second output parameter.

```
[y,zf] = filter(b,a,x,zi)
```

Access to initial and final conditions is useful for filtering data in sections, especially if memory limitations are a consideration. Suppose you have collected data in two segments of 5000 points each.

```
x1 = randn(5000,1);  % two random sequences to
x2 = randn(5000,1);  % serve as simulated data
```

Perhaps the first sequence, `x1`, corresponds to the first 10 minutes of data and the second, `x2`, to an additional 10 minutes. The whole sequence is `x = [x1; x2]`. If there is not sufficient memory to hold the combined sequence,

Meta Platforms, Inc. - Exhibit 1009
Page 37 of 720

filter the subsequences x1 and x2 one at a time. To ensure continuity of the filtered sequences, use the final conditions from x1 as initial conditions to filter x2.

```
[y1,zf] = filter(b,a,x1);
y2 = filter(b,a,x2,zf);
```

The filtic function generates initial conditions for filter. filtic computes the delay vector to make the behavior of the filter reflect past inputs and outputs that you specify. To obtain the same output delay values zf as above using filtic

```
zf = filtic(b,a,flipud(y1),flipud(x1));
```

This can be useful when filtering short data sequences, as appropriate initial conditions help reduce transient startup effects.

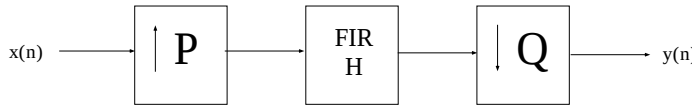
Meta Platforms, Inc. - Exhibit 1009
Page 38 of 720

Other Functions for Filtering

In addition to `filter`, several other functions in the Signal Processing Toolbox perform the basic filtering operation. These functions include `upfirdn`, which performs FIR filtering with resampling, `filtfilt`, which eliminates phase distortion in the filtering process, `fftfilt`, which performs the FIR filtering operation in the frequency domain, and `latcfilt`, which filters using a lattice implementation.

Multirate Filter Bank Implementation

The function `upfirdn` alters the sampling rate of a signal by an integer ratio P/Q. It computes the result of the cascade of three systems: (1) upsampling (zero insertion) by integer factor `p`, (2) filtering by FIR filter `h`, and (3) downsampling by integer factor `q`.



For example, to change the sample rate of a signal from 44.1 kHz to 48 kHz, we first find the smallest integer conversion ratio `p/q`.

```
d = gcd(48000,44100);
p = 48000/d;
q = 44100/d;
```

where we find that `p = 160` and `q = 147`. Sample rate conversion is then accomplished by `y = upfirdn(x,h,p,q)`. This cascade of operations is implemented in an efficient manner using polyphase filtering techniques, and it is a central concept of multirate filtering (see reference [1] for details on multirate filter theory). Note that the quality of the resampling result relies on the quality of the FIR filter `h`.

Filter banks may be implemented using `upfirdn` by allowing the filter `h` to be a matrix, with one FIR filter per column. A signal vector is passed independently through each FIR filter, resulting in a matrix of output signals.

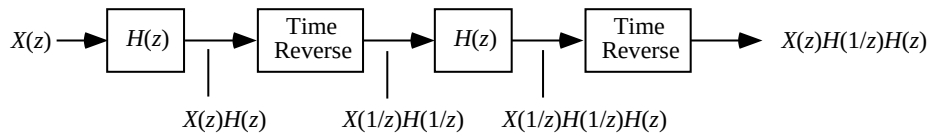
Other functions that perform multirate filtering (with fixed filter) include `resample`, `interp`, and `decimate`.

Meta Platforms, Inc. - Exhibit 1009
Page 39 of 720

Anti-Causal, Zero-Phase Filter Implementation

In the case of FIR filters, it is possible to design linear phase filters that, when applied to data (using `filter` or `conv`), simply delay the output by a fixed number of samples. For IIR filters, however, the phase distortion is usually highly nonlinear. The `filtfilt` function uses the information in the signal at points before and after the current point, in essence "looking into the future," to eliminate phase distortion.

To see how `filtfilt` does this, recall that if the z-transform of a real sequence $x(n)$ is $X(z)$, the z-transform of the time reversed sequence $x(n)$ is $X(1/z)$. Consider the processing scheme



When $|z| = 1$, that is $z = e^{j\omega}$, the output reduces to $X(e^{j\omega})|H(e^{j\omega})|^2$. Given all the samples of the sequence $x(n)$, a doubly filtered version of x that has zero-phase distortion is possible.

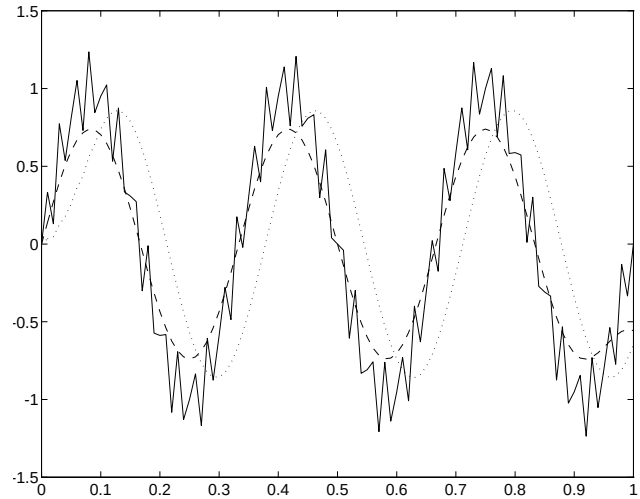
For example, a one-second duration signal sampled at 100 Hz, composed of two sinusoidal components at 3 Hz and 40 Hz, is

```
Fs = 100;
t = 0:1/Fs:1;
x = sin(2*pi*t*3)+.25*sin(2*pi*t*40);
```


Meta Platforms, Inc. - Exhibit 1009
Page 40 of 720

Now create a 10-point averaging FIR filter, and filter x using both `filter` and `filtfilt` for comparison:

```
b = ones(1,10)/10;              % 10 point averaging filter
y = filtfilt(b,1,x);            % non-causal filtering
yy = filter(b,1,x);             % normal filtering
plot(t,x,t,y,'--',t,yy,':')
```



Both filtered versions eliminate the 40 Hz sinusoid evident in the original, solid line. The plot also shows how `filter` and `filtfilt` differ; the dashed (`filtfilt`) line is in phase with the original 3 Hz sinusoid, while the dotted (`filter`) line is delayed by about five samples. Also, the amplitude of the dashed line is smaller due to the magnitude squared effects of `filtfilt`.

`filtfilt` reduces filter startup transients by carefully choosing initial conditions, and by prepending onto the input sequence a short, reflected piece of the input sequence. For best results, make sure the sequence you are filtering has length at least three times the filter order and tapers to zero on both edges.

Meta Platforms, Inc. - Exhibit 1009
Page 41 of 720

Frequency Domain Filter Implementation

Duality between the time domain and the frequency domain makes it possible to perform any operation in either domain. Usually one domain or the other is more convenient for a particular operation, but you can always accomplish a given operation in either domain.

To implement general IIR filtering in the frequency domain, multiply the discrete Fourier transform (DFT) of the input sequence with the quotient of the DFT of the filter,

```
n = length(x);
y = ifft(fft(x).*fft(b,n)./fft(a,n));
```

This computes results that are identical to `filter`, but with different startup transients (edge effects). For long sequences, this computation is very inefficient because of the large zero-padded FFT operations on the filter coefficients, and because the FFT algorithm becomes less efficient as the number of points `n` increases.

For FIR filters, however, it is possible to break longer sequences into shorter, computationally efficient FFT lengths. The function

```
y = fftfilt(b,x)
```

uses the overlap add method (see reference [1] at the end of this chapter) to filter a long sequence with multiple medium-length FFTs. Its output is equivalent to `filter(b,1,x)`.

Meta Platforms, Inc. - Exhibit 1009
Page 42 of 720

Impulse Response

The impulse response of a digital filter is the output arising from the input sequence

$$x(n) = \begin{cases} 1, & n = 1 \\ 0, & n \neq 1 \end{cases}$$

In MATLAB, you can generate an impulse sequence a number of ways; one straightforward way is

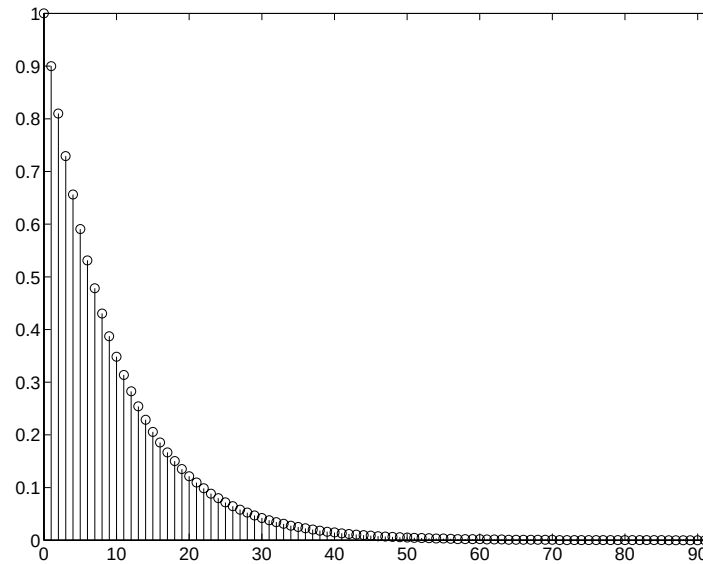
```
imp = [1; zeros(49,1)];
```

The impulse response of the simple filter `b = 1` and `a = [1 –0.9]` is

```
h = filter(b,a,imp);
```

The `impz` function in the toolbox simplifies this operation, choosing the number of points to generate and then making a stem plot (using the `stem` function).

```
impz(b,a)
```



The plot shows the exponential decay `h(n) = 0.9n` of the single pole system.

Meta Platforms, Inc. - Exhibit 1009
Page 43 of 720

Frequency Response

The Signal Processing Toolbox enables you to perform frequency domain analysis of both analog and digital filters.

Digital Domain

`freqz` uses an FFT-based algorithm to calculate the *z*-transform frequency response of a digital filter. Specifically, the statement

```
[h,w] = freqz(b,a,n)
```

returns the *n*-point complex frequency response, $H(e^{j\omega})$, of the digital filter.

$$H(e^{j\omega}) = \frac{b(1) + b(2)e^{-j\omega} + \cdots + b(nb+1)e^{-j\omega(nb)}}{a(1) + a(2)e^{-j\omega} + \cdots + a(na+1)e^{-j\omega(na)}}$$

In its simplest form, `freqz` accepts the filter coefficient vectors `b` and `a`, and an integer `n` specifying the number of points at which to calculate the frequency response. `freqz` returns the complex frequency response in vector `h`, and the actual frequency points in vector `w` in radians/second.

`freqz` can accept other parameters, such as a sampling frequency or a vector of arbitrary frequency points. The example below finds the 256-point frequency response for a 12th-order Chebyshev type I filter. The call to `freqz` specifies a sampling frequency `Fs` of 1000 Hz.

```
[b,a] = cheby1(12,0.5,200/500);
[h,f] = freqz(b,a,256,1000);
```

Because the parameter list includes a sampling frequency, `freqz` returns a vector `f` that contains the 256 frequency points between 0 and `Fs/2` used in the frequency response calculation.

Meta Platforms, Inc. - Exhibit 1009
Page 44 of 720

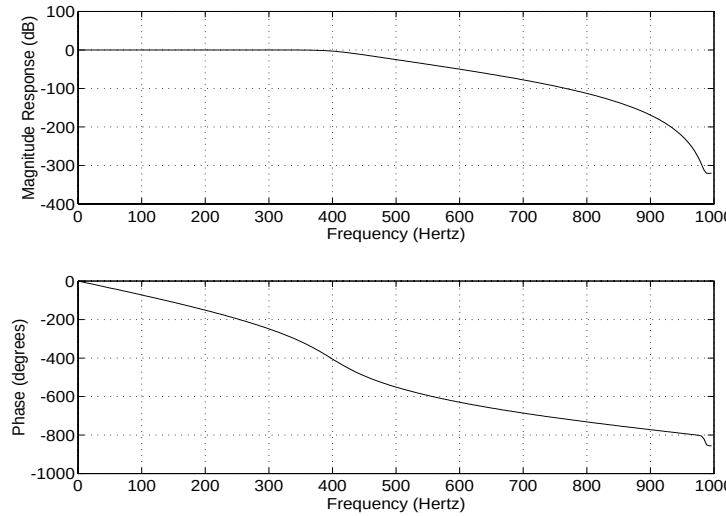
Frequency Normalization This toolbox uses the convention that unit frequency is the Nyquist frequency, defined as half the sampling frequency. The cutoff frequency parameter for all basic filter design functions is normalized by the Nyquist frequency. For a system with a 1000 Hz sampling frequency, for example, 300 Hz is 300/500 = 0.6. To convert normalized frequency to angular frequency around the unit circle, multiply by π. To convert normalized frequency back to Hertz, multiply by half the sample frequency.

If you call `freqz` with no output arguments, it automatically plots both magnitude versus frequency and phase versus frequency. For example, a ninth-order Butterworth lowpass filter with a cutoff frequency of 400 Hz, based on a 2000 Hz sampling frequency, is

```
[b,a] = butter(9,400/1000);
```

Now calculate the 256-point complex frequency response for this filter, and plot the magnitude and phase with a call to `freqz`.

```
freqz(b,a,256,2000)
```



Meta Platforms, Inc. - Exhibit 1009
Page 45 of 720

freqz can also accept a vector of arbitrary frequency points for use in the frequency response calculation. For example,

```
w = linspace(0,pi);
h = freqz(b,a,w);
```

calculates the complex frequency response at the frequency points in w for the filter defined by vectors b and a. The frequency points can range from 0 to 2π. To specify a frequency vector that ranges from zero to your sampling frequency, include both the frequency vector and the sampling frequency value in the parameter list.

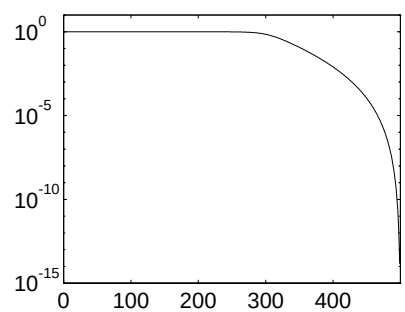
Analog Domain

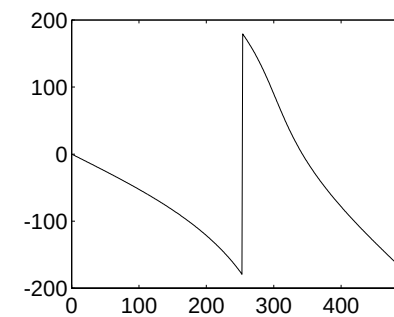
freqs evaluates frequency response for an analog filter defined by two input coefficient vectors b and a. Its operation is similar to that of freqz; you can specify a number of frequency points to use (by default, the function uses 200), supply a vector of arbitrary frequency points, and plot the magnitude and phase response of the filter.

Magnitude and Phase

MATLAB provides functions to extract magnitude and phase from a frequency response vector h. The function abs returns the magnitude of the response; angle returns the phase angle in radians. To extract and plot the magnitude and phase of a Butterworth filter.

```
[b,a] = butter(6,300/500); [h,w] = freqz(b,a,512,1000);
m = abs(h); p = angle(h);
semilogy(w,m);
plot(w,p*180/pi)
```





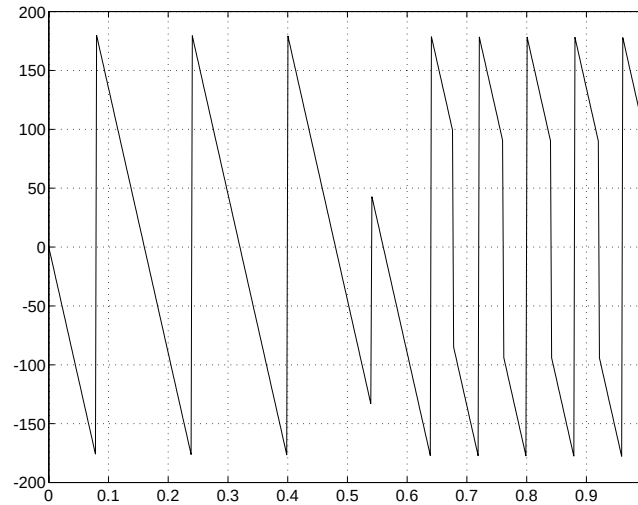
Meta Platforms, Inc. - Exhibit 1009
Page 46 of 720

The `unwrap` function is also useful in frequency analysis. `unwrap` unwraps the phase to make it continuous across 360° phase discontinuities by adding multiples of ±360°, as needed. To see how `unwrap` is useful, design a 25th-order lowpass FIR filter.

```
h = fir1(25,0.4);
```

Obtain the filter's frequency response with `freqz`, and plot the phase in degrees.

```
[H,f] = freqz(h,1,512,2);
plot(f,angle(H)*180/pi); grid
```

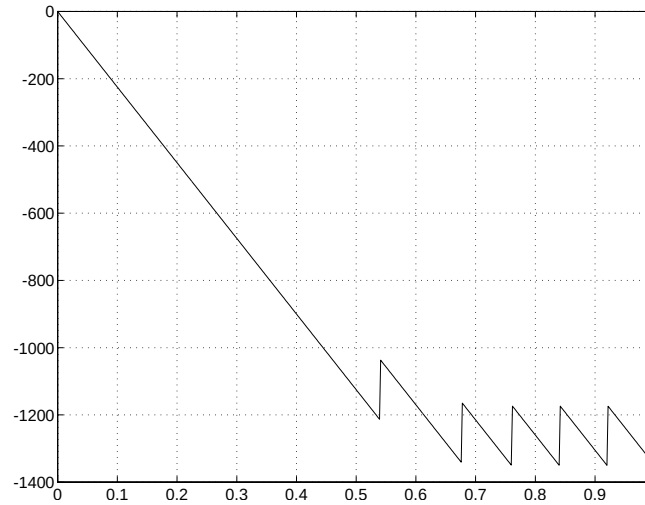


It is difficult to distinguish the 360° jumps (an artifact of the arctangent function inside `angle`) from the 180° jumps that signify zeros in the frequency response.

Meta Platforms, Inc. - Exhibit 1009
Page 47 of 720

Use unwrap to eliminate the 360° jumps.

```
plot(f,unwrap(angle(H))*180/pi); grid
```



Delay

The *group delay* of a filter is a measure of the average delay of the filter as a function of frequency. It is defined as the negative first derivative of a filter's phase response. If the complex frequency response of a filter is $H(e^{j\omega})$, then the group delay is

$$\tau_g(\omega) = -\frac{d\theta(\omega)}{d\omega}$$

where θ is the phase angle of $H(e^{j\omega})$. Compute group delay with

```
[gd,w] = grpdelay(b,a,n)
```

which returns the n-point group delay, $\tau_g(\omega)$, of the digital filter specified by b and a, evaluated at the frequencies in vector w.

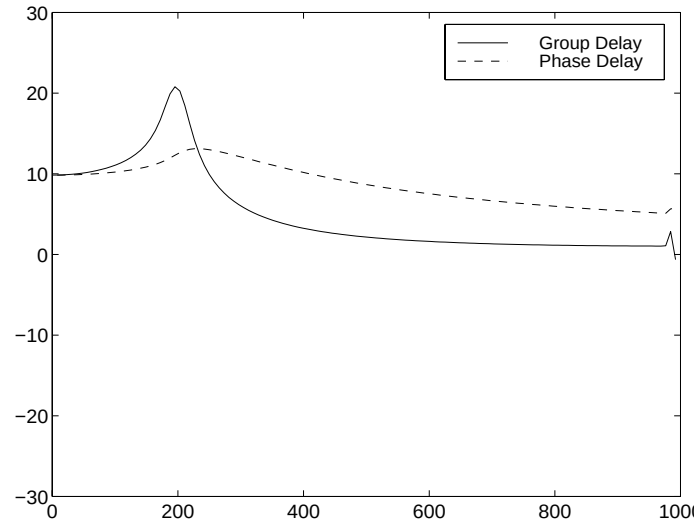
The *phase delay* of a filter is the negative of phase divided by frequency

$$\tau_p(\omega) = -\frac{\theta(\omega)}{\omega}$$

Meta Platforms, Inc. - Exhibit 1009
Page 48 of 720

To plot both the group and phase delays of a system on the same graph.

```
[b,a] = butter(10,200/1000);
gd = grpdelay(b,a,128);
[h,f] = freqz(b,a,128,2000);
pd = -unwrap(angle(h))*(2000/(2*pi))./f;
plot(f,gd,'-',f,pd,'- -')
axis([0 1000 -30 30])
legend('Group Delay','Phase Delay')
```



Meta Platforms, Inc. - Exhibit 1009
Page 49 of 720

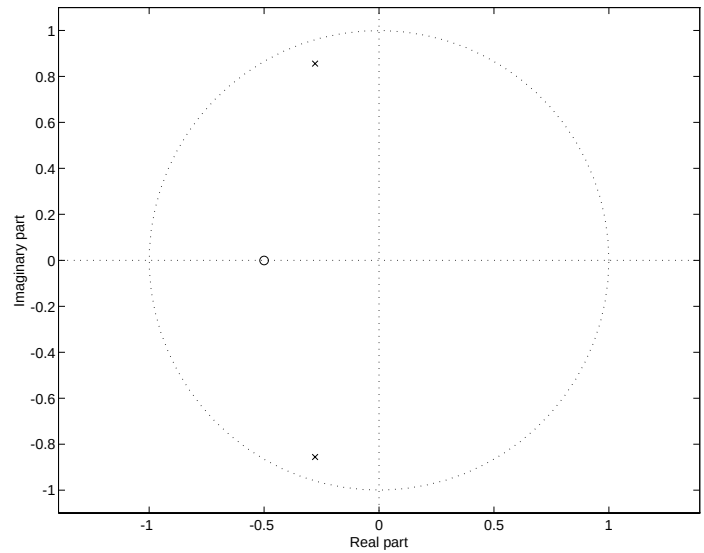
Zero-Pole Analysis

The `zplane` function plots poles and zeros of a linear system. For example, a simple filter with a 0 at -1/2 and a complex pole pair at $0.9e^{j2\pi(0.3)}$ and $0.9e^{-j2\pi(0.3)}$ is

```
zer = -0.5;
pol = .9*exp(j*2*pi*[-0.3 .3]');
```

The zero-pole plot for the filter is

```
zplane(zer,pol)
```



For a system in zero-pole form, supply column vector arguments `z` and `p` to `zplane`.

```
zplane(z,p)
```

For a system in transfer function form, supply row vectors `b` and `a` as arguments to `zplane`.

```
zplane(b,a)
```

In this case `zplane` finds the roots of `b` and `a` using the `roots` function and plots the resulting zeros and poles.

Meta Platforms, Inc. - Exhibit 1009
Page 50 of 720

See “Linear System Models” on page 1-32 for details on zero-pole and transfer function representation of systems.

Meta Platforms, Inc. - Exhibit 1009
Page 51 of 720

Linear System Models

The Signal Processing Toolbox provides several models for representing linear time-invariant systems. This flexibility lets you choose the representational scheme that best suits your application and, within the bounds of numeric stability, convert freely to and from most other models. This section provides a brief overview of supported linear system models and describes how to work with these models in MATLAB.

Discrete-Time System Models

The discrete-time system models are representational schemes for digital filters. MATLAB supports several discrete-time system models:

- Transfer function
- Zero-pole-gain form
- State-space form
- Partial fraction expansion
- Second-order section form
- Lattice structure form
- Convolution matrices

Transfer Function

The *transfer function* is a basic z-domain representation of a digital filter, expressing the filter as a ratio of two polynomials. It is the principal discrete-time model for this toolbox. The transfer function model description for the z-transform of a digital filter's difference equation is

$$Y(z) = \frac{b(1) + b(2)z^{-1} + \cdots + b(nb+1)z^{-nb}}{a(1) + a(2)z^{-1} + \cdots + a(na+1)z^{-na}} X(z)$$

Here, the constants $b(i)$ and $a(i)$ are the filter coefficients, and the order of the filter is the maximum of na and nb. In MATLAB, you store these coefficients in two vectors (row vectors by convention), one row vector for the numerator and one for the denominator. See "Filters and Transfer Functions" on page 1-15 for more details on the transfer function form.

Meta Platforms, Inc. - Exhibit 1009
Page 52 of 720

Zero-Pole-Gain

The factored or *zero-pole-gain* form of a transfer function is

$$H(z) = \frac{q(z)}{p(z)} = k\frac{(z-q(1))(z-q(2))\cdots(z-q(n))}{(z-p(1))(z-p(2))\cdots(z-p(n))}$$

By convention, MATLAB stores polynomial coefficients in row vectors and polynomial roots in column vectors. In zero-pole-gain form, therefore, the zero and pole locations for the numerator and denominator of a transfer function reside in column vectors. The factored transfer function gain k is a MATLAB scalar.

The `poly` and `roots` functions convert between polynomial and zero-pole-gain representations. For example, a simple IIR filter is

```
b = [2 3 4];
a = [1 3 3 1];
```

The zeros and poles of this filter are

```
q = roots(b)

q =

   -0.7500 + 1.1990i
   -0.7500 - 1.1990i

p = roots(a)

p =

   -1.0000
   -1.0000 + 0.0000i
   -1.0000 - 0.0000i

k = b(1)/a(1)

k =

     2
```


Meta Platforms, Inc. - Exhibit 1009
Page 53 of 720

Returning to the original polynomials,

```
bb = k*poly(q)

bb =

    2.0000    3.0000    4.0000

aa = poly(p)

aa =

    1.0000    3.0000    3.0000    1.0000
```

Note that b and a in this case represent the transfer function

$$H(z) = \frac{2 + 3z^{-1} + 4z^{-2}}{1 + 3z^{-1} + 3z^{-2} + z^{-3}} = \frac{2z^3 + 3z^2 + 4z}{z^3 + 3z^2 + 3z + 1}$$

For `b = [2 3 4]`, the `roots` function misses the zero for z equal to 0. In fact, it misses poles and zeros for z equal to 0 whenever the input transfer function has more poles than zeros, or vice versa. This is acceptable in most cases. To circumvent the problem, however, simply append zeros to make the vectors the same length before using the `roots` function, for example, `b = [b 0]`.

State-Space

It is always possible to represent a digital filter, or a system of difference equations, as a set of first-order difference equations. In matrix or *state-space* form, you can write the equations as

$$\begin{aligned} x(n+1) &= Ax(n) + Bu(n) \\ y(n) &= Cx(n) + Du(n) \end{aligned}$$

where u is the input, x is the state vector, and y is the output. For single-channel systems, A is an m-by-m matrix where m is the order of the filter, B is a column vector, C is a row vector, and D is a scalar. State-space notation is especially convenient for multichannel systems where input u and output y become vectors, and B, C, and D become matrices.

State-space representation extends easily to the MATLAB environment. In MATLAB, A, B, C, and D are rectangular arrays; MATLAB treats them as individual variables.

Meta Platforms, Inc. - Exhibit 1009
Page 54 of 720

Taking the z-transform of the state-space equations and combining them shows the equivalence of state-space and transfer function forms.

$$Y(z) = H(z)U(z), \qquad \text{where } H(z) = C(zI - A)^{-1}B + D$$

Don't be concerned if you are not familiar with the state-space representation of linear systems. Some of the filter design algorithms use state-space form internally but do not require any knowledge of state-space concepts to use them successfully. If your applications use state-space based signal processing extensively, however, consult the Control System Toolbox for a comprehensive library of state-space tools.

Partial Fraction Expansion (Residue Form)

Each transfer function also has a corresponding *partial fraction expansion* or *residue* form representation, given by

$$\frac{b(z)}{a(z)} = \frac{r(1)}{1 - p(1)z^{-1}} + \cdots + \frac{r(n)}{1 - p(n)z^{-1}} + k(1) + k(2)z^{-1} + \cdots + k(m - n + 1)z^{-(m-n)}$$

provided $H(z)$ has no repeated poles. Here, n is the degree of the denominator polynomial of the rational transfer function $b(z)/a(z)$. If r is a pole of multiplicity s_r, then $H(z)$ has terms of the form

$$\frac{r(j)}{1 - p(j)z^{-1}} + \frac{r(j+1)}{(1 - p(j)z^{-1})^2} + \cdots + \frac{r(j + s_r - 1)}{(1 - p(j)z^{-1})^{s_r}}$$

The `residuez` function in the Signal Processing Toolbox converts transfer functions to and from the partial fraction expansion form. The "`z`" on the end of `residuez` stands for z-domain, or discrete domain. `residuez` returns the poles in a column vector `p`, the residues corresponding to the poles in a column vector `r`, and any improper part of the original transfer function in a row vector `k`. `residuez` determines that two poles are the same if the magnitude of their difference is smaller than 0.1 percent of either of the poles' magnitudes.

Partial fraction expansion arises in signal processing as one method of finding the inverse z-transform of a transfer function. For example, the partial fraction expansion of

$$H(z) = \frac{-4 + 8z^{-1}}{1 + 6z^{-1} + 8z^{-2}}$$

Meta Platforms, Inc. - Exhibit 1009
Page 55 of 720

is

```
b = [-4 8];
a = [1 6 8];
[r,p,k] = residuez(b,a)

r =
    -12
     8

p =
    -4
    -2

k =
    []
```

corresponding to

$$H(z) = \frac{-12}{1+4z^{-1}} + \frac{8}{1+2z^{-1}}$$

To find the inverse z-transform of $H(z)$, find the sum of the inverse z-transforms of the two addends of $H(z)$, giving the causal impulse response

$$h(n) = -12(-4)^n + 8(-2)^n, \qquad n = 0, 1, 2, \ldots$$

To verify this in MATLAB

```
imp = [1 0 0 0 0];
resptf = filter(b,a,imp)

resptf =

        -4     32     -160     704     -2944

respres = filter(r(1),[1 -p(1)],imp) + filter(r(2),[1 -p(2)],imp)

respres =

        -4     32     -160     704     -2944
```

Second-Order Sections (SOS)

Any transfer function $H(z)$ has a second-order sections representation

Meta Platforms, Inc. - Exhibit 1009
Page 56 of 720

$$H(z) = \prod_{k=1}^{L} H_k(z) = \prod_{k=1}^{L} \frac{b_{0k} + b_{1k}z^{-1} + b_{2k}z^{-2}}{a_{0k} + a_{1k}z^{-1} + a_{2k}z^{-2}}$$

where L is the number of second-order sections that describe the system. MATLAB represents the second-order section form of a discrete-time system as an L-by-6 array `sos`. Each row of `sos` contains a single second-order section, where the row elements are the three numerator and three denominator coefficients that describe the second-order section.

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

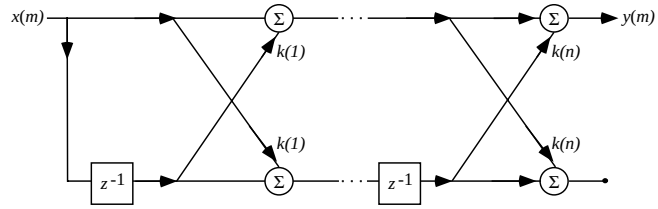
There are an uncountable number of ways to represent a filter in second-order section form. Through careful pairing of the pole and zero pairs, ordering of the sections in the cascade, and multiplicative scaling of the sections, it is possible to reduce quantization noise gain and avoid overflow in some fixed-point filter implementations. The functions `zp2sos` and `ss2sos`, described later in "Linear System Transformations," perform pole-zero pairing, section scaling, and section ordering.

Lattice Structure

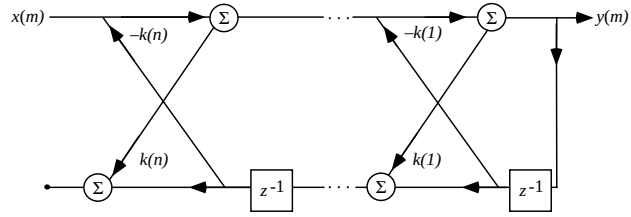
For a discrete Nth order all-pole or all-zero filter described by the polynomial coefficients $a(n)$, $n = 1,2,\ldots,N+1$, there are N corresponding lattice structure coefficients $k(n)$, $n = 1,2,\ldots,N$. The parameters $k(n)$ are also called the *reflection*

Meta Platforms, Inc. - Exhibit 1009
Page 57 of 720

coefficients of the filter. Given these reflection coefficients, you can implement a discrete filter as

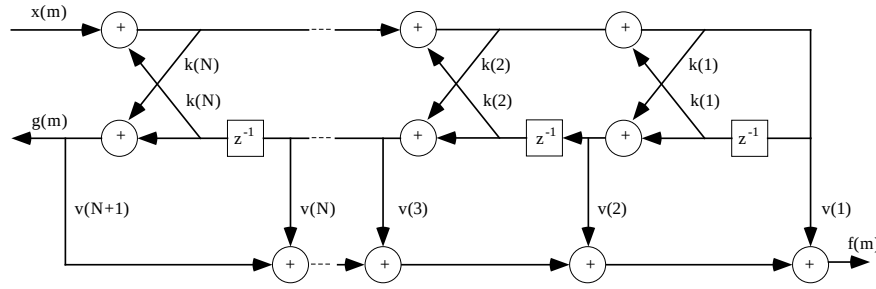


FIR Lattice Filter



IIR Lattice Filter

For a general pole-zero IIR filter described by polynomial coefficients a and b, there are both lattice coefficients $k(n)$ for the denominator a and ladder coefficients $v(n)$ for the numerator b. The lattice/ladder filter may be implemented as



Meta Platforms, Inc. - Exhibit 1009
Page 58 of 720

The toolbox function `tf2latc` accepts an FIR or IIR filter in polynomial form and returns the corresponding reflection coefficients. An example IIR filter in polynomial form is

```
a = [1.0000   0.6149   0.9899   0.0000   0.0031  -0.0082];
```

This filter's lattice (reflection coefficient) representation is

```
k = tf2latc(a)

k =

    0.3090
    0.9800
    0.0031
    0.0081
   -0.0082
```

The magnitude of the reflection coefficients provides an easy stability check for a filter. If all the reflection coefficients corresponding to a polynomial have magnitude less than 1, all of that polynomial's roots are inside the unit circle.

The function `latc2tf` calculates the polynomial coefficients for a filter from its lattice (reflection) coefficients. Given the reflection coefficient vector `k` (above), the corresponding polynomial form is

```
a = latc2tf(k)

a =

    1.0000    0.6149    0.9899 -0.0000    0.0031  -0.0082
```

The lattice or lattice/ladder coefficients can be used to implement the filter using the function `latcfilt`.

Convolution Matrix

In signal processing, convolving two vectors or matrices is equivalent to filtering one of the input operands by the other. This relationship permits the representation of a digital filter as a *convolution matrix*.

Given any vector, the toolbox `convmtx` function generates a matrix whose inner product with another vector is equivalent to the convolution of the two vectors. The generated matrix represents a digital filter that you can apply to any

Meta Platforms, Inc. - Exhibit 1009
Page 59 of 720

vector of appropriate length; the inner dimension of the operands must agree to compute the inner product.

The convolution matrix for a vector b, representing the numerator coefficients for a digital filter, is

```
b = [1 2 3]; x = randn(3,1);
C = convmtx(b',3)

C =

     1     0     0
     2     1     0
     3     2     1
     0     3     2
     0     0     3
```

Two ways to convolve b with x are

```
y1 = C*x;
y2 = conv(b,x);
```

Type this example into MATLAB; the results for y1 and y2 are equal.

Continuous-Time System Models

The continuous-time system models are representational schemes for analog filters. Many of the discrete-time system models described earlier are also appropriate for the representation of continuous-time systems:

- State-space form
- Partial fraction expansion
- Transfer function
- Zero-pole-gain form

It is possible to represent any system of linear time-invariant differential equations as a set of first-order differential equations. In matrix or *state-space* form, you can express the equations as

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

Meta Platforms, Inc. - Exhibit 1009
Page 60 of 720

where u is a vector of nu inputs, x is an nx-element state vector, and y is a vector of ny outputs. In MATLAB, store `A`, `B`, `C`, and `D` in separate rectangular arrays.

An equivalent representation of the state-space system is the Laplace transform transfer function description

$$Y(s) = H(s)U(s)$$

where

$$H(s) = C(sI - A)^{-1}B + D$$

For single-input, single-output systems, this form is given by

$$H(s) = \frac{b(s)}{a(s)} = \frac{b(1)s^{nb} + b(2)s^{nb-1} + \cdots + b(nb+1)}{a(1)s^{na} + a(2)s^{na-1} + \cdots + a(na+1)}$$

Given the coefficients of a Laplace transform transfer function, `residue` determines the partial fraction expansion of the system. See the description of `residue` in the *MATLAB Language Reference Manual* for details.

The factored zero-pole-gain form is

$$H(s) = \frac{z(s)}{p(s)} = k\frac{(s - z(1))(s - z(2))\cdots(s - z(n))}{(s - p(1))(s - p(2))\cdots(s - p(n))}$$

As in the discrete-time case, MATLAB stores polynomial coefficients in row vectors in descending powers of `s`. MATLAB stores polynomial roots, or zeros and poles, in column vectors.

Linear System Transformations

The Signal Processing Toolbox provides a number of functions that convert between the various linear system models; see the reference description in Chapter 6 for a complete description of each. You can use the following chart to find an appropriate transfer function: find the row of the model to convert *from* on the left side of the chart and the column of the model to convert *to* on the top of the chart and read the function name(s) at the intersection of the row and column.

Meta Platforms, Inc. - Exhibit 1009
Page 61 of 720

	Transfer function	State-space	Zero-pole gain	Partial fraction	Lattice filter	Second-order sections	Convolution matrix
Transfer function		tf2ss	tf2zp roots	residuez residue	tf2latc		convmtx
State-space	ss2tf		ss2zp			ss2sos	
Zero-pole gain	zp2tf poly	zp2ss				zp2sos	
Partial fraction	residuez residue						
Lattice filter	latc2tf						
Second-order sections	sos2tf	sos2ss	sos2zp				
Convolution matrix							

Many of the toolbox filter design functions use these functions internally. For example, the zp2ss function converts the poles and zeros of an analog prototype into the state-space form required for creation of a Butterworth, Chebyshev, or elliptic filter. Once in state-space form, the filter design function performs any required frequency transformation, that is, it transforms the initial lowpass design into a bandpass, highpass, or bandstop filter, or a lowpass filter with the desired cutoff frequency. See Chapter 6 and the reference descriptions of the individual filter design functions for more details.

Meta Platforms, Inc. - Exhibit 1009
Page 62 of 720

Discrete Fourier Transform

The discrete Fourier transform, or DFT, is the primary tool of digital signal processing. The foundation of the Signal Processing Toolbox is the Fast Fourier Transform (FFT), a method for computing the DFT with reduced execution time. Many of the toolbox functions (including *z*-domain frequency response, spectrum and cepstrum analysis, and some filter design and implementation functions) incorporate the FFT.

MATLAB provides the functions `fft` and `ifft` to compute the discrete Fourier transform and its inverse, respectively. For the input sequence *x* and its transformed version *X* (the discrete-time Fourier transform at equally spaced frequencies around the unit circle), the two functions implement the relationships

$$X(k+1) = \sum_{n=0}^{N-1} x(n+1) W_n^{kn}$$

$$x(n+1) = \frac{1}{N} \sum_{k=0}^{N-1} X(k+1) W_n^{-kn}$$

In these equations, the series subscripts begin with 1 instead of 0 because of MATLAB's vector indexing scheme, and

$$W_N = e^{-j\left(\frac{2\pi}{N}\right)}$$

NOTE MATLAB uses a negative *j* for the `fft` function. This is an engineering convention; physics and pure mathematics typically use a positive *j*.

`fft`, with a single input argument `x`, computes the DFT of the input vector or matrix. If `x` is a vector, `fft` computes the DFT of the vector; if `x` is a rectangular array, `fft` computes the DFT of each array column.

Meta Platforms, Inc. - Exhibit 1009
Page 63 of 720

For example, create a time vector and signal.

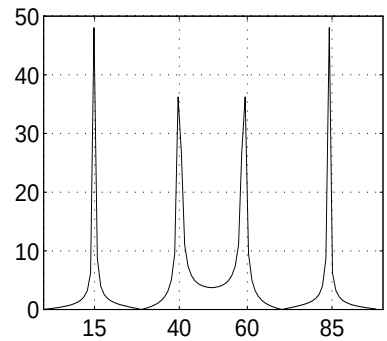
```
t = (0:1/99:1);                        % time vector
x = sin(2*pi*15*t) + sin(2*pi*40*t);% signal
```

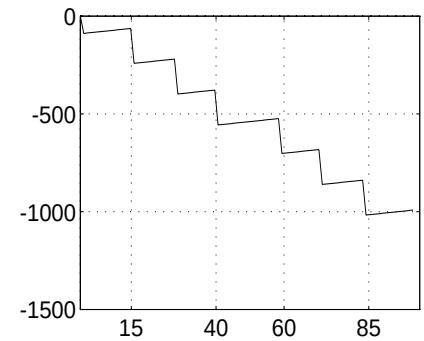
The DFT of the signal, and the magnitude and phase of the transformed sequence, are then

```
y = fft(x);                          % Compute DFT of x.
m = abs(y); p = unwrap(angle(y)); % mag. and phase
```

To plot the magnitude and phase

```
f = (0:length(y)-1)*99/length(y); % frequency vector
plot(f,m)
set(gca,'XTick',[15 40 60 85]);
plot(f,p*180/pi)
set(gca,'XTick',[15 40 60 85]);
```





A second argument to `fft` specifies a number of points `n` for the transform, representing DFT length.

```
y = fft(x,n);
```

In this case, `fft` pads the input sequence with zeros if it is shorter than `n`, or truncates the sequence if it is longer than `n`. If `n` is not specified, it defaults to the length of the input sequence.

Meta Platforms, Inc. - Exhibit 1009
Page 64 of 720

Execution time for `fft` depends on the length `n` of the DFT it performs:

- For any `n` that is a power of two, `fft` uses the high-speed radix-2 algorithm. This results in the fastest execution time. Additionally, the algorithm for power of two `n` is highly optimized for real `x`, providing a 40% speed-up over the complex case.
- For any composite number `n` that is not a power of two, `fft` uses a prime factor algorithm. The speed of this algorithm depends on both the size of `n` and number of prime factors it has. Although 1013 and 1000 are close in magnitude, `fft` transforms a sequence of length 1000 much more quickly than a sequence of length 1013.
- For a prime number `n`, `fft` cannot use an FFT algorithm, and instead performs the slower, computation-intensive DFT directly.

The inverse discrete Fourier transform function `ifft` also accepts an input sequence and, optionally, the number of desired points for the transform. Try the example below; the original sequence `x` and the reconstructed sequence are identical (within rounding error).

```
t = (0:1/255:1);
x = sin(2*pi*120*t);
y = real(ifft(fft(x)));
```

This toolbox also includes functions for the two-dimensional FFT and its inverse, `fft2` and `ifft2`. These functions are useful for two-dimensional signal or image processing; see the reference descriptions in Chapter 6 for details.

It is sometimes convenient to rearrange the output of the `fft` or `fft2` function so the zero frequency component is at the center of the sequence. The MATLAB function `fftshift` moves the zero frequency component to the center of a vector or matrix.

Meta Platforms, Inc. - Exhibit 1009
Page 65 of 720

References

Algorithm development for the Signal Processing Toolbox has drawn heavily upon the references listed below. All are recommended to the interested reader who needs to know more about signal processing than is covered in this manual.

1 Crochiere, R.E., and L.R. Rabiner. *Multi-Rate Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1983. Pgs. 88-91.

2 IEEE. *Programs for Digital Signal Processing*. IEEE Press. New York: John Wiley & Sons, 1979.

3 Jackson, L.B. *Digital Filters and Signal Processing*. Third Ed. Boston: Kluwer Academic Publishers, 1989.

4 Kay, S.M. *Modern Spectral Estimation*. Englewood Cliffs, NJ: Prentice Hall, 1988.

5 Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

6 Parks, T.W., and C.S. Burrus. *Digital Filter Design*. New York: John Wiley & Sons, 1987.

7 Pratt,W.K. *Digital Image Processing*. New York: John Wiley & Sons, 1991.

8 Percival, D.B., and A.T. Walden. *Spectral Analysis for Physical Applications: Multitaper and Conventional Univariate Techniques*. Cambridge: Cambridge University Press, 1993.

9 Proakis, J.G., and D.G. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications*. Upper Saddle River, NJ: Prentice Hall, 1996.

10 Rabiner, L.R., and B. Gold. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1975.

11 Welch, P.D. “The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified Periodograms.” *IEEE Trans. Audio Electroacoust*. Vol. AU-15 (June 1967). Pgs. 70-73.

Meta Platforms, Inc. - Exhibit 1009
Page 66 of 720

2

Filter Design

Filter Requirements and Specification 2-2

IIR Filter Design 2-4
Classical IIR Filter Design Using Analog Prototyping 2-6
Comparison of Classical IIR Filter Types 2-8

FIR Filter Design 2-16
Linear Phase Filters 2-17
Windowing Method 2-18
Multiband FIR Filter Design with Transition Bands 2-22
Constrained Least Squares FIR Filter Design 2-27
Arbitrary-Response Filter Design 2-31

Special Topics in IIR Filter Design 2-37
Analog Prototype Design 2-38
Frequency Transformation 2-38
Filter Discretization 2-41

References 2-45

Filter Requirements and Specification

The Signal Processing Toolbox provides functions that support a range of filter design methodologies. This chapter explains how to apply the filter design tools to *Infinite Impulse Response* (IIR) and *Finite Impulse Response* (FIR) filter design problems.

The goal of filter design is to perform frequency dependent alteration of a data sequence. A possible requirement might be to remove noise above 30 Hz from a data sequence sampled at 100 Hz. A more rigorous specification might call for a specific amount of passband ripple, stopband attenuation, or transition width. A very precise specification could ask to achieve the performance goals with the minimum filter order, or it could call for an arbitrary magnitude shape, or it might require an FIR filter.

Filter design methods differ primarily in how performance is specified. For "loosely specified" requirements, as in the first case above, a Butterworth IIR filter is often sufficient. To design a fifth-order 30 Hz lowpass Butterworth filter and apply it to the data in vector `x`,

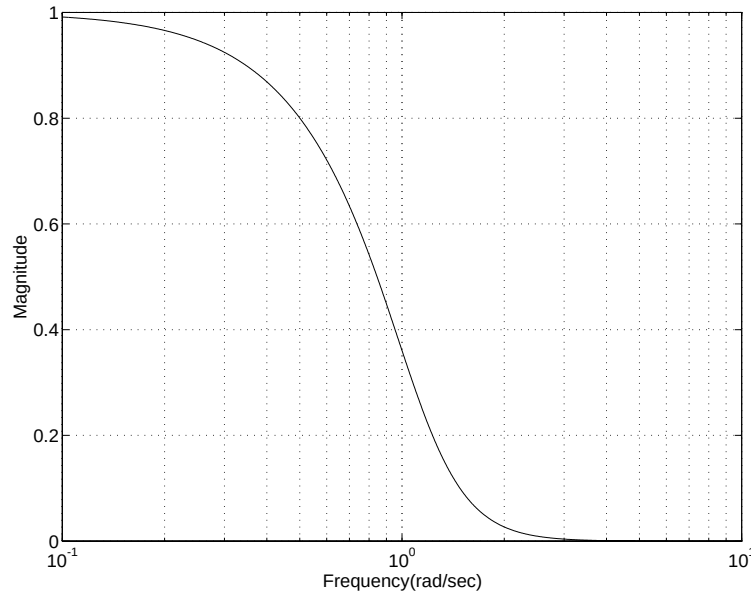
```
[b,a] = butter(5,30/50);
y = filter(b,a,x);
```

The second input argument to `butter` indicates the cutoff frequency, normalized to half the sampling frequency (the Nyquist frequency).

Frequency Normalization in the Signal Processing Toolbox All of the filter design functions operate with normalized frequencies, so they do not require the system sampling rate as an extra input argument. This toolbox uses the convention that unit frequency is the Nyquist frequency, defined as half the sampling frequency. The normalized frequency, therefore, is always in the interval $0 \le f \le 1$. For a system with a 1000 Hz sampling frequency, 300 Hz is 300/500 = 0.6. To convert normalized frequency to angular frequency around the unit circle, multiply by π. To convert normalized frequency back to Hertz, multiply by half the sample frequency.

Meta Platforms, Inc. - Exhibit 1009
Page 68 of 720

More rigorous filter requirements traditionally include passband ripple (`Rp`, in decibels), stopband attenuation (`Rs`, in decibels), and transition width (`Ws–Wp`, in Hertz).



You can design Butterworth, Chebyshev type I, Chebyshev type II, and elliptic filters that meet this type of performance specification. The toolbox order selection functions estimate the minimum filter order that meets a given set of requirements.

To meet specifications with more rigid constraints like linear phase or arbitrary filter shape, use the FIR and direct IIR filter design routines.

Meta Platforms, Inc. - Exhibit 1009
Page 69 of 720

IIR Filter Design

The primary advantage of IIR filters over FIR filters is that they typically meet a given set of specifications with a much lower filter order than a corresponding FIR filter. Although IIR filters have nonlinear phase, data processing within MATLAB is commonly performed "off-line," that is, the entire data sequence is available prior to filtering. This allows for a noncausal, zero-phase filtering approach (via the `filtfilt` function), which eliminates the nonlinear phase distortion of an IIR filter.

The classical IIR filters, Butterworth, Chebyshev types I and II, elliptic, and Bessel, all approximate the ideal "brickwall" filter in different ways. This toolbox provides functions to create all these types of classical IIR filters in both the analog and digital domains (except Bessel, for which only the analog case is supported), and in lowpass, highpass, bandpass, and bandstop configurations. For most filter types, you can also find the lowest filter order that fits a given filter specification in terms of passband and stopband attenuation, and transition width(s).

The direct filter design function `yulewalk` finds a filter with magnitude response approximating a desired function. This is one way to create a multiband bandpass filter.

You can also use the parametric modeling or system identification functions to design IIR filters. These functions are discussed in the "Parametric Modeling" section of Chapter 4.

The generalized Butterworth design function `maxflat` is discussed in the section "Generalized Butterworth Filter Design" on page 2-14.

The following table summarizes the various filter methods in the toolbox and lists the functions available to implement these methods.

Meta Platforms, Inc. - Exhibit 1009
Page 70 of 720

Method	Description	Functions

Analog Prototyping	Using the poles and zeros of a classical lowpass prototype filter in the continuous (Laplace) domain, obtain a digital filter through frequency transformation and filter discretization.	Complete design functions:
		`besself`, `butter`, `cheby1`, `cheby2`, `ellip`
		Order estimation functions:
		`buttord`, `cheb1ord`, `cheb2ord`, `ellipord`
		Lowpass analog prototype functions:
		`besselap`, `buttap`, `cheb1ap`, `cheb2ap`, `ellipap`
		Frequency transformation functions:
		`lp2bp`, `lp2bs`, `lp2hp`, `lp2lp`
		Filter discretization functions:
		`bilinear`, `impinvar`

Direct Design	Design digital filter directly in the discrete domain by approximating a piecewise linear magnitude response.	`yulewalk`

Parametric Modeling*	Find a digital filter that approximates a prescribed time or frequency domain response.	Time-domain modeling functions:
		`lpc`, `prony`, `stmcb`
		Frequency-domain modeling functions:
		`invfreqs`, `invfreqz`

Generalized Butterworth Design	Design lowpass Butterworth filters with more zeros than poles.	`maxflat`

* See the System Identification Toolbox for an extensive collection of parametric modeling tools.

Meta Platforms, Inc. - Exhibit 1009
Page 71 of 720

Classical IIR Filter Design Using Analog Prototyping

The principal IIR digital filter design technique this toolbox provides is based on the conversion of classical lowpass analog filters to their digital equivalents. The following sections describe how to design filters and summarize the characteristics of the supported filter types. See "Special Topics in IIR Filter Design" on page 2-37 for detailed steps on the filter design process.

Complete Classical IIR Filter Design

You can easily create a filter of any order with a lowpass, highpass, bandpass, or bandstop configuration using the filter design functions.

Filter Type	Design Function

Butterworth	`[b,a] = butter(n,Wn,options)`
	`[z,p,k] = butter(n,Wn,options)`
	`[A,B,C,D] = butter(n,Wn,options)`

Chebyshev type I	`[b,a] = cheby1(n,Rp,Wn,options)`
	`[z,p,k] = cheby1(n,Rp,Wn,options)`
	`[A,B,C,D] = cheby1(n,Rp,Wn,options)`

Chebyshev type II	`[b,a] = cheby2(n,Rs,Wn,options)`
	`[z,p,k] = cheby2(n,Rs,Wn,options)`
	`[A,B,C,D] = cheby2(n,Rs,Wn,options)`

Elliptic	`[b,a] = ellip(n,Rp,Rs,Wn,options)`
	`[z,p,k] = ellip(n,Rp,Rs,Wn,options)`
	`[A,B,C,D] = ellip(n,Rp,Rs,Wn,options)`

Bessel (analog only)	`[b,a] = besself(n,Wn,options)`
	`[z,p,k] = besself(n,Wn,options)`
	`[A,B,C,D] = besself(n,Wn,options)`

By default, each of these functions returns a lowpass filter; you need only specify the desired cutoff frequency `Wn` in normalized frequency (Nyquist frequency = 1 Hz). For a highpass filter, append the string `'high'` to the function's parameter list. For a bandpass or bandstop filter, specify `Wn` as a two-element vector containing the passband edge frequencies, appending the string `'stop'` for the bandstop configuration.

Meta Platforms, Inc. - Exhibit 1009
Page 72 of 720

Here are some example digital filters.

```
[b,a] = butter(5,0.4);  % lowpass Butterworth
[b,a] = cheby1(4,1,[0.4 0.7]);  % bandpass Chebyshev type I
[b,a] = cheby2(6,60,0.8,'high');  % highpass Chebyshev type II
[b,a] = ellip(3,1,60,[0.4 0.7],'stop');  % bandstop elliptic
```

To design an analog filter, perhaps for simulation, use a trailing `'s'` and specify cutoff frequencies in radians/second.

```
[b,a] = butter(5,.4,'s'); % analog Butterworth filter
```

All filter design functions return a filter in the transfer function, zero-pole-gain, or state-space linear system model representation, depending on how many output arguments are present.

NOTE All classical IIR lowpass filters are ill-conditioned for extremely low cut-off frequencies. Therefore, instead of designing a lowpass IIR filter with a very narrow passband, it can be better to design a wider passband and decimate the input signal.

Designing IIR Filters to Frequency Domain Specifications

This toolbox provides order selection functions that calculate the minimum filter order that meets a given set of requirements.

Filter Type	Order Estimation Function
Butterworth	`[n,Wn] = buttord(Wp,Ws,Rp,Rs)`
Chebyshev type I	`[n,Wn] = cheb1ord(Wp, Ws, Rp, Rs)`
Chebyshev type II	`[n,Wn] = cheb2ord(Wp, Ws, Rp, Rs)`
Elliptic	`[n,Wn] = ellipord(Wp, Ws, Rp, Rs)`

These are useful in conjunction with the filter design functions. Suppose you want a bandpass filter with a passband from 1000 to 2000 Hz, stopbands starting 500 Hz away on either side, a 10 kHz sampling frequency, at most 1 dB

Meta Platforms, Inc. - Exhibit 1009
Page 73 of 720

of passband ripple, and at least 60 dB of stopband attenuation. To meet these specifications with the `butter` function.

```
[n,Wn] = buttord([1000 2000]/5000,[500 2500]/5000,1,60)

n =
    12

Wn =
    0.1951    0.4080

[b,a] = butter(n,Wn);
```

An elliptic filter that meets the same requirements is given by

```
[n,Wn] = ellipord([1000 2000]/5000,[500 2500]/5000,1,60)

n =
    5

Wn =
    0.2000    0.4000

[b,a] = ellip(n,1,60,Wn);
```

These functions also work with the other standard band configurations, as well as for analog filters; see Chapter 6 for details.

Comparison of Classical IIR Filter Types

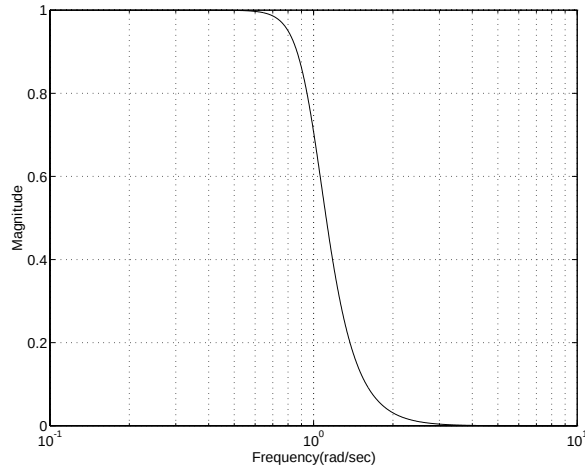
The toolbox provides five different types of classical IIR filters, each optimal in some way. This section shows the basic analog prototype form for each and summarizes major characteristics.

Butterworth Filter

The Butterworth filter provides the best Taylor Series approximation to the ideal lowpass filter response at $\Omega = 0$ and $\Omega = \infty$; for any order N, the magnitude squared response has $2N-1$ zero derivatives at these locations (*maximally flat*

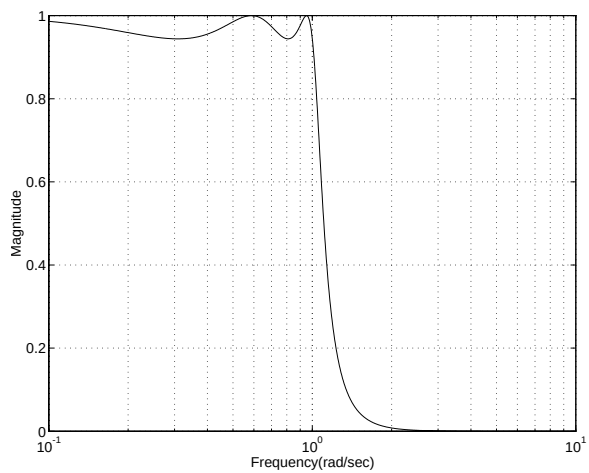
Meta Platforms, Inc. - Exhibit 1009
Page 74 of 720

at $\Omega = 0$ and $\Omega = \infty$). Response is monotonic overall, decreasing smoothly from $\Omega = 0$ to $\Omega = \infty$. $|H(j\Omega)|$ = `sqrt(1/2)` at $\Omega = 1$.



Chebyshev Type I Filter

The Chebyshev type I filter minimizes the absolute difference between the ideal and actual frequency response over the entire passband by incorporating an equal ripple of `Rp` dB in the passband. Stopband response is maximally flat. The transition from passband to stopband is more rapid than for the Butterworth filter. $|H(j\Omega)| = 10^{-Rp/20}$ at $\Omega = 1$.

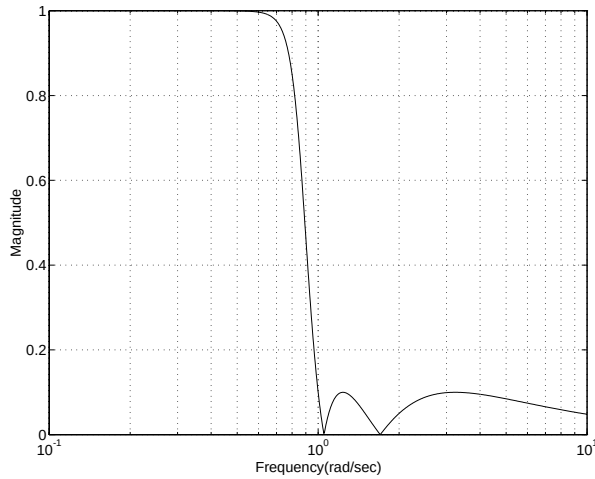


Meta Platforms, Inc. - Exhibit 1009
Page 75 of 720

Chebyshev Type II Filter

The Chebyshev type II filter minimizes the absolute difference between the ideal and actual frequency response over the entire stopband, by incorporating an equal ripple of Rs dB in the stopband. Passband response is maximally flat.

The stopband does not approach zero as quickly as the type I filter (and does not approach zero at all for even-valued n). The absence of ripple in the passband, however, is often an important advantage. $|H(j\Omega)| = 10^{-Rs/20}$ at $\Omega = 1$.

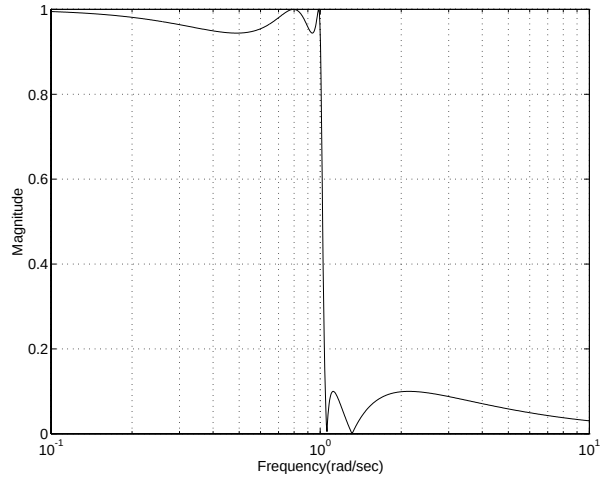


Elliptic Filter

Elliptic filters are equiripple in both the passband and stopband. They generally meet filter requirements with the lowest order of any supported filter type. Given a filter order n, passband ripple Rp in decibels, and stopband ripple

Meta Platforms, Inc. - Exhibit 1009
Page 76 of 720

Rs in decibels, elliptic filters minimize transition width. $|H(j\Omega)| = 10^{-Rp/20}$ at $\Omega = 1$.

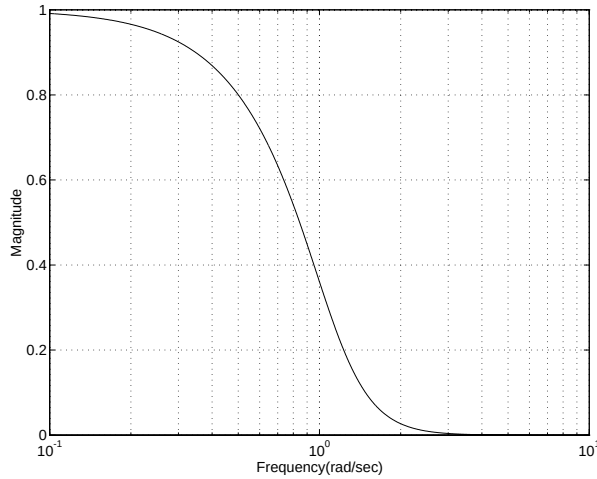


Bessel Filter

Analog Bessel lowpass filters have maximally flat group delay at zero frequency and retain nearly constant group delay across the entire passband. Filtered signals therefore maintain their waveshapes in the passband frequency range. Frequency mapped and digital Bessel filters, however, do not have this maximally flat property; this toolbox supports only the analog case for the complete Bessel filter design function.

Meta Platforms, Inc. - Exhibit 1009
Page 77 of 720

Bessel filters generally require a higher filter order than other filters for satisfactory stopband attenuation. $|H(j\Omega)| < 1/\sqrt{2}$ at $\Omega = 1$ and decreases as n increases.



NOTE The lowpass filters shown above were created with the analog prototype functions `besselap`, `buttap`, `cheb1ap`, `cheb2ap`, and `ellipap`. These functions find the zeros, poles, and gain of an order `n` analog filter of the appropriate type with cutoff frequency of 1 rad/sec. The complete filter design functions (`besself`, `butter`, `cheby1`, `cheby2`, and `ellip`) call the prototyping functions as a first step in the design process. See "Special Topics in IIR Filter Design" on page 2-37 for details.

To create similar plots, use `n = 5` and, as needed, `Rp = 0.5` and `Rs = 20`. For example, to create the elliptic filter plot:

```
[z,p,k] = ellipap(5,0.5,20);
w = logspace(-1,1,1000);
h = freqs(k*poly(z),poly(p),w);
semilogx(w,abs(h)), grid
```


Meta Platforms, Inc. - Exhibit 1009
Page 78 of 720

Direct IIR Filter Design

This toolbox uses the term *direct methods* to describe techniques for IIR design that find a filter based on specifications in the discrete domain. Unlike the analog prototyping method, direct design methods are not constrained to the standard lowpass, highpass, bandpass, or bandstop configurations. Rather, these functions design filters with an arbitrary, perhaps multiband, frequency response. This section discusses the `yulewalk` function, which is intended specifically for filter design; "Parametric Modeling" in Chapter 4 discusses other methods that may also be considered direct, such as Prony's method, Linear Prediction, the Steiglitz-McBride method, and inverse frequency design.

`yulewalk` designs recursive IIR digital filters by fitting a specified frequency response. `yulewalk`'s name reflects its method for finding the filter's denominator coefficients: it finds the inverse FFT of the ideal desired power spectrum and solves the "modified Yule-Walker equations" using the resulting autocorrelation function samples. The statement

```
[b,a] = yulewalk(n,f,m)
```

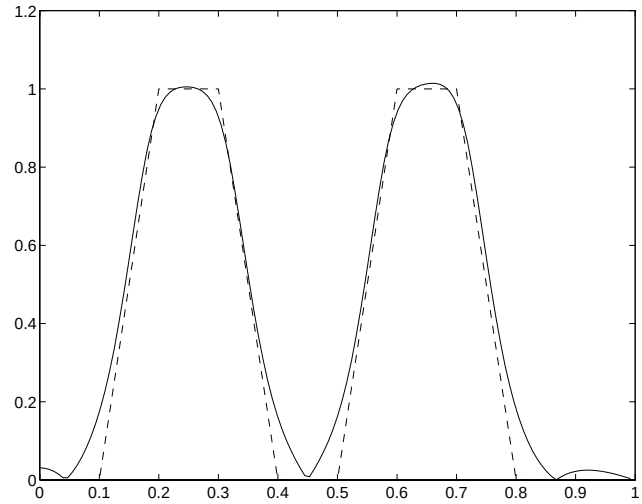
returns row vectors `b` and `a` containing the `n+1` numerator and denominator coefficients of the order `n` IIR filter whose frequency-magnitude characteristics approximate those given in vectors `f` and `m`. `f` is a vector of frequency points ranging from 0 to 1, where 1 represents the Nyquist frequency. `m` is a vector containing the desired magnitude response at the points in `f`. `f` and `m` can describe any piecewise linear shape magnitude response, including a multiband response. The FIR counterpart of this function is `fir2`, which also designs a filter based on an arbitrary piecewise linear magnitude response. See "FIR Filter Design" on page 2-16 for details.

Note that `yulewalk` does not accept phase information, and no statements are made about the optimality of the resulting filter.

Meta Platforms, Inc. - Exhibit 1009
Page 79 of 720

Design a multiband filter with `yulewalk`, and plot the desired and actual frequency response.

```
m = [0   0   1   1   0   0   1   1   0 0];
f = [0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 1];
[b,a] = yulewalk(10,f,m);
[h,w] = freqz(b,a,128);
plot(f,m,w/pi,abs(h))
```



Generalized Butterworth Filter Design

The toolbox function `maxflat` enables you to design generalized Butterworth filters, that is, Butterworth filters with differing numbers of zeros and poles. This is desirable in some implementations where poles are more expensive computationally than zeros. `maxflat` is just like the `butter` function, except that it you can specify *two* orders (one for the numerator and one for the denominator) instead of just one. These filters are *maximally flat*. This means that the resulting filter is optimal for any numerator and denominator orders, with the maximum number of derivatives at 0 and the Nyquist frequency $\omega=\pi$ both set to 0.

Meta Platforms, Inc. - Exhibit 1009
Page 80 of 720

For example, when the two orders are the same, `maxflat` is the same as `butter`.

```
[b,a] = maxflat(3,3,0.25)

b =
    0.0317    0.0951    0.0951    0.0317

a =
    1.0000   -1.4590    0.9104   -0.1978

[b,a] = butter(3,0.25)

b =
    0.0317    0.0951    0.0951    0.0317

a =
    1.0000   -1.4590    0.9104   -0.1978
```

However, `maxflat` is more versatile, because you can design a filter with more zeros than poles.

```
[b,a] = maxflat(3,1,0.25)

b =
    0.0950    0.2849    0.2849    0.0950

a =
    1.0000   -0.2402
```

The third input to maxflat is the *half-power frequency*, a frequency between 0 and 1 with a desired magnitude response of $1/\sqrt{2}$.

You can also design linear phase filters that have the maximally flat property using the `'sym'` option.

```
maxflat(4,'sym',0.3)

ans =
    0.0331    0.2500    0.4337    0.2500    0.0331
```

For complete details of the `maxflat` algorithm, see Selesnick and Burrus [2].

Meta Platforms, Inc. - Exhibit 1009
Page 81 of 720

FIR Filter Design

Digital filters with finite-duration impulse response (all-zero, or FIR filters) have both advantages and disadvantages compared to infinite-duration impulse response (IIR) filters.

FIR filters have the following primary advantages:

- They can have exactly linear phase.
- They are always stable.
- The design methods are generally linear.
- They can be realized efficiently in hardware.
- The filter startup transients have finite duration.

The primary disadvantage of FIR filters is that they often require a much higher filter order than IIR filters to achieve a given level of performance. Correspondingly, the delay of these filters is often much greater than for an equal performance IIR filter.

Method	Description	Functions
Windowing	Apply window to truncated inverse Fourier transform of desired “brickwall” filter	`fir1,fir2,kaiserord`
Multiband with Transition Bands	Equiripple or least squares approach over sub-bands of the frequency range	`firls,remez,remezord`
Constrained Least Squares	Minimize squared integral error over entire frequency range subject to maximum error constraints	`fircls,fircls1`

Meta Platforms, Inc. - Exhibit 1009
Page 82 of 720

Method	Description	Functions
Arbitrary Response	Arbitrary responses, including nonlinear phase and complex filters	`cremez`
Raised Cosine	Lowpass response with smooth, sinusoidal transition	`firrcos`

Linear Phase Filters

Except for `cremez`, all of the FIR filter design functions design linear phase filters only. The filter coefficients, or "taps," of such filters obey either an even or odd symmetry relation. Depending on this symmetry, and on whether the order n of the filter is even or odd, a linear phase filter (stored in length n+1 vector `b`) has certain inherent restrictions on its frequency response.

Linear Phase Filter Type	Filter Order n	Symmetry of Coefficients	Response H(f), f = 0	Response H(f), f = 1 (Nyquist)
Type I	Even	even: $b(k) = b(n+2-k), \quad k = 1, \ldots, n+1$	No restriction	No restriction
Type II	Odd		No restriction	$H(1) = 0$
Type III	Even	odd: $b(k) = -b(n+2-k), \quad k = 1, \ldots, n+1$	$H(0) = 0$	$H(1) = 0$
Type IV	Odd		$H(0) = 0$	No restriction

The phase delay and group delay of linear phase FIR filters are equal and constant over the frequency band. For an order n linear phase FIR filter, the group delay is $n/2$, and the filtered signal is simply delayed by $n/2$ time steps (and the magnitude of its Fourier transform is scaled by the filter's magnitude response). This property preserves the wave shape of signals in the passband, that is, there is no phase distortion.

The functions `fir1`, `fir2`, `firls`, `remez`, `fircls`, `fircls1`, and `firrcos` all design type I and II linear phase FIR filters by default. Both `firls` and `remez` design type III and IV linear phase FIR filters given a `'hilbert'` or `'differentiator'` flag. `cremez` can design any type of linear phase filter, and nonlinear phase filters as well.

Meta Platforms, Inc. - Exhibit 1009
Page 83 of 720

NOTE Because the frequency response of a type II filter is zero at the Nyquist rate ("high" frequency), `fir1` does not design type II highpass and bandstop filters. For odd-valued `n` in these cases, `fir1` adds 1 to the order and returns a type I filter.

Windowing Method

Consider the ideal, or "brick-wall," digital lowpass filter with a cutoff frequency of ω_0 rad/sec. This filter has magnitude 1 at all frequencies with magnitude less than ω_0, and magnitude 0 at frequencies with magnitude between ω_0 and π. Its impulse response sequence $h(n)$ is

$$h(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H(\omega)e^{j\omega n}d\omega = \frac{1}{2\pi}\int_{-\omega_0}^{\omega_0} e^{j\omega n}d\omega = \frac{\omega_0}{\pi}\mathrm{sinc}(\frac{\omega_0}{\pi}n)$$

This filter is not implementable since its impulse response is infinite and noncausal. To create a finite-duration impulse response, truncate it by applying a window. By retaining the central section of impulse response in this truncation, you obtain a linear phase FIR filter. For example, a length 51 filter with a lowpass cutoff frequency ω_0 of 0.4π rad/sec is

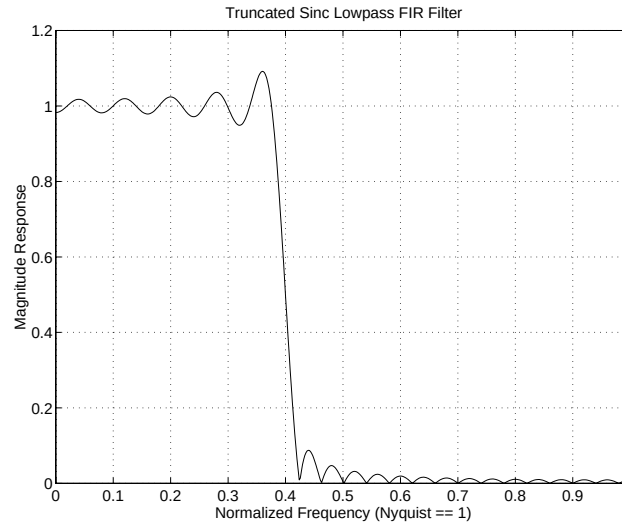
```
b = 0.4*sinc(0.4*(-25:25));
```

The window applied here is a simple rectangular or "boxcar" window. By Parseval's theorem, this is the length 51 filter that best approximates the ideal

Meta Platforms, Inc. - Exhibit 1009
Page 84 of 720

lowpass filter, in the integrated least squares sense. To view its frequency response:

```
[H,w] = freqz(b,1,512,2);
plot(w,abs(H)), grid
```

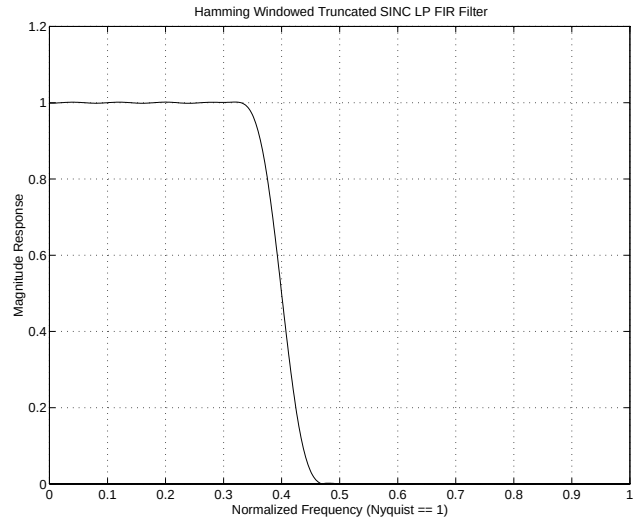


Note the ringing and ripples in the response, especially near the band edge. This “Gibbs effect” does not vanish as the filter length increases, but a nonrectangular window reduces its magnitude. Multiplication by a window in

Meta Platforms, Inc. - Exhibit 1009
Page 85 of 720

the time domain causes a convolution or smoothing in the frequency domain. Apply a length 51 Hamming window to the filter.

```
b = b.*hamming(51)';
[H,w] = freqz(b,1,512,2);
plot(w,abs(H)), grid
```



As you can see, this greatly reduces the ringing. This improvement is at the expense of transition width (the windowed version takes longer to ramp from passband to stopband) and optimality (the windowed version does not minimize the integrated squared error).

The functions `fir1` and `fir2` are based on this windowing process. Given a filter order and description of an ideal desired filter, these functions return a windowed inverse Fourier transform of that ideal filter. Both use a Hamming window by default, but they accept any window function. See the “Windows” section of Chapter 4 for an overview of windows and their properties.

Standard Band FIR Filter Design: fir1

`fir1` implements the classical method of windowed linear phase FIR digital filter design. It resembles the IIR filter design functions in that it is formulated to design filters in standard band configurations: lowpass, bandpass, highpass, and bandstop.

Meta Platforms, Inc. - Exhibit 1009
Page 86 of 720

The statements

```
n = 50;
Wn = 0.4;
b = fir1(n,Wn);
```

create row vector b containing the coefficients of the order n Hamming-windowed filter. This is a lowpass, linear phase FIR filter with cutoff frequency Wn. Wn is a number between 0 and 1, where 1 corresponds to the Nyquist frequency, half the sampling frequency. (Unlike other methods, here Wn corresponds to the 6 dB point.) For a highpass filter, simply append the string 'high' to the function's parameter list. For a bandpass or bandstop filter, specify Wn as a two-element vector containing the passband edge frequencies; append the string 'stop' for the bandstop configuration.

b = fir1(n,Wn,window) uses the window specified in column vector window for the design. The vector window must be n+1 elements long. If you do not specify a window, fir1 applies a Hamming window.

Kaiser Window Order Estimation. The kaiserord function estimates the filter order, cutoff frequency, and Kaiser window beta parameter needed to meet a given set of specifications. Given a vector of frequency band edges and a corresponding vector of magnitudes, as well as maximum allowable ripple, kaiserord returns appropriate input parameters for the fir1 function. For details on kaiserord, see the reference description in Chapter 6.

Multiband FIR Filter Design: fir2

The function fir2 also designs windowed FIR filters, but with an arbitrarily shaped piecewise linear frequency response. This is in contrast to fir1, which only designs filters in standard lowpass, highpass, bandpass, and bandstop configurations.

The commands

```
n = 50;
f = [0 .4 .5 1];
m = [1  1  0 0];
b = fir2(n,f,m);
```

return row vector b containing the n+1 coefficients of the order n FIR filter whose frequency-magnitude characteristics match those given by vectors f and m. f is a vector of frequency points ranging from 0 to 1, where 1 represents the

Meta Platforms, Inc. - Exhibit 1009
Page 87 of 720

Nyquist frequency. `m` is a vector containing the desired magnitude response at the points specified in `f`. (The IIR counterpart of this function is `yulewalk`, which also designs filters based on arbitrary piecewise linear magnitude responses. See "IIR Filter Design" for details.)

Multiband FIR Filter Design with Transition Bands

The `firls` and `remez` functions provide a more general means of specifying the ideal desired filter than the `fir1` and `fir2` functions. These functions design Hilbert transformers, differentiators, and other filters with odd symmetric coefficients (type III and type IV linear phase). They also let you include transition or "don't care" regions in which the error is not minimized, and perform band dependent weighting of the minimization.

`firls` is an extension of the `fir1` and `fir2` functions in that it minimizes the integral of the square of the error between the desired frequency response and the actual frequency response.

`remez` implements the Parks-McClellan algorithm, which uses the Remez exchange algorithm and Chebyshev approximation theory to design filters with optimal fits between the desired and actual frequency responses. The filters are optimal in the sense that they minimize the maximum error between the desired frequency response and the actual frequency response; they are sometimes called *minimax* filters. Filters designed in this way exhibit an equiripple behavior in their frequency response, and hence are also known as *equiripple* filters. The Parks-McClellan FIR filter design algorithm is perhaps the most popular and widely used FIR filter design methodology.

The syntax for `firls` and `remez` is the same; the only difference is their minimization schemes. The next example shows how filters designed with `firls` and `remez` reflect these different schemes.

Basic Configurations

The default mode of operation of `firls` and `remez` is to design type I or type II linear phase filters, depending on whether the order you desire is even or odd, respectively. A lowpass example with approximate amplitude 1 from 0 to 0.4 Hz, and approximate amplitude 0 from 0.5 to 1.0 Hz is

```
n = 20;            % filter order
f = [0 .4 .5 1];   % frequency band edges
a = [1  1  0 0];   % desired amplitudes
b = remez(n,f,a);
```


Meta Platforms, Inc. - Exhibit 1009
Page 88 of 720

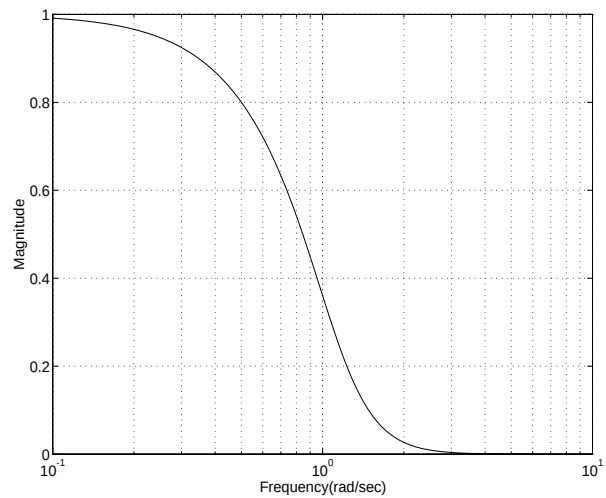
From 0.4 to 0.5 Hz, `remez` performs no error minimization; this is a transition band or "don't care" region. A transition band minimizes the error more in the bands that you do care about, at the expense of a slower transition rate. In this way, these types of filters have an inherent trade-off similar to FIR design by windowing.

To compare least squares to equiripple filter design, use `firls` to create a similar filter.

```
bb = firls(n,f,a);
```

and compare their frequency responses:

```
[H,w]=freqz(b);
[HH,w]=freqz(bb);
plot(w/pi,abs(H),w/pi,abs(HH),'--'), grid
```



You can see that the filter designed with `remez` exhibits equiripple behavior. Also note that the `firls` filter has a better response over most of the passband and stopband, but at the band edges (`f = 0.4` and `f = 0.5`), the response is further away from the ideal than the `remez` filter. This shows that the `remez` filter's *maximum* error over the pass- and stopbands is smaller and, in fact, it is the smallest possible for this band edge configuration and filter length.

Think of frequency bands as lines over short frequency intervals. `remez` and `firls` use this scheme to represent any piecewise linear desired function with

Meta Platforms, Inc. - Exhibit 1009
Page 89 of 720

any transition bands. firls and remez design lowpass, highpass, bandpass, and bandstop filters; a bandpass example is

```
f = [0 0.3  0.4  0.7  0.8  1];  % band edges in pairs
a = [0  0    1    1    0   0];  % bandpass filter amplitude
```

Technically, these f and a vectors define five bands:

- Two stopbands, from 0.0 to 0.3 and from 0.8 to 1.0
- A passband from 0.4 to 0.7
- Two transition bands, from 0.3 to 0.4 and from 0.7 to 0.8

Example highpass and bandstop filters are

```
f = [0 0.7  0.8  1];  % band edges in pairs
a = [0  0    1   1];  % highpass filter amplitude
```

```
f = [0 0.3  0.4  0.5  0.8  1];  % band edges in pairs
a = [1  1    0    0    1   1];  % bandstop filter amplitude
```

An example multiband bandpass filter is

```
f = [0 0.1 0.15 0.25 0.3 0.4 0.45 0.55 0.6 0.7 0.75 0.85 0.9 1];
a = [1  1   0    0    1   1   0    0    1   1   0    0    1  1];
```

Another possibility is a filter that has as a transition region the line connecting the passband with the stopband; this can help control “runaway” magnitude response in wide transition regions:

```
f = [0 0.4 0.42 0.48 0.5  1];
a = [1   1  0.8  0.2   0  0]; % passband,linear transition,stopband
```

The Weight Vector

Both firls and remez allow you to place more or less emphasis on minimizing the error in certain frequency bands relative to others. To do this, specify a weight vector following the frequency and amplitude vectors. An example

Meta Platforms, Inc. - Exhibit 1009
Page 90 of 720

lowpass equiripple filter with 10 times less ripple in the stopband than the passband is

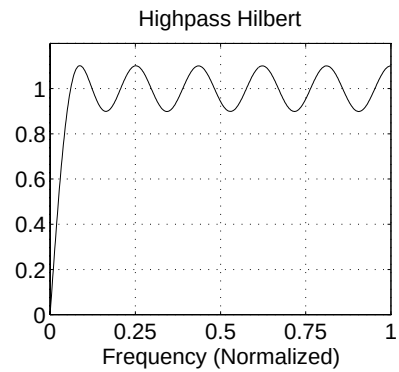
```
n = 20;              % filter order
f = [0 0.4 0.5 1];   % frequency band edges
a = [1  1   0  0];   % desired amplitudes
w = [1 10];          % weight vector
b = remez(n,f,a,w);
```

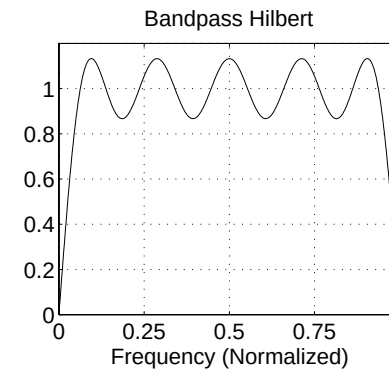
A legal weight vector is always half the length of the `f` and `a` vectors; there must be exactly one weight per band.

Anti-Symmetric Filters / Hilbert Transformers

When called with a trailing `'h'` or `'Hilbert'` option, `remez` and `firls` design FIR filters with odd symmetry, that is, type III (for even order) or type IV (for odd order) linear phase filters. An ideal Hilbert transformer has this anti-symmetry property and an amplitude of 1 across the entire frequency range. Try the following approximate Hilbert transformers:

```
b = remez(21,[0.05 1],[1 1],'h');    % highpass Hilbert
bb = remez(20,[0.05 0.95],[1 1],'h'); % bandpass Hilbert
```





You can find the delayed Hilbert transform of a signal `x` by passing it through these filters:

```
Fs = 1000;           % sampling frequency
t = (0:1/Fs:2)';     % two second time vector
x = sin(2*pi*300*t); % 300 Hz sine wave example signal
xh = filter(bb,1,x); % Hilbert transform of x
```


Meta Platforms, Inc. - Exhibit 1009
Page 91 of 720

The analytic signal corresponding to x is the complex signal that has x as its real part and the Hilbert transform of x as its imaginary part. For this FIR method (an alternative to the hilbert function), you must delay x by half the filter order to create the analytic signal.

```
xd = [zeros(10,1); x(1:length(x)-10)]; % delay 10 samples
xa = xd + j*xh;  % analytic signal
```

This method does not work directly for filters of odd order, which require a noninteger delay. In this case, the hilbert function, described in the "Specialized Transforms" section in Chapter 4, estimates the analytic signal. Alternatively, use the resample function to delay the signal by a noninteger number of samples.

Differentiators

Differentiation of a signal in the time domain is equivalent to multiplication of the signal's Fourier transform by an imaginary ramp function. That is, to differentiate a signal, pass it through a filter that has a response $H(w) = jw$. Approximate the ideal differentiator (with a delay) using remez or firls with a 'd' or 'differentiator' option.

```
b = remez(21,[0 1],[0 pi*Fs],'d');
```

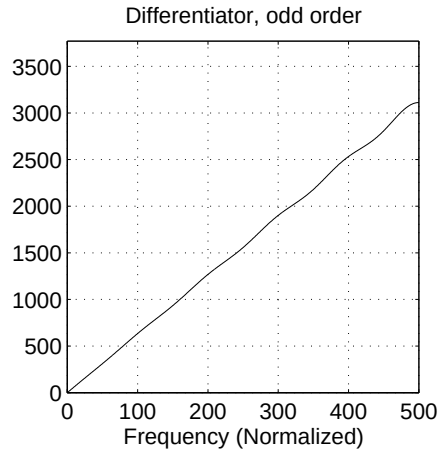
To obtain the correct derivative, scale by pi*Fs rad/sec, where Fs is the sampling frequency in Hertz. For a type III filter, the differentiation band should stop short of the Nyquist frequency, and the amplitude vector must reflect that change to ensure the correct slope.

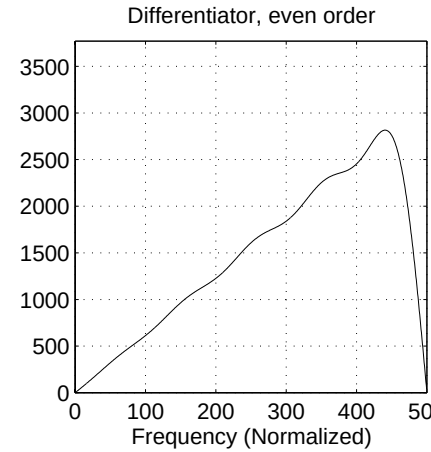
```
bb = remez(20,[0 0.9],[0 0.9*pi*Fs],'d');
```

In the 'd' mode, remez weights the error by $1/w$ in nonzero amplitude bands to minimize the maximum *relative* error. firls weights the error by $(1/w)^2$ in nonzero amplitude bands in the 'd' mode.

Meta Platforms, Inc. - Exhibit 1009
Page 92 of 720

The magnitude response plots for the differentiators shown above are





Constrained Least Squares FIR Filter Design

The Constrained Least Squares (CLS) FIR filter design functions implement a technique that enables you to design FIR filters without explicitly defining the transition bands for the magnitude response. The ability to omit the specification of transition bands is useful in several situations. For example, it may not be clear where a rigidly defined transition band should appear if noise and signal information appear together in the same frequency band. Similarly, it may make sense to omit the specification of transition bands if they appear only to control the results of Gibbs phenomena that appear in the filter's response. See Selesnick, Lang, and Burrus [2] for discussion of this method.

Instead of defining passbands, stopbands, and transition regions, the CLS method accepts a cutoff frequency (for the highpass, lowpass, bandpass, or bandstop cases), or passband and stopband edges (for multiband cases), for the desired response. In this way, the CLS method defines transition regions implicitly, rather than explicitly.

The key feature of the CLS method is that it enables you to define upper and lower thresholds that contain the maximum allowable ripple in the magnitude response. Given this constraint, the technique applies the least square error minimization technique over the frequency range of the filter's response, instead of over specific bands. The error minimization includes any areas of discontinuity in the ideal, "brick wall" response. An additional benefit is that

Meta Platforms, Inc. - Exhibit 1009
Page 93 of 720

the technique enables you to specify arbitrarily small peaks resulting from Gibbs' phenomena.

There are two toolbox functions that implement this design technique.

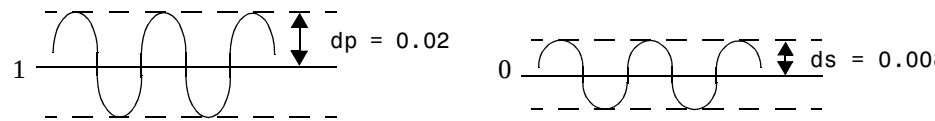
Description	Function
Constrained least square multiband FIR filter design.	`fircls`
Constrained least square filter design for lowpass and highpass linear phase filters	`fircls1`

For details on the calling syntax for these functions, see their reference descriptions in Chapter 6.

Basic Lowpass and Highpass CLS Filter Design

The most basic of the CLS design functions, `fircls1`, uses this technique to design lowpass and highpass FIR filters. As an example, consider designing a filter with order 61 impulse response and cutoff frequency of 0.3 (normalized). Further, define the upper and lower bounds that constrain the design process as:

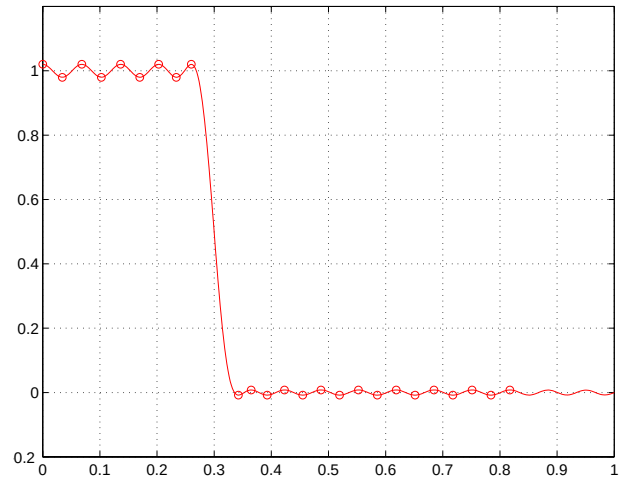
- Maximum passband deviation from 1 (passband ripple) of 0.02.
- Maximum stopband deviation from 0 (stopband ripple) of 0.008.



Meta Platforms, Inc. - Exhibit 1009
Page 94 of 720

To approach this design problem using `fircls1`

```
n = 61;
wo = 0.3;
dp = 0.02;
ds = 0.008;
h = fircls1(n,wo,dp,ds,'plot');
```



Multiband CLS Filter Design

`fircls` uses the same technique to design FIR filters with a desired piecewise constant magnitude response. In this case, you can specify a vector of band edges and a corresponding vector of band amplitudes. In addition, you can specify the maximum amount of ripple for each band.

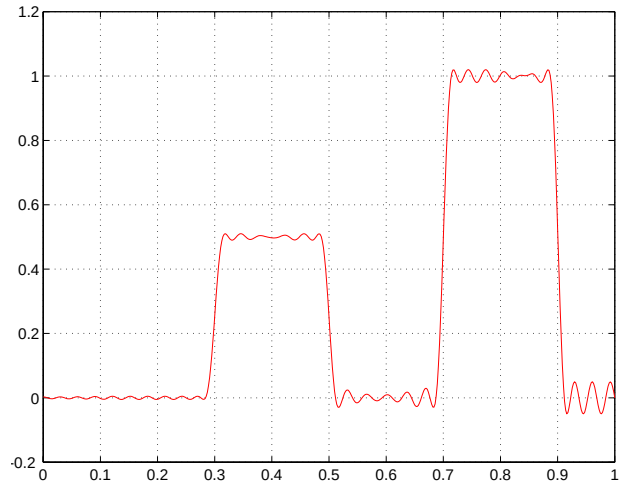
For example, assume the specifications for a filter call for:

- From 0 to 0.3 (normalized): amplitude 0, upper bound 0.005, lower bound -0.005
- From 0.3 to 0.5: amplitude 0.5, upper bound 0.51, lower bound 0.49
- From 0.5 to 0.7: amplitude 0, upper bound 0.03, lower bound -0.03
- From 0.7 to 0.9: amplitude 1, upper bound 1.02, lower bound 0.98
- From 0.9 to 1: amplitude 0, upper bound 0.05, lower bound -0.05

Meta Platforms, Inc. - Exhibit 1009
Page 95 of 720

Design a CLS filter with impulse response order 129 that meets these specifications.

```
n = 129;
f = [0 0.3 0.5 0.7 0.9 1];
a = [0 0.5 0 1 0];
up = [0.005 0.51 0.03 1.02 0.05];
lo = [-0.005 0.49 -0.03 0.98 -0.05];
h = fircls(n,f,a,up,lo,'plot');
```



Weighted CLS Filter Design

Weighted CLS filter design lets you design lowpass or highpass FIR filters with relative weighting of the error minimization in each band. The `fircls1` function enables you to specify the passband and stopband edges for the least squares weighting function, as well as a constant `k` that specifies the ratio of the stopband to passband weighting.

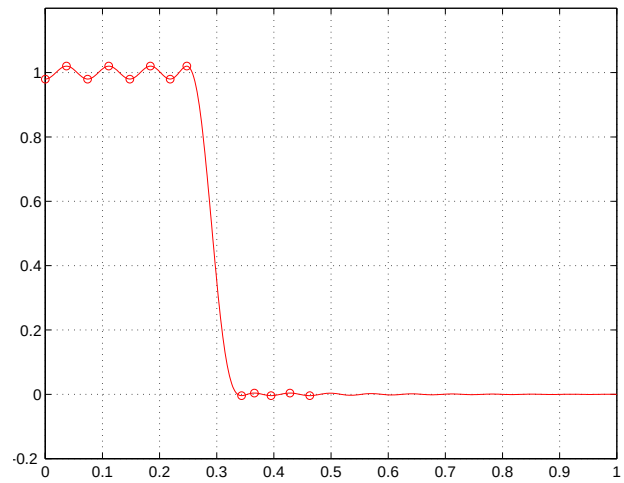
For example, consider specifications that call for an FIR filter with impulse response order of 55 and cutoff frequency of 0.3 (normalized). Also assume maximum allowable passband ripple of 0.02 and maximum allowable stopband ripple of 0.004. In addition, add weighting requirements:

Meta Platforms, Inc. - Exhibit 1009
Page 96 of 720

- Passband edge for the weight function of 0.28 (normalized)
- Stopband edge for the weight function of 0.32
- Weight error minimization 10 times as much in the stopband as in the passband

To approach this using `fircls1`:

```
n = 55;
wo = 0.3;
dp = 0.02;
ds = 0.004;
wp = 0.28;
ws = 0.32;
k = 10;
h = fircls1(n,wo,dp,ds,wp,ws,k,'plot');
```



Arbitrary-Response Filter Design

The `cremez` filter design function provides a tool for designing FIR filters with arbitrary complex responses. It differs from the other filter design functions in how the frequency response of the filter is specified: it accepts the name of a function which returns the filter response calculated over a grid of frequencies. This capability makes `cremez` a highly versatile and powerful technique for filter design.

Meta Platforms, Inc. - Exhibit 1009
Page 97 of 720

This design technique may be used to produce nonlinear-phase FIR filters, asymmetric frequency-response filters (with complex coefficients), or more symmetric filters with custom frequency responses.

The design algorithm optimizes the Chebyshev (or minimax) error using an extended Remez-exchange algorithm for an initial estimate. If this exchange method fails to obtain the optimal filter, the algorithm switches to an ascent-descent algorithm that takes over to finish the convergence to the optimal solution.

For details on the calling syntax for `cremez`, see the reference description in Chapter 6.

Multiband Filter Design

Consider a multiband filter with the following special frequency-domain characteristics:

Band	Amplitude	Optimization Weighting
[-1 -0.5]	[5 1]	1
[-0.4 +0.3]	[2 2]	10
[+0.4 +0.8]	[2 1]	5

A linear-phase multiband filter may be designed using the predefined frequency-response function `multiband`, as follows.

```
b = cremez(38, [-1 -0.5 -0.4 0.3 0.4 0.8], ...
               {'multiband', [5 1 2 2 2 1]}, [1 10 5]);
```

For the specific case of a multiband filter, we can use a shorthand filter design notation similar to the syntax for `remez`:

```
b = cremez(38,[-1 -0.5 -0.4 0.3 0.4 0.8], ...
              [5 1 2 2 2 1], [1 10 5]);
```

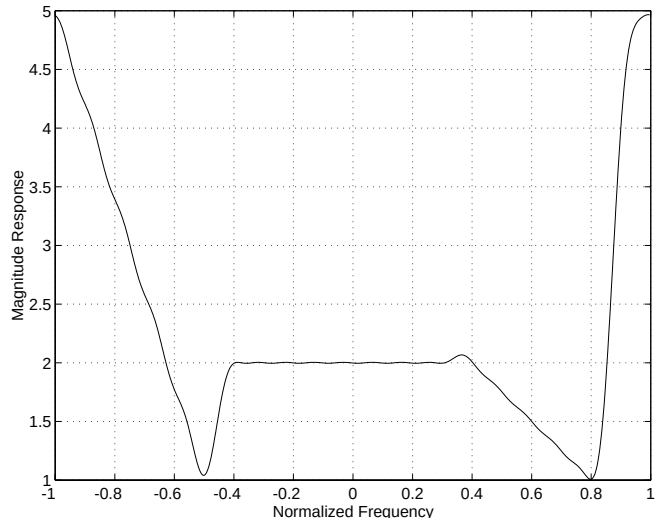
As with `remez`, a vector of band edges is passed to `cremez`. This vector defines the frequency bands over which optimization is performed; note that there are two transition bands, from -0.5 to -0.4 and from 0.3 to 0.4.

Meta Platforms, Inc. - Exhibit 1009
Page 98 of 720

In either case, the frequency response is obtained and plotted using linear scale.

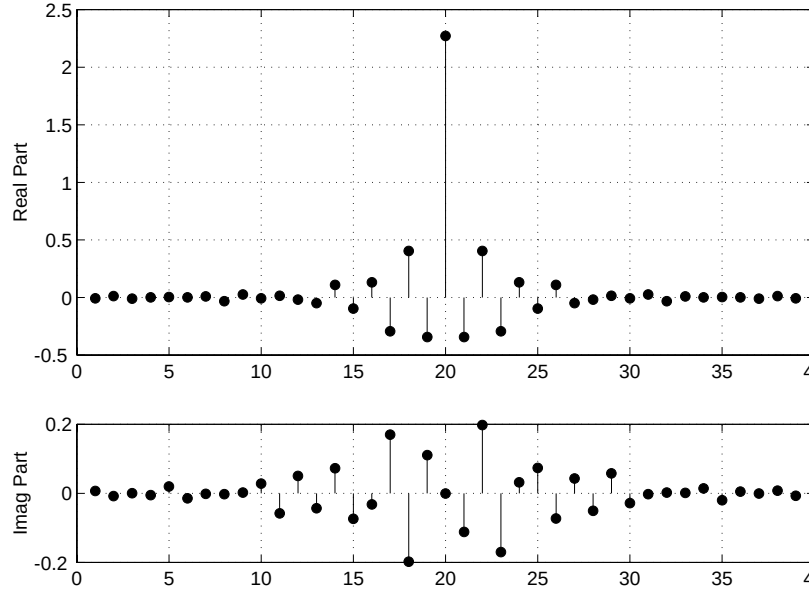
```
[h,w] = freqz(b,1,512,'whole');
plot(w/pi-1,fftshift(abs(h)));
```

Note that the frequency response has been calculated over the entire normalized frequency range [-1 +1] by passing the option `'whole'` to `freqz`. In order to plot the negative frequency information in a natural way, the response has been "wrapped," just as FFT data is, using `fftshift`.



Meta Platforms, Inc. - Exhibit 1009
Page 99 of 720

The filter response for this multiband filter is complex, which is expected because of the asymmetry in the frequency domain. The filter response is



Filter Design with Reduced Delay

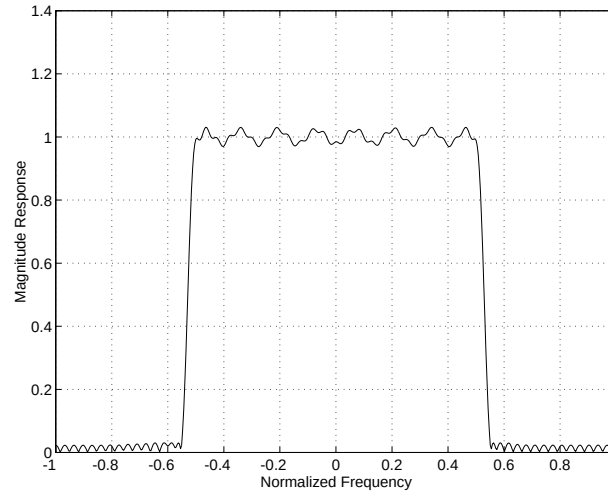
Consider the design of a 62-tap lowpass filter with a half-Nyquist cutoff. If we specify a negative offset value to the `lowpass` filter design function, the group delay offset for the design is significantly less than that obtained for a standard linear-phase design. This filter design may be computed as follows.

```
b = cremez(61, [0 0.5 0.55 1], {'lowpass', -16});
```

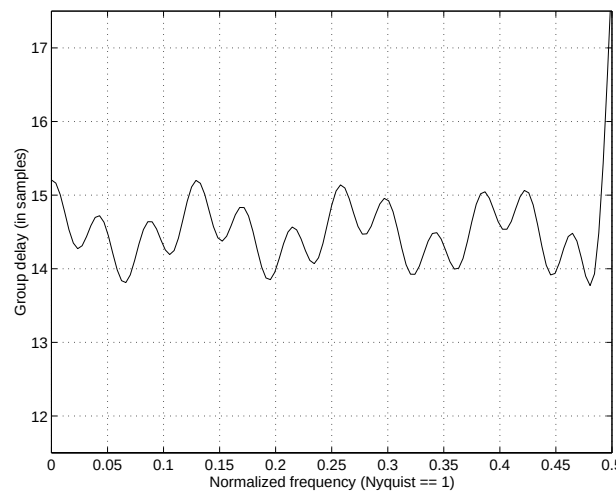

Meta Platforms, Inc. - Exhibit 1009
Page 100 of 720

The resulting magnitude response is

```
[h,w] = freqz(b,1,512,'whole');
plot(w/pi-1,fftshift(abs(h)));
```



The group delay of the filter reveals that the offset has been reduced from `N/2=30.5` to `N/2-16=14.5`. Now, however, the group delay is no longer flat in the passband region (plotted over the normalized frequency range 0 to 0.5 for clarity).



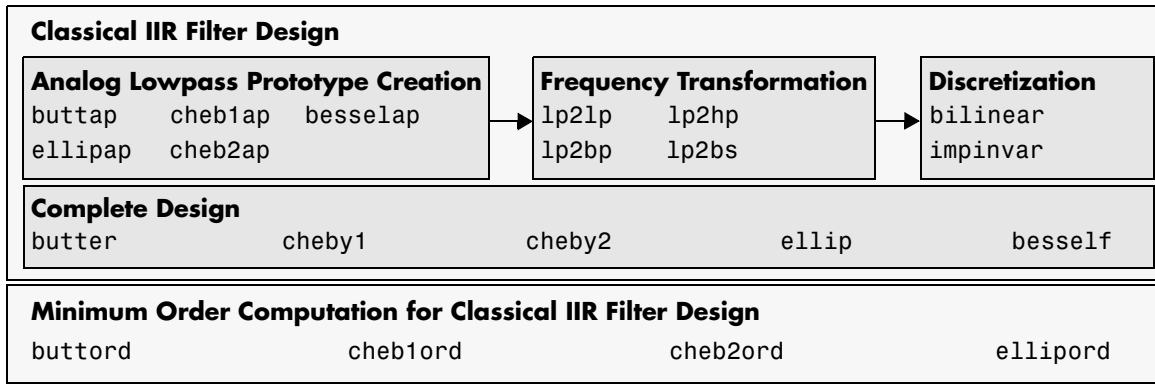
Meta Platforms, Inc. - Exhibit 1009
Page 101 of 720

If we compare this nonlinear-phase filter to a linear-phase filter that has exactly 14.5 samples of group delay, the resulting filter is of order 2*14.5 or 29. Using `b = cremez(29,[0 0.5 0.55 1],'lowpass')`, the passband and stopband ripple is much greater for the order 29 filter. These comparisons can assist you in deciding which filter is more appropriate for a specific application.

Meta Platforms, Inc. - Exhibit 1009
Page 102 of 720

Special Topics in IIR Filter Design

The classic IIR filter design technique finds an analog lowpass filter with cutoff frequency of 1, translates this “prototype” filter to the desired band configuration, then transforms the filter to the digital domain. The toolbox provides functions for each step of this process.



The `butter`, `cheby1`, `cheby2`, and `ellip` functions are sufficient for many design problems, and the lower level functions are generally not needed. But if you do have an application where you need to transform the band edges of an analog filter, or discretize a rational transfer function, this section describes tools to do so.

Meta Platforms, Inc. - Exhibit 1009
Page 103 of 720

Analog Prototype Design

This toolbox provides a number of functions to create lowpass analog prototype filters with cutoff frequency of 1, the first step in the classical approach to IIR filter design. The table below summarizes the analog prototype design functions for each supported filter type; plots for each type are shown in the "IIR Filter Design" section above.

Filter Type	Analog Prototype Function
Bessel	`[z,p,k] = besselap(n)`
Butterworth	`[z,p,k] = buttap(n)`
Chebyshev type I	`[z,p,k] = cheb1ap(n,Rp)`
Chebyshev type II	`[z,p,k] = cheb2ap(n,Rs)`
Elliptic	`[z,p,k] = ellipap(n,Rp,Rs)`

Frequency Transformation

The second step in the analog prototyping design technique is the frequency transformation of a lowpass prototype. The toolbox provides a set of functions to transform analog lowpass prototypes (with cutoff frequency of 1 rad/sec) into bandpass, highpass, bandstop, and lowpass filters of the desired cutoff frequency.

Meta Platforms, Inc. - Exhibit 1009
Page 104 of 720

Freq. Transformation	Transformation Function
Lowpass to lowpass $s' = s/\omega_0$	`[numt,dent] = lp2lp(num,den,Wo)` `[At,Bt,Ct,Dt] = lp2lp(A,B,C,D,Wo)`
Lowpass to highpass $s' = \frac{\omega_0}{s}$	`[numt,dent] = lp2hp(num,den,Wo)` `[At,Bt,Ct,Dt] = lp2hp(A,B,C,D,Wo)`
Lowpass to bandpass $s' = \frac{\omega_0}{B_\omega}\frac{(s/\omega_0)^2+1}{s/\omega_0}$	`[numt,dent] = lp2bp(num,den,Wo,Bw)` `[At,Bt,Ct,Dt] = lp2bp(A,B,C,D,Wo,Bw)`
Lowpass to bandstop $s' = \frac{B_\omega}{\omega_0}\frac{s/\omega_0}{(s/\omega_0)^2+1}$	`[numt,dent] = lp2bs(num,den,Wo,Bw)` `[At,Bt,Ct,Dt] = lp2bs(A,B,C,D,Wo,Bw)`

As shown, all of the frequency transformation functions can accept two linear system models: transfer function and state-space form. For the bandpass and bandstop cases

$$\omega_0 = \sqrt{\omega_1\omega_2}$$

and

$$B_\omega = \omega_2 - \omega_1$$

where ω_1 is the lower band edge and ω_2 is the upper band edge.

The frequency transformation functions perform frequency variable substitution. In the case of `lp2bp` and `lp2bs`, this is a second-order substitution, so the output filter is twice the order of the input. For `lp2lp` and `lp2hp`, the output filter is the same order as the input.

Meta Platforms, Inc. - Exhibit 1009
Page 105 of 720

To begin designing an order 10 bandpass Chebyshev type I filter with a value of 3 dB for passband ripple

```
[z,p,k] = cheb1ap(5,3);
```

z, p, and k contain the poles, zeros, and gain of a lowpass analog filter with cutoff frequency Ω_c equal to 1 rad/sec. Use the lp2bp function to transform this lowpass prototype to a bandpass analog filter with band edges $W_1 = \pi/5$ and $W_2 = \pi$. First, convert the filter to state-space form so the lp2bp function can accept it.

```
[A,B,C,D] = zp2ss(z,p,k); % Convert to state-space form.
```

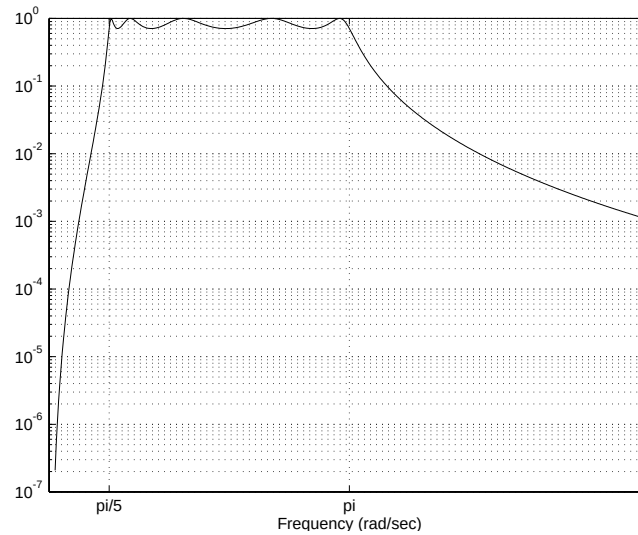
Now, find the bandwidth and center frequency, and call lp2bp.

```
u1 = 0.1*2*pi;  u2 = 0.5*2*pi; % in radians per second
Bw = u2-u1;
Wo = sqrt(u1*u2);
[At,Bt,Ct,Dt] = lp2bp(A,B,C,D,Wo,Bw);
```


Meta Platforms, Inc. - Exhibit 1009
Page 106 of 720

Finally, calculate the frequency response and plot its magnitude.

```
[b,a] = ss2tf(At,Bt,Ct,Dt);% Convert to TF form.
w = linspace(.01,1,500)*2*pi;% Generate frequency vector.
h = freqs(b,a,w);% Compute frequency response.
semilogy(w/2/pi,abs(h)), grid% Plot log magnitude vs. freq.
```



Filter Discretization

The third step in the analog prototyping technique is the transformation of the filter to the discrete-time domain. The toolbox provides two methods for this: the impulse invariant and bilinear transformations. The filter design functions

Meta Platforms, Inc. - Exhibit 1009
Page 107 of 720

`butter`, `cheby1`, `cheby2`, and `ellip` use the bilinear transformation for discretization in this step.

Analog to Digital Transformation	Transformation Function
Impulse invariance	`[numd,dend] = impinvar(num,den,Fs)`

Bilinear transform	`[zd,pd,kd] = bilinear(z,p,k,Fs,Fp)`
	`[numd,dend] = bilinear(num,den,Fs,Fp)`
	`[Ad,Bd,Cd,Dd] = bilinear(At,Bt,Ct,Dt,Fs,Fp)`

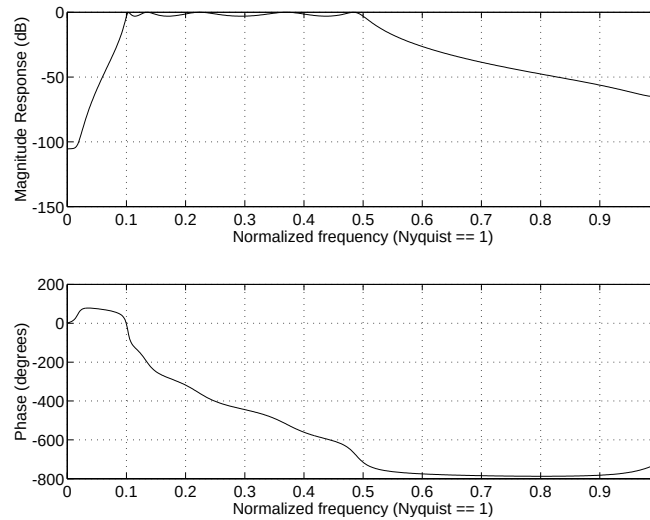
Impulse Invariance

The toolbox function `impinvar` creates a digital filter whose impulse response is the samples of the continuous impulse response of an analog filter. This function works only on filters in transfer function form. For best results, the analog filter should have negligible frequency content above half the sampling frequency, because such high frequency content is aliased into lower bands upon sampling. Impulse invariance works for some lowpass and bandpass filters, but is not appropriate for highpass and bandstop filters.

Meta Platforms, Inc. - Exhibit 1009
Page 108 of 720

Design a Chebyshev type I filter and plot its frequency response.

```
[bz,az] = impinvar(b,a,2);
freqz(bz,az)
```



Impulse invariance retains the cutoff frequencies of 0.1 Hz and 0.5 Hz.

Bilinear Transformation

The bilinear transformation is a nonlinear mapping of the continuous domain to the discrete domain; it maps the *s*-plane into the *z*-plane by

$$H(z) = H(s)\Big|_{s = k\frac{z-1}{z+1}}$$

Bilinear transformation maps the jΩ axis of the continuous domain to the unit circle of the discrete domain according to

$$\omega = 2\tan^{-1}\left(\frac{\Omega}{k}\right)$$

The toolbox function `bilinear` implements this operation, where the frequency warping constant k is equal to twice the sampling frequency (`2*Fs`) by default and equal to `2*pi*Fp/tan(pi*Fp/Fs)` if you give `bilinear` a trailing argument that represents a “match” frequency `Fp`. If a match frequency `Fp` (in Hertz) is

Meta Platforms, Inc. - Exhibit 1009
Page 109 of 720

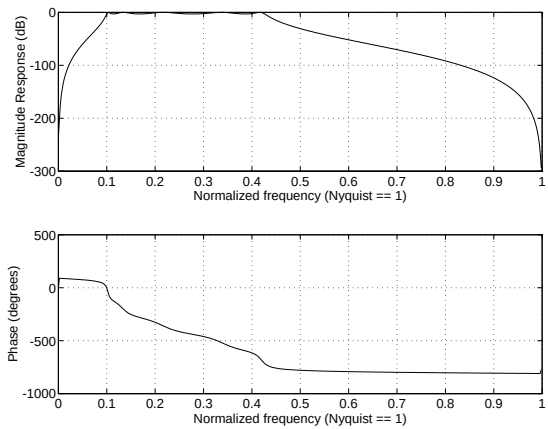
present, `bilinear` maps the frequency $\Omega = 2\pi f_p$ (in radians/second) to the same frequency in the discrete domain, normalized to the sampling rate: $\omega = 2\pi f_p / f_s$ (also in radians/second).

The `bilinear` function can perform this transformation on three different linear system representations: zero-pole-gain, transfer function, and state-space form. Try calling `bilinear` with the state-space matrices that describe the Chebyshev type I filter from the previous section, using a sampling frequency of 2 Hz, and retaining the lower band edge of 0.1 Hz.

```
[Ad,Bd,Cd,Dd] = bilinear(At,Bt,Ct,Dt,2,0.1);
```

The frequency response of the resulting digital filter is

```
[bz,az] = ss2tf(Ad,Bd,Cd,Dd); % convert to TF
freqz(bz,az)
```



The lower band edge is at 0.1 Hz as expected. Notice, however, that the upper band edge is slightly less than 0.5 Hz, although in the analog domain it was exactly 0.5 Hz. This illustrates the nonlinear nature of the bilinear transformation. To counteract this nonlinearity, it is necessary to create analog domain filters with “prewarped” band edges, which map to the correct locations

Meta Platforms, Inc. - Exhibit 1009
Page 110 of 720

upon bilinear transformation. Here the prewarped frequencies u1 and u2 generate Bw and Wo for the lp2bp function.

```
Fs = 2;                          % sampling frequency (Hertz)
u1 = 2*Fs*tan(.1*(2*pi/Fs)/2); % lower band edge (radians/second)
u2 = 2*Fs*tan(.5*(2*pi/Fs)/2); % upper band edge (radians/second)
Bw = u2-u1;                      % bandwidth
Wo = sqrt(u1*u2);                % center frequency
[At,Bt,Ct,Dt] = lp2bp(A,B,C,D,Wo,Bw);
```

A digital bandpass filter with correct band edges 0.1 and 0.5 times the Nyquist frequency is

```
[Ad,Bd,Cd,Dd] = bilinear(At,Bt,Ct,Dt,Fs);
```

The example bandpass filters from the last two sections could also be created in one statement using the complete IIR design function cheby1. For instance, an analog version of the example Chebyshev filter is

```
[b,a] = cheby1(5,3,[0.1 0.5]*2*pi,'s');
```

Note that the band edges are in radians/second for analog filters, whereas for the digital case, frequency is normalized (the Nyquist frequency is equal to 1 Hz).

```
[bz,az] = cheby1(5,3,[0.1 0.5]);
```

All of the complete design functions call bilinear internally. They prewarp the band edges as needed to obtain the correct digital filter. See Chapter 6 for more on these functions.

Meta Platforms, Inc. - Exhibit 1009
Page 111 of 720

References

1. Karam, L.J., and J.H. McClellan. "Complex Chebyshev Approximation for FIR Filter Design." *IEEE Trans. on Circuits and Systems II*. March 1995.

2. Selesnick, I.W., and C.S. Burrus. "Generalized Digital Butterworth Filter Design." *Proceedings of the IEEE Int. Conf. Acoust., Speech, Signal Processing*. Vol. 3 (May 1996).

3. Selesnick, I.W., M. Lang, and C.S. Burrus. "Constrained Least Square Design of FIR Filters without Specified Transition Bands." *Proceedings of the IEEE Int. Conf. Acoust., Speech, Signal Processing*. Vol. 2 (May 1995). Pgs. 1260-1263.

Meta Platforms, Inc. - Exhibit 1009
Page 112 of 720

3

Statistical Signal Processing

Correlation and Covariance 3-2
Bias and Normalization 3-3
Multiple Channels 3-4

Spectral Analysis 3-5
Welch's Method 3-6
Multitaper Method 3-16
Yule-Walker AR Method 3-19
Burg Method 3-20
Covariance and Modified Covariance Methods 3-22
MUSIC and Eigenvector Analysis Methods 3-23

References 3-27

Correlation and Covariance

The Signal Processing Toolbox provides tools for estimating important functions of random signals. In particular, there are tools to estimate correlation and covariance sequences and spectral density functions of discrete signals. This chapter explains the correlation and covariance functions and discusses the mathematically related functions for estimating the power spectrum.

The functions `xcorr` and `xcov` estimate the cross-correlation and cross-covariance sequences of random processes. They also handle autocorrelation and autocovariance as special cases.

The true cross-correlation sequence is a statistical quantity defined as

$$\gamma_{xy}(m) = E\{x_n y^*_{n+m}\}$$

where x_n and y_n are stationary random processes, $-\infty < n < \infty$, and $E\{\ \}$ is the expected value operator. The cross-covariance sequence is the mean-removed cross-correlation sequence

$$C_{xy}(m) = E\{(x_n - \mu_x)(y^*_{n+m} - \mu^*_x)\}$$

or, in terms of the cross-correlation,

$$C_{xy}(m) = \gamma_{xy}(m) - \mu_x \mu^*_y$$

In practice, you must estimate these sequences, because it is possible to access only a finite segment of the infinite-length random process. A common estimate based on N samples of x_n and y_n is the deterministic cross-correlation sequence (also called the time-ambiguity function)

$$\hat{R}_{xy}(m) = \begin{cases} \displaystyle\sum_{n=0}^{N-|m|-1} x_n y^*_{n+m} & m \geq 0 \\ \hat{R}^*_{yx}(-m) & m < 0 \end{cases}$$

Meta Platforms, Inc. - Exhibit 1009
Page 114 of 720

where we assume for this discussion that x_n and y_n are indexed from 0 to N-1,

and $\hat{R}^*_{yx}(-m)$ from -(N-1) to N-1. The `xcorr` function evaluates this sum with an efficient FFT-based algorithm, given inputs x_n and y_n stored in length N vectors `x` and `y`. Its operation is equivalent to convolution with one of the two subsequences reversed in time.

For example,

```
x = [1 1 1 1 1]';
y = x;
xyc = xcorr(x,y)

xyc =

    1.0000
    2.0000
    3.0000
    4.0000
    5.0000
    4.0000
    3.0000
    2.0000
    1.0000
```

Notice that the resulting sequence is twice the length of the input sequence minus 1. Thus, the Nth element is the correlation at lag 0. Also notice the triangular pulse of the output that results when convolving two square pulses.

The `xcov` function estimates autocovariance and cross-covariance sequences. This function has the same options and evaluates the same sum as `xcorr`, but first removes the means of `x` and `y`.

Bias and Normalization

An estimate of a quantity is *biased* if its expected value is not equal to the quantity it estimates. The expected value of the output of `xcorr` is

$$E\{\hat{R}_{xy}(m)\} = \sum_{n=0}^{N-|m|-1} E\{x_n y^*_{n+m}\} = (N-|m|)\gamma_{xy}(m)$$

Meta Platforms, Inc. - Exhibit 1009
Page 115 of 720

`xcorr` provides the unbiased estimate, dividing by $N-|m|$, when you specify an `'unbiased'` flag after the input sequences.

```
xcorr(x,y,'unbiased')
```

Although this estimate is unbiased, the end points (near $-(N-1)$ and $N-1$) suffer from large variance because `xcorr` computes them using only a few data points. A possible trade-off is to simply divide by N using the `'biased'` flag.

```
xcorr(x,y,'biased')
```

With this scheme, only the sample of the correlation at 0 lag (the Nth output element) is unbiased. This estimate is often more desirable than the unbiased one because it avoids random large variations at the end points of the correlation sequence.

`xcorr` provides one other normalization scheme. The syntax

```
xcorr(x,y,'coeff')
```

divides the output by `norm(x)*norm(y)` so that, for autocorrelations, the sample at 0 lag is 1.

Multiple Channels

For a multichannel signal, `xcorr` and `xcov` estimate the autocorrelation and cross-correlation and covariance sequences for all of the channels at once. If `S` is an M-by-N signal matrix representing N channels in its columns, `xcorr(S)` returns a $(2M-1)$-by-N^2 matrix with the autocorrelations and cross-correlations of the channels of `S` in its N^2 columns. If `S` is a 3-channel signal

```
S = [s1 s2 s3]
```

then the result of `xcorr(S)` is organized as

```
R = [Rs1s1 Rs1s2 Rs1s3 Rs2s1 Rs2s2 Rs2s3 Rs3s1 Rs3s2 Rs3s3]
```

Two related functions, `cov` and `corrcoef`, are available in the standard MATLAB environment. They estimate covariance and normalized covariance respectively between the different channels at lag 0 and arrange them in a square matrix.

Meta Platforms, Inc. - Exhibit 1009
Page 116 of 720

Spectral Analysis

Spectral analysis seeks to describe the frequency content of a signal, random process, or system, based on a finite set of data. Estimation of power spectra is useful in a variety of applications, including the detection of signals buried in wide-band noise.

The *power spectral density* (PSD) of a stationary random process x_n is related mathematically to the correlation sequence by the discrete-time Fourier transform,

$$P_{xx}(\omega) = \sum_{m=-\infty}^{\infty} \gamma_{xx}(m) e^{-j\omega m}$$

This function of frequency has the property that its integral over a frequency band is equal to the power of the signal x_n in that band.

The PSD is a special case of the *cross spectral density* (CSD) function, defined between two signals x_n and y_n as

$$P_{xy}(\omega) = \sum_{m=-\infty}^{\infty} \gamma_{xy}(m) e^{-j\omega m}$$

As is the case for the correlation and covariance sequences, the toolbox *estimates* the PSD and CSD because signal lengths are finite.

The various methods of PSD estimation can be identified as *parametric* or *nonparametric*. One technique offered in the Signal Processing Toolbox is the popular nonparametric scheme developed by Welch [5]. This is complemented by more modern nonparametric techniques such as the *multitaper method* (MTM) and the *multiple signal classification* (MUSIC) or *eigenvector* (EV) *method*, which is well suited for line spectra (data made up of sinusoids). The *Yule-Walker autoregressive* (AR) *method* is a parametric method that estimates the autocorrelation function to solve for the AR model parameters. The *Burg method* is another parametric spectral estimation method that minimizes the forward and backward linear prediction errors while satisfying the Levinson-Durbin recursion. These methods are listed in the table below with the corresponding toolbox function name. The number below each method name indicates the page that describes the method in greater detail. See

Meta Platforms, Inc. - Exhibit 1009
Page 117 of 720

"Parametric Modeling" in Chapter 4 for details about `lpc` and other parametric estimation functions.

Method	Description	Functions
Burg (3-20)	Autoregressive (AR) spectral estimation of a time-series by minimization of linear prediction errors	`pburg`
Covariance (3-22)	Autoregressive (AR) spectral estimation of a time-series by minimization of the forward prediction errors	`pcov`
Modified Covariance (3-22)	Autoregressive (AR) spectral estimation of a time-series by minimization of the forward and backward prediction errors	`pmcov`
Multitaper (3-16)	Spectral estimate from combination of multiple orthogonal windows (or "tapers")	`pmtm`
MUSIC (3-23)	Multiple signal classification or eigenvector method	`pmusic`
Welch (3-6)	Averaged periodograms of overlapped, windowed signal sections	`pwelch, csd, tfe, cohere`
Yule-Walker AR (3-19)	Autoregressive (AR) spectral estimate of a time-series from its estimated autocorrelation function	`pyulear`

Welch's Method

One way of estimating the power spectrum of a process is to simply find the discrete-time Fourier transform of the samples of the process (usually done on

Meta Platforms, Inc. - Exhibit 1009
Page 118 of 720

a grid with an FFT) and take the magnitude squared of the result. An example 1001-element signal xn, which consists of two sinusoids plus noise, is given by

```
Fs = 1000;      % sampling frequency
t = 0:1/Fs:1;   % one second worth of samples
xn = sin(2*pi*50*t) + 2*sin(2*pi*120*t) + randn(size(t));
```

A crude estimate of the PSD of xn is

```
Pxx = abs(fft(xn,1024)).^2/1001;
```

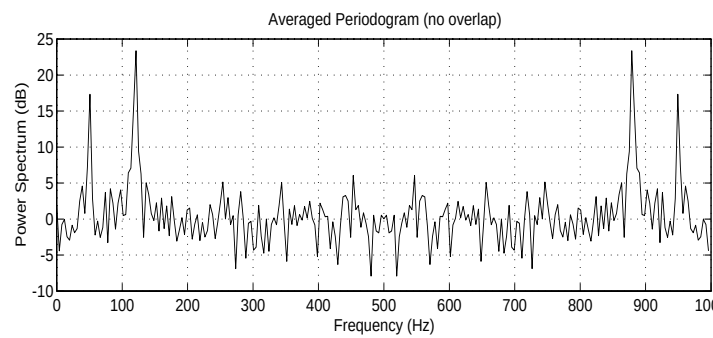
This estimate is called the *periodogram*. Scale the magnitude squared of the FFT by the square of the norm of the data window applied to the signal (in this case, a length 1001 rectangular window) to ensure that the estimate is *asymptotically unbiased*. That is, as the number of samples increases, the expected value of the periodogram approaches the true PSD.

The problem with the periodogram estimate is that its variance is large (on the order of the PSD squared) and does not decrease as the number of samples increases. The following two examples show this; as FFT length increases, the periodogram does not become smoother.

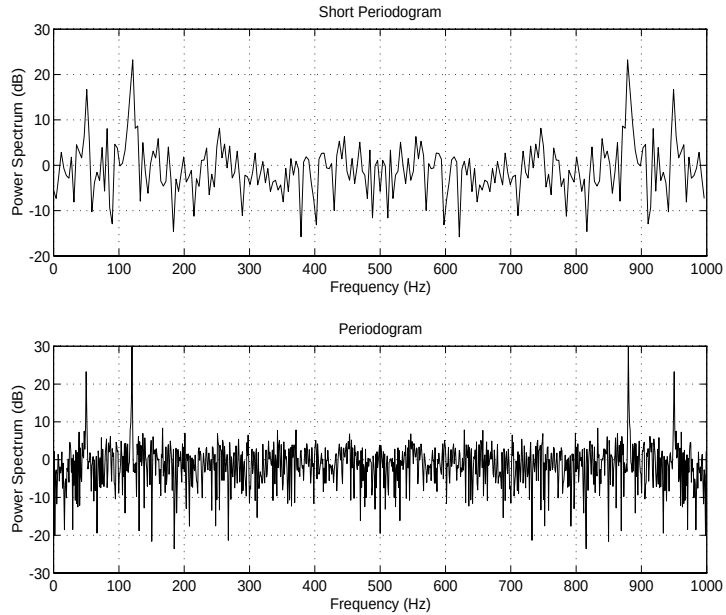
```
Pxx_short = abs(fft(xn,256)).^2/256;
plot((0:255)/256*Fs,10*log10(Pxx_short))
plot((0:1023)/1024*Fs,10*log10(Pxx))
```

Reduce the variance of the PSD estimate by breaking the signal into nonoverlapping sections and averaging the periodograms of these sections.

```
Pxx = (abs(fft(xn(  1:256))).^2 + abs(fft(xn(257:512))).^2 + ...
       abs(fft(xn(513:768))).^2 ) / (256*3);
plot((0:255)/256*Fs,10*log10(Pxx))
```



Meta Platforms, Inc. - Exhibit 1009
Page 119 of 720

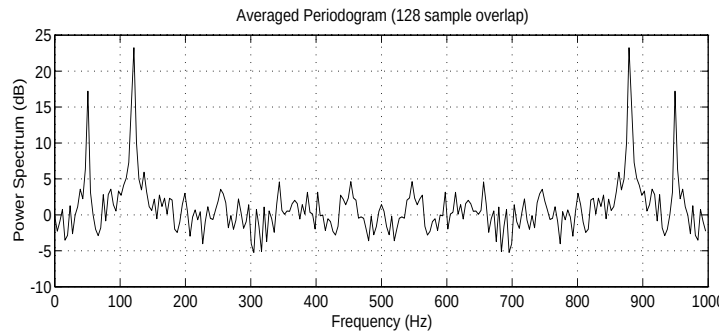


This averaged estimate has one third the variance of the length 256 periodogram shown earlier. The more sections you average, the lower the variance of the result. However, the signal length limits the number of sections

Meta Platforms, Inc. - Exhibit 1009
Page 120 of 720

possible (to three sections of length 256 in the previous example). To obtain more sections, break the signal into overlapping sections.

```
Pxx = (abs(fft(xn(  1:256))).^2 + abs(fft(xn(129:384))).^2 + ...
       abs(fft(xn(257:512))).^2 + abs(fft(xn(385:640))).^2 + ...
       abs(fft(xn(513:768))).^2 + ...
       abs(fft(xn(641:896))).^2 ) / (256*6);
plot((0:255)/256*Fs,10*log10(Pxx))
```



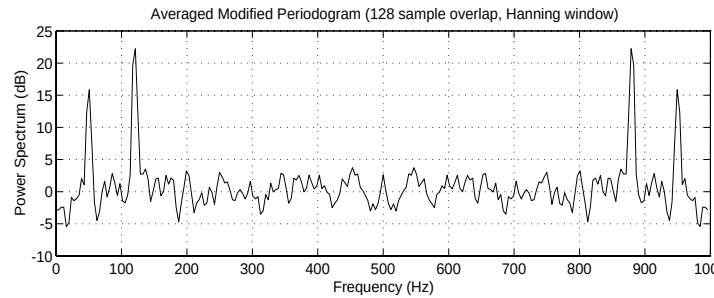
In this case the sections are statistically dependent, resulting in higher variance; thus there is a trade-off between the number of sections and the overlap rate.

Another way to improve the periodogram estimate is to apply a nonrectangular data window to the sections before computing the periodogram, resulting in a *modified periodogram*. This reduces the effect of section dependence due to overlap, because the window is tapered to 0 on the edges. Also, a nonrectangular window diminishes the side-lobe interference or “spectral leakage” while increasing the width of spectral peaks. With a suitable window (such as Hamming, Hanning, or Kaiser), overlap rates of about half the section length have been found to lower the variance of the estimate significantly.

Meta Platforms, Inc. - Exhibit 1009
Page 121 of 720

The application of a Hanning window results in

```
w = hanning(256)';
Pxx =  ( abs(fft(w.*xn(  1:256))).^2 + ...
         abs(fft(w.*xn(129:384))).^2 + ...
         abs(fft(w.*xn(257:512))).^2 + ...
         abs(fft(w.*xn(385:640))).^2 + ...
         abs(fft(w.*xn(513:768))).^2 + ...
         abs(fft(w.*xn(641:896))).^2 ) / (norm(w)^2*6);
plot((0:255)/256*Fs,10*log10(Pxx))
```



Notice in this plot that the spectral peaks have widened, and the *noise floor*, or level of the noise, seems to be the flattest of any estimate so far. This method of averaged, modified periodograms is Welch's method of PSD estimation.

The functions `pwelch` and `csd` provide control over all the parameters discussed so far (FFT length, window, and amount of overlap) in computing the PSD and CSD of signals using Welch's method.

For a more detailed discussion of Welch's method of PSD estimation, see Kay [1] and Welch [5].

Power Spectral Density Function

The `pwelch` function averages and scales the modified periodograms of sections of a signal. Simply specify the parameters that control the algorithm as arguments to the function.

An estimate for the PSD of a sequence `xn` using `pwelch`'s default FFT length (256), window (Hanning of length 256), and overlap samples (none) is

```
Pxx = pwelch(xn);
```


Meta Platforms, Inc. - Exhibit 1009
Page 122 of 720

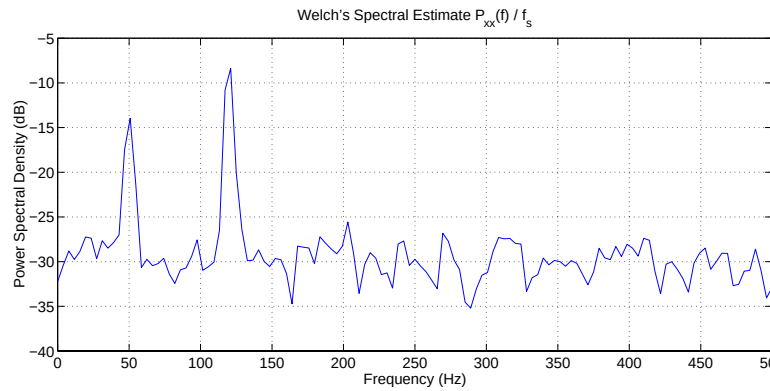
Pxx has units of power per unit frequency interval. For example, if the original sequence xn has units of volts, Pxx has units of Watts/Hz.

To recreate the last example accurately, specify 128 as the number of samples to overlap:

```
nfft = 256;                 % length of FFT
window = hanning(256);      % window function
noverlap = 128;             % number of samples overlap
Pxx = pwelch(xn,nfft,Fs,window,noverlap);
```

Pxx is scaled by the reciprocal of the sampling frequency, 1/Fs. pwelch without any outputs generates a plot of the PSD over the frequency range [0,Fs/2):

```
pwelch(xn,nfft,Fs,window,noverlap)
```



If you want to plot the PSD yourself, obtain the frequency vector through an additional output argument.

```
[Pxx,f] = pwelch(xn,nfft,Fs,256,noverlap);
plot(f,10*log10(Pxx))
```

All the spectral estimation functions allow you to specify an empty matrix, [], in place of an input argument to use the default value of that argument. In the command above, because the values for nfft and window are actually the same as the defaults, you could replace them both with the empty matrix.

```
[Pxx,f] = pwelch(xn,[],Fs,[],noverlap);
```


Meta Platforms, Inc. - Exhibit 1009
Page 123 of 720

Since the signal `xn` is real, `pwelch` returns only the frequencies from 0 through the Nyquist frequency. In contrast, the earlier FFT example generated PSD estimates ranging from 0 through `Fs`.

Bias and Normalization in Welch's Method

In studying the output of `pwelch` shown earlier, several revealing characteristics about the signal `xn` are evident. The noise floor is flat at 0 decibels (dB), implying white noise of variance 1. Furthermore, the "signal" part of `xn` is concentrated in two peaks at 50 and 120 Hz. The relation of the peak heights is meaningful. For instance, the 50 Hz peak is 6 dB below the 120 Hz peak, verifying that the higher frequency sinusoid has twice the magnitude as the lower ($10^{6/20} = 2.0$). Unlike the relative heights, the actual height of the peaks does not tell us much about the original amplitude of the sinusoids without additional analysis.

To obtain useful information about the peak amplitudes of the underlying sinusoids, note that the expected value of the estimated PSD is

$$E\{\hat{P}_{xx}(\omega)\} = \frac{1}{2\pi\|w\|^2}\int_{-\pi}^{\pi} P_{xx}(\theta)|W(\omega-\theta)|\,d\theta$$

Because the expected value is not equal to the true PSD, the estimate is *biased*. This quantity is the convolution of the true PSD with the squared magnitude of the window's discrete-time Fourier transform $W(\omega)$, scaled by the squared norm of the window. The scaling factor is the sum of the squares of the window function:

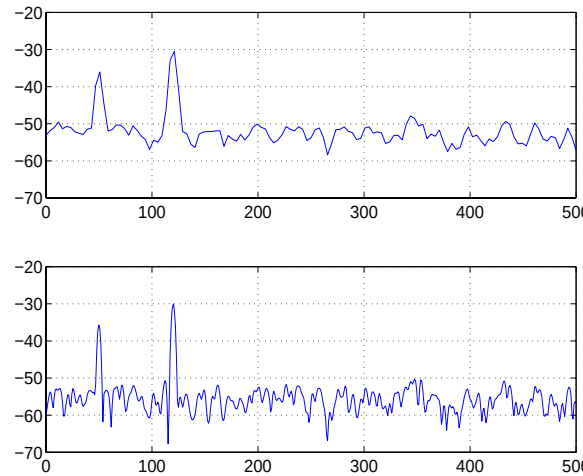
$$\|w\|^2 = \sum w(n)^2$$

This says that if $P_{xx}(\omega)$ has a peak of height 1 at a particular frequency ω_0, the estimate will have approximate height $|W(0)|^2/\|w\|^2$ at that frequency, provided the window $W(\omega)$ is narrow with respect to the spacing between the peak and other spectral features. So, to obtain an estimate which on average reflects the height of the original peaks, multiply the result of `pwelch` by

Meta Platforms, Inc. - Exhibit 1009
Page 124 of 720

`norm(w)^2/sum(w)^2`, where `w` is the window vector. This scaling is independent of window length and shape. For example:

```
w1 = hanning(256); w2 = hanning(500);
[Pxx1,f1] = pwelch(xn,256,Fs,w1,128);
[Pxx2,f2] = pwelch(xn,1024,Fs,w2,250);
subplot(2,1,1)
plot(f1,10*log10(Pxx1*norm(w1)^2/sum(w1)^2))
axis([0 500 –70 –20]); grid
subplot(2,1,2)
plot(f2,10*log10(Pxx2*norm(w2)^2/sum(w2)^2))
axis([0 500 –70 –20]); grid
```



In both plots, which show the spectrum at positive frequencies only (the negative frequencies are the same), the higher frequency peak has a value of 0 dB, and the lower frequency peak is at -6 dB. The 120 Hz sinusoid height of 0 dB corresponds to a squared amplitude of 1. This results from the sinusoid of amplitude 2 having complex exponential components of amplitude 1 at both positive and negative frequency. Similarly, the 50 Hz sinusoid has both positive and negative frequency components with squared amplitude of $(½)^2$ = ¼, or 10*log10(.25) = -6 dB, as shown in the plot. Also, note that the second plot reflects a slightly lower noise floor, which is the result of a longer window length.

Meta Platforms, Inc. - Exhibit 1009
Page 125 of 720

Cross-Spectral Density Function

To estimate the cross-spectral density of two equal length signals x and y using Welch's method, the csd function forms the periodogram as the product of the FFT of x and the conjugate of the FFT of y. Unlike the real-valued PSD, the CSD is a complex function. csd handles the sectioning and windowing of x and y in the same way as the pwelch function.

```
Pxy = csd(x,y,nfft,Fs,window,noverlap)
```

Confidence Intervals

Both the pwelch and csd functions can compute confidence intervals. Simply provide an input argument p, which specifies the percentage of the confidence interval.

```
[Pxx,Pxxc,f] = pwelch(x,nfft,Fs,window,noverlap,p)
[Pxy,Pxyc,f] = csd(x,y,nfft,Fs,window,noverlap,p)
```

p must be a scalar between 0 and 1. The functions assume chi-squared distributed periodograms of nonoverlapping sections in computing the confidence intervals. (This assumption is valid when the signal is a Gaussian distributed random process.) Provided these assumptions are correct, there is a p*100% probability that the confidence interval

```
[Pxx–Pxxc(:,1) Pxx+Pxxc(:,2)]
```

covers the true PSD. If the sections overlap, the confidence interval is not reliable and the functions display a warning message.

Transfer Function Estimate

One application of Welch's method is nonparametric system identification. Assume that H is a linear, time invariant system, and $x(n)$ and $y(n)$ are the input to and output of H, respectively. Then the PSD of $x(n)$ is related to the CSD of $x(n)$ and $y(n)$ by

$$P_{xy}(\omega) = H(\omega)P_{xx}(\omega)$$

An estimate of the transfer function between $x(n)$ and $y(n)$ is

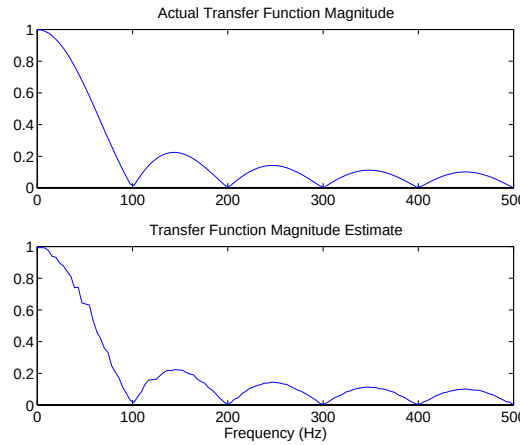
$$\hat{H}(\omega) = \frac{\hat{P}_{xy}(\omega)}{\hat{P}_{xx}(\omega)}$$

Meta Platforms, Inc. - Exhibit 1009
Page 126 of 720

This method estimates both magnitude and phase information. The `tfe` function uses Welch's method to compute the CSD and PSD and then forms their quotient for the transfer function estimate. Use `tfe` the same way that you use the `csd` function.

Filter the signal `xn` with an FIR filter, then plot the actual magnitude response and the estimated response.

```
h = ones(1,10)/10; % moving average filter
yn = filter(h,1,xn);
[HEST,f] = tfe(xn,yn,256,Fs,256,128,'none');
H = freqz(h,1,f,Fs);
subplot(2,1,1); plot(f,abs(H));
title('Actual Transfer Function Magnitude');
subplot(2,1,2); plot(f,abs(HEST));
title('Transfer Function Magnitude Estimate');
xlabel('Frequency (Hz)');
```



Coherence Function

The magnitude-squared coherence between two signals $x(n)$ and $y(n)$ is

$$C_{xy}(\omega) = \frac{|P_{xy}(\omega)|^2}{P_{xx}(\omega)P_{yy}(\omega)}$$

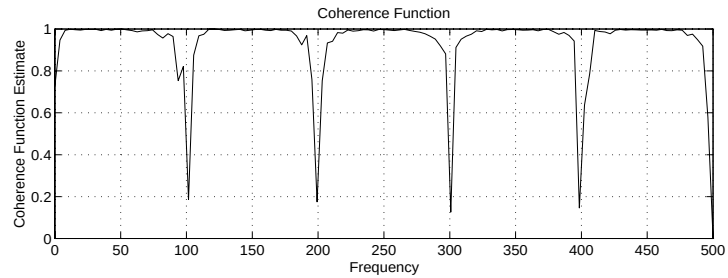
Meta Platforms, Inc. - Exhibit 1009
Page 127 of 720

This quotient is a real number between 0 and 1 that measures the correlation between $x(n)$ and $y(n)$ at the frequency ω.

The `cohere` function takes sequences `x` and `y`, computes their PSDs and CSD, and returns the quotient of the magnitude squared of the CSD and the product of the PSDs. Its options and operation are similar to the `csd` and `tfe` functions.

The coherence function of `xn` and the filter output `yn` versus frequency is

```
cohere(xn,yn,256,Fs,256,128,'none')
```



If the input sequence length, window length, and overlap are such that `cohere` operates on only a single record, the function returns all ones.

Multitaper Method

The *multitaper method* (MTM) uses orthogonal windows (or "tapers") to obtain approximately independent estimates of the power spectrum and then combines them to yield an estimate. This estimate exhibits more degrees of freedom and allows for easier quantification of the bias and variance trade-offs, compared to conventional periodogram methods. Many conventional spectral estimates use a single taper (or "window"), with some irretrievable loss of information at the beginning and the end of the series. In the multitaper method, additional tapers are used to recover some of the lost information.

This brief discussion of the multitaper method provides an intuitive look at the algorithm to assist in determining when to use it. For a more detailed and thorough explanation, see Percival and Walden [3].

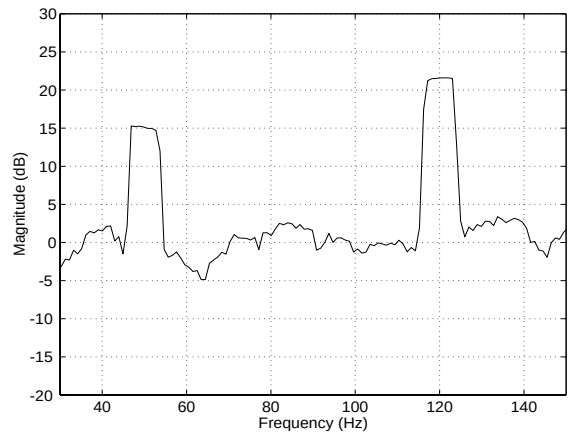
The simple parameter for the multitaper method is the time-bandwidth product, NW. This parameter is a "resolution" parameter directly related to the number of tapers used to compute the spectrum. There are always $2*NW-1$ tapers used to form the estimate. This means that, as NW increases, there are

Meta Platforms, Inc. - Exhibit 1009
Page 128 of 720

more estimates of the power spectrum, and the variance of the estimate decreases. However, the bandwidth of each taper is also proportional to *NW*, so as *NW* increases, each estimate exhibits more spectral leakage (i.e., wider peaks) and the overall spectral estimate is more biased. For each data set, there is usually a value for *NW* that allows an optimal trade-off between bias and variance.

Using `pmtm` on the data from the previous section, `xn`, yields

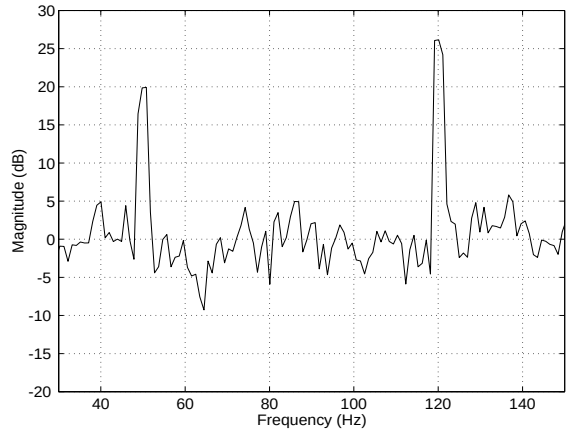
```
Fs = 1000;
t = 0:1/Fs:1;
randn('seed',0)
xn = sin(2*pi*50*t) + 2*sin(2*pi*120*t) + randn(size(t));

[P,f] = pmtm(xn,4,1024,Fs);
plot(f,10*log10(P))           % plot in decibels
axis([30 150 –20 30])
```



Meta Platforms, Inc. - Exhibit 1009
Page 129 of 720

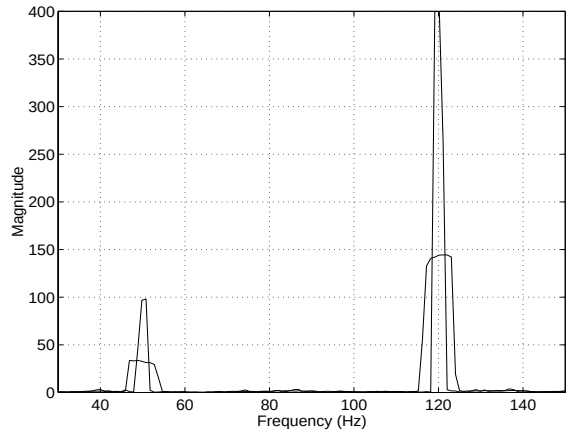
By lowering the time-bandwidth product, the peaks become narrower.

```
[P1,f] = pmtm(xn,3/2,1024,Fs);
plot(f,10*log10(P1))            % plot in decibels
axis([30 150 –20 30])
```



Note that the area under the peaks remains about the same, as can be seen when both are plotted together on a linear scale.

```
plot(f,[P P1])
axis([30 150 0 400])
```



Meta Platforms, Inc. - Exhibit 1009
Page 130 of 720

This conservation of total power is verifiable numerically.

```
sum(P)

ans =
    1.8447e+03

sum(P1)

ans =
    1.8699e+03
```

Note that total power is only approximately conserved in this case. This is because the adaptive weighting procedure that is used to minimize leakage does not strictly conserve total power.

This method is more expensive computationally than Welch's method, because of the cost of computing the discrete prolate spheroidal sequences (DPSSs, also known as Slepian sequences). For long data series (10,000 points or more), it is useful to compute the DPSSs once and save them in a MAT-file. The M-files `dpsssave`, `dpssload`, `dpssdir`, and `dpssclear` are provided, to keep a database of saved DPSSs in the MAT-file `dpss.mat`.

Yule-Walker AR Method

The *Yule-Walker AR method* is an autoregressive technique for spectral density estimation (see Marple [2], Chapter 7, and Proakis[4], Section 12.3.2). This method solves for the AR model parameters by the autocorrelation method.

The Yule-Walker AR estimate is obtained by solution of the normal equations.

$$\begin{bmatrix} r(1) & r(2)^* & \cdots & r(n)^* \\ r(2) & r(1) & \cdots & r(n-1)^* \\ \vdots & \ddots & \ddots & \vdots \\ r(n) & \cdots & r(2) & r(1) \end{bmatrix} \begin{bmatrix} a(2) \\ a(3) \\ \vdots \\ a(n+1) \end{bmatrix} = \begin{bmatrix} -r(2) \\ -r(3) \\ \vdots \\ -r(n+1) \end{bmatrix}$$

Here, $a = [1\ a(2) \ldots a(n+1)]$ is a vector of autoregressive coefficients, the elements of vector $r = [r(1)\ r(2) \ldots r(n+1)]$ are correlations, and the left-hand side autocorrelation matrix is Hermitian Toeplitz and positive definite.

Meta Platforms, Inc. - Exhibit 1009
Page 131 of 720

The spectral density estimate is

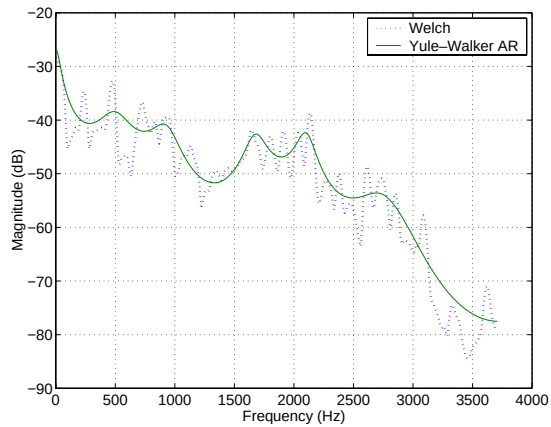
$$P_{YuleAR}(f) = \frac{1}{|a^H \mathbf{e}(f)|^2}$$

where $\mathbf{e}(f)$ is a complex sinusoid.

The toolbox function `pyulear` implements the Yule-Walker AR method.

For example, compare the spectrum of a speech signal using Welch's method and the Yule-Walker AR method:

```
load mtlb
[P1,f] = pwelch(mtlb,1024,Fs,256);
[P2,f] = pyulear(mtlb,14,1024,Fs);  % 14th order model
plot(f,10*log10(P1),':',f,10*log10(P2)); grid
ylabel('Magnitude (dB)'); xlabel('Frequency (Hz)');
legend('Welch','Yule–Walker AR')
```



The solid Yule-Walker AR spectrum is smoother than the periodogram because of the simple underlying all-pole model.

Burg Method

The Burg method for AR spectral estimation is based on minimizing the forward and backward prediction errors while satisfying the Levinson-Durbin recursion (see Marple[2], Chapter 7, and Proakis[4], Section 12.3.3). In contrast to other AR estimation methods, the Burg method avoids calculating

Meta Platforms, Inc. - Exhibit 1009
Page 132 of 720

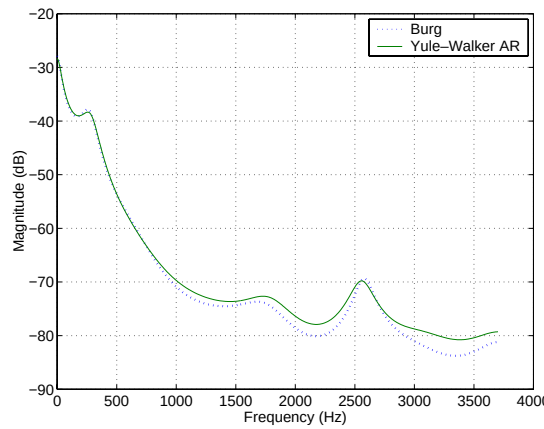
the autocorrelation function, and instead estimates the reflection coefficients directly.

The primary advantages of the Burg method are resolving closely spaced sinusoids in signals with low noise levels, and estimating short data records, in which case the AR power spectral density estimates are very close to the true values. In addition, the Burg method ensures a stable AR model and is computationally efficient.

The accuracy of the Burg method is lower for high-order models, long data records, and high signal-to-noise ratios (which can cause *line splitting*, or the generation of extraneous peaks in the spectrum estimate). The spectral density estimate computed by the Burg method is also susceptible to frequency shifts (relative to the true frequency) resulting from the initial phase of noisy sinusoidal signals. This effect is magnified when analyzing short data sequences.

The toolbox function `pburg` implements the Burg method. Compare the spectrum of the speech signal generated by both the Burg method and the Yule-Walker AR method. They are very similar for large signal lengths.

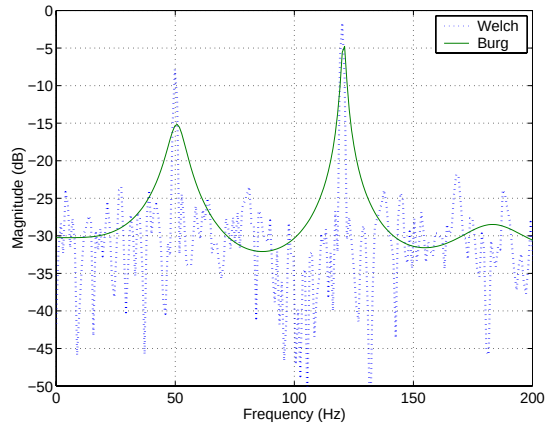
```
load mtlb
[P1,f] = pburg(mtlb(1:512),14,1024,Fs);    % 14th order model
[P2,f] = pyulear(mtlb(1:512),14,1024,Fs);  % 14th order model
plot(f,10*log10(P1),':',f,10*log10(P2)); grid
ylabel('Magnitude (dB)'); xlabel('Frequency (Hz)');
legend('Burg','Yule–Walker AR')
```



Meta Platforms, Inc. - Exhibit 1009
Page 133 of 720

Compare the spectrum of a noisy signal computed using the Burg method and the Welch method.

```
Fs = 1000;
t = 0:1/Fs:1;
xn = sin(2*pi*50*t) + 2*sin(2*pi*120*t) + randn(size(t));
[P1,f] = pwelch(xn,1024 ,Fs);
[P2,f] = pburg(xn,17,1024,Fs);   % 17th order model
plot(f,10*log10(P1),':',f,10*log10(P2)), grid
ylabel('Magnitude (dB)'); xlabel('Frequency (Hz)');
axis([0 200 –50 0])
legend('Welch','Burg')
```



Note that, as the model order for the Burg method is reduced, a frequency shift due to the initial phase of the sinusoids will become apparent.

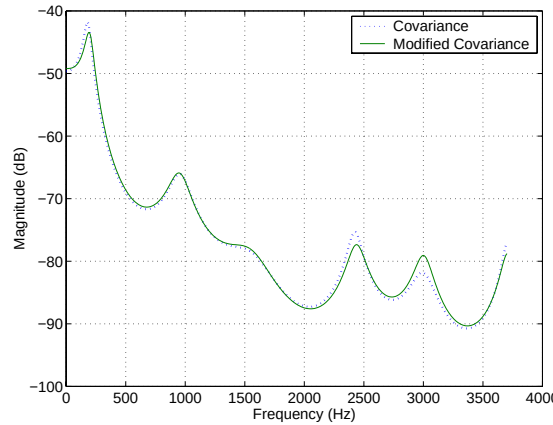
Covariance and Modified Covariance Methods

The covariance method for AR spectral estimation is based on minimizing the forward prediction error. The modified covariance method is based on minimizing the forward and backward prediction errors. The toolbox functions `pcov` and `pmcov` implement the respective methods.

Meta Platforms, Inc. - Exhibit 1009
Page 134 of 720

Compare the spectrum of the speech signal generated by both the covariance method and the modified covariance method. They are nearly identical, even for a short signal length.

```
load mtlb
[P1,f] = pcov(mtlb(1:64),14,1024,Fs);% 14th order model
[P2,f] = pmcov(mtlb(1:64),14,1024,Fs);% 14th order model
plot(f,10*log10(P1),':',f,10*log10(P2)); grid
ylabel('Magnitude (dB)'); xlabel('Frequency (Hz)');
legend('Covariance','Modified Covariance')
```



MUSIC and Eigenvector Analysis Methods

The `pmusic` function provides two related spectral analysis methods:

- The multiple signal classification method (MUSIC) developed by Schmidt
- The eigenvector (EV) method developed by Johnson

See Marple [2] (pgs. 373-378) for a summary of these methods.

Both of these methods are frequency estimator techniques based on eigenanalysis of the autocorrelation matrix. This type of spectral analysis categorizes the information in a correlation or data matrix, assigning information to either a signal subspace or a noise subspace.

Meta Platforms, Inc. - Exhibit 1009
Page 135 of 720

Eigenanalysis Overview

Consider a number of complex sinusoids embedded in white noise. You can write the autocorrelation matrix R for this system as the sum of the signal autocorrelation matrix (S) and the noise autocorrelation matrix (W).

$$R = S + W$$

There is a close relationship between the eigenvectors of the signal autocorrelation matrix and the signal and noise subspaces. The eigenvectors v of S span the same signal subspace as the signal vectors. If the system contains M complex sinusoids and the order of the autocorrelation matrix is p, eigenvectors v_{M+1} through v_{p+1} span the noise subspace of the autocorrelation matrix.

Frequency Estimator Functions. To generate their frequency estimates, eigenanalysis methods calculate functions of the vectors in the signal and noise subspaces. Both the MUSIC and EV techniques choose a function that theoretically goes to infinity at one of the sinusoidal frequencies in the input signal. Using digital technology, the resulting estimate has sharp peaks at the frequencies of interest; this means that there won't be infinity values in the vectors.

The MUSIC estimate is given by the formula

$$P_{music}(f) = \frac{1}{\mathbf{e}^H(f)\left(\sum_{k=p+1}^{N} \mathbf{v}_k \mathbf{v}_k^H\right)\mathbf{e}(f)} = \frac{1}{\sum_{k=p+1}^{N} \left|\mathbf{v}_k^H \mathbf{e}(f)\right|^2}$$

where N is the size of the eigenvectors and $\mathbf{e}(f)$ is a complex sinusoid vector:

$$\mathbf{e}(f) = [1 \;\; \exp(j2\pi f) \; \exp(j2\pi f \cdot 2) \; \exp(j2\pi f \cdot 4) \; \ldots \; \exp(j2\pi f \cdot (n-1))]^H$$

$\mathbf{v}$ represents the eigenvectors of the input signal's correlation matrix; $\mathbf{v}_k$ is the kth eigenvector. H is the conjugate transpose operator. The eigenvectors used in the sum correspond to the smallest eigenvalues and span the noise subspace (p is the size of the signal subspace).

The expression $\mathbf{v}_k^H \mathbf{e}(f)$ is equivalent to a Fourier transform (the vector $\mathbf{e}(f)$ consists of complex exponentials). This form is useful for numeric computation because the FFT can be computed for each $\mathbf{v}_k$ and then the squared magnitudes can be summed.

Meta Platforms, Inc. - Exhibit 1009
Page 136 of 720

The EV method weights the summation by the eigenvalues of the correlation matrix:

$$P_{ev}(f) = \frac{1}{\left(\sum_{k=p+1}^{N} \left| \mathbf{v}_k^H \mathbf{e}(f) \right|^2 \right) / \lambda_k}$$

The `pmusic` function in this toolbox uses the `svd` (singular value decomposition) function in the signal case and the `eig` function for analyzing the correlation matrix and assigning eigenvectors to the signal or noise subspaces. When `svd` is used, `pmusic` never computes the correlation matrix explicitly, but the singular values are the eigenvalues.

Controlling Subspace Thresholds

To provide user control over the assignments of eigenvectors to the signal and noise subspaces, the `pmusic` function accepts a threshold argument `thresh`. `thresh` is a two-element vector where the first element is the number of eigenvectors spanning the signal subspace and the second element is a threshold test:

- If `thresh(2)` ≤ 1, then `thresh(1)` specifies the number of eigenvectors spanning the signal subspace. In this case the values of `thresh(1)` must be in the range [0, N), where N is:
  - The column length of `xR` if `xR` is a data matrix
  - The matrix size if `xR` is a correlation matrix
  - The window length if `xR` is a signal vector
- If `thresh(1)`≥ N, then `thresh(2)` is a value greater than or equal to 1 that specifies the absolute threshold for splitting the eigenvalues between the signal and noise subspaces. That is, if a given eigenvalue is less than or equal to the product `thresh(2)`min{λ_k}, then the given eigenvector is assigned to the noise subspace.
- If `thresh(1)` < N and `thresh(2)` ≥ 1, `thresh(1)` still specifies the maximum number of eigenvectors in the signal subspace. However, the threshold test specified by `thresh(2)` can also assign eigenvectors to the noise subspace.
- If `thresh(1)` ≥ N and `thresh(2)` < 1, there are no noise eigenvectors. This is an invalid case and `pmusic` generates an error.

Meta Platforms, Inc. - Exhibit 1009
Page 137 of 720

For complete details on using the `thresh` parameter, see the reference description of `pmusic` in Chapter 6.

Meta Platforms, Inc. - Exhibit 1009
Page 138 of 720

References

1. Kay, S.M. *Modern Spectral Estimation*. Englewood Cliffs, NJ: Prentice Hall, 1988.

2. Marple, S.L. *Digital Spectral Analysis*. Englewood Cliffs, NJ: Prentice Hall, 1987.

3. Percival, D.B., and A.T. Walden. *Spectral Analysis for Physical Applications: Multitaper and Conventional Univariate Techniques*. Cambridge: Cambridge University Press, 1993.

4. Proakis, J.G., and D.G. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications*. Englewood Cliffs, NJ: Prentice Hall, 1996.

5. Welch, P.D. "The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified Periodograms." *IEEE Trans. Audio Electroacoust*. Vol. AU-15 (June 1967). Pgs. 70-73.

Meta Platforms, Inc. - Exhibit 1009
Page 139 of 720

Meta Platforms, Inc. - Exhibit 1009
Page 140 of 720

4

Special Topics

Windows 4-2
Basic Shapes 4-2
Generalized Cosine Windows 4-4
Kaiser Window 4-4
Chebyshev Window 4-9

Parametric Modeling 4-10
Time-Domain Based Modeling 4-11
Frequency-Domain Based Modeling 4-16

Resampling 4-20

Cepstrum Analysis 4-23
Inverse Complex Cepstrum 4-25

FFT-Based Time-Frequency Analysis 4-27

Median Filtering 4-28

Communications Applications 4-29

Deconvolution 4-33

Specialized Transforms 4-34
Chirp z-Transform 4-34
Discrete Cosine Transform 4-36
Hilbert Transform 4-38

References 4-40

Windows

In both digital filter design and power spectrum estimation, the choice of a windowing function can play an important role in determining the quality of overall results. The main role of the window is to damp out the effects of the Gibbs phenomenon that results from truncation of an infinite series.

The toolbox window functions are shown in the table below.

Window	Function
Bartlett window	`bartlett`
Blackman window	`blackman`
Rectangular window	`boxcar`
Chebyshev window	`chebwin`
Hamming window	`hamming`
Hanning window	`hanning`
Kaiser window	`kaiser`
Triangular window	`triang`

Basic Shapes

The basic window is the *rectangular window*, a vector of ones of the appropriate length. A rectangular window of length 50 is

```
n = 50;
w = boxcar(n);
```

This toolbox stores windows in column vectors by convention, so an equivalent expression is

```
w = ones(50,1);
```

The *Bartlett* (or triangular) *window* is the convolution of two rectangular windows. The functions `bartlett` and `triang` compute similar triangular windows, with three important differences. The `bartlett` function always

Meta Platforms, Inc. - Exhibit 1009
Page 142 of 720

returns a window with two zeros on the ends of the sequence, so that for n odd, the center section of bartlett(n+2) is equivalent to triang(n):

```
bartlett(7)

ans =

         0
    0.3333
    0.6667
    1.0000
    0.6667
    0.3333
         0

triang(5)

ans =

    0.3333
    0.6667
    1.0000
    0.6667
    0.3333
```

For n even, bartlett is still the convolution of two rectangular sequences. There is no standard definition for the triangular window for n even; the slopes of the line segments of triang's result are slightly steeper than those of bartlett's in this case:

```
w = bartlett(8);
[w(2:7)  triang(6)]

ans =

    0.2857    0.1667
    0.5714    0.5000
    0.8571    0.8333
    0.8571    0.8333
    0.5714    0.5000
    0.2857    0.1667
```

The final difference between the Bartlett and triangular windows is evident in the Fourier transforms of these functions. The Fourier transform of a Bartlett

Meta Platforms, Inc. - Exhibit 1009
Page 143 of 720

window is negative for n even. The Fourier transform of a triangular window, however, is always nonnegative.

This difference can be important when choosing a window for some spectral estimation techniques, such as the Blackman-Tukey method. Blackman-Tukey forms the spectral estimate by calculating the Fourier transform of the autocorrelation sequence. The resulting estimate might be negative at some frequencies if the window's Fourier transform is negative (see Kay [1], pg. 80).

Generalized Cosine Windows

Blackman, Hamming, Hanning, and rectangular windows are all special cases of the *generalized cosine window*. These windows are combinations of sinusoidal sequences with frequencies 0, $2\pi/(N-1)$, and $4\pi/(N-1)$, where N is the window length. One way to generate them is

```
ind = (0:n-1)'*2*pi/(n-1);
w = A - B*cos(ind) + C*cos(2*ind);
```

where A, B, and C are constants you define. The concept behind these windows is that by summing the individual terms to form the window, the low frequency peaks in the frequency domain combine in such a way as to decrease sidelobe height. This has the side effect of increasing the mainlobe width.

The Hamming and Hanning windows are two-term generalized cosine windows, given by A = 0.54, B = 0.46 for Hamming and A = 0.5, B = 0.5 for Hanning (C = 0 in both cases). The hamming and hanning functions, respectively, compute these windows.

Note that the definition of the generalized cosine window shown in the earlier MATLAB code yields zeros at samples 1 and n for A = 0.5 and B = 0.5. To eliminate these zeros on the edges of the window, hanning uses a cosine of frequency $2\pi/(N+1)$ instead of $2\pi/(N-1)$.

The Blackman window is a popular three-term window, given by A = 0.42, B = 0.5, C = 0.08. The blackman function computes this window.

Kaiser Window

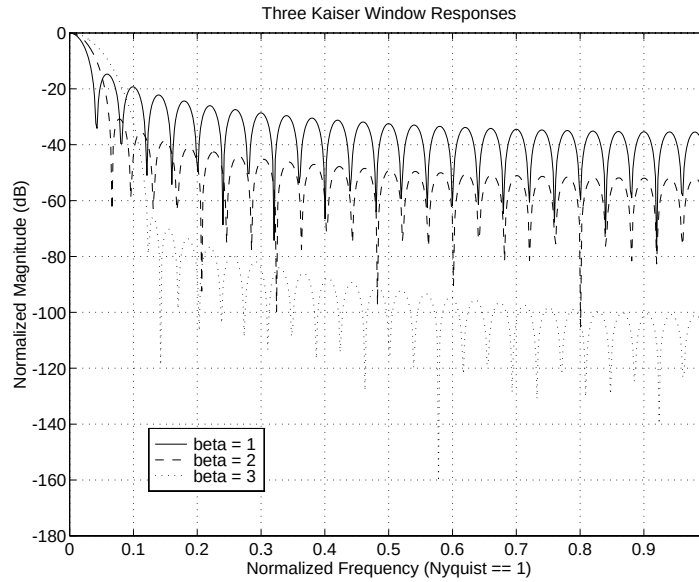
The *Kaiser window* is an approximation to the prolate-spheroidal window, for which the ratio of the mainlobe energy to the sidelobe energy is maximized. For a Kaiser window of a particular length, the parameter β controls the sidelobe height. For a given β, the sidelobe height is fixed with respect to window length.

Meta Platforms, Inc. - Exhibit 1009
Page 144 of 720

The statement `kaiser(n,beta)` computes a length `n` Kaiser window with parameter `beta`.

Examples of Kaiser windows with length 50 and various values for the `beta` parameter are

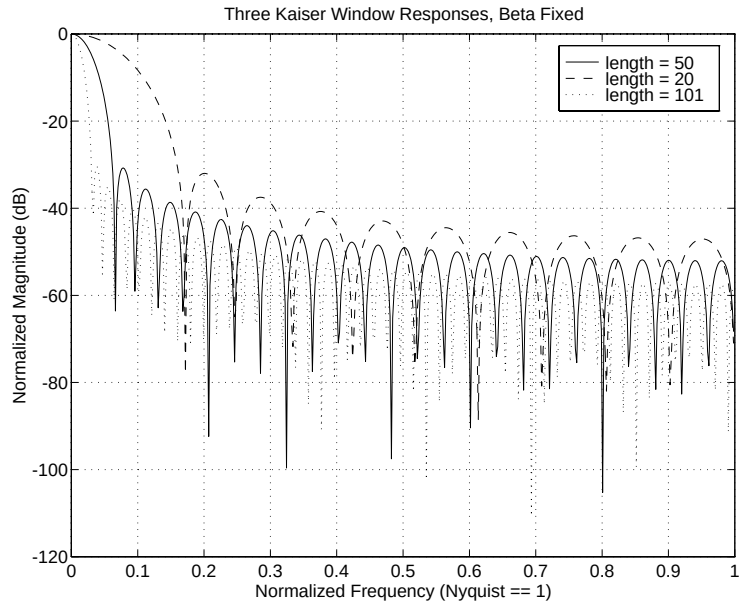
```
n = 50;
w1 = kaiser(n,1);
w2 = kaiser(n,4);
w3 = kaiser(n,9);
[W1,f] = freqz(w1/sum(w1),1,512,2);
[W2,f] = freqz(w2/sum(w2),1,512,2);
[W3,f] = freqz(w3/sum(w3),1,512,2);
plot(f,20*log10(abs([W1 W2 W3])))
```



Meta Platforms, Inc. - Exhibit 1009
Page 145 of 720

As β increases, the sidelobe height decreases and the mainlobe width increases. To see how the sidelobe height stays the same for a fixed β parameter as the length is varied, try

```
w1 = kaiser(50,4);
w2 = kaiser(20,4);
w3 = kaiser(101,4);
[W1,f] = freqz(w1/sum(w1),1,512,2);
[W2,f] = freqz(w2/sum(w2),1,512,2);
[W3,f] = freqz(w3/sum(w3),1,512,2);
plot(f,20*log10(abs([W1 W2 W3])))
```



Meta Platforms, Inc. - Exhibit 1009
Page 146 of 720

Kaiser Windows in FIR Design

There are two design formulas that can help you design FIR filters to meet a set of filter specifications using a Kaiser window. To achieve a sidelobe height of $-\alpha$ dB, the `beta` parameter is

$$\beta = \begin{cases} 0.1102(\alpha - 8.7), & \alpha > 50 \\ 0.5842(\alpha - 21)^{0.4} + 0.07886(\alpha - 21), & 50 \geq \alpha \geq 21 \\ 0, & \alpha < 21 \end{cases}$$

For a transition width of $\Delta\omega$ rad/sec, use the length

$$n = \frac{\alpha - 8}{2.285\Delta\omega} + 1$$

Filters designed using these heuristics will meet the specifications approximately, but you should verify this. To design a lowpass filter with cutoff frequency 0.5π rad/sec, transition width 0.2π rad/sec, and 40 dB of attenuation in the stopband, try

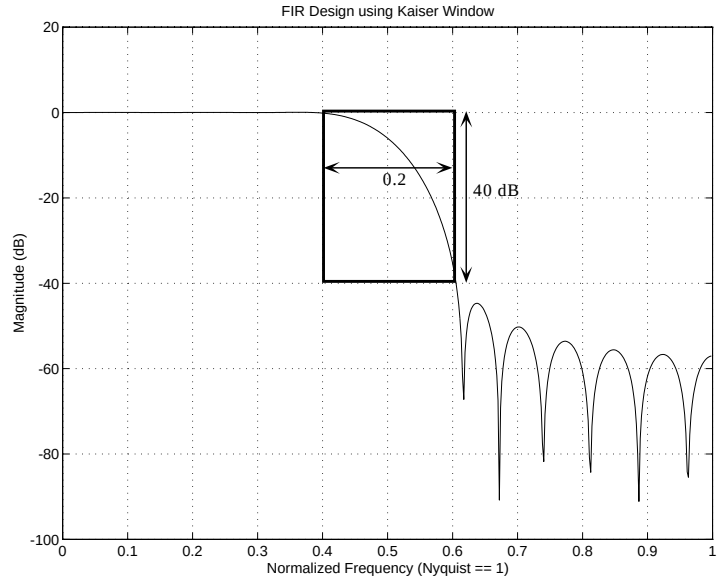
```
[n,wn,beta] = kaiserord([0.4 0.6]*pi,[1 0],[0.01 0.01],2*pi);
h = fir1(n,wn,kaiser(n+1,beta),'noscale');
```

The `kaiserord` function estimates the filter order, cutoff frequency, and Kaiser window beta parameter needed to meet a given set of frequency domain specifications.

Meta Platforms, Inc. - Exhibit 1009
Page 147 of 720

The ripple in the passband is roughly the same as the ripple in the stopband. As you can see from the frequency response, this filter nearly meets the specifications.

```
[H,f] = freqz(h,1,512,2);
plot(f,20*log10(abs(H))), grid
```



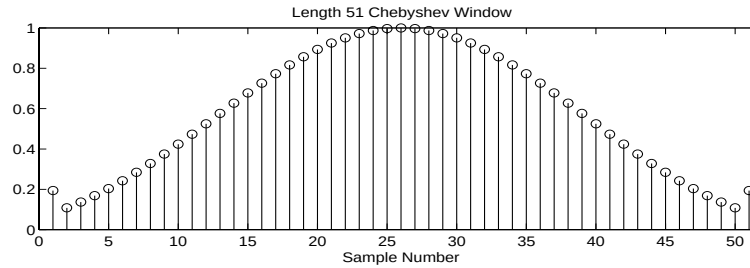
For details on `kaiserord`, see the reference description in Chapter 6.

Meta Platforms, Inc. - Exhibit 1009
Page 148 of 720

Chebyshev Window

The Chebyshev window minimizes the mainlobe width, given a particular sidelobe height. It is characterized by an equiripple behavior, that is, its sidelobes all have the same height. The `chebwin` function, with length and sidelobe height parameters, computes a Chebyshev window.

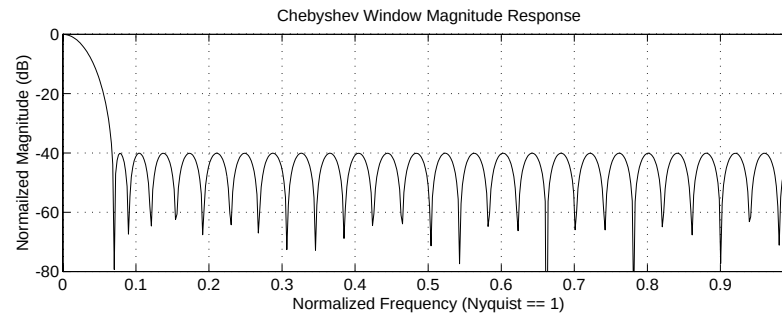
```
n = 51;
Rs = 40;  % sidelobe height in decibels
w = chebwin(n,Rs);
stem(w)
```



As shown in the plot, the Chebyshev window has large spikes at its outer samples.

Plot the frequency response to see the equiripples at -40 dB.

```
[W,f] = freqz(w,1,512,2);
plot(f,20*log10(abs(W)/sum(w))), grid
```



For a detailed discussion of the characteristics and applications of the various window types, see [2] Oppenheim and Schafer, pgs. 444-462, and [3] Parks and Burrus, pgs. 71-73.

Meta Platforms, Inc. - Exhibit 1009
Page 149 of 720

Parametric Modeling

Parametric modeling techniques find the parameters for a mathematical model describing a signal, system, or process. These techniques use known information about the system to determine the model. Applications for parametric modeling include speech and music synthesis, data compression, high-resolution spectral estimation, communications, manufacturing, and simulation.

The toolbox parametric modeling functions operate with the rational transfer function model. Given appropriate information about an unknown system (impulse or frequency response data, or input and output sequences), these functions find the coefficients of a linear system that models the system.

One important application of the parametric modeling functions is in the design of filters that have a prescribed time or frequency response. These functions provide a data-oriented alternative to the IIR and FIR filter design functions discussed in Chapter 2.

Here is a summary of the parametric modeling functions in this toolbox. Note that the System Identification Toolbox provides a more extensive collection of parametric modeling functions.

Meta Platforms, Inc. - Exhibit 1009
Page 150 of 720

Domain	Functions	Description
Time	`arburg`	Generate all-pole filter coefficients that model an input data sequence using the Levinson-Durbin algorithm.
	`arcov`	Generate all-pole filter coefficients that model an input data sequence by minimizing the forward prediction error.
	`armcov`	Generate all-pole filter coefficients that model an input data sequence by minimizing the forward and backward prediction errors.
	`aryule`	Generate all-pole filter coefficients that model an input data sequence using an estimate of the autocorrelation function.
	`lpc`, `levinson`	Linear Predictive Coding. Generate all-pole recursive filter whose impulse response matches a given sequence.
	`prony`	Generate IIR filter whose impulse response matches a given sequence.
	`stmcb`	Find IIR filter whose output, given a specified input sequence, matches a given output sequence.
Frequency	`invfreqz`, `invfreqs`	Generate digital or analog filter coefficients given complex frequency response data.

Because `yulewalk` is geared explicitly toward ARMA filter design, it is discussed in Chapter 2. `pburg` and `pyulear` are discussed in Chapter 3 along with the other (nonparametric) spectral estimation methods.

Time-Domain Based Modeling

The `lpc`, `prony`, and `stmcb` functions find the coefficients of a digital rational transfer function that approximates a given time-domain impulse response. The algorithms differ in complexity and accuracy of the resulting model.

Meta Platforms, Inc. - Exhibit 1009
Page 151 of 720

Linear Prediction

Linear prediction modeling assumes that each output sample of a signal, `x(k)`, is a linear combination of the past `n` outputs (that is, it can be "linearly predicted" from these outputs), and that the coefficients are constant from sample to sample.

$$x(k) = -a(2)x(k-1) - a(3)x(k-2) - \cdots - a(n+1)x(k-n)$$

An `n`th-order all-pole model of a signal `x` is

```
a = lpc(x,n)
```

To illustrate `lpc`, create a sample signal that is the impulse response of an all-pole filter with additive white noise.

```
randn('seed',0)
x = impz(1,[1 0.1 0.1 0.1 0.1],10) + randn(10,1)/10;
```

The coefficients for a fourth-order all-pole filter that models the system are

```
a = lpc(x,4)

a =
     1.0000     0.0395     0.0338     0.0668     0.1264
```

`lpc` first calls `xcorr` to find a biased estimate of the correlation function of `x`, and then uses the Levinson-Durbin recursion, implemented in the `levinson` function, to find the model coefficients `a`. The Levinson-Durbin recursion is a fast algorithm for solving a system of symmetric Toeplitz linear equations. `lpc`'s entire algorithm for `n = 4` is

```
r = xcorr(x);
r(1:length(x)-1) = []; % remove corr. at negative lags
a = levinson(r,4)

a =
     1.0000     0.0395     0.0338     0.0668     0.1264
```


Meta Platforms, Inc. - Exhibit 1009
Page 152 of 720

You could form the linear prediction coefficients with other assumptions by passing a different correlation estimate to `levinson`, such as the unbiased correlation estimate.

```
r = xcorr(x,'unbiased');
r(1:length(x)-1) = []; % remove corr. at negative lags
a = levinson(r,4)

a =
     1.0000     0.0554     0.0462     0.0974     0.2115
```

Prony's Method (ARMA Modeling)

The `prony` function models a signal using a specified number of poles and zeros. Given a sequence `x` and numerator and denominator orders `nb` and `na`, respectively, the statement

```
[b,a] = prony(x,nb,na)
```

finds the numerator and denominator coefficients of an IIR filter whose impulse response approximates the sequence `x`.

`prony` implements the method described in [3] Parks and Burrus (pgs. 226-228). This method uses a variation of the covariance method of AR modeling to find the denominator coefficients `a`, and then finds the numerator coefficients `b` for which the resulting filter's impulse response matches exactly the first `nb + 1` samples of `x`. The filter is not necessarily stable, but it can potentially recover the coefficients exactly if the data sequence is truly an autoregressive moving average (ARMA) process of the correct order.

NOTE The functions `prony` and `stmcb` (described next) are more accurately described as ARX models in system identification terminology. ARMA modeling assumes noise only at the inputs, while ARX assumes an external input. `prony` and `stmcb` know the input signal: it is an impulse for `prony` and is arbitrary for `stmcb`.

Meta Platforms, Inc. - Exhibit 1009
Page 153 of 720

A model for the test sequence x (from the earlier lpc example) using a third-order IIR filter is

```
[b,a] = prony(x,3,3)

b =
     1.1165    -0.2181    -0.6084     0.5369

a =
     1.0000    -0.1619    -0.4765     0.4940
```

The impz command shows how well this filter's impulse response matches the original sequence.

```
format long
[x impz(b,a,10)]

ans =

   1.11649535105007   1.11649535105007
  -0.03731609173676  -0.03731609173676
  -0.08249198453223  -0.08249198453223
  -0.04583930972315  -0.04583930972315
  -0.14255125351637  -0.02829072973977
   0.20400424807471   0.01433198229497
   0.02685697779814   0.01148698991026
   0.18956307836948   0.02266475846451
   0.02717716288172   0.00206242734272
   0.08057060786906   0.00545783754743
```

Notice that the first four samples match exactly. For an example of exact recovery, recover the coefficients of a Butterworth filter from its impulse response.

```
[b,a] = butter(4,.2);
h = impz(b,a,26);
[bb,aa] = prony(h,4,4);
```

Try this example; you'll see that bb and aa match the original filter coefficients to within a tolerance of 10^{-13}.

Meta Platforms, Inc. - Exhibit 1009
Page 154 of 720

Steiglitz-McBride Method (ARMA Modeling)

stmcb determines the coefficients for the system $b(z)/a(z)$ given an approximate impulse response x, as well as the desired number of zeros and poles. This function identifies an unknown system based on both input and output sequences that describe the system's behavior, or just the impulse response of the system. In its default mode, stmcb works like prony

```
[b,a] = stmcb(x,3,3)

b =
    1.1165   -0.6213   -0.8365    1.3331

a =
    1.0000   -0.5401   -0.6109    1.1298
```

stmcb also finds systems that match given input and output sequences

```
y = filter(1,[1 1],x);  % Create an output signal.
[b,a] = stmcb(y,x,0,1)

b =
     1

a =
     1     1
```

In this example, stmcb correctly identifies the system used to create y from x.

The Steiglitz-McBride method is a fast iterative algorithm that solves for the numerator and denominator coefficients simultaneously in an attempt to minimize the signal error between the filter output and the given output signal. This algorithm usually converges rapidly, but might not converge if the model order is too large. As for prony, stmcb's resulting filter is not necessarily stable due to its exact modeling approach.

stmcb provides control over several important algorithmic parameters; modify these parameters if you are having trouble modeling the data. To change the number of iterations from the default of five and provide an initial estimate for the denominator coefficients

```
n = 10;            % number of iterations
a = lpc(x,3);      % initial estimates for denominator
[b,a] = stmcb(x,3,3,n,a);
```


Meta Platforms, Inc. - Exhibit 1009
Page 155 of 720

The function uses an all-pole model created with prony as an initial estimate when you do not provide one of your own.

To compare the functions lpc, prony, and stmcb, compute the signal error in each case

```
a1 = lpc(x,3);
[b2,a2] = prony(x,3,3);
[b3,a3] = stmcb(x,3,3);
[ x-impz(1,a1,10)  x-impz(b2,a2,10)  x-impz(b3,a3,10) ]

ans =

    0.1165         0         0
   -0.0058         0   -0.0190
   -0.0535    0.0000    0.0818
    0.0151   -0.0000   -0.0176
   -0.1473   -0.1143   -0.0476
    0.2005    0.1897    0.0869
    0.0233    0.0154   -0.0103
    0.1901    0.1669   -0.0093
    0.0275    0.0251    0.0294
    0.0808    0.0751    0.0022

sum(ans.^2)

ans =

    0.1226    0.0834    0.0182
```

In comparing modeling capabilities for a given order IIR model, the last result shows that for this example, stmcb performs best, followed by prony, then lpc. This relative performance is typical of the modeling functions.

Frequency-Domain Based Modeling

The invfreqs and invfreqz functions implement the inverse operations of freqs and freqz; they find an analog or digital transfer function of a specified order that matches a given complex frequency response. Though the following examples demonstrate invfreqz, the discussion also applies to invfreqs.

Meta Platforms, Inc. - Exhibit 1009
Page 156 of 720

To recover the original filter coefficients from the frequency response of a simple digital filter

```
[b,a] = butter(4,.4)    % design Butterworth lowpass

b =
    0.0466    0.1863    0.2795    0.1863    0.0466

a =
    1.0000   -0.7821    0.6800   -0.1827    0.0301

[h,w] = freqz(b,a,64);          % compute frequency resp.
[bb,aa] = invfreqz(h,w,4,4)     % model: nb = 4, na = 4

bb =

    0.0466    0.1863    0.2795    0.1863    0.0466

aa =

    1.0000   -0.7821    0.6800   -0.1827    0.0301
```

The vector of frequencies `w` has the units in rads/sample, and the frequencies need not be equally spaced. `invfreqz` finds a filter of any order to fit the frequency data; a third-order example is

```
[bb,aa] = invfreqz(h,w,3,3)   % find third-order IIR

bb =

    0.0464    0.1785    0.2446    0.1276

aa =

    1.0000   -0.9502    0.7382   -0.2006
```

Both `invfreqs` and `invfreqz` design filters with real coefficients; for a data point at positive frequency `f`, the functions fit the frequency response at both `f` and `-f`.

Meta Platforms, Inc. - Exhibit 1009
Page 157 of 720

By default `invfreqz` uses an equation error method to identify the best model from the data. This finds b and a in

$$\min_{b,a} \sum_{k=1}^{n} wt(k)\left|h(k)A(w(k)) - B(w(k))\right|^2$$

by creating a system of linear equations and solving them with MATLAB's \ operator. Here $A(w(k))$ and $B(w(k))$ are the Fourier transforms of the polynomials `a` and `b` respectively at the frequency $w(k)$, and n is the number of frequency points (the length of `h` and `w`). $wt(k)$ weights the error relative to the error at different frequencies. The syntax

```
invfreqz(h,w,nb,na,wt)
```

includes a weighting vector. In this mode, the filter resulting from `invfreqz` is not guaranteed to be stable.

`invfreqz` provides a superior ("output-error") algorithm that solves the direct problem of minimizing the weighted sum of the squared error between the actual frequency response points and the desired response

$$\min_{b,a} \sum_{k=1}^{n} wt(k)\left|h(k) - \frac{B(w(k))}{A(w(k))}\right|^2$$

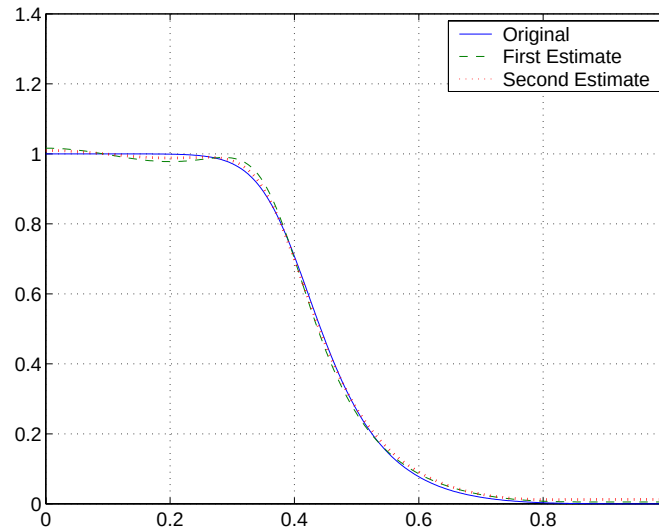
To use this algorithm, specify a parameter for the iteration count after the weight vector parameter

```
wt = ones(size(w));    % create unity weighting vector
[bbb,aaa] = invfreqz(h,w,3,3,wt,30)  % 30 iterations

bbb =

    0.0464    0.1829    0.2572    0.1549

aaa =

    1.0000   -0.8664    0.6630   -0.1614
```


Meta Platforms, Inc. - Exhibit 1009
Page 158 of 720

The resulting filter is always stable. Graphically compare the results of the first and second algorithms to the original Butterworth filter.

```
[H1,w1] = freqz(b,a,'magnitude','linear','phase','no');
[H2,w2] = freqz(bb,aa,'magnitude','linear','phase','no');
[H3,w3] = freqz(bbb,aaa,'magnitude','linear','phase','no');
plot(w1/pi,abs(H1),w2/pi,abs(H2),'--',w3/pi,abs(H3),':')
legend('Original','First Estimate','Second Estimate');
grid on
```



To verify the superiority of the fit numerically

```
sum(abs(h-freqz(bb,aa,w)).^2)   % total error, algorithm 1

ans =

    0.0200

sum(abs(h-freqz(bbb,aaa,w)).^2) % total error, algorithm 2

ans =

    0.0096
```


Meta Platforms, Inc. - Exhibit 1009
Page 159 of 720

Resampling

The toolbox provides a number of functions that resample a signal at a higher or lower rate.

Operation	Function
Resample at new rate	`resample`
Decimation	`decimate`
Interpolation	`interp`
Apply FIR filter with resampling	`upfirdn`
Cubic spline interpolation	`spline`
Other 1-D interpolation	`interp1`

The `resample` function changes the sampling rate for a sequence to any rate that is a ratio of two integers. The basic syntax for `resample` is

```
y = resample(x,p,q)
```

where the function resamples the sequence `x` at `p/q` times the original sampling rate. The length of the result `y` is `p/q` times the length of `x`.

One resampling application is the conversion of digitized audio signals from one sampling rate to another, such as from 48 kHz (the Digital Audio Tape standard) to 44.1 kHz (the Compact Disc standard). In the next example, the sampling rates are different but the idea is the same.

Meta Platforms, Inc. - Exhibit 1009
Page 160 of 720

The example file contains a length 4001 vector of speech sampled at 7418 Hz.

```
clear
load mtlb
whos

Name        Size        Bytes        Class

  Fs           1x1          8        double array
  mtlb      4001x1      32008        double array

Grand total is 4002 elements using 32016 bytes

Fs

Fs =
        7418
```

To play this speech signal on a workstation that can only play sound at 8192 Hz, use the `rat` function to find integers `p` and `q` that yield the correct resampling factor.

```
[p,q] = rat(8192/Fs,.0001)

p =
   127

q =
   115
```

Since `p/q*Fs = 8192.05` Hz, the tolerance of 0.0001 is acceptable; to resample the signal at very close to 8192 Hz.

```
y = resample(mtlb,p,q);
```

`resample` applies a lowpass filter to the input sequence to prevent aliasing during resampling. It designs this filter using the `firls` function with a Kaiser window. The syntax

```
resample(x,p,q,l,beta)
```

controls the filter's length and the beta parameter of the Kaiser window. Alternatively, use the function `intfilt` to design an interpolation filter `b` and use it with

```
resample(x,p,q,b)
```


Meta Platforms, Inc. - Exhibit 1009
Page 161 of 720

The `decimate` and `interp` functions do the same thing as `resample` with `p = 1` and `q = 1`, respectively. These functions provide different anti-alias filtering options, and they incur a slight signal delay due to filtering. The `interp` function is significantly less efficient than the `resample` function with `q = 1`.

The toolbox also contains a function, `upfirdn`, that applies an FIR filter to an input sequence and outputs the filtered sequence at a different sample rate than its original rate. See "Multirate Filter Bank Implementation" on page 1-19 and the reference description of `upfirdn` in Chapter 6 for more details.

The standard MATLAB environment contains a function, `spline`, that works with irregularly spaced data. The MATLAB function `interp1` performs interpolation, or table lookup, using various methods including linear and cubic interpolation. See the online *MATLAB Function Reference* for information on `spline` and `interp1`.

Meta Platforms, Inc. - Exhibit 1009
Page 162 of 720

Cepstrum Analysis

Cepstrum analysis is a nonlinear signal processing technique with a variety of applications in areas such as speech and image processing. The Signal Processing Toolbox provides three functions for cepstrum analysis.

Operation	Function
Complex cepstrum	`cceps`
Real cepstrum	`rceps`
Inverse complex cepstrum	`icceps`

The complex cepstrum for a sequence x is calculated by finding the complex natural logarithm of the Fourier transform of x, then the inverse Fourier transform of the resulting sequence.

$$\hat{x} = \frac{1}{2\pi}\int_{-\pi}^{\pi} \log[X(e^{j\omega})]e^{j\omega n}\,d\omega$$

The toolbox function `cceps` performs this operation, estimating the complex cepstrum for an input sequence. It returns a real sequence the same size as the input sequence

```
xhat = cceps(x)
```

The complex cepstrum transformation is central to the theory and application of *homomorphic systems*, that is, systems that obey certain general rules of superposition. See [2] Oppenheim and Schafer for a discussion of the complex cepstrum and homomorphic transformations, with details on speech processing applications.

Try using `cceps` in an echo detection application. First, create a 45 Hz sine wave sampled at 100 Hz

```
t = 0:0.01:1.27;
s1 = sin(2*pi*45*t);
```

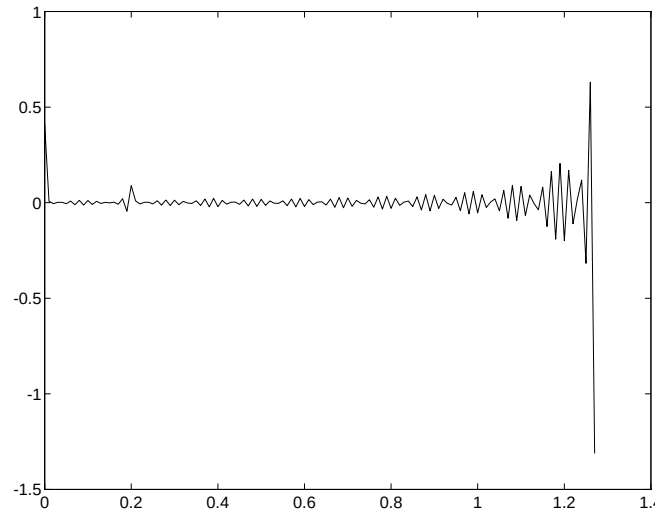

Meta Platforms, Inc. - Exhibit 1009
Page 163 of 720

Add an echo of the signal, with half the amplitude, 0.2 seconds after the beginning of the signal.

```
s2 = s1 + 0.5*[zeros(1,20) s1(1:108)];
```

The complex cepstrum of this new signal is

```
c = cceps(s2);
plot(t,c)
```



Note that the complex cepstrum shows a peak at 0.2 seconds, indicating the echo.

The *real cepstrum* of a signal *x*, sometimes called simply the cepstrum, is calculated by determining the natural logarithm of magnitude of the Fourier transform of *x*, then obtaining the inverse Fourier transform of the resulting sequence

$$c_x = \frac{1}{2\pi}\int_{-\pi}^{\pi} \log\left|X(e^{j\omega})\right| e^{j\omega n}\, d\omega$$

Meta Platforms, Inc. - Exhibit 1009
Page 164 of 720

The toolbox function `rceps` performs this operation, returning the real cepstrum for a sequence `x`. The returned sequence is a real-valued vector the same size as the input vector.

```
y = rceps(x)
```

By definition, you cannot reconstruct the original sequence from its real cepstrum transformation, as the real cepstrum is based only on the magnitude of the Fourier transform for the sequence (see [2]). The `rceps` function, however, can reconstruct a minimum-phase version of the original sequence by applying a windowing function in the cepstral domain. To obtain both the real cepstrum and the minimum phase reconstruction for a sequence, use

```
[y,ym] = rceps(x)
```

where `y` is the real cepstrum and `ym` is the minimum phase reconstruction of `x`.

Inverse Complex Cepstrum

To invert the complex cepstrum, use the `icceps` function. Inversion is complicated by the fact that the `cceps` function performs a data dependent phase modification so that the unwrapped phase of its input is continuous at zero frequency. The phase modification is equivalent to an integer delay. This delay term is returned by `cceps` if you ask for a second output. For example,

```
x = 1:10;
[xh,nd] = cceps(x)

xh =
  Columns 1 through 7
   2.2428   -0.0420   -0.0210    0.0045    0.0366    0.0788    0.1386
  Columns 8 through 10
    0.2327    0.4114    0.9249

nd =
     1
```


Meta Platforms, Inc. - Exhibit 1009
Page 165 of 720

To invert the complex cepstrum, use `icceps` with the original delay parameter.

```
icceps(xh,nd)

ans =
  Columns 1 through 7
   1.0000   2.0000   3.0000   4.0000   5.0000   6.0000   7.0000
  Columns 8 through 10
    8.0000    9.0000   10.0000
```

NOTE With any modification of the complex cepstrum, the original delay term may no longer be valid. Use the `icceps` function with care.

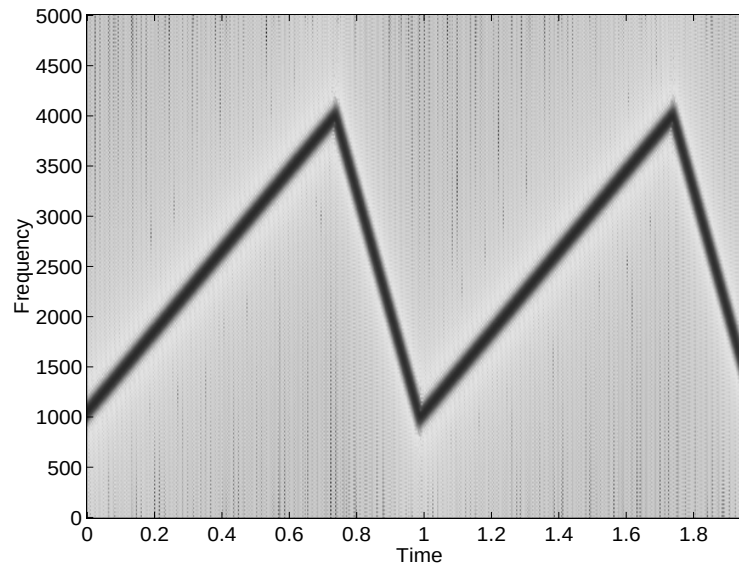
Meta Platforms, Inc. - Exhibit 1009
Page 166 of 720

FFT-Based Time-Frequency Analysis

The Signal Processing Toolbox provides a function, `specgram`, that returns the time-dependent Fourier transform for a sequence, or displays this information as a spectrogram. The *time-dependent Fourier transform* is the discrete-time Fourier transform for a sequence, computed using a sliding window. This form of the Fourier transform, also known as the short-time Fourier transform (STFT), has numerous applications in speech, sonar, and radar processing. The *spectrogram* of a sequence is the magnitude of the time-dependent Fourier transform versus time.

To display the spectrogram of a linear FM signal

```
Fs = 10000;
t = 0:1/Fs:2;
x = vco(sawtooth(2*pi*t,.75),[0.1 0.4]*Fs,Fs);
specgram(x,512,Fs,kaiser(256,5),220)
```



Note that the spectrogram display is an image, not a plot.

Meta Platforms, Inc. - Exhibit 1009
Page 167 of 720

Median Filtering

The function `medfilt1` implements one-dimensional median filtering, a nonlinear technique that applies a sliding window to a sequence. The median filter replaces the center value in the window with the median value of all the points within the window [4]. In computing this median, `medfilt1` assumes zeros beyond the input points.

When the number of elements `n` in the window is even, `medfilt1` sorts the numbers, then takes the average of the `((n–1)/2 & ((n–1)/2)+1)` elements.

Two simple examples with fourth- and third-order median filters are

```
medfilt1([4 3 5 2 8 9 1],4)

ans =

    1.500 3.500 3.500 4.000 6.500 5.000 4.500

medfilt1([4 3 5 2 8 9 1],3)

ans =

     3     4     3     5     8     8     1
```

See the *Image Processing Toolbox User's Guide* for information on two-dimensional median filtering.

Meta Platforms, Inc. - Exhibit 1009
Page 168 of 720

Communications Applications

The toolbox provides three functions for communications simulation.

Operation	Function
Modulation	`modulate`
Demodulation	`demod`
Voltage controlled oscillation	`vco`

Modulation varies the amplitude, phase, or frequency of a *carrier signal* with reference to a *message signal*. The `modulate` function modulates a message signal with a specified modulation method.

The basic syntax for the `modulate` function is

```
y = modulate(x,Fc,Fs,'method',opt)
```

where:

- `x` is the message signal.
- `Fc` is the carrier frequency.
- `Fs` is the sampling frequency.
- *`method`* is a flag for the desired modulation method.
- `opt` is any additional argument that the method requires. (Not all modulation methods require an option argument.)

The table below summarizes the modulation methods provided; see Chapter 6 for complete details on each.

Method	Description
`amdsb–sc` or `am`	Amplitude modulation, double side-band, suppressed carrier
`amdsb–tc`	Amplitude modulation, double side-band, transmitted carrier

Meta Platforms, Inc. - Exhibit 1009
Page 169 of 720

Method	Description
amssb	Amplitude modulation, single side-band
fm	Frequency modulation
pm	Phase modulation
ptm	Pulse time modulation
pwm	Pulse width modulation
qam	Quadrature amplitude modulation

If the input x is an array rather than a vector, modulate modulates each column of the array.

To obtain the time vector that modulate uses to compute the modulated signal, specify a second output parameter.

```
[y,t] = modulate(x,Fc,Fs,'method',opt)
```

The demod function performs *demodulation*, that is, it obtains the original message signal from the modulated signal.

The syntax for demod is

```
x = demod(y, Fc,Fs,'method',opt)
```

demod uses any of the methods shown for modulate, but the syntax for quadrature amplitude demodulation requires two output parameters.

```
[X1,X2] = demod(y,Fc,Fs,'qam')
```

If the input y is an array, demod demodulates all columns.

Try modulating and demodulating a signal. A 50 Hz sine wave sampled at 1000 Hz is

```
t = (0:1/1000:2);
x = sin(2*pi*50*t);
```

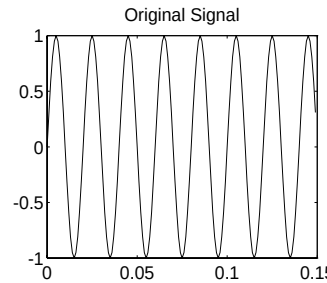

Meta Platforms, Inc. - Exhibit 1009
Page 170 of 720

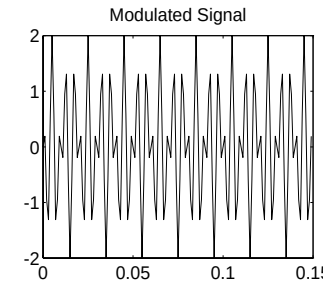
With a carrier frequency of 200 Hz, the modulated and demodulated versions of this signal are

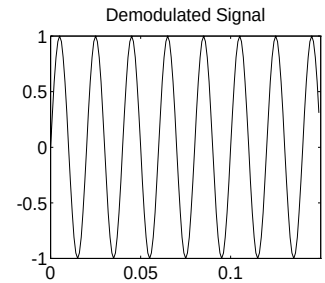
```
y = modulate(x,200,1000,'am');
z = demod(y,200,1000,'am');
```

To plot portions of the original, modulated, and demodulated signal

```
figure; plot(t(1:150),x(1:150)); title('Original Signal');
figure; plot(t(1:150),y(1:150)); title('Modulated Signal');
figure; plot(t(1:150),z(1:150)); title('Demodulated Signal');
```







The voltage controlled oscillator function `vco` creates a signal that oscillates at a frequency determined by the input vector. The basic syntax for `vco` is

```
y = vco(x,Fc,Fs)
```

where `Fc` is the carrier frequency and `Fs` is the sampling frequency.

To scale the frequency modulation range

```
y = vco(x,[Fmin Fmax],Fs)
```


Meta Platforms, Inc. - Exhibit 1009
Page 171 of 720

In this case, `vco` scales the frequency modulation range so values of `x` on the interval `[-1 1]` map to oscillations of frequency on `[Fmin Fmax]`.

If the input `x` is an array, `vco` produces an array whose columns oscillate according to the columns of `x`.

See "FFT-Based Time-Frequency Analysis" on page 4-27 for an example using the `vco` function.

Meta Platforms, Inc. - Exhibit 1009
Page 172 of 720

Deconvolution

Deconvolution, or polynomial division, is the inverse operation of convolution. Deconvolution is useful in recovering the input to a known filter, given the filtered output. This method is very sensitive to noise in the coefficients, however, so use caution in applying it.

The syntax for `deconv` is

```
[q,r] = deconv(b,a)
```

where `b` is the polynomial dividend, `a` is the divisor, `q` is the quotient, and `r` is the remainder.

To try `deconv`, first convolve two simple vectors `a` and `b` (see Chapter 1 for a description of the convolution function)

```
a = [1 2 3];
b = [4 5 6];
c = conv(a,b)

c =
    4   13   28   27   18
```

Now use `deconv` to deconvolve `b` from `c`

```
[q,r] = deconv(c,a)

q =
    4    5    6

r =
    0    0    0    0    0
```

See the *System Identification Toolbox User's Guide* for advanced applications of signal deconvolution.

Meta Platforms, Inc. - Exhibit 1009
Page 173 of 720

Specialized Transforms

In addition to the discrete Fourier transform (DFT) described in Chapter 1, the Signal Processing Toolbox and the MATLAB environment together provide the following transform functions:

- The chirp *z*-transform (CZT), useful in evaluating the *z*-transform along contours other than the unit circle. The chirp *z*-transform is also more efficient than the DFT algorithm for the computation of prime-length transforms, and it is useful in computing a subset of the DFT for a sequence.
- The discrete cosine transform (DCT), closely related to the DFT. The DCT's energy compaction properties are useful for applications like signal coding.
- The Hilbert transform, which facilitates the formation of the analytic signal. The analytic signal is useful in the area of communications, particularly in bandpass signal processing.

Chirp *z*-Transform

The chirp *z*-transform, or CZT, computes the *z*-transform along spiral contours in the *z*-plane for an input sequence. Unlike the DFT, the CZT is not constrained to operate along the unit circle, but can evaluate the *z*-transform along contours described by

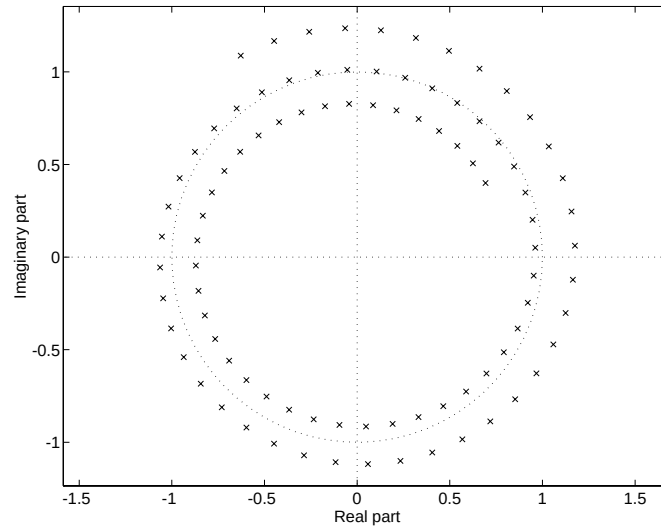
$$z_l = AW^{-l}, \quad l = 0, \ldots, M-1$$

where A is the complex starting point, W is a complex scalar describing the complex ratio between points on the contour, and M is the length of the transform.

Meta Platforms, Inc. - Exhibit 1009
Page 174 of 720

One possible spiral is

```
A = 0.8*exp(j*pi/6);
W = 0.995*exp(-j*pi*.05);
M = 91;
z = A*(W.^(-(0:M-1)));
zplane([],z.')
```



`czt(x,M,W,A)` computes the *z*-transform of `x` on these points.

An interesting and useful spiral set is `m` evenly spaced samples around the unit circle, parameterized by `A = 1` and `W = exp(-j*pi/M)`. The *z*-transform on this contour is simply the DFT, obtained by

```
y = czt(x)
```

`czt` is faster than the `fft` function for computing the DFT of sequences with certain odd lengths, particularly long prime-length sequences. (Try comparing the execution time for the `fft` and `czt` functions for a sequence of length 1013.)

Meta Platforms, Inc. - Exhibit 1009
Page 175 of 720

Discrete Cosine Transform

The toolbox function `dct` computes the unitary discrete cosine transform, or DCT, for an input vector or matrix. Mathematically, the unitary DCT of an input sequence x is

$$y(k) = \sum_{n=1}^{N} w(n)x(n)\cos\frac{\pi(2n-1)(k-1)}{2N}, \qquad k = 1, \ldots, N$$

where

$$w(n) = \begin{cases} \frac{1}{\sqrt{N}}, & n = 1 \\ \sqrt{\frac{2}{N}}, & 2 \le n \le N \end{cases}$$

The DCT is closely related to the discrete Fourier transform; the DFT is actually one step in the computation of the DCT for a sequence. The DCT, however, has better *energy compaction* properties, with just a few of the transform coefficients representing the majority of the energy in the sequence. The energy compaction properties of the DCT make it useful in applications such as data communications.

The function `idct` computes the inverse DCT for an input sequence, reconstructing a signal from a complete or partial set of DCT coefficients. The inverse discrete cosine transform is

$$x(n) = \sum_{k=1}^{N} w(k)y(k)\cos\frac{\pi(2n-1)(k-1)}{2N}, \qquad n = 1, \ldots, N$$

where

$$w(k) = \begin{cases} \frac{1}{\sqrt{N}}, & k = 1 \\ \sqrt{\frac{2}{N}}, & 2 \le k \le N \end{cases}$$

Meta Platforms, Inc. - Exhibit 1009
Page 176 of 720

Because of the energy compaction mentioned above, it is possible to reconstruct a signal from only a fraction of its DCT coefficients. For example, generate a 10 Hz sinusoidal sequence, sampled at 1000 Hz.

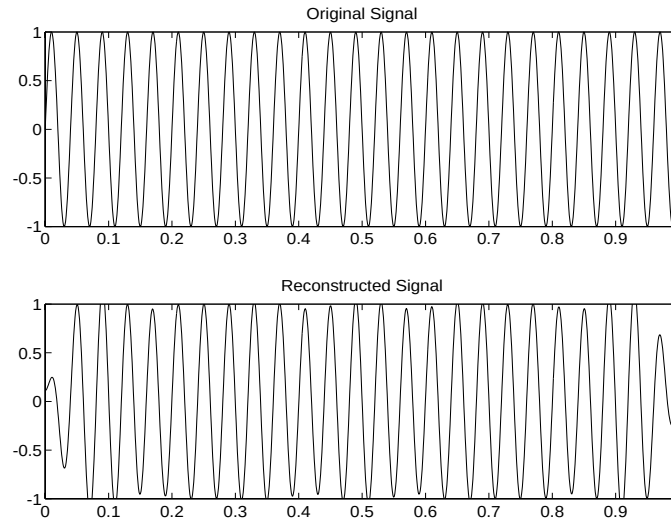
```
t = (0:1/999:1);
x = sin(2*pi*25*t);
```

Compute the DCT of this sequence and reconstruct the signal using only those components with value greater than 53 (12 of the original 1000 DCT coefficients).

```
y = dct(x);                 % compute DCT
y2 = find(abs(y) < 53);     % use 12 coefs.
y(y2) = zeros(size(y2));    % zero out points < 53
z = idct(y);                % reconstruct signal using inverse DCT
```

Plot the original and reconstructed sequences.

```
plot(t,x)
plot(t,z), axis([0 1 –1 1])
```



Meta Platforms, Inc. - Exhibit 1009
Page 177 of 720

One measure of the accuracy of the reconstruction is

```
norm(x-z)/norm(x)
```

that is, the norm of the difference between the original and reconstructed signals, divided by the norm of the original signal. In this case, the relative error of reconstruction is 0.1778. The reconstructed signal retains approximately 82% of the energy in the original signal.

Hilbert Transform

The toolbox function `hilbert` computes the Hilbert transform for a real input sequence `x` and returns a complex result of the same length

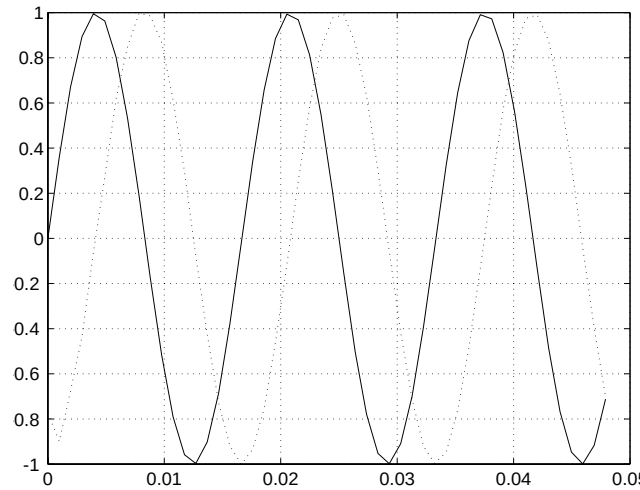
```
y = hilbert(x)
```

where the real part of `y` is the original real data and the imaginary part is the actual Hilbert transform. `y` is sometimes called the *analytic signal*, in reference to the continuous-time analytic signal. A key property of the discrete-time analytic signal is that its *z*-transform is 0 on the lower half of the unit circle. Many applications of the analytic signal are related to this property; for example, the analytic signal is useful in avoiding aliasing effects for bandpass sampling operations. The magnitude of the analytic signal is the complex envelope of the original signal.

Meta Platforms, Inc. - Exhibit 1009
Page 178 of 720

The Hilbert transform is related to the actual data by a 90° phase shift; sines become cosines and vice versa. To plot a portion of data (solid line) and its Hilbert transform (dotted line)

```
t = (0:1/1023:1);
x = sin(2*pi*60*t);
y = hilbert(x);
plot(t(1:50),real(y(1:50))), hold on
plot(t(1:50),imag(y(1:50)),':'), hold off
```



The analytic signal is useful in calculating *instantaneous attributes* of a time series, the attributes of the series at any point in time. The instantaneous amplitude of the input sequence is the amplitude of the analytic signal. The instantaneous phase angle of the input sequence is the (unwrapped) `angle` of the analytic signal; the instantaneous frequency is the time rate of change of the instantaneous phase angle. You can calculate the instantaneous frequency using `diff`, as described in the online *MATLAB Function Reference*.

Meta Platforms, Inc. - Exhibit 1009
Page 179 of 720

References

1 Kay, S.M. *Modern Spectral Estimation*. Englewood Cliffs, NJ: Prentice Hall, 1988.

2 Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

3 Parks, T.W., and C.S. Burrus. *Digital Filter Design*. New York: John Wiley & Sons, 1987.

4 Pratt,W.K. *Digital Image Processing*. New York: John Wiley & Sons, 1991.

Meta Platforms, Inc. - Exhibit 1009
Page 180 of 720

5

Interactive Tools

SPTool: An Interactive Signal Processing Environment . 5-2

Using SPTool 5-3

Using the Signal Browser: Interactive Signal Analysis . 5-43

Using the Filter Designer: Interactive Filter Design . . . 5-59

Using the Filter Viewer: Interactive Filter Analysis . . . 5-84

Using the Spectrum Viewer: Interactive PSD Analysis . 5-97

Example: Generation of Bandlimited Noise 5-113

SPTool: An Interactive Signal Processing Environment

The Signal Processing Toolbox includes an interactive graphical user interface (GUI), called SPTool, for performing digital signal processing tasks. SPTool provides an easy-to-use interface to many of the most important toolbox functions. With it, you can use the mouse and on-screen controls to import, view, measure, and print digital signals; design, view, and implement digital filters; and analyze the frequency content of signals.

This chapter describes how to use the different components of SPTool. Where appropriate, we point you to other areas of the manual that describe how to perform similar tasks by calling functions from the command line or from M-files.

The section "Example: Generation of Bandlimited Noise" at the end of this chapter describes how to use this graphical environment for a complete filter design and analysis task.

Overview

SPTool is a graphical environment for analyzing and manipulating digital signals, filters, and spectra. It is the starting point for using the interactive signal processing environment. In SPTool, you can import signals, filters, and spectra either from the workspace or as MAT-files. Through SPTool, you access four additional GUI tools that provide an integrated environment for signal browsing, filter design, analysis, and implementation. The four components of the interactive signal processing environment include:

- The *Signal Browser*, which provides a graphical view of the signal objects currently selected in SPTool and enables you to display, measure, analyze, and print these signals interactively
- The *Filter Designer*, which enables you to create and edit lowpass, highpass, bandpass, and bandstop FIR and IIR digital filters of various lengths and types using the filter design functions of the Signal Processing Toolbox
- The *Filter Viewer*, which enables you to view various characteristics of a filter that you've imported or designed, including its magnitude and phase responses, group delay, zero-pole plot, and impulse and step responses
- The *Spectrum Viewer*, which enables you to create, view, modify, and print spectra interactively, and to perform graphical analysis of frequency domain data using a variety of common methods of spectral estimation

Meta Platforms, Inc. - Exhibit 1009
Page 182 of 720

Using SPTool

SPTool is the data management tool for the interactive GUI environment of the Signal Processing Toolbox. Using SPTool you can:

- Load a saved session
- Import a signal, filter, or spectrum
- Duplicate or clear a signal, filter, or spectrum
- Change the name of a signal, filter, or spectrum
- Change the sampling frequency of a signal or filter
- Activate the Signal Browser, Filter Viewer, Filter Designer, or Spectrum Viewer
- Save a session
- Use the **Window** menu to change to any open MATLAB figure window

Opening SPTool

Open SPTool from the MATLAB command window by typing

```
sptool
```

and pressing **Enter**.

Quick Start

Once SPTool is open, you can import data from the workspace or a file. You can then view it in the Signal Browser or generate its spectrum in the Spectrum Viewer.

To get started right away, work through the following example. Then continue through this chapter to learn the details of using SPTool and its component tools.

Or, you can skip the example, read through the rest of this section (from "Basic SPTool Functions" to "Using the Signal Browser: Interactive Signal Analysis"), and then work through the example.

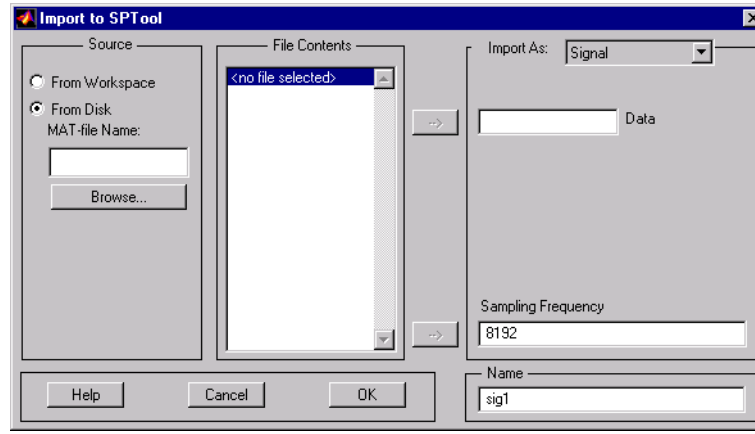
Example: Importing Signal Data from a MAT-File

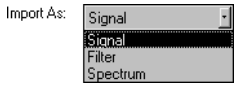
This example uses the sample file `mtlb.mat`, which is in the `toolbox/signal/signal` directory.

Meta Platforms, Inc. - Exhibit 1009
Page 183 of 720

1 Select **Import** from the **File** menu.

The **Import to SPTool** window is displayed.





2 Make sure that **Signal** is displayed in the **Import As** pop-up menu.

3 Press the **From Disk** radio button.

You can either enter a MAT-file name or press **Browse** to open the file dialog box and select a MAT-file.

4 To get started, type the file name `mtlb` and press **Tab** or **Enter**. Note that SPTool adds the `.mat` extension automatically.

The data from the file you selected is displayed in the **File Contents** list. In this example, `mtlb` is the signal data and `Fs` is the sampling frequency.

Notice that `sig1` is displayed in the **Name** field. This is the default name of the imported signal.

5 Click on `mtlb` to select it, and then press the arrow at the left of the **Data** field. `mtlb` is transferred to the **Data** field.

6 Click on `Fs` to select it, and then press the arrow at the left of the **Sampling Frequency** field.

`Fs` is transferred to the **Sampling Frequency** field.

Meta Platforms, Inc. - Exhibit 1009
Page 184 of 720

7 Press **OK**.

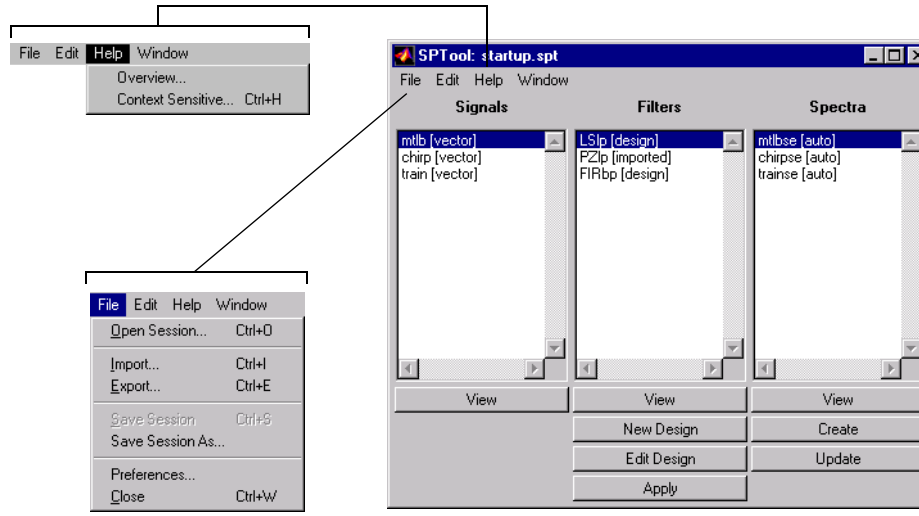
The signal has been imported into SPTool with the name `sig1`.

You can look at this signal in the Signal Browser by pressing the **View** button under **Signals**.

You can look at the frequency content of the signal in the Spectrum Viewer by pressing the **Create** button under **Spectra** and then pressing **Apply** in the Spectrum Viewer.

Basic SPTool Functions

When you first open SPTool, it contains a collection of default signals, filters, and spectra. You can import additional signals, filters, and spectra into SPTool, and you can also design filters using the Filter Designer and create spectra using the Spectrum Viewer. You can save and export data from SPTool and customize many properties of the SPTool environment. The following figure shows the SPTool window and its **File** and **Help** menus, which are described below.



Meta Platforms, Inc. - Exhibit 1009
Page 185 of 720

File Menu

Open Session. Select **Open Session...** from the **File** menu to load a saved session file. An SPTool session is saved in a file with an `.spt` extension.

Import. Select **Import...** from the **File** menu to import a signal, filter, or spectrum into SPTool from either the workspace or from a file. You can import variables from any MAT-file into SPTool. See "Importing Signals, Filters, and Spectra" on page 5-7 and "Example: Importing Signal Data from a MAT-File" on page 5-3 for more information.

Export. Select **Export...** from the **File** menu to export signals, filters, and spectra to the MATLAB workspace as structure variables. See "Saving Signal Data" on page 5-57, "Saving Filter Data" on page 5-79, and "Saving Spectrum Data" on page 5-110 for complete information.

Save Session. Select **Save Session** and **Save Session As...** from the **File** menu to save the current session. **Save Session** overwrites the existing session file. **Save Session As...** saves the current session with a name you specify. An SPTool session is saved in a MAT-file with an `.spt` extension.

Preferences. Select **Preferences...** from the **File** menu to customize preferences for the behavior of all the Signal Processing GUI tools. See "Customizing Preferences" on page 5-20 for a complete discussion.

Close. Select **Close** from the **File** menu to close SPTool and all other active Signal Processing GUI tools. SPTool prompts you to save if you have not recently saved the current session.

When you close SPTool, all signal and filter customization and ruler information set in any of the GUI tools are lost. Settings you changed and saved using the **Preferences** dialog box in SPTool are used the next time you open SPTool.

Help Menu

Overview... Select **Overview...** from the **File** menu to get general help on SPTool and the Signal Processing Toolbox GUI environment. This also gives you access to the MATLAB Help Desk.

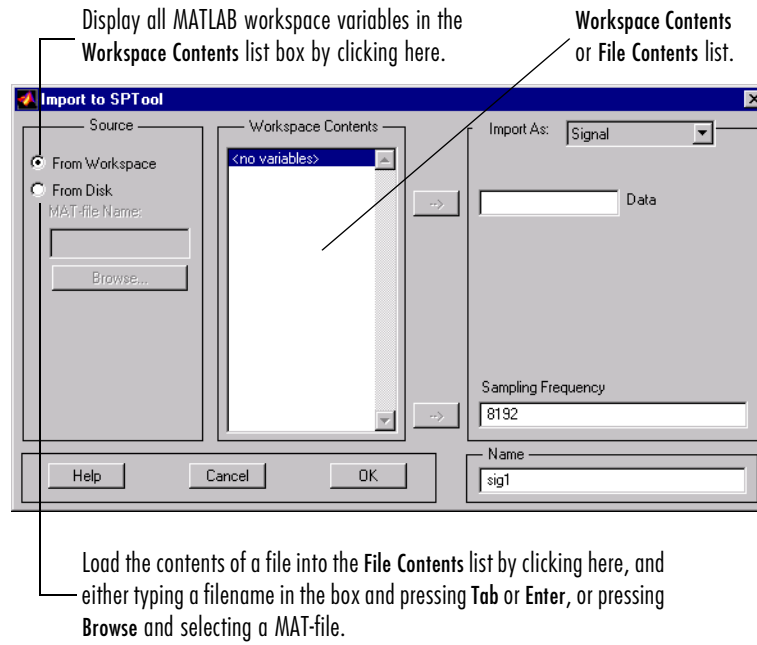
Meta Platforms, Inc. - Exhibit 1009
Page 186 of 720

Context Sensitive... Select **Context Sensitive...** from the **File** menu for help on a specific part of SPTool. When you click on **Context Sensitive...**, the mouse pointer becomes an arrow with a question mark symbol. You can then click on anything in SPTool, including menu items, to find out what it is and how to use it.

Importing Signals, Filters, and Spectra

You can import a signal, filter, or spectrum into SPTool from either the workspace or from a file.

Select **Import...** from the **File** menu to open the **Import to SPTool** window:



Loading Variables from the MATLAB Workspace

To import variables from the MATLAB workspace, first list the workspace variables in the **Workspace Contents** list. Then select the variables to be imported into SPTool:

Meta Platforms, Inc. - Exhibit 1009
Page 187 of 720

1 Click the **From Workspace** radio button.

   The contents of the MATLAB workspace are displayed in the **Workspace Contents** list.

2 You can now import one or more variables from the **Workspace Contents** list into SPTool. See "Importing Workspace Contents and File Contents" below.

Loading Variables from Disk

To import variables from a MAT-file on disk, first list the file's variables in the **File Contents** list. Then select the variables to be imported into SPTool:

1 Click the **From Disk** radio button.

2 Type the name of the file you want to import into the **MAT-file Name** field and press **Tab** or **Enter**.

   or

   Press **Browse**, and then find and select the file you want to import using the File Search window. Press **OK**.

   The data from the file you selected is displayed in the **File Contents** list.

3 You can now import one or more variables from the **File Contents** list into SPTool (see below).

Importing Workspace Contents and File Contents

Once you've loaded the contents of the workspace or a file into the **Workspace Contents** or **File Contents** list, you can select one or more variables from the list to import into SPTool. You can import a variable as a signal, a filter, or a spectrum. You can also import a variable whose value represents a sampling frequency or other design parameter.

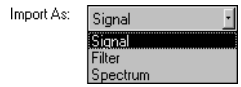
Depending on whether you're importing a signal, a filter, or a spectrum, you can customize different parameters before you import the data into SPTool. In each case, however, the general procedure for specifying a variable or a value is the same. In the following example, the selected variable is being imported as a signal. See "Importing a Signal" on page 5-10, "Importing a Filter" on page

Meta Platforms, Inc. - Exhibit 1009
Page 188 of 720

5-11, and “Importing a Spectrum” on page 5-13 for details on customizing variables that are imported into SPTool.

1 Click on a variable name in the **Workspace Contents** list or the **File Contents** list to select it.

   If the variable is not a saved data object from SPTool, select the appropriate data type (**Signal**, **Filter**, or **Spectrum**) from the **Import As** pop-up menu and type a name into the **Name** field.



   If the variable is a saved data object from SPTool, its name is displayed in the **Name** field, and its type (**Signal**, **Filter**, or **Spectrum**) is automatically selected in the **Import As** pop-up menu.

2 Press the arrow at the left of the **Data** field. The selected variable is transferred to the **Data** field.

NOTE You can also type a variable name into the **Data** field directly.

---

3 To change the sampling frequency of the variable you’re importing, you can either:

   a Click on a variable in the **Workspace Contents** list or **File Contents** list whose value is a sampling frequency, and then press the arrow at the left of the **Sampling Frequency** field.

      The selected variable is transferred to the **Sampling Frequency** field.

      or

   b Type a value or variable name in the **Sampling Frequency** field.

Meta Platforms, Inc. - Exhibit 1009
Page 189 of 720

4 Press **OK**.

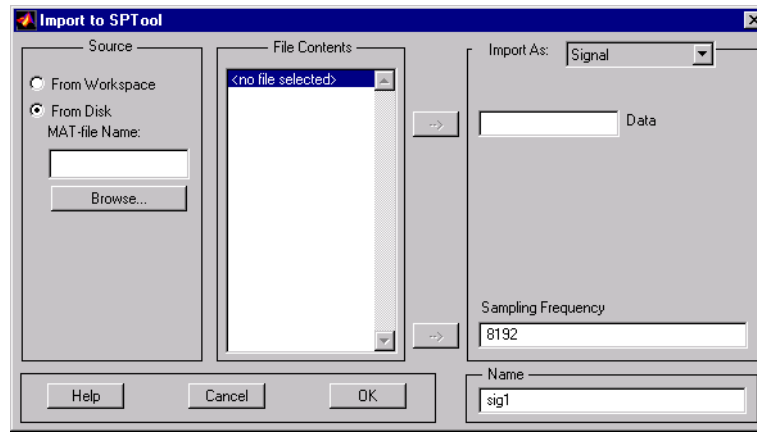
The signal is imported into SPTool with the specified name and sampling frequency.

5 To import another variable, select **Import...** again, press the **From Workspace** or **From File** radio button, and repeat steps 1 through 4 for each variable that you want to load into SPTool.

NOTE When you're importing from the workspace, you can specify either a variable or a value for each data field. When you're importing from a disk, you can only specify variables.

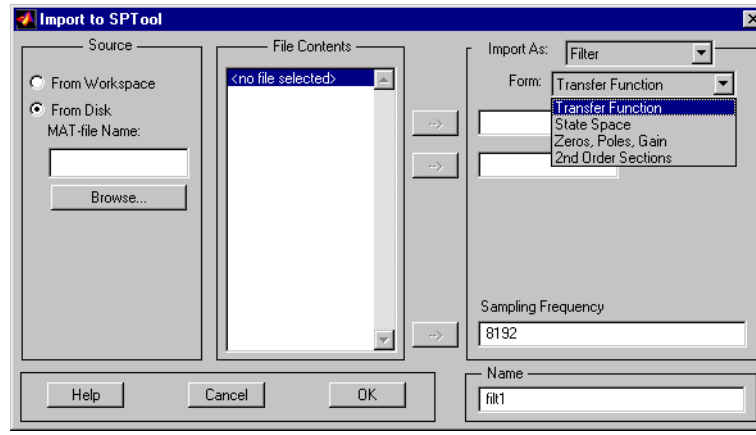
Importing a Signal. When you import a signal, you specify:

- A variable name for the signal data (or the signal data values) in the **Data** field
- A variable or a value for the signal's sampling frequency in the **Sampling Frequency** field



Meta Platforms, Inc. - Exhibit 1009
Page 190 of 720

Importing a Filter. When you import a filter, first select the appropriate filter form from the **Form** pop-up menu.



Each filter form requires different variables.

- For **Transfer Function**, you specify the filter by its transfer function representation:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + \cdots + b(m+1)z^{-m}}{a(1) + a(2)z^{-1} + \cdots + a(n+1)z^{-n}}$$

  - The **Numerator** field specifies a variable name or value for the numerator coefficient vector b, which contains m+1 coefficients in descending powers of z.
  - The **Denominator** field specifies a variable name or value for the denominator coefficient vector a, which contains n+1 coefficients in descending powers of z.
- For **State Space**, you specify the filter by its state-space representation:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

  The **A-Matrix**, **B-Matrix**, **C-Matrix**, and **D-Matrix** fields specify a variable name or a value for each matrix in this system.
- For **Zeros, Poles, Gain**, you specify the filter by its zero-pole-gain representation:

Meta Platforms, Inc. - Exhibit 1009
Page 191 of 720

$$H(z) = \frac{Z(z)}{P(z)} = k\frac{(z-z(1))(z-z(2))\cdots(z-z(m))}{(z-p(1))(z-p(2))\cdots(z-p(n))}$$

- The **Zeros** field specifies a variable name or value for the zeros vector z, which contains the locations of m zeros.
- The **Poles** field specifies a variable name or value for the zeros vector p, which contains the locations of n poles.
- The **Gain** field specifies a variable name or value for the gain k.

- For **2nd Order Sections** you specify the filter by its second-order section representation:

$$H(z) = \prod_{k=1}^{L} H_k(z) = \prod_{k=1}^{L} \frac{b_{0k} + b_{1k}z^{-1} + b_{2k}z^{-2}}{1 + a_{1k}z^{-1} + a_{2k}z^{-2}}$$

  The **SOS Matrix** field specifies a variable name or a value for the L-by-6 SOS matrix

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

  whose rows contain the numerator and denominator coefficients b_{ik} and a_{ik} of the second-order sections of $H(z)$:

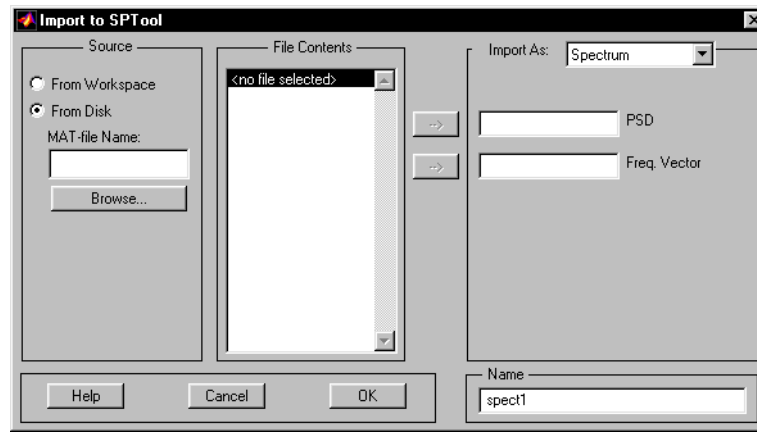
- For every filter, you specify:

  A variable name or a value for the filter's sampling frequency in the **Sampling Frequency** field

Meta Platforms, Inc. - Exhibit 1009
Page 192 of 720

Importing a Spectrum. When you import a spectrum, you specify:

- A variable name or a value for the power spectral density (PSD) vector in the **PSD** field
- A variable name or a value for the frequency vector in the **Freq. Vector** field

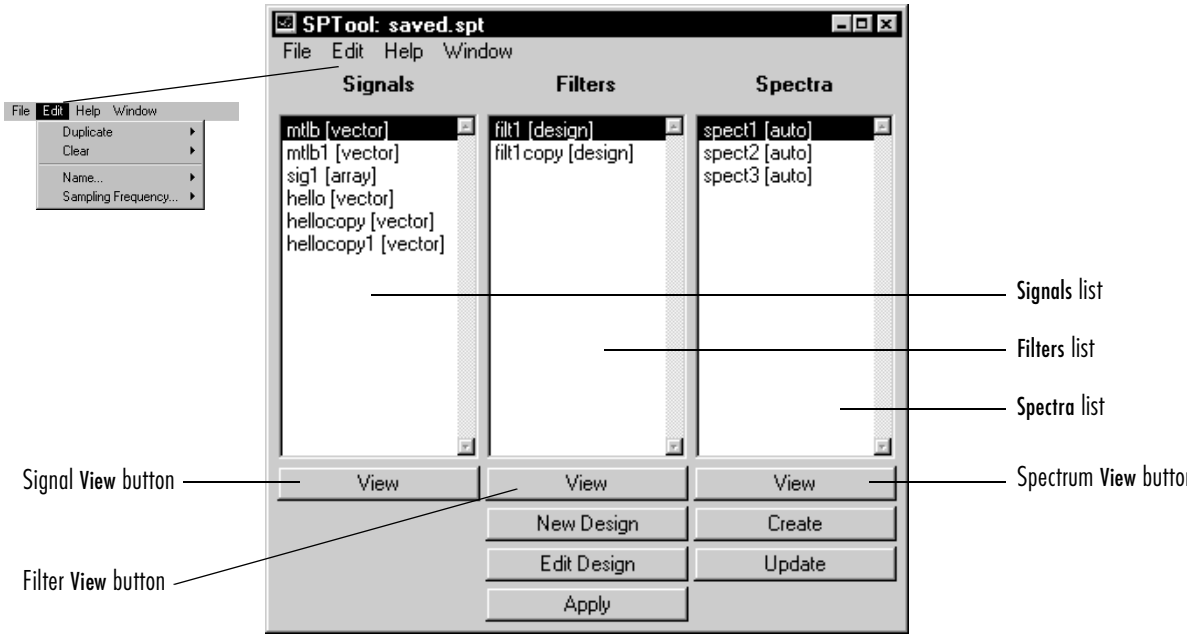


The PSD values in the **PSD** vector correspond to the frequencies contained in the **Freq. Vector** vector; the two vectors must have the same length.

Working with Signals, Filters, and Spectra

When a signal, filter, or spectrum is imported into SPTool or created in SPTool, it is displayed in the appropriate list box, as shown below. Using the **Edit** menu functions and SPTool buttons, you can edit various properties of the data in SPTool and invoke all of SPTool's digital signal processing functions.

Meta Platforms, Inc. - Exhibit 1009
Page 193 of 720



Component Lists in SPTool

Each signal, filter, and spectrum in SPTool is displayed in the appropriate **Signals** list, **Filters** list, or **Spectra** list.

- Signals are displayed with the *signal type* `[vector]` or `[array]`:
  - A *vector signal* (`[vector]`) has one column of data.
  - An *array signal* (`[array]`) has more than one column of data.
- Filters are displayed with the *filter type* `[design]` or `[imported]`:
  - A *designed filter* (`[design]`) is a filter that was created using the Filter Designer. This type of filter can also be edited in the Filter Designer.
  - An *imported filter* (`[imported]`) is a filter that was imported from the MATLAB workspace or a file, or one that was edited in the Pole/Zero Editor.

Meta Platforms, Inc. - Exhibit 1009
Page 194 of 720

- Spectra are displayed with the *spectrum type* `[auto]`:

  An *auto-spectrum* (`[auto]`) is a spectrum whose source is a single signal, as opposed to the cross-spectrum of two channels of data. `spectrum[auto]` is the only spectrum type in SPTool.

Selecting Data Objects in SPTool

Each signal, filter, and spectrum in SPTool is one data object. A data object is selected when it is highlighted. When you first import or create a data object, it is selected.

The **Signals** list shows all vector and array signals in the current SPTool session.

The **Filters** list shows all designed and imported filters in the current SPTool session.

The **Spectra** list shows all spectra in the current SPTool session.

You can select a single data object in a list, a range of data objects in a list, or multiple separate data objects in a list. You can also have data objects simultaneously selected in different lists:

- To select a single unselected data object, click on it. All other data objects in that list box become unselected.
- To add or remove a range of data objects, **Shift**-click on the data objects at the top and bottom of the section of the list that you want to add.
- To add a single data object to a selection or remove a single data object from a multiple selection, **Ctrl**-click on the object. You can also use the right mouse button.

Editing Data Objects in SPTool

The **Edit** menu entries are available only when there is at least one selected data object (signal, filter, or spectrum) in SPTool. Use the **Edit** menu to duplicate and clear objects in SPTool and to edit object names and change sampling frequencies.

A signal, filter, or spectrum must be selected to be edited. When you click on an **Edit** menu entry, all *selected* data objects are displayed in a pop-up menu.

Meta Platforms, Inc. - Exhibit 1009
Page 195 of 720

To edit an SPTool object:

1 Select a signal, filter, or spectrum.

2 Select the appropriate **Edit** menu function.

The pop-up menu shows the names of all selected data objects.

3 Drag to choose a specific signal, filter, or spectrum for editing.

Duplicate. Use **Duplicate** from the **Edit** menu to make a copy of the selected signal, filter, or spectrum in SPTool.

Select **Duplicate** and drag to choose the signal, filter, or spectrum you want to copy. When you select a data object to duplicate it, a new data object of the same type is automatically created. The new data object is named as a copy of the selected data object. It is placed at the bottom of the list and is selected. You can change its name using **Name...** from the **Edit** menu.

Clear. Use **Clear** from the **Edit** menu to delete the selected signal, filter, or spectrum from SPTool.

Select **Clear** and drag to choose the signal, filter, or spectrum you want to remove. The data object is deleted from the current SPTool session.

Name. Use **Name...** from the **Edit** menu to give the selected signal, filter, or spectrum a new, unique name.

1 Select **Name...** and drag to choose the signal, filter, or spectrum you want to rename.

The **Name Change** dialog box is displayed.

2 Type in the new name and press **OK**.

Sampling Frequency. Use **Sampling Frequency...** from the **Edit** menu to supply a sampling frequency for a selected signal or filter. The sampling frequency may be a number, such as 1, 0.001, or 1/5000, or a valid MATLAB expression including workspace variables, such as `Fs`, `1/Ts`, or `cos(0.1*pi)`.

Meta Platforms, Inc. - Exhibit 1009
Page 196 of 720

1 Select **Sampling Frequency...** and drag to choose the signal or filter you want to change.

   The **Sampling Frequency...** dialog box is displayed.

2 Type in the value, variable name, or expression and press **OK**.

Viewing a Signal

Use the Signal **View** button to make the Signal Browser active and view one or more imported signals. The Signal Browser provides tools for graphical analysis of the selected signal(s).

Select one or more signals from the **Signals** list and press the **View** button in the signal panel. The Signal Browser displays the selected signal(s). See "Using the Signal Browser: Interactive Signal Analysis" on page 5-43 for a full description of Signal Browser functions and operations.

Viewing a Filter

Use the **View** button in the **Filters** panel to make the Filter Viewer active and view imported filters or filters designed/edited in the Filter Designer. The Filter Viewer provides tools for analyzing filters; you can investigate the magnitude response, phase, group delay, zeros and poles, and impulse and step responses of the selected filters.

Select one or more filters from the **Filters** list and press the **View** button in the filter panel. The Filter Viewer displays the selected filters. See "Using the Filter Viewer: Interactive Filter Analysis" on page 5-84 for a full description of Filter Viewer functions and operations.

Designing a Filter

New Design. Use the Filter **New Design** button to make the Filter Designer active and design a filter. The Filter Designer lets you create FIR and IIR digital filters of various lengths and types using the filter design functions in the Signal Processing Toolbox.

Press the **New Design** button in the **Filter** panel. The Filter Designer displays a filter created with the default settings and assigns it a default name of `filt`*n*, where *n* is a unique sequential identifying digit. Once the default filter is created, you can use it as is or edit it with the Filter Designer. You can rename it using **Name...** from the **Edit** menu.

Meta Platforms, Inc. - Exhibit 1009
Page 197 of 720

Edit Design. Use the **Filter** panel's **Edit Design** button to make the Filter Designer active and edit a filter. Imported filters always open in the Pole/Zero Editor.

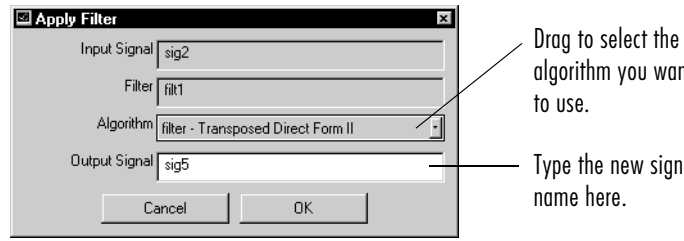
Select one or more filters from the **Filters** list and press the **Edit Design** button in the filter panel. The Filter Designer displays the first of the selected filters. The other selected filters can be edited, one at a time, by selecting them from the **Filter** pop-up menu. See "Using the Filter Designer: Interactive Filter Design" on page 5-59 for a full description of Filter Designer functions and operations.

Applying a Filter

Use the **Apply Filter** button to apply a filter to a selected signal. This creates a new signal:

1 Select one signal from the **Signals** list and one filter from the **Filters** list and press the **Apply Filter** button in the filter panel.

The **Apply Filter** dialog box is displayed.



2 Select the filtering algorithm from the **Algorithm** pop-up menu, type the name for the new signal in the **Output Signal** field, and press **OK**.

The available filter algorithms are:

- Transposed direct-form II (uses the `filter` function)
- Zero-phase IIR (uses the `filtfilt` function)
- FFT-based FIR (uses the `fftfilt` function)

The selected filter is applied to the selected input signal and the new output signal is listed in the **Signals** list.

Meta Platforms, Inc. - Exhibit 1009
Page 198 of 720

Creating a Spectrum

Use the **Create** button to activate the Spectrum Viewer and generate a default spectrum of a selected signal. Once you've generated a spectrum, you can view it in a variety of ways, measure it, and modify it in the Spectrum Viewer.

1. Select one signal from the **Signals** list and press the **Create** button in the **Spectra** panel.

   The Spectrum Viewer is activated and a spectrum object with default parameters is created in the **Spectra** panel. The PSD data is not computed or displayed yet. The newly created spectrum object has a default name of spec*n*, where *n* is a unique sequential identifying digit. Once the spectrum object is created, you can use it with the default settings, or continue to edit it in the Spectrum Viewer. You can rename it using **Name...** from the **Edit** menu.

2. Modify the spectrum parameters (e.g., **Method** and **Nfft**) as desired.

3. Press **Apply** in the Spectrum Viewer to compute the spectrum data. This button is enabled when the spectrum has just been created or when you have changed one or more parameters in the Spectrum Viewer.

   The updated spectrum is displayed in the Spectrum Viewer. See "Using the Spectrum Viewer: Interactive PSD Analysis" on page 5-97 for a full description of Spectrum Viewer parameters and displays.

Viewing a Spectrum

Select one or more spectra from the **Spectra** list and press **View** in the **Spectra** panel. The Spectrum Viewer displays the selected spectrum or spectra.

Updating a Spectrum

Use the **Update** button to update the selected spectrum so that it reflects the data in the currently selected signal.

1. Select one signal from the **Signals** list and one spectrum from the **Spectra** list and press **Update** in the **Spectra** panel.

   The spectral data from the current spectrum is removed from the Spectrum Viewer. The Spectrum Viewer is activated. No spectrum is displayed yet.

Meta Platforms, Inc. - Exhibit 1009
Page 199 of 720

2 Press **Apply** in the Spectrum Viewer to compute the spectrum and complete the update.

The spectrum is regenerated with the same parameters but using the data in the currently selected signal. This feature is useful when you have altered a signal (by filtering it, for example), and want to update the existing spectrum to reflect the change.

Customizing Preferences

Use **Preferences...** from the **File** menu to customize displays and certain parameters for SPTool and its four component tools. The new settings are saved on disk and are used when you restart MATLAB.

In the **Preferences** panels, you can:

- Select colors and markers for rulers, and set the initial ruler style
- Select color and line style sequence for displayed signals
- Configure axis labels, and enable/disable rulers, panner, and mouse zoom in the Signal Browser
- Configure axis parameters, and enable/disable rulers and mouse zoom in the Spectrum Viewer
- Configure filter and axis parameters and enable/disable mouse zoom in the Filter Viewer
- Configure tiling preferences in the Filter Viewer
- Specify FFT length, and enable/disable mouse zoom and grid in the Filter Designer
- Enable/disable use of a default session file
- Configure filters for export to the Control System Toolbox (for users of that product)
- Enable/disable search for plug-ins at start-up

When you first select **Preferences...**, the **Preferences** dialog box displays the **Rulers** panel. You can change the settings for rulers, or choose on any of the other settings categories to customize other settings.

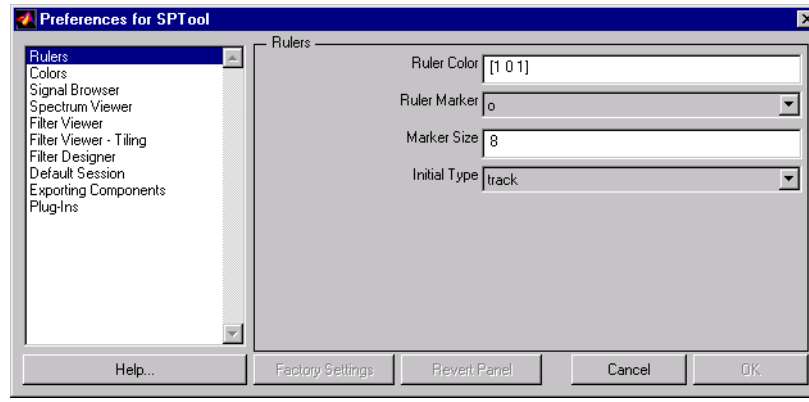
Click once on a settings category to select it.

Meta Platforms, Inc. - Exhibit 1009
Page 200 of 720

The following sections describe all of the settings you can modify. The illustrations show the default settings for each category. For additional information on preference settings, use the **Help...** button at the bottom of the **Preferences** dialog box.

Ruler Settings

The **Rulers** preferences apply to the rulers in the Signal Browser, Spectrum Viewer, and Filter Viewer. These preferences also apply to printouts from the Signal Browser and Spectrum Viewer.



Ruler Color. Specifies the color of the rulers. Color is specified as a string (e.g., `'r'`) or RGB triple (e.g., `[1 0 0]`).

Ruler Marker. Specifies the marker used in the track and slope rulers.

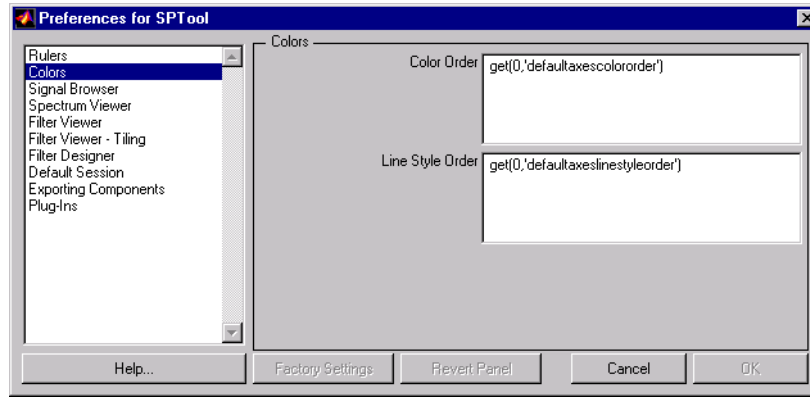
Marker Size. Specifies the size of the ruler marker in points. This can be any positive value.

Initial Type. Specifies the type of ruler (horizontal, vertical, track, or slope) that is selected when you first open the Signal Browser, Spectrum Viewer, or Filter Viewer.

Meta Platforms, Inc. - Exhibit 1009
Page 201 of 720

Color Settings

The **Colors** preferences apply to signals displayed in the Signal Browser, Spectrum Viewer, and Filter Viewer. These preferences also apply to printouts from the Signal Browser and Spectrum Viewer.



Color Order. Specifies the color order to cycle through for data plotted in the Signal Browser, Spectrum Viewer, and Filter Viewer. The default is the axis color order.

Type in a new value or value string to change the color order. Color is specified as a string (e.g., `'r'`) or RGB triple (e.g., `[1 0 0]`), an *n*-by-3 matrix of *n* colors, or an *n*-by-1 cell array of such objects.

Line Style Order. Specifies the line styles to cycle through for data plotted in the Signal Browser, Spectrum Viewer, and Filter Viewer. The default is the axis line style order.

Type in a new string value (e.g., `'--'`) or an array of strings (e.g., `{'-','--',':'}`) to change the line style order.

Meta Platforms, Inc. - Exhibit 1009
Page 202 of 720

Signal Browser Settings

The **Signal Browser** preferences let you set optional *x*-axis and *y*-axis labels, enable and disable the display of the rulers and the panner, and toggle zoom persistence.

Except for zoom persistence, these preferences also apply to printing from the Signal Browser. See "Printing Signal Data" on page 5-54 for details on printing from the Signal Browser.

X Label, Y Label. Type in a string for the *x*-axis label and the *y*-axis label in the Signal Browser. The default is `Time` for the *x*-axis.

Rulers. Click in the check box to display (checked) or hide (unchecked) the ruler buttons and the ruler panel in the Signal Browser. See "Ruler Controls" on page 5-33 for details on using the rulers in the Signal Browser.

Panner. Click in the check box to display (checked) or hide (unchecked) the panner in the Signal Browser. See "Panner Display" on page 5-52 for details on using the panner in the Signal Browser.

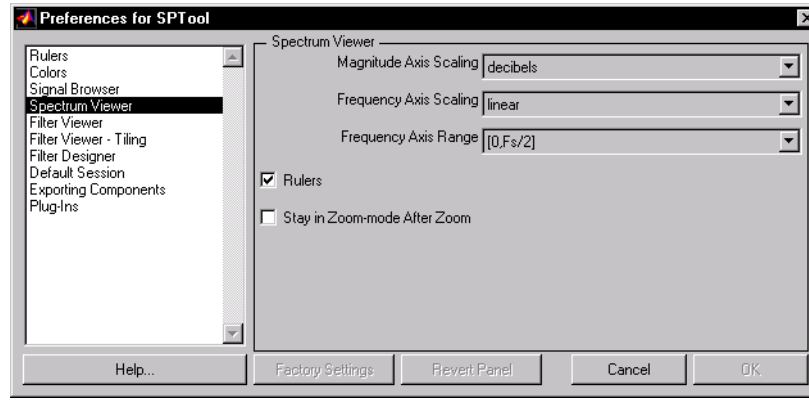
Stay in Zoom-mode After Zoom. Click in the check box to enable (checked) or disable (unchecked) zoom persistence in the Signal Browser. See "Zoom Controls" on page 5-31 for details on zoom controls in the Signal Browser.

Meta Platforms, Inc. - Exhibit 1009
Page 203 of 720

Spectrum Viewer Settings

The **Spectrum Viewer** preferences let you set axis parameters, enable and disable the display of the rulers, and toggle zoom persistence.

Except for zoom persistence, these preferences also apply to printing from the Spectrum Viewer. See "Printing Spectrum Data" on page 5-107 for details on printing from the Spectrum Viewer.



Magnitude Axis Scaling. Specifies the scaling units for the magnitude (*y*) axis in the Spectrum Viewer. Scaling units can be **decibels** or **linear**.

Frequency Axis Scaling. Specifies the scaling units for the frequency (*x*) axis in the Spectrum Viewer. Scaling units can be **linear** or **log**.

Frequency Axis Range. Specifies the numerical range for the frequency (*x*) axis in the Spectrum Viewer. Scaling options are **[0,Fs/2]**, **[0,Fs]**, or **[-Fs/2,Fs/2]**.

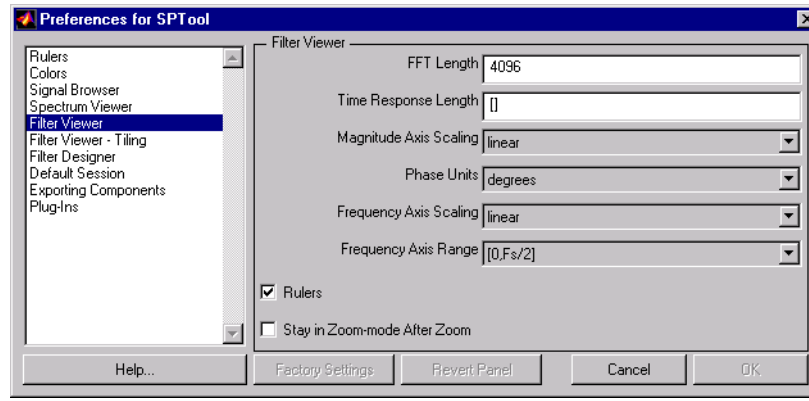
Rulers. Click in the check box to display (checked) or hide (unchecked) the ruler buttons and the ruler panel in the Spectrum Viewer. See "Ruler Controls" on page 5-33 for details on using the rulers in the Spectrum Viewer.

Stay in Zoom-mode After Zoom. Click in the check box to enable (checked) or disable (unchecked) zoom persistence in the Spectrum Viewer. See "Zoom Controls" on page 5-31 for details on zoom controls in the Spectrum Viewer.

Meta Platforms, Inc. - Exhibit 1009
Page 204 of 720

Filter Viewer Settings

The **Filter Viewer** preferences let you set key filter plot configuration parameters and toggle zoom persistence.



FFT Length. Specifies the number of points at which the frequency response is computed (for the magnitude, phase, and group delay plots).

Time Response Length. Specifies the time response length (in samples) for the impulse and step response plots. An empty value, [ ], for an FIR filter indicates that the complete response length will be shown; an empty value for an IIR filter indicates that the response length will be automatically determined using the `impz` function.

Magnitude Axis Scaling. Specifies the scaling units for the magnitude (y) axis in the Filter Viewer. Scaling units can be **linear**, **log**, or **decibels**.

Phase Units. Specifies the phase units for the phase response plot. Phase units can be **degrees** or **radians**.

Frequency Axis Scaling. Specifies the scaling units for the frequency (x) axis in the Filter Viewer. Scaling units can be **linear** or **log**.

Frequency Axis Range. Specifies the numerical range for the frequency (x) axis in the Filter Viewer. Scaling options are **[0,Fs/2]**, **[0,Fs]**, or **[-Fs/2,Fs/2]**.

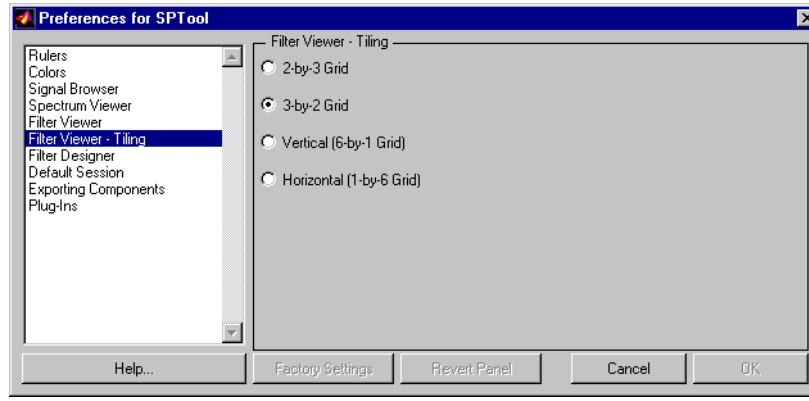
Meta Platforms, Inc. - Exhibit 1009
Page 205 of 720

Rulers. Click in the check box to display (checked) or hide (unchecked) the ruler buttons and the ruler panel in the Filter Viewer. See "Ruler Controls" on page 5-33 for details on using the rulers in the Filter Viewer.

Stay in Zoom-mode After Zoom. Click in the check box to enable (checked) or disable (unchecked) zoom persistence in the Filter Viewer. See "Zoom Controls" on page 5-31 for details on zoom controls in the Filter Viewer.

Filter Viewer Tiling Settings

The **Filter Viewer–Tiling** preferences let you change the way the Filter Viewer displays the analysis plots.



Click the radio button to select how the plots are tiled in the display area. Options are **2-by-3 Grid**, **3-by-2 Grid**, **Vertical (6-by-1 Grid)**, and **Horizontal (1-by-6 Grid)**.

This specifies how the plots are arranged when all six plot options are turned on. When fewer options are turned on, the plots are displayed as symmetrically as possible.

Meta Platforms, Inc. - Exhibit 1009
Page 206 of 720

Filter Designer Settings

The **Filter Designer** preferences let you set key filter configuration and plot parameters.

FFT Length. Specifies the number of points used to calculate a filter's frequency response.

Display grid lines. Turns plot grid lines on (checked) or off (unchecked).

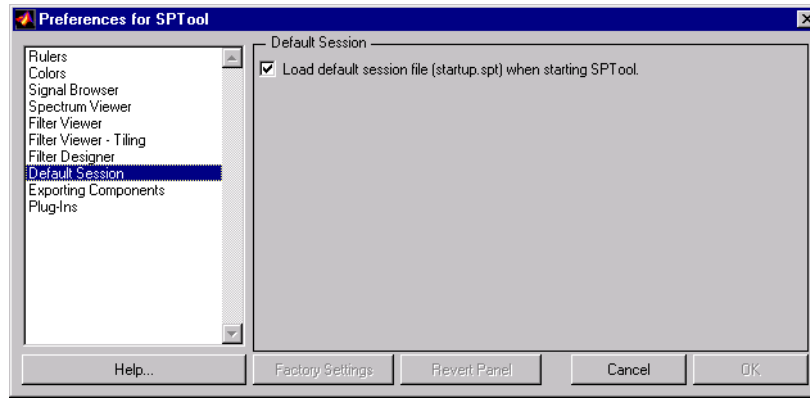
Auto Design – initial value. Specifies the default setting for the **Auto Design** check box in the Filter Designer. When the Filter Designer is first launched, the **Auto Design** check box will have the same setting (checked or unchecked) as the **Auto Design – initial value** check box here.

Stay in Zoom-mode After Zoom. Turns persistent zooming on and off, as described in "Zoom Controls" on page 5-31.

Meta Platforms, Inc. - Exhibit 1009
Page 207 of 720

Default Session Setting

The **Default Session** preference lets you specify whether or not to load a default session file when SPTool is launched.



Default Session. Enables (checked) or disables (unchecked) loading of a default session file, which must have the name `startup.spt`, when starting SPTool. An SPTool session file is a MAT-file (see *Using MATLAB*) with a `.spt` extension.

The **Default Session** preference only affects the session contents when you start SPTool. You can load a new session file after starting SPTool using the **Open Session** option in the SPTool **File** menu. You can also save a session file at any time using the **Save Session** option in the **File** menu. If you close SPTool without having recently saved a session file, you will be prompted to do so. (See "File Menu" on page 5-6.)

NOTE Changes that you make in the SPTool preference panels are saved in a separate MAT-file, which is reloaded the next time you open SPTool (see "Saving and Discarding Changes to Preferences Settings" on page 5-30). The **Default Session** preference does *not* affect these preference settings.

Meta Platforms, Inc. - Exhibit 1009
Page 208 of 720

Exporting Components Setting

The **Exporting Components** preference lets you specify whether or not to export filters created in SPTool as TF objects for the Control System Toolbox.

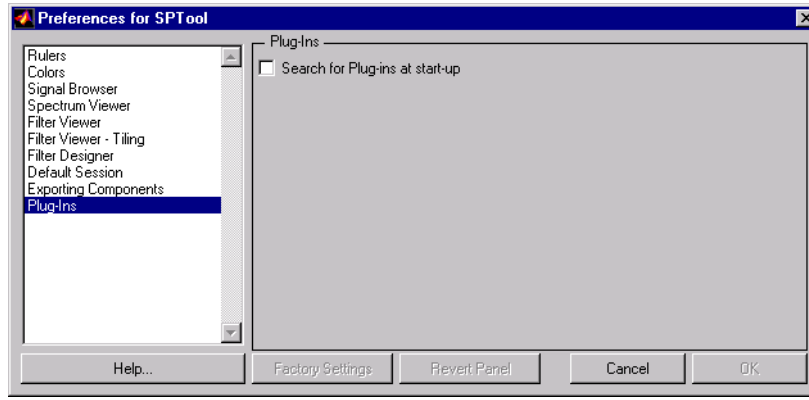
Export Filters as TF objects. Enables (checked) or disables (unchecked) exporting of SPTool filters as TF (transfer function) objects, for use in the Control System Toolbox (see the *Control System Toolbox User's Guide*).

NOTE This preference category only appears in the in SPTool **Preferences** dialog box if you have installed the Control System Toolbox.

Meta Platforms, Inc. - Exhibit 1009
Page 209 of 720

Plug-Ins Setting

The **Plug-Ins** preference lets you search for plug-ins when SPTool is started up.



Search for Plug-Ins at start-up. Enables (checked) or disables (unchecked) searching for installed plug-ins.

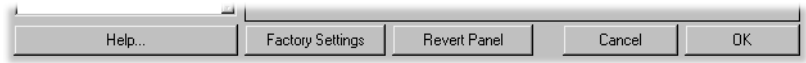
A *plug-in* is an extension to SPTool. Plug-ins include customized add-on panels and new buttons in the panels in SPTool, new spectral methods in the Spectrum Viewer, and new SPTool preferences. You can also plug one or more toolboxes into SPTool.

You need to use this setting only when you have installed extensions or have other toolboxes plugged into SPTool.

To use SPTool with extensions, check **Search for Plug-ins at start-up**, close SPTool, and restart it.

Saving and Discarding Changes to Preferences Settings

The buttons at the bottom of the **Preferences** panels let you save or discard any changes you have made, or return to the default settings.



Factory Settings. Restores the preferences in the current panel to their original settings; that is, the settings at the time the Signal Processing Toolbox was first installed.

Meta Platforms, Inc. - Exhibit 1009
Page 210 of 720

Revert Panel. Cancels changes in the current panel only. Settings in the current panel revert to the previous settings.

Cancel. Cancels changes in all preference categories and closes the **Preferences** dialog box. Settings in all panels revert to their previous state.

OK. Applies changes in all **Preferences** panels and closes the **Preferences** dialog box. Settings in all panels are saved in a MAT-file called `sigprefs.mat`.

If `sigprefs.mat` does not exist, either on the current MATLAB path or in the current directory, you are prompted for a location to save the file. The saved settings are used the next time you open SPTool.

Controls for Viewing and Measuring

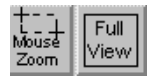
The GUI tools share common controls for viewing and measuring signals. These controls are described in this section. Not all tools use all of the viewing and measuring controls; specific details about the tools and procedures for viewing and measuring are described in the section on each tool.

Zoom Controls

The GUI tools share a common set of zoom control buttons. The Signal Browser and Spectrum Viewer use the same set of common zoom control buttons, which affect both on-screen and printed images.

The Filter Designer has one additional viewing button, the **Pass Band** button.

The Filter Viewer has a subset of the zoom control buttons.



Each button works the same way in every GUI tool in which it occurs.

In ordinary use, you press a button once to zoom in or out of the signal display.

Meta Platforms, Inc. - Exhibit 1009
Page 211 of 720

Zoom In-X, Zoom Out-X, Zoom In-Y, and Zoom Out-Y. Click once to perform one zoom operation (in or out) on the *x*- or *y*-axis. Each zoom operation changes the axes limits by a factor of two on the specified axis, about the center of the displayed signal. You can press repeatedly on one or more buttons to continue to change the scale in one or both axes.

When you zoom in the *x*-axis (horizontal scaling), the *y* limits (vertical scaling) of the main axes are not changed. Similarly, when you zoom in the *y*-axis, the *x* limits of the main axes are not changed.

Full View. Click once to restore the displayed signal to its full range in both axes.

Mouse Zoom. Click once to activate zoom mode. The cursor changes to a crosshair. You can either zoom in without specifying a zoom window, or you can use a zoom rectangle to select a specific zoom window. In either case, the *x*- and *y*-axis are automatically adjusted to display the selected signal:

- To zoom in without specifying a zoom window, click on the plot. The position of the crosshair is the center of a zoom operation that halves both the *x*- and *y*-axis limits.
- To use a zoom rectangle, click where you want the rectangle to begin, drag the mouse diagonally to where you want it to end, and release the mouse button.
- To get out of mouse zoom mode without zooming in or out, press the **Mouse Zoom** button again.

Zoom Persistence. Mouse zooming can either be *one-time* or *persistent*:

- One-time zooming is activated when you press the **Mouse Zoom** button. It automatically turns itself off after you click in the display area and the zoom operation occurs. This is the default for all the tools.
- Persistent zooming is also activated by pressing the **Mouse Zoom** button. It does not turn off after you click in the display area and a zoom operation occurs; you can continue to click and zoom without resetting the **Mouse Zoom** button.

You can change whether zooming is one-time or persistent by selecting **Preferences...** from the **File** menu and toggling **Stay in Zoom-mode After Zoom** in the **Preferences** panels for the Signal Browser, the Spectrum Viewer, the Filter Designer, and the Filter Viewer.

Meta Platforms, Inc. - Exhibit 1009
Page 212 of 720

When **Stay in Zoom-mode After Zoom** is selected, zooming is persistent. To turn off mouse zooming when **Stay in Zoom-mode After Zoom** is selected, press the **Mouse Zoom** button.

Passband Zoom (Filter Designer). Click once to zoom in on the passband of the response.

Both the *x*- and *y*-limits of the main axes are changed so that the passband fills the main axes.

There is no stopband zoom button. To zoom the stopband, use standard **Mouse Zoom**, centering the crosshair on the area of the stopband you want to view. If you are in passband zoom, first press **Full View** to return to the standard view.

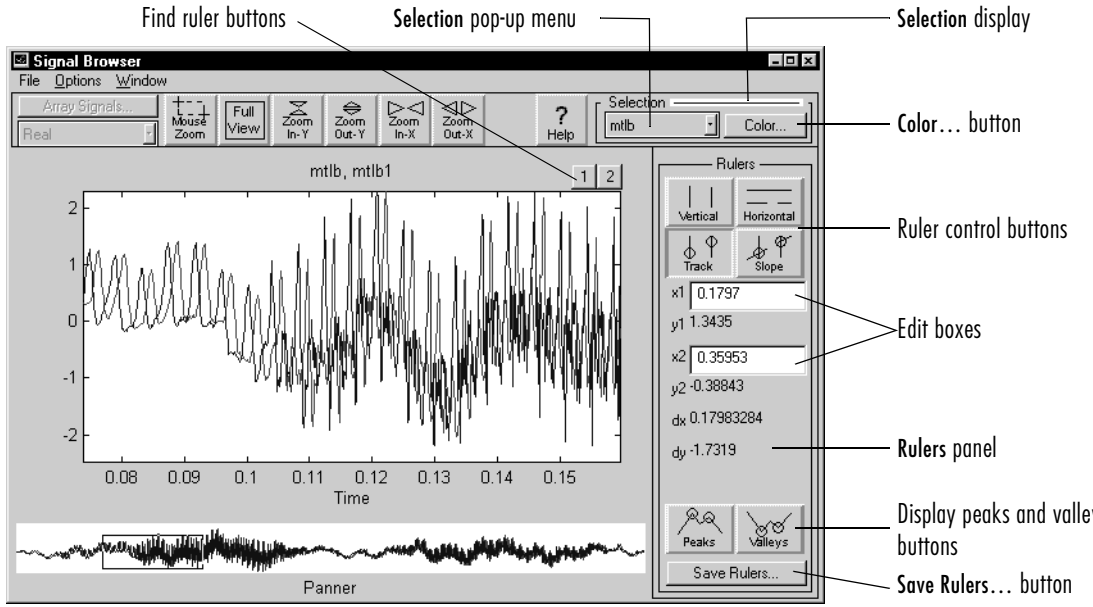
Ruler Controls

The Signal Browser, Filter Viewer, and Spectrum Viewer share a common set of ruler controls. Use the rulers to make measurements on the signals or spectra in the main axes (display) area. The ruler controls give you a variety of ways to read and control the values of the rulers in the main axes. With the rulers you can measure such information as the vertical and horizontal distance between features in a signal or spectrum, the dimensions of peaks and valleys, and slope information.

In the following discussion, the Signal Browser is shown. The ruler controls include the **Selection** controls at the top right of the window and the buttons and edit boxes in the **Rulers** panel. The controls in the Filter Viewer and Spectrum Viewer work the same way. The controls in the Signal Browser and the Spectrum Viewer also affect images printed from these tools.

In the Filter Viewer, the rulers appear on only one subplot at a time. You can choose which subplot the rulers appear on by selecting the subplot from the pop-up menu at the top of the **Rulers** panel, or by clicking on a line in the plot that you want to measure. If a subplot is not currently visible when you select its name from the pop-up menu, the Filter Viewer creates the subplot and places the rulers in it.

Meta Platforms, Inc. - Exhibit 1009
Page 213 of 720



Selecting a Line to Measure. When there is only one signal displayed, the displayed signal is automatically selected and is measured when you use the rulers. When there is more than one signal displayed, only one signal (line) may be selected and measured at a time.

When a signal (line) is selected, you can use the ruler controls (**Vertical**, **Horizontal**, **Track**, or **Slope**) and the **Peaks** and/or **Valleys** controls on the selected line. The label of the selected signal (line) is displayed in the **Selection** pop-up menu.

There are two ways to select a signal (line):

- Click on the **Selection** pop-up menu and drag to select the line to measure.

  All signals that are currently selected in SPTool are listed. Vector signals in the Signal Browser, spectra in the Spectrum Viewer, and filters in the Filter Viewer are listed as single variables; in the Signal Browser, each column of a two-dimensional signal matrix is listed as a separate variable.
- Move the mouse pointer over any point in the line you want to select and click on it.

Meta Platforms, Inc. - Exhibit 1009
Page 214 of 720

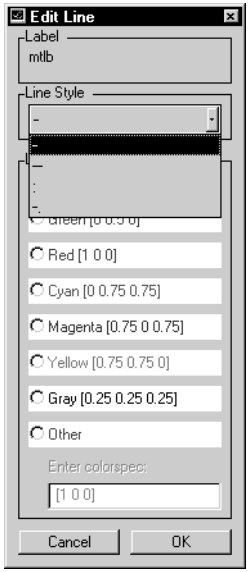
The label of the signal, including the column number if the line is one column of a matrix, is displayed in the **Selection** pop-up menu.

The line selection display changes to the color and pattern of the selected signal, spectrum, or filter.

Line Selection Pop-Up Menu. Use to select a line (vector signal, array column, filter, or spectrum) to measure.

Click the **Selection** pop-up menu and drag to select the line.

Line Selection Display. The line color and style of the selected signal are displayed.



Color... Button. Use to edit the line style or display color of the selected line.

Click on the **Color...** button at the top right of the window to display the **Edit Line** pop-up menu, which is shown on the left. The label of the selected line is displayed in the **Label** field.

- Click the **Line Style** pop-up menu and drag to select a line style, as shown on the left.
- Click a radio button to select a color. If you select **Other**, you can type a color value in the **Enter colorspec** box; the color value can be a string (e.g., `'r'`) or an RGB triple (e.g., `[1 0 0]`).
- Click **OK** to apply the line style and color you selected.

Find Ruler Buttons. Use the find ruler buttons to bring one or both rulers into the viewing area of the main axes. When both rulers are within the signal display (main axes) area, the **Find Ruler** buttons, at the top right of the main axes area, are not displayed.

If the rulers are not within the signal display area, both **Find Ruler** buttons are displayed, as shown on the right.

If one ruler is within the signal display area, the button for the other ruler is displayed, as shown on the right.

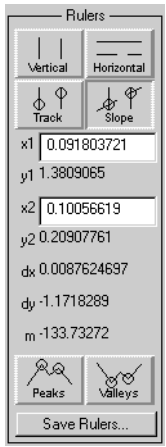
Click a **Find Ruler** button to bring the specified ruler into the display area.

When a ruler is visible, you can click on it and drag it to make a measurement on the selected signal. See “Making Signal Measurements” on page 5-37 for details on manipulating the rulers and measuring the selected signal.

Meta Platforms, Inc. - Exhibit 1009
Page 215 of 720

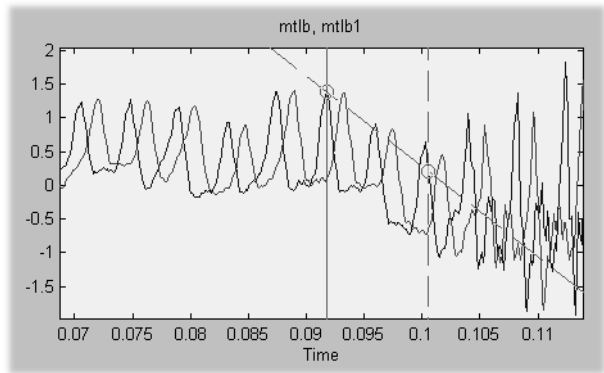
Ruler Control Buttons. Use the ruler control buttons to select the type of measurement you want to make: **Vertical**, **Horizontal**, **Track**, or **Slope**. The default setting is **Track**.

Click a ruler control button to select it. The **Rulers** panel changes depending on which ruler control is selected. See "Making Signal Measurements" on page 5-37 for details on the four kinds of measurements you can make and the parameters for each one.



Rulers Panel and Edit Boxes. The **Rulers** panel changes depending on which ruler control is selected: **Vertical**, **Horizontal**, **Track**, or **Slope**. It shows the parameters for the selected ruler control. Depending on which ruler control is selected, the following fields are displayed: **x1**, **y1**, **x2**, **y2**, **dx**, **dy**, **m**. The picture on the left shows the **Rulers** panel when **Slope** is selected.

When you press a ruler control button, rulers are displayed superimposed on the signal(s) in the main axes display area. The rulers are either vertical (for **Vertical**, **Track**, and **Slope**) or horizontal (for **Horizontal**). For **Track** and **Slope**, ruler markers are also displayed. The rulers and ruler markers are associated with the currently selected signal. The following picture shows the rulers and ruler markers that are displayed when **Slope** is selected.



To position a ruler, you can click and drag on it. When you drag a ruler, the parameters in the **Rulers** panel change to reflect the measurements on the selected signal.

You can also position a ruler by specifying parameters in the edit boxes in the **Rulers** panel. The parameters are either the **x1** and **x2** values or the **y1** and **y2** values, depending on which ruler control is selected.

Meta Platforms, Inc. - Exhibit 1009
Page 216 of 720

Type the value or variable for the ruler parameter in the **x1** and **x2** boxes or the **y1** and **y2** boxes. See "Making Signal Measurements" on page 5-37 for details on manipulating the rulers and the parameters you can measure with each one.

Peaks and Valleys. Use these buttons to show or hide the local maxima and/or local minima of the currently selected signal, filter response, or spectrum. Only peaks or valleys, or both peaks and valleys may be displayed.

- Click **Peaks** to toggle showing (down) or hiding (up) the maxima of the signal.
- Click **Valleys** to toggle showing (down) or hiding (up) the minima of the signal.

In track and slope mode (see "Making Signal Measurements" on page 5-37), the rulers are constrained to the peaks or valleys. In horizontal and vertical mode, the peaks and valleys are only visual and do not affect the behavior of the rulers.

Save Rulers... Button. Once you've set up and made a certain set of measurements, you may find it useful to save them for future reference. Use the **Save Rulers...** button to save a structure in the MATLAB workspace with the fields `x1`, `y1`, `x2`, `y2`, `dx`, `dy`, `m`, `peaks`, and `valleys`. Undefined values are set to `NaN`.

1. Click **Save Rulers...** to save the current measurements as a variable in the workspace.

   The Save Rulers dialog box is displayed.

2. Type a variable name in the edit field and press **OK**.

Making Signal Measurements

Use the rulers to make measurements on a selected line, which is a vector or a column of a matrix in the Signal Browser, a filter response in the Filter Viewer, or a spectrum in the Spectrum Viewer. To make a measurement:

1. Select a line as described in "Selecting a Line to Measure" on page 5-34.

2. Apply a ruler to the display as described in "Ruler Control Buttons" on page 5-36.

Meta Platforms, Inc. - Exhibit 1009
Page 217 of 720

3 Position a ruler where you want it in the main axes area by clicking and dragging it:

   a Move the mouse over the ruler (1 or 2) that you want to drag.

      The hand cursor is displayed when you're over a ruler, with the ruler number inside it:

   b Click and drag the ruler to where you want it on the signal.

      Depending on which ruler control is selected, you can drag the ruler to the right and left (**Vertical**, **Track**, and **Slope**) or up and down (**Horizontal**).

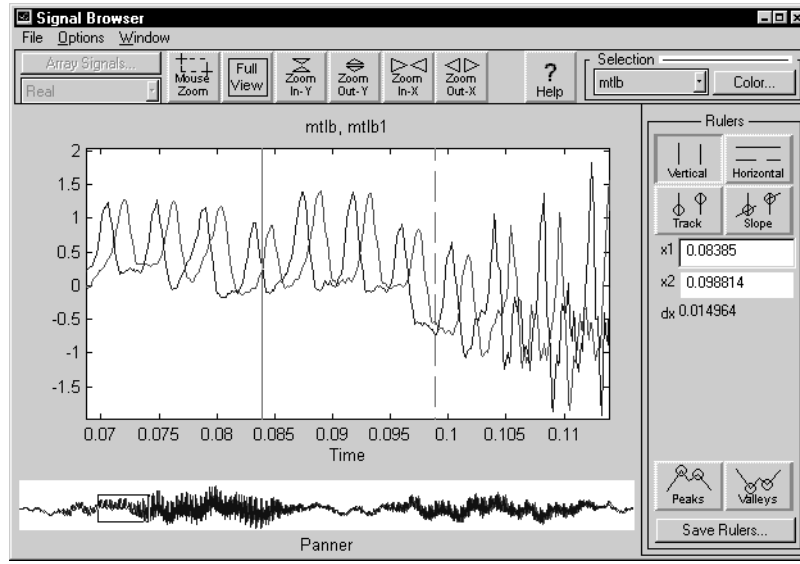
   As you drag a ruler, the **Rulers** panel shows the current position of both rulers. Depending on which ruler control is selected, the following fields are displayed: **x1**, **y1**, **x2**, **y2**, **dx**, **dy**, **m**. These fields will also be displayed in printouts from the Signal Browser, unless you suppress them. See "Printing Signal Data" on page 5-54 and "Printing Spectrum Data" on page 5-107 for details on printing from these tools.

You can also position a ruler by typing its **x1** and **x2** or **y1** and **y2** values in the **Rulers** panel, as described on page 5-36.

Ruler Controls: Vertical. There are two vertical rulers, called ruler 1 and ruler 2. When vertical rulers are in use, the measurements displayed in the **Rulers**

Meta Platforms, Inc. - Exhibit 1009
Page 218 of 720

panel are **x1** (the position of ruler 1 on the *x*-axis), **x2** (the position of ruler 2 on the *x*-axis), and **dx** (the value of **x2-x1**).



Press **Vertical** to put the rulers in vertical mode.

In vertical mode, you may change the *x*-values of the rulers (that is, their horizontal position). As the **x1** and **x2** values change, the value of **dx** changes automatically.

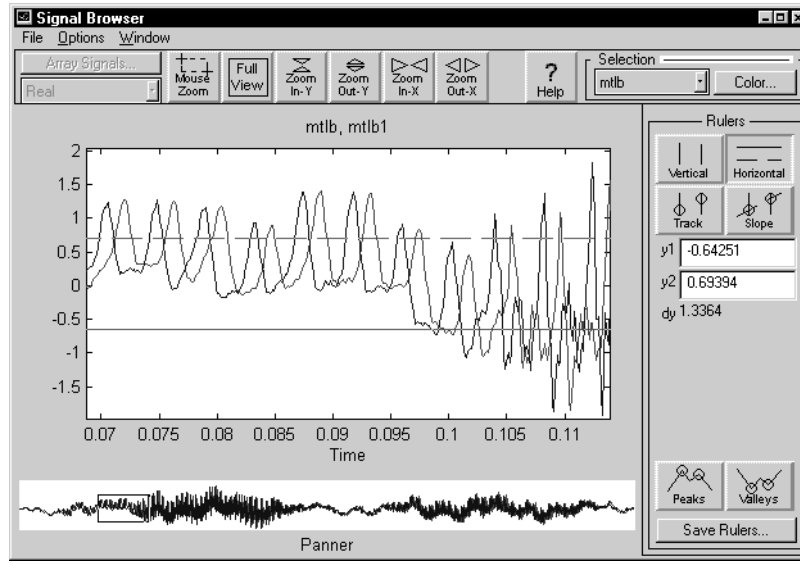
Change the **x1** and **x2** values in one of two ways:

- Drag the rulers to the left and the right with the mouse
- Enter their values in the **x1** and **x2** edit boxes in the **Rulers** panel

Ruler Controls: Horizontal. There are two horizontal rulers, called ruler 1 and ruler 2. When horizontal rulers are in use, the measurements displayed in the

Meta Platforms, Inc. - Exhibit 1009
Page 219 of 720

Rulers panel are **y1** (the position of ruler 1 on the *y*-axis), **y2** (the position of ruler 2 on the *y*-axis), and **dy** (the value of **y2-y1**).



Press **Horizontal** to put the rulers in horizontal mode.

In horizontal mode, you may change the *y*-values of the rulers (that is, their vertical position). As the **y1** and **y2** values change, the value of **dy** changes automatically.

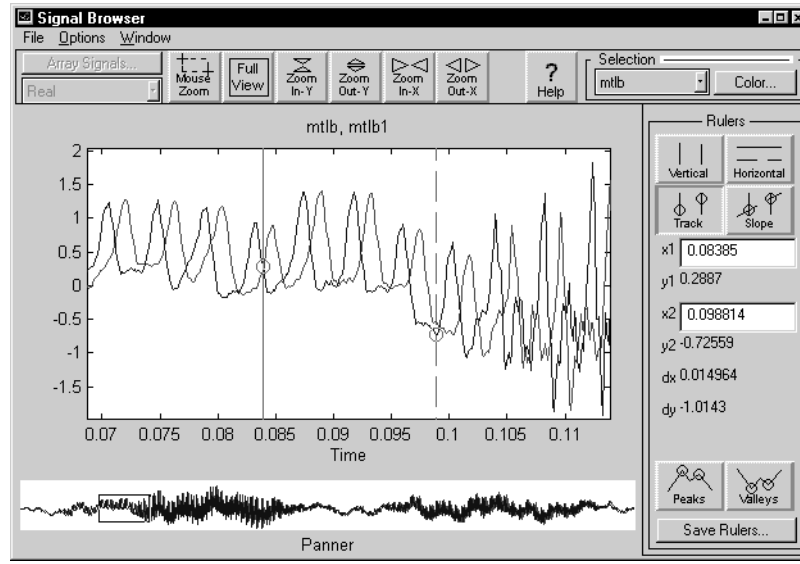
Change the **y1** and **y2** values in one of two ways:

- Dragging the rulers up and down with the mouse
- Entering their values in the **y1** and **y2** edit boxes in the **Rulers** panel

Ruler Controls: Track. There are two vertical rulers, called ruler 1 and ruler 2, with a marker on each that shows the *y*-values of the signal at the *x*-values of the rulers. When track rulers are in use, the measurements displayed in the **Rulers** panel are **x1** (the position of ruler 1 on the *x*-axis), **y1** (the position of

Meta Platforms, Inc. - Exhibit 1009
Page 220 of 720

ruler 1 on the *y*-axis), **x2** (the position of ruler 2 on the *x*-axis), **y2** (the position of ruler 2 on the *y*-axis), **dx** (the value of **x2-x1**), and **dy** (the value of **y2-y1**).



Press **Track** to put the rulers in track mode.

You can change the track marker in the **Rulers** panel from the **Preferences** dialog box; see "Ruler Settings" on page 5-21.

In track mode, you may change the *x*-values of the rulers (that is, their horizontal position). As the **x1** and **x2** values change, the values of **y1**, **y2**, **dx**, and **dy** change automatically.

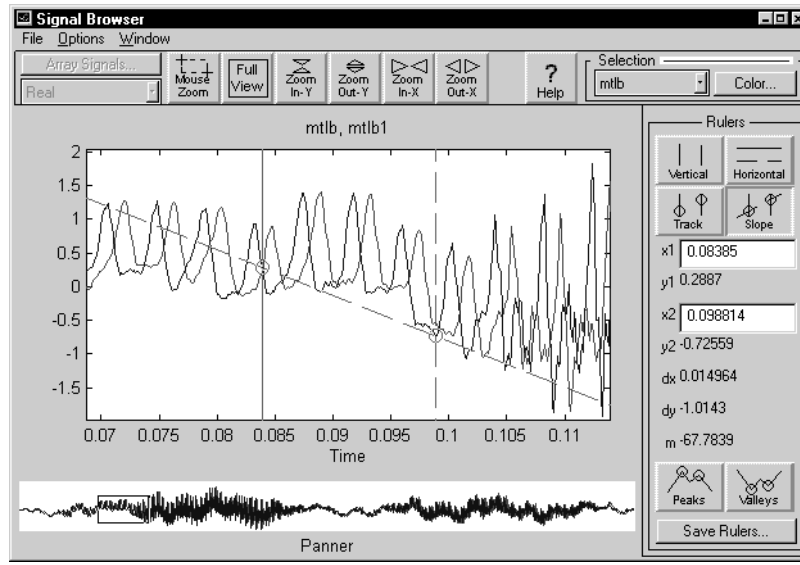
Change the **x1** and **x2** values in one of two ways:

- Dragging the rulers to the left and the right with the mouse
- Entering their values in the **x1** and **x2** edit boxes in the **Rulers** panel

Ruler Controls: Slope. There are two vertical rulers, called ruler 1 and ruler 2, with the slope line passing through the *y*-axis intersections of the two vertical rulers and the signal. The rulers also track the signal with markers on each ruler that shows the *y*-values of the signal at the *x*-values of the rulers. The line connecting (**x1**, **y1**) and (**x2**, **y2**) is included in the main plot, so you can approximate derivatives and slopes of the signal.

Meta Platforms, Inc. - Exhibit 1009
Page 221 of 720

When slope rulers are in use, the measurements displayed in the **Rulers** panel are **x1** (the position of ruler 1 on the x-axis), **y1** (the position of ruler 1 on the *y*-axis), **x2** (the position of ruler 2 on the x-axis), **y2** (the position of ruler 2 on the *y*-axis), **dx** (the value of **x2 - x1**), **dy** (the value of **y2 - y1**), and **m** (equal to **dy/dx**, the slope of the line between **x1** and **x2**).



Press **Slope** to put the rulers in slope mode.

In slope mode, you may change the *x*-values of the rulers (that is, their horizontal position). As the **x1** and **x2** values change, the values of **dy** and **m** change automatically.

Change the **x1** and **x2** values in one of two ways:

- Dragging the rulers to the left and the right with the mouse
- Entering their values in the **x1** and **x2** edit boxes in the **Rulers** panel

Meta Platforms, Inc. - Exhibit 1009
Page 222 of 720

Using the Signal Browser: Interactive Signal Analysis

The Signal Browser tool is an interactive signal exploration environment. It provides a graphical view of the signal object(s) currently selected in the **Signals** column of SPTool.

Using the Signal Browser you can:

- View and compare vector or array signals
- Zoom in on a range of signal data to examine it more closely
- Measure a variety of characteristics of signal data
- Play signal data on audio hardware
- Print signal data

Opening the Signal Browser

To open or activate the Signal Browser from SPTool:

1. Click on one or more signals in the **Signals** list of SPTool.
2. Press **View** in the **Signals** panel of SPTool.

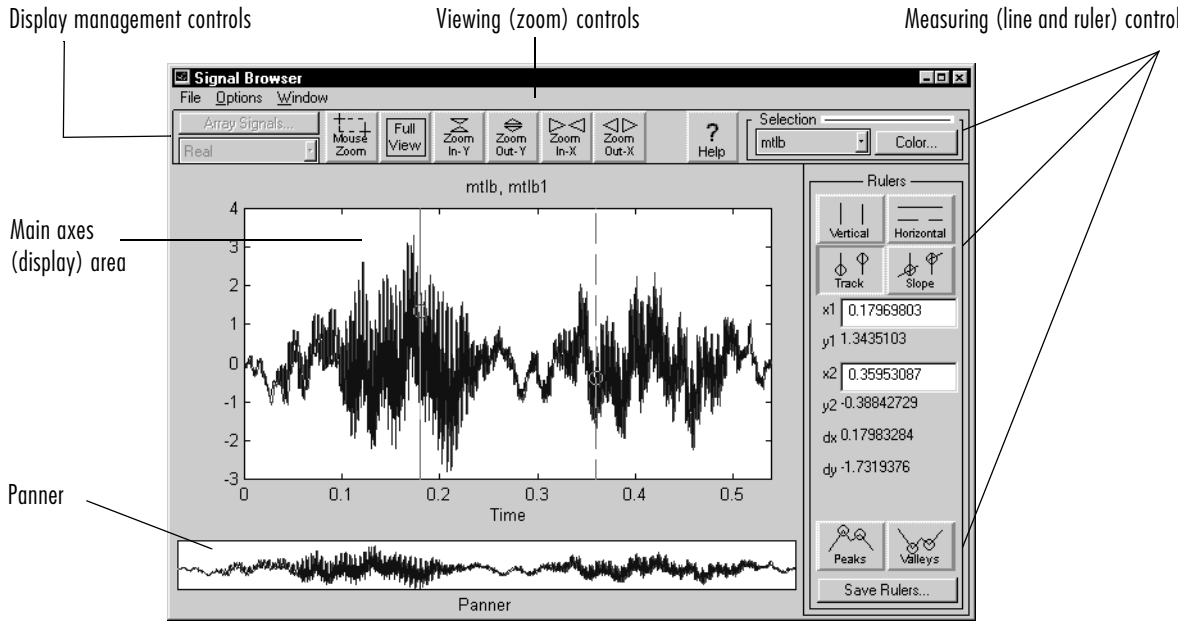
   The Signal Browser is activated and the selected signal(s) are loaded into the Signal Browser and displayed.

Meta Platforms, Inc. - Exhibit 1009
Page 223 of 720

Basic Signal Browser Functions

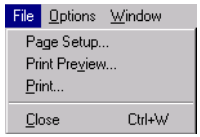
The Signal Browser has the following components:

- A main axes (display) area for viewing signals graphically
- Display management controls: **Array Signals...** and the complex signal display pop-up menu
- Zoom controls for getting a closer look at signal features
- Rulers and line display controls for making signal measurements and comparisons
- A panner for seeing what part of the signal is currently being displayed and for quickly moving the view to other features of the signal
- Menu options for printing signal data
- A menu option for playing a selected signal through audio equipment

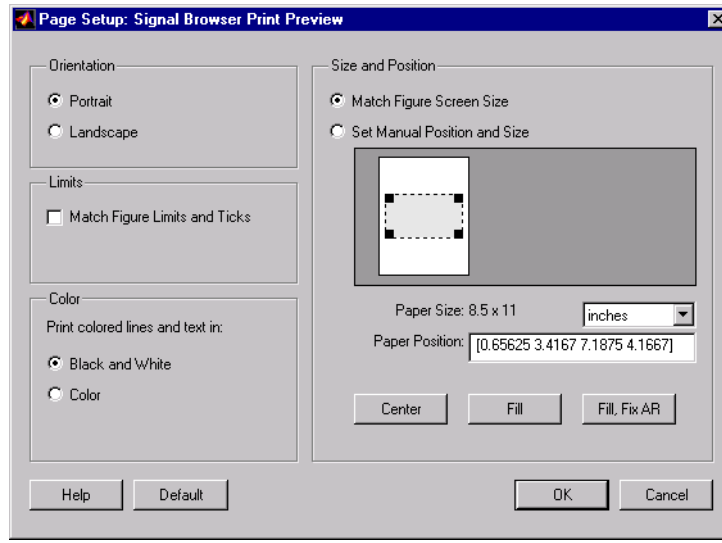


Meta Platforms, Inc. - Exhibit 1009
Page 224 of 720

Menus



File Menu. Use **Page Setup...** from the **File** menu to open the standard MATLAB **Page Setup** dialog box, in which you can set the orientation, size and position, limits, and color of your printout.



See *Using MATLAB Graphics* for more information about the **Page Setup** dialog box.

Use **Print Preview...** from the **File** menu to open a MATLAB figure window with a preview of the information from the Signal Browser window that will appear in your printout. See "Signal Browser Settings" on page 5-23 and "Printing Signal Data" on page 5-54 for details about printing from the Signal Browser.

Use **Print...** from the **File** menu to open the standard print dialog box, from which you can print your signal data.

Use **Close** from the **File** menu to close the Signal Browser. All signal selection and ruler information will be lost. (You may want to save ruler information to the workspace using the **Save Rulers...** button, as described on page 5-37.) Settings you changed and saved using the **Preferences** dialog box in SPTool are saved and used the next time you open a Signal Browser.

Meta Platforms, Inc. - Exhibit 1009
Page 225 of 720

Options Menu. Use **Play** from the **Options** menu to play the selected signal.

Play works only when you have sound capabilities on your computer. If your computer does not have sound capabilities, this menu choice does nothing.

The entire selected signal is played at either Fs (the sampling frequency of the signal) or at the default platform sampling frequency if Fs is less than 25 Hz. The real part and the imaginary part of a complex signal are played in separate channels.

Window Menu. Use the **Window** menu to select a currently open MATLAB window.

Zoom Controls

The available zoom controls in the Signal Browser are **Mouse Zoom**, **Full View**, **Zoom In-Y**, **Zoom Out-Y**, **Zoom In-X**, and **Zoom Out-X**. See "Zoom Controls" on page 5-31 for details on using the zoom controls in the Signal Browser.

Zoom persistence is off by default in the Signal Browser; use the Signal Browser settings panel in the **Preferences** dialog box in SPTool to toggle zoom persistence on and off. See "Signal Browser Settings" on page 5-23.

Ruler and Line Display Controls

Using the rulers and line display controls, you can measure a variety of characteristics of signals in the Signal Browser. See "Ruler Controls" on page 5-33 for details on using rulers and modifying line displays in the Signal Browser.

The rulers are displayed by default in the Signal Browser; you can turn off the ruler display in the **Signal Browser** settings panel in the **Preferences** dialog box. See "Signal Browser Settings" on page 5-23.

Help Button

To use context-sensitive help, press the **Help** button. The mouse pointer becomes an arrow with a question mark symbol. You can then click on anything in the Signal Browser, including menu items, to find out what it is and how to use it.

Meta Platforms, Inc. - Exhibit 1009
Page 226 of 720

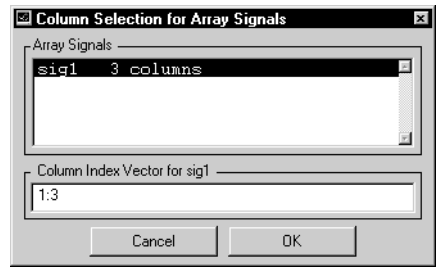
Display Management Controls

Array Signals... Button. Use this to enter a column index vector for a selected array signal. All array signals start out with only the first column displayed.

The **Array Signals...** button is enabled when at least one array signal is selected in SPTool:

1 Press **Array Signals...**

The **Column Selection for Array Signals** dialog box is displayed.

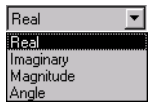


All array signals that are selected in SPTool are shown in the list.

2 Select a signal from the list.

3 Type a column index vector for the selected signal.

Valid index vectors are of the form `1` or `1:3` or `[1 3 5]`.



Complex Signal Display. Use to specify whether the Signal Browser plots the real part, the imaginary part, the magnitude, or the angle of a complex signal.

This menu is enabled when at least one of the signal variables selected in SPTool is complex. The Complex Display mode affects all of the variables in the current selection, even those that are strictly real.

Click and drag to select the plotting mode.

Main Axes Display Area

The **Signals** list in SPTool shows all signals in the current SPTool session. One or more signals may be selected. The signal data of all selected signals are displayed graphically in the main axes display area of the Signal Browser.

Meta Platforms, Inc. - Exhibit 1009
Page 227 of 720

When there is only one signal displayed, its properties are reflected in the display management controls and its measurements are displayed in the ruler display panel. When more than one signal is displayed, select the line you want to focus on.

When a signal is selected, you can use the ruler controls on the selected line (see "Making Signal Measurements" on page 5-37), you can choose how to display the signal (see "Display Management Controls" on page 5-47), and you can play the signal (see "Options Menu" on page 5-46). The label of the selected signal (line) is displayed in the **Selection** pop-up menu.

Use one of three ways to select a signal (line) in the Signal Browser:

- Click on the **Selection** pop-up menu and drag to select the line to measure.
- Move the mouse pointer over any point in the line in the main axes display and click on it.
- Move the mouse pointer over any point in the line in the panner and click on it.

See "Selecting a Line to Measure" on page 5-34 for details.

Axes Labels. By default, the *x*-axis in the Signal Browser is labeled `Time`. You can change the *x*-axis label and add a *y*-axis label using the **Signal Browser** settings panel in the **Preferences** dialog box in SPTool. See "Signal Browser Settings" on page 5-23.

Click-and-Drag Panning. You can use the mouse to pan around the main axes display; click on a line in the main axes, hold down the mouse button, and drag the mouse.

Click-and-drag panning is not enabled in mouse zoom mode.

Panner

The panner gives a panoramic view of the signal(s) displayed in the main axes. The panner always displays the entire signal sample. When you zoom in on the main axes, a patch in the panner shows the section of the plot that is currently in view in the main axes. Click-and-drag the patch in the panner window to pan dynamically across the signal data in the main axes.

You can also select a line by clicking on it in the panner; the selected line is highlighted in both the panner and in the main axes display area.

Meta Platforms, Inc. - Exhibit 1009
Page 228 of 720

See “Panner Display” on page 5-52 for more details.

The panner is displayed by default in the Signal Browser; you can turn off the panner in the Signal Browser settings panel in the **Preferences** dialog box in SPTool. See “Signal Browser Settings” on page 5-23.

Making Signal Measurements

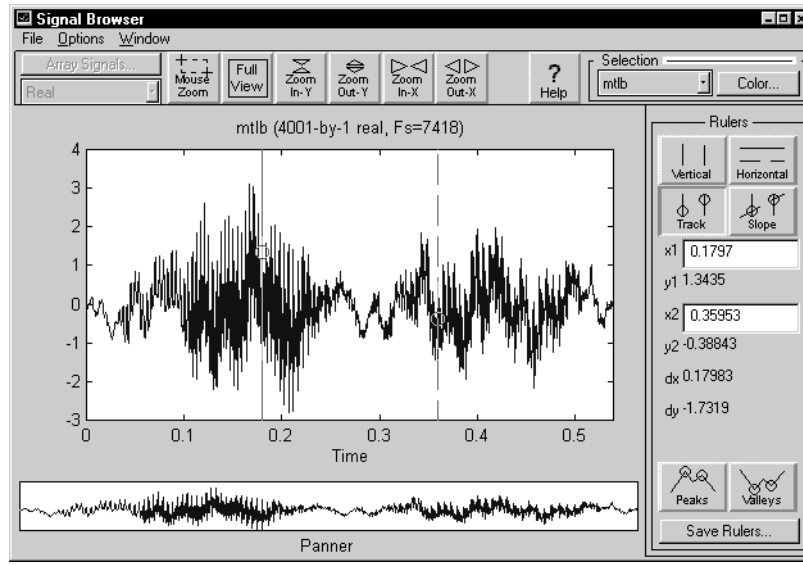
Use the rulers to make a variety of measurements on the selected signal. See “Making Signal Measurements” on page 5-37 for details.

Viewing and Exploring Signals

You can open or activate the Signal Browser in SPTool by selecting one or more signals and pressing **View** in the Signal panel. The selected signals are loaded into the Signal Browser. See “Viewing a Signal” on page 5-17 for details.

Selecting and Displaying a Signal

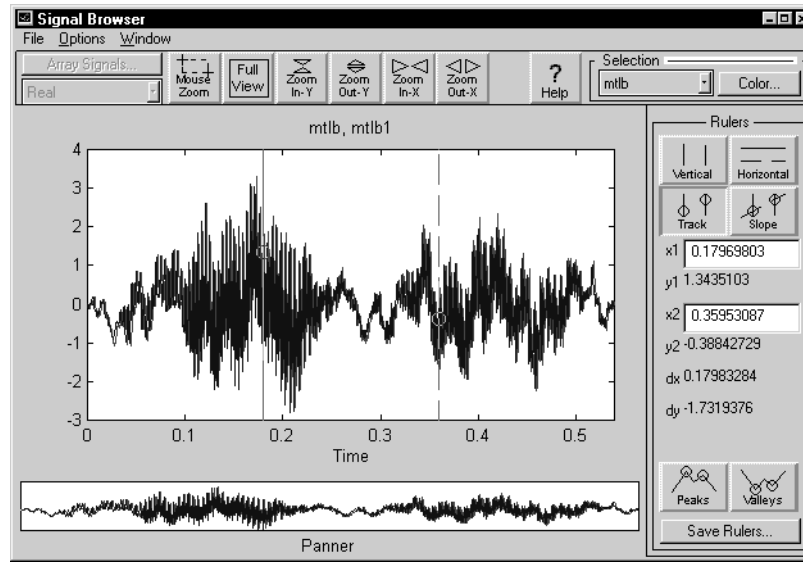
When the Signal Browser is activated, all selected signals are displayed in the main axes display area and in the panner. The data of the selected signals are plotted against an equally spaced time vector in both the main axes display and the panner.



Meta Platforms, Inc. - Exhibit 1009
Page 229 of 720

When the variable is a vector, one signal is displayed, as in the example above. It is automatically selected and its name, size, type, and sampling frequency are displayed above the main axes display; the name is also highlighted in the **Selection** pop-up menu.

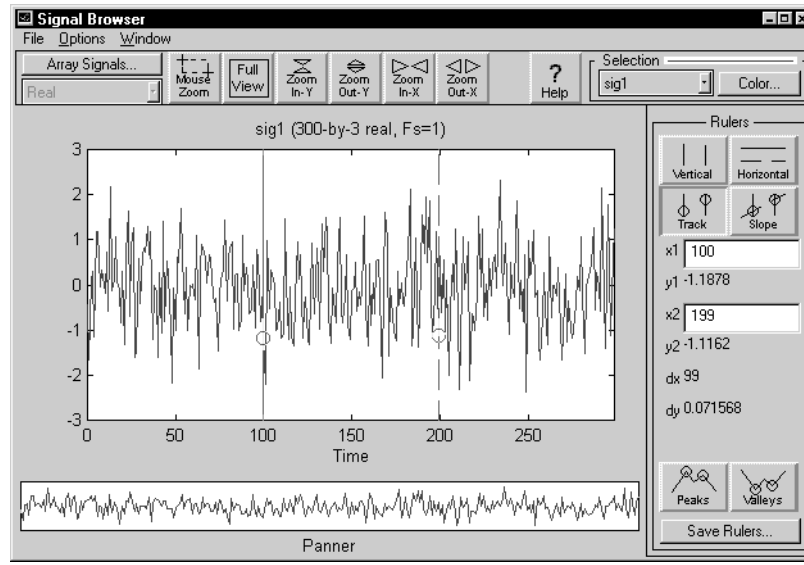
When more than one signal is selected, each signal is displayed in a different color in both the main axes display and the panner.



The names of all signals are displayed above the main axes display. The first signal in the list is automatically selected in both the main axes display and the panner, its name is highlighted in the **Selection** pop-up menu, and its color is shown in the **Selection** display.

Meta Platforms, Inc. - Exhibit 1009
Page 230 of 720

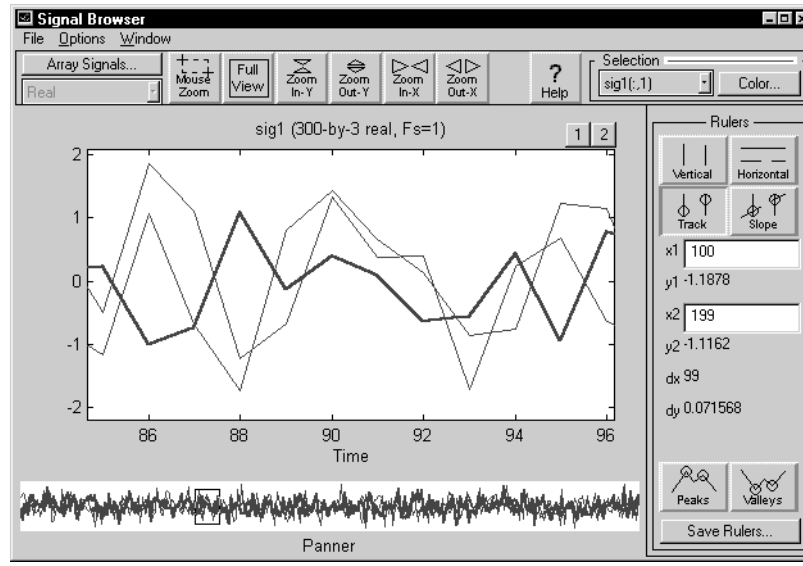
When the signal is an array, only the first column is initially displayed in both the main axes and the panner.



To display a different array column, or more than one column of the array, press the **Array Signals...** button and specify the columns to be displayed (see "Array Signals... Button" on page 5-47). All displayed columns of an array are shown in the same color; the selected column is emphasized with a heavier line

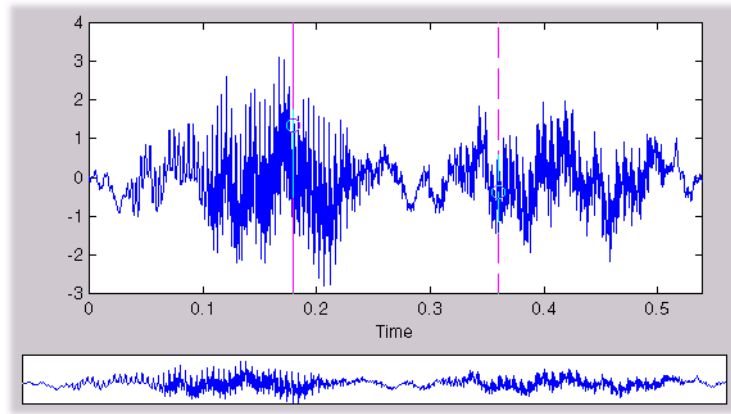
Meta Platforms, Inc. - Exhibit 1009
Page 231 of 720

in both the main axes and the panner, and its label is displayed in the **Selection** pop-up menu.



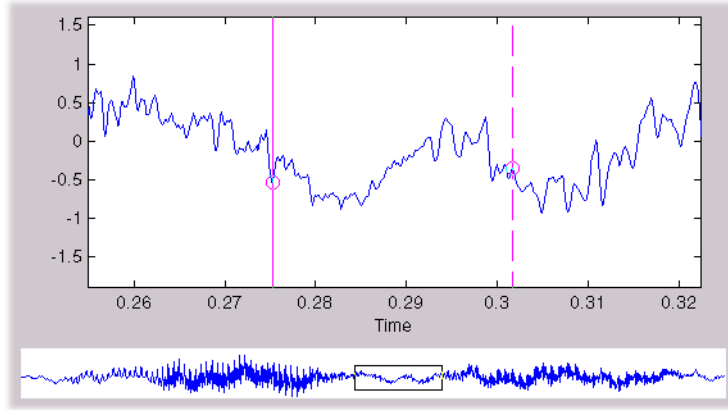
Panner Display

The panner displays the entire signal sample at all times.



Meta Platforms, Inc. - Exhibit 1009
Page 232 of 720

When the signal in the main axes is zoomed, the part of the signal that is shown in the main axes is shown in the panner with a window around it.



Each time you zoom, the panner is updated to frame the region of data displayed in the main axes.

Click-and-drag on the panner window to move it. As the window moves over the signal in the panner, the signal in the main axes area is panned.

Manipulating Displays

Changing Signal Displays. The signals are displayed in the default line colors and default line styles. You can change the defaults using the **Color Order** and **Line Style Order** fields in the **Colors** settings panel (see "Color Settings" on page 5-22).

Changing the Sample Interval. You can change the sample interval by selecting **Sampling Frequency...** from the **Edit** menu in SPTool. See "Editing Data Objects in SPTool" on page 5-15.

Displaying Complex Signals. You can change how complex numbers are displayed by selecting **Real**, **Imaginary**, **Magnitude**, or **Angle** from the pop-up menu. See "Array Signals... Button" on page 5-47.

Changing Signal Browser Displays. Using the **Signal Browser** settings panel in the **Preferences** dialog box in SPTool, you can set optional *x*-axis and *y*-axis labels,

Meta Platforms, Inc. - Exhibit 1009
Page 233 of 720

enable and disable the display of the rulers and the panner, and toggle zoom persistence. See “Signal Browser Settings” on page 5-23.

Working with Signals

Once a signal is displayed, you can browse it in a variety of ways:

- You can zoom in on a specific area of the display (see “Zoom Controls” on page 5-31).
- You can mark off a segment of the display with the rulers (see “Ruler Controls” on page 5-33) and save ruler settings (see “Save Rulers... Button” on page 5-37).
- You can select a segment of the display with the panner (see “Panner Display” on page 5-52).
- You can make certain measurements on the displayed signals (see “Making Signal Measurements” on page 5-37).
- When there is more than one signal in the display, you can select which one you want to measure (see “Selecting a Line to Measure” on page 5-34).

You can use the other GUI tools to manipulate signals in a variety of ways:

- You can interactively design and analyze filters to be applied to signals (see “Using the Filter Designer: Interactive Filter Design” on page 5-59 and “Using the Filter Viewer: Interactive Filter Analysis” on page 5-84).
- You can create a spectrum for a signal and interactively analyze its spectral density with a variety of estimation methods (see “Using the Spectrum Viewer: Interactive PSD Analysis” on page 5-97).

You access the Filter Designer, Filter Viewer, and Spectrum Viewer tools from SPTool. You can access SPTool from the Signal Browser in one of two ways:

- Click on an active SPTool window
- Activate SPTool using the **Window** menu in the Signal Browser

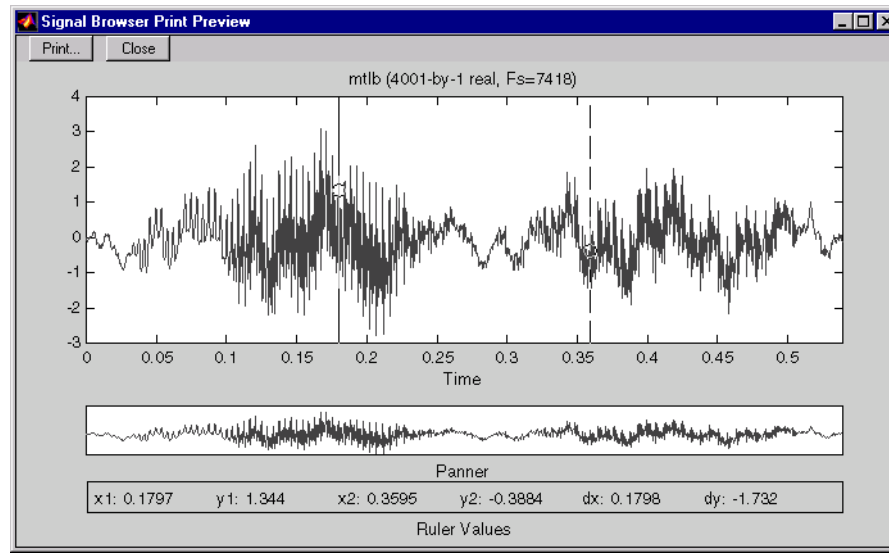
Printing Signal Data

You can print all of the signal data that you can display and manipulate in the Signal Browser. Your printout will contain up to three components:

Meta Platforms, Inc. - Exhibit 1009
Page 234 of 720

- The plots of one or more signals, as displayed in the main axes area
- The panner display of the signal(s) displayed in the main axes area
- The ruler measurements of the selected signal

You can preview your printout by choosing **Print Preview...** from the **File** menu in the Signal Browser window. This opens the **Signal Browser Print Preview** window, shown below with default settings for signal `mtlb`.



The **Close** button on the **Signal Browser Print Preview** window closes the print preview window and returns you to the Signal Browser.

The **Print...** button on the **Signal Browser Print Preview** window opens the standard print dialog box, from which you can print the contents of your print preview window.

If you choose to print from the print dialog, both that dialog and the **Signal Browser Print Preview** window will close, and the contents of the print preview will print.

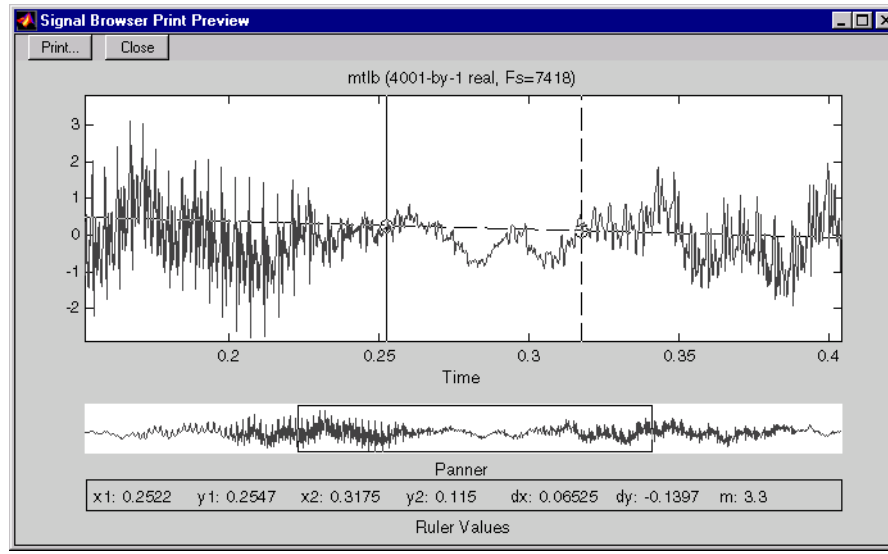
If you choose not to print, the print dialog and the **Signal Browser Print Preview** window will close, and you will return to the Signal Browser.

Any changes you make, before opening the print preview window, that affect the display of data in the Signal Browser (see “Viewing and Exploring Signals”

Meta Platforms, Inc. - Exhibit 1009
Page 235 of 720

on page 5-49) will affect the display of data in the **Signal Browser Print Preview** window in exactly the same way. For example, in the print preview shown below:

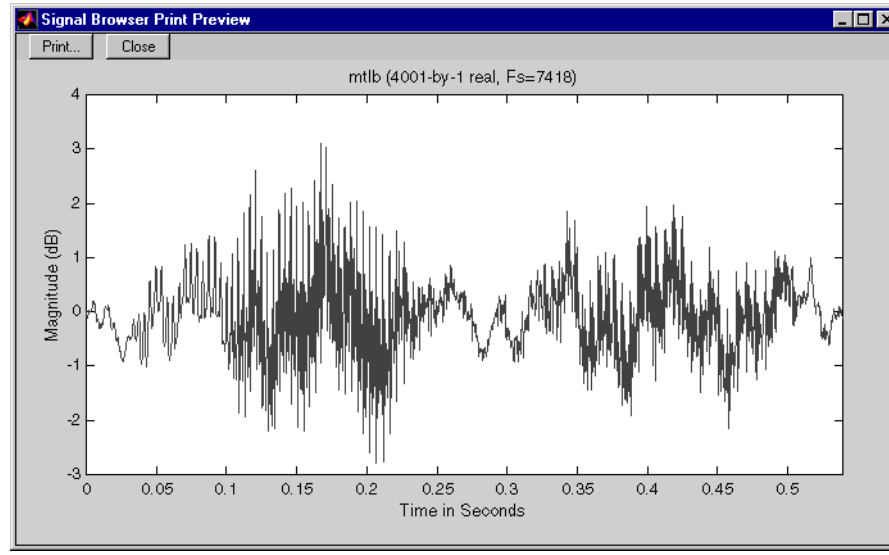
- The plot of the `mtlb` data has been mouse zoomed (see "Zoom Controls" on page 5-31).
- The panner display reflects the current zoom level by enclosing the zoomed portion of the whole signal in a window (see "Panner Display" on page 5-52).
- The rulers have been moved into the zoomed area and the ruler controls have been changed from **Track** to **Slope** (see "Ruler Controls" on page 5-33).
- The ruler measurements displayed reflect the changes to the rulers in the display area (see "Making Signal Measurements" on page 5-37).



Changes you make in the **Preferences** dialog box in SPTool, before opening the print preview window, will also affect the **Signal Browser Print Preview** window in the same way that they affect the Signal Browser itself. For example, in the print preview shown below, the panner display and the ruler measurements have been suppressed, and the axis labels have been changed,

Meta Platforms, Inc. - Exhibit 1009
Page 236 of 720

by changes to the Signal Browser preferences (see “Signal Browser Settings” on page 5-23).



NOTE The `HandleVisibility` property of the **Print Preview** figure window is set to `'on'`. This allows you to modify the preview using Handle Graphics commands from the MATLAB command line or the Plot Editor tools (see *Using MATLAB Graphics* for more information). Changes that you make in the preview window print exactly as they appear on the screen; however, they are *not* saved when the **Print Preview** figure window is closed.

Saving Signal Data

After creating a signal in SPTool (by applying a filter to an imported signal, for example), you can export the signal information to the workspace or to disk using **Export...** from the **File** menu in SPTool. The signal information is stored in a structure that you can access to retrieve the signal data and sample frequency. The signal structure also contains a number of fields that are used internally by SPTool.

To see the fields of the signal structure, try exporting a signal to the workspace:

Meta Platforms, Inc. - Exhibit 1009
Page 237 of 720

1. Import a signal into SPTool if there are none currently loaded (see "Importing Signals, Filters, and Spectra" on page 5-7). Label the imported signal `sig1`.

2. Export the signal. Select **Export...** from the **File** menu.

3. In the **Export** dialog box, select `sig1` and press the **Export to Workspace** button.

4. Type `who` at the MATLAB command line to look at the variables in the workspace. The variable called `sig1` is the signal structure you exported from SPTool.

5. Type `sig1` at the command line to list the fields of the signal structure.

The `data` and `Fs` fields of the signal structure contain the information that defines the signal. The other fields are used internally by SPTool and are subject to change in future releases:

- The `data` field is a vector or array containing the signal's data.
- The `Fs` field contains the sample frequency of the signal in Hertz.

Meta Platforms, Inc. - Exhibit 1009
Page 238 of 720

Using the Filter Designer: Interactive Filter Design

The Filter Designer provides an interactive graphical environment for the design of digital IIR and FIR filters based on specifications that you enter on a magnitude or pole-zero plot. Using the Filter Designer, you can design IIR and FIR filters of various lengths and types, with standard frequency band configurations (highpass, lowpass, bandpass, bandstop, and multiband).

Using the Filter Designer you can:

- Design IIR filters with standard band configurations, using the Butterworth, Chebyshev type I, Chebyshev type II, and elliptic design options
- Design FIR filters with standard band configurations, using the equiripple, least squares, and Kaiser window design options
- Design FIR and IIR filters with arbitrary band configurations, using the Pole/Zero Editor
- Redesign a filter by manually adjusting indicators in the magnitude plot, or by graphically repositioning the transfer function's poles and zeros
- Overlay a spectrum on the filter's magnitude response plot

When you have designed a filter to your specifications, you can apply the filter to a selected signal using the **Apply** button in SPTool (see "Applying a Filter" on page 5-18).

NOTE For information on using filter design functions from the command line or from M-files, see Chapter 2.

Opening the Filter Designer

You can open or activate the Filter Designer from SPTool by pressing the **New Design** or **Edit Design** buttons:

- To create a filter, press **New Design** in SPTool. A default filter is generated and displayed in the Filter Designer. You can view the filter in a variety of ways in the Filter Viewer and modify it in the Filter Designer.
- To edit a filter, select it in SPTool and press **Edit Design**. You can modify the filter in a variety of ways in the Filter Designer.

Meta Platforms, Inc. - Exhibit 1009
Page 239 of 720

See “Designing a Filter” on page 5-17 for complete details.

Basic Filter Designer Functions

The Filter Designer has the following components:

- A pop-up menu for selecting different filter design methods
- A magnitude plot (display) area for graphically adjusting filter responses
- A pole-zero plot (display) area for graphically adjusting filter transfer functions
- A **Specifications** panel for viewing and modifying the design parameters or pole-zero locations of the current filter
- A **Measurements** panel for viewing the response characteristics and stability of the current filter
- Specification lines for graphically adjusting the magnitude response parameters of a filter
- Measurement lines for measuring the magnitude response parameters of a filter
- Zoom controls for getting a closer look at the magnitude response or pole-zero features

The following sections describe the different components of the Filter Designer’s interface and how you can use them together to create and edit filters.

Menus

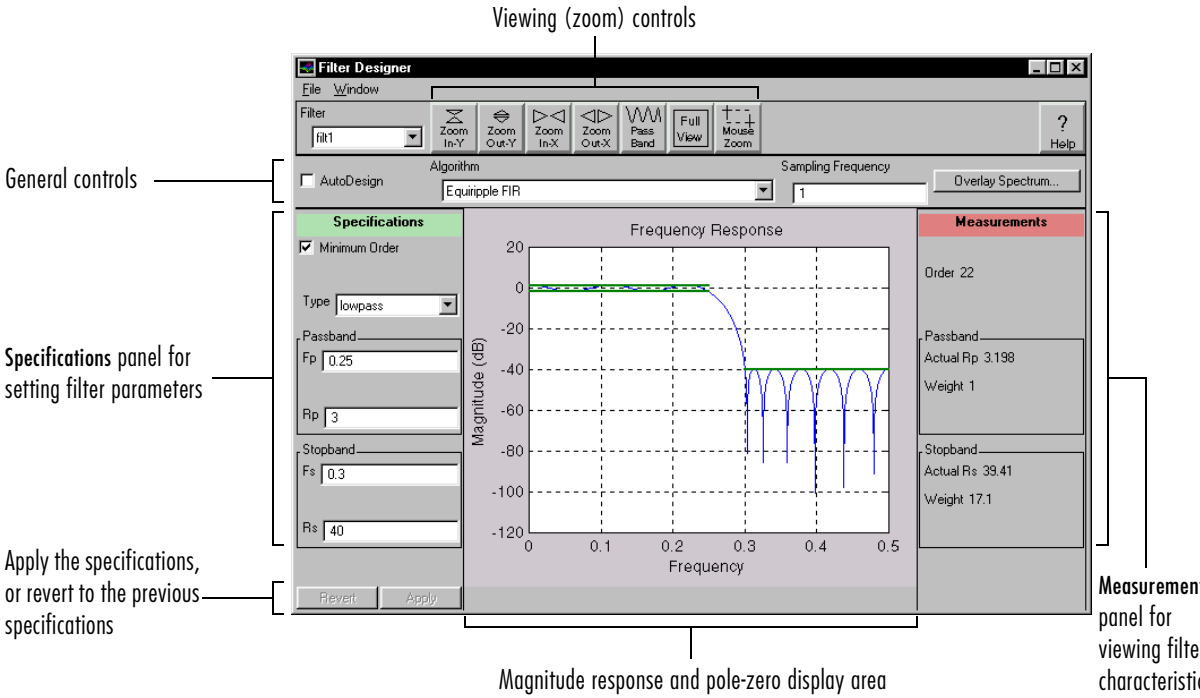
File Menu. Use **Close** from the **File** menu to close the Filter Designer. Settings you changed and saved using the **Preferences** dialog box in SPTool are saved and used the next time you open the Filter Designer.

Window Menu. Use the **Window** menu to select a MATLAB figure window.

Filter Pop-Up Menu

The **Filter** pop-up menu displays all of the filters currently selected in SPTool. Select a filter in the menu to make it the current filter in the Filter Designer.

Meta Platforms, Inc. - Exhibit 1009
Page 240 of 720



Zoom Controls

The available zoom controls in the Filter Designer are **Zoom In-Y**, **Zoom Out-Y**, **Zoom In-X**, **Zoom Out-X**, **Pass Band**, **Full View**, and **Mouse Zoom**.

See “Zoom Controls” on page 5-31 for details on using the zoom controls.

Zoom persistence is off by default in the Filter Designer; use the Filter Designer settings panel in the **Preferences** dialog box to toggle zoom persistence on and off. See “Filter Designer Settings” on page 5-27.

Help Button

To use context-sensitive help, press the **Help** button. The mouse pointer becomes an arrow with a question mark symbol. You can then click on any

Meta Platforms, Inc. - Exhibit 1009
Page 241 of 720

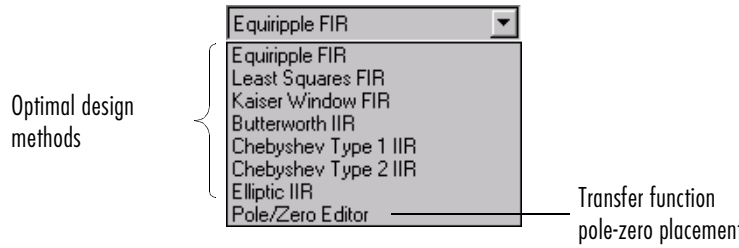
object in the Filter Designer, including menu items, to find out what it is and how to use it.

General Controls

Beneath the zoom controls are several general controls for filter design and display. Some of these controls are available only when an optimal filter design method (e.g., **Equiripple FIR**, **Butterworth IIR**) is selected from the **Algorithm** menu, and are not available when the **Pole/Zero Editor** is selected.

Algorithm. Use the **Algorithm** pop-up menu to select a design method for the current filter. The first seven options in the **Algorithm** menu – the *optimal design methods* – all design filters based on a high-level specification of filter magnitude response characteristics (such as passband frequencies or stopband attenuation). The filter characteristics are specified numerically in the **Specifications** panel or graphically on the magnitude response plot.

The eighth option, **Pole/Zero Editor**, is not an optimal design method but allows low-level specification of the filter's transfer function via numerical or graphical pole-zero placement.



When you select one of the optimal design methods from the menu, the magnitude response plot, **Specifications** panel, and **Measurements** panel all update to reflect the design parameters available for that method. When you select **Pole/Zero Editor**, the magnitude response plot is replaced by the *z*-plane for pole-zero placement.

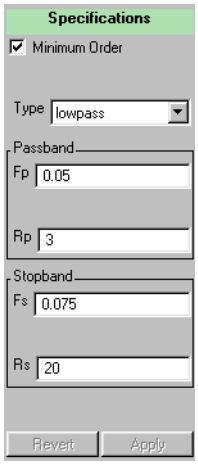
Auto Design (optimal design methods only). When the **Auto Design** check box is checked, the filter's magnitude response is redrawn whenever you change a filter specification, either by entering a value in the **Specifications** panel or by dragging a specification line on the plot. When the box is not checked, the new response is computed and redrawn only when you press the **Apply** button or release the specification line. **Auto Design** is initially off by default; use the

Meta Platforms, Inc. - Exhibit 1009
Page 242 of 720

Filter Designer settings panel in the **Preferences** dialog box to change this default setting. See "Filter Designer Settings" on page 5-27.

Sampling Frequency. The **Sampling Frequency** field allows you to specify the filter's sampling frequency in Hertz. To change the sampling frequency, type a value in the box and press **Enter**. This is the same as changing the sampling frequency by selecting **Sampling Frequency...** from the SPTool **Edit** menu (see "Editing Data Objects in SPTool" on page 5-15). The frequency axis of the magnitude response plot is updated to reflect the new sampling frequency. Changing the **Sampling Frequency** in the Pole/Zero Editor does not alter the pole-zero plot, but does scale the corresponding magnitude plot in the Filter Viewer (if the Viewer is open).

Overlay Spectrum (design methods only). The Filter Designer allows you to overlay a signal spectrum on the filter's magnitude response plot. Press the **Overlay Spectrum...** button to display a list of the current spectra in SPTool. Select a spectrum from the list and press **OK** to overlay it on the current magnitude response plot. Note that the spectrum is plotted on the existing frequency axis, which is scaled to the filter's sampling frequency.



Filter Specifications Panel—Design Methods

When you design a new filter with the design methods, the Filter Designer initially contains the specifications and magnitude response plot for an order 22, lowpass, equiripple filter.

Use the **Type** pop-up menu in the **Specifications** panel to select a band configuration. Use the edit boxes below it in the panel to change the band edge frequencies and the amount of ripple in the passband and attenuation in the stopband. Check the **Minimum Order** box to let the Filter Designer automatically determine the lowest filter order that achieves the current specifications.

The design parameters available in the **Specifications** panel depend on the filter design selected in the **Algorithm** pop-up menu, the band configuration selected in the **Type** pop-up menu, and the state of the **Minimum Order** check box.

Specifications Parameters – Automatic Order Selection. When the **Minimum Order** box is checked, all of the filter designs except **Least Squares FIR** display the same set of parameters in the **Specifications** panel. (The order for the **Least Squares FIR** design cannot be automatically computed). For lowpass and

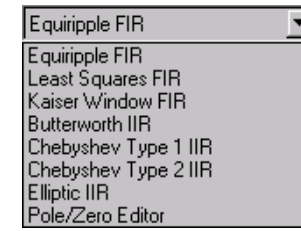
Meta Platforms, Inc. - Exhibit 1009
Page 243 of 720

highpass band configurations, these parameters include the passband edge frequency Fp, the stopband edge frequency Fs, the passband ripple Rp, and the stopband attenuation Rs. For bandpass and bandstop configurations, the parameters include the lower and upper passband edge frequencies, Fp1 and Fp2, the lower and upper stopband edge frequencies, Fs1 and Fs2, the passband ripple Rp, and the stopband attenuation Rs. Frequency values are in Hertz, and ripple and attenuation values are in dB.

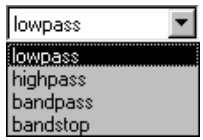
Specifications Parameters—Manual Order Selection. When the **Minimum Order** box is not checked, an **Order** parameter for setting the filter order appears below it, and each filter design displays a unique set of parameters in the **Specifications** panel. These are shown in the table below, where Wp and Ws are the weights for the passband and stopband, Beta is the Kaiser window β parameter, Fc is the cutoff frequency, and F3db is the -3 dB frequency.

	Lowpass	Highpass	Bandpass	Bandstop
Equiripple FIR	Fp, Fs, Wp, Ws	Fp, Fs, Wp, Ws	Fp1, Fp2, Fs1, Fs2, Wp, Ws	Fp1, Fp2, Fs1, Fs2, Wp, Ws
Least Squares FIR	Fp, Fs, Wp, Ws	Fp, Fs, Wp, Ws	Fp1, Fp2, Fs1, Fs2, Wp, Ws	Fp1, Fp2, Fs1, Fs2, Wp, Ws
Kaiser Window FIR	Fc, Beta	Fc, Beta	Fc1, Fc2, Beta	Fc1, Fc2, Beta
Butterworth IIR	F3db	F3db	F3db 1, F3db 2	F3db 1, F3db 2
Chebyshev Type I IIR	Fp, Rp	Fp, Rp	Fp1, Fp2, Rp	Fp1, Fp2, Rp
Chebyshev Type II IIR	Fs, Rs	Fs, Rs	Fs1, Fs2, Rs	Fs1, Fs2, Rs
Elliptic IIR	Fp, Rp, Rs	Fp, Rp, Rs	Fp1, Fp2, Rp, Rs	Fp1, Fp2, Rp, Rs

Designing a New Filter With the Design Methods. In general, follow these steps to design a new filter using the **Specifications** panel parameters:



1 Click-and-drag to select an appropriate optimal filter design method from the **Algorithm** pop-up menu.



2 Click-and-drag to select the band configuration from the **Type** pop-up menu.

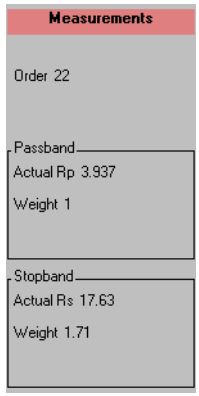
3 Type in the **Specifications** panel's editable fields to change the values for band edge frequencies, ripple, attenuation, and other filter characteristics.

Meta Platforms, Inc. - Exhibit 1009
Page 244 of 720

4 Press **Enter** after typing in an editable text box to enter the value. If **Auto Design** is not checked, press the **Apply** button below the **Specifications** panel to update the magnitude plot.

5 To specify a filter order, press **Minimize Order** to deselect the check box and disable automatic filter order selection. Then type a value for the **Order** parameter. If **Auto Design** is not checked, press **Apply**.

Edit a new filter or an existing filter in the same way. Note that when **Auto Design** is disabled and you change a filter's parameter values, the magnitude response plot changes from solid lines to dashed lines. This indicates that the plot no longer reflects the current specifications. When you press the **Apply** button, the new response is computed and the lines revert to the solid style. When **Auto Design** is enabled, the plot updates whenever you change a filter specification.



Filter Measurements Panel—Design Methods

When you design a filter, the **Measurements** panel (shown at left) displays the values of filter parameters that *do not* appear in the **Specifications** panel. For example, when the Filter Designer provides an **Fs** parameter in the **Specifications** panel, it displays the **Actual Fs** value in the **Measurements** panel. Similarly, when the **Minimum Order** option is selected in the **Specifications** panel, the computed filter order is displayed in the **Order** field of the **Measurements** panel. The values in the **Measurements** panel are updated whenever the magnitude plot is redrawn.

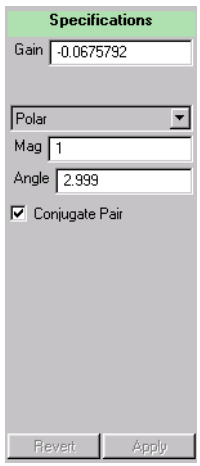
Measurement Parameters – Automatic Order Selection. The parameter combinations that appear in the **Measurements** panel are shown in the following two tables. The first table lists the parameters that appear when **Minimum Order** is checked (automatic order selection). The stopband edge frequency parameters listed (Fs, Fs1, Fs2) are the *actual* edge frequencies for the design, rather than the *desired* frequencies entered in the **Specifications** panel.

Meta Platforms, Inc. - Exhibit 1009
Page 245 of 720

	Lowpass	Highpass	Bandpass	Bandstop
Equiripple FIR	Order, Rp, Rs, Wp, Ws	Order, Rp, Rs, Wp, Ws	Order, Rp, Rs, Wp, Ws	Order, Rp, Rs, Wp, Ws
Least Squares FIR	Order cannot be automatically computed.			
Kaiser Window FIR	Order, Fc, Beta, Rp, Rs	Order, Fc, Beta, Rp, Rs	Order, Fc1, Fc2, Beta, Rp, Rs	Order, Fc1, Fc2, Beta, Rp, Rs
Butterworth IIR	Order, Rp, F3db	Order, Rp, F3db	Order, Rp, F3db 1, F3db 2	Order, Rp, F3db 1, F3db 2
Chebyshev Type I IIR	Order, Fs	Order, Fs	Order, Fs1, Fs2	Order, Fs1, Fs2
Chebyshev Type II IIR	Order, Fs	Order, Fs	Order, Fs1, Fs2	Order, Fs1, Fs2
Elliptic IIR	Order, Fs	Order, Fs	Order, Fs1, Fs2	Order, Fs1, Fs2

Measurement Parameters – Manual Order Selection. The next table shows the parameter sets that appear in the **Measurements** panel when **Minimum Order** is not selected. The measurements that can be interactively changed by dragging the red measurement lines on the response plot are shown shaded.

	Lowpass	Highpass	Bandpass	Bandstop
Equiripple FIR	Rp, Rs	Rp, Rs	Rp, Rs	Rp, Rs
Least Squares FIR	Rp, Rs	Rp, Rs	Rp, Rs	Rp, Rs
Kaiser Window FIR	*Fp, Fs, Rp, Rs*	*Fp, Fs, Rp, Rs*	*Fp1, Fp2, Fs1, Fs2, Rp, Rs*	*Fp1, Fp2, Fs1, Fs2, Rp, Rs*
Butterworth IIR	*Fp, Fs, Rp, Rs*	*Fp, Fs, Rp, Rs*	*Fp1, Fp2, Fs1, Fs2, Rp, Rs*	*Fp1, Fp2, Fs1, Fs2, Rp, Rs*
Chebyshev Type I IIR	*Fs, Rs*	*Fs, Rs*	*Fs1, Fs2, Rs*	*Fs1, Fs2, Rs*
Chebyshev Type II IIR	*Fp, Rp*	*Fp, Rp*	*Fp1, Fp2, Rp*	*Fp1, Fp2, Rp*
Elliptic IIR	Fs	Fs	Fs1, Fs2	Fs1, Fs2



Filter Specifications Panel—Pole/Zero Editor

When you design a new filter using the Pole/Zero Editor, the Filter Designer initially displays a pole-zero plot for the default order-22 filter.

The general pole-zero (factored) form of the transfer function of a digital filter is:

$$H(z) = \frac{Z(z)}{P(z)} = k\frac{(z-z_1)(z-z_2)\cdot\cdot(z-z_m)}{(z-p_1)(z-p_2)\cdot\cdot(z-p_n)}$$

where $z_1, z_2, ..., z_m$ are the filter zeros, $p_1, p_2, ..., p_n$ are the filter poles, and k is the overall transfer function gain. The number of zeros and poles do not need to be the same.

The controls in the **Specifications** panel allow you to set the overall gain and to numerically adjust the position of each pole and zero using polar or rectangular coordinates.

Meta Platforms, Inc. - Exhibit 1009
Page 246 of 720

The value you enter in the **Gain** edit box specifies the gain, k. The **Coordinates** pop-up menu lets you position the poles and zeros using either polar or rectangular measurements. When **Polar** is selected, you can specify the coordinates of the currently selected pole or zero by entering values in the **Mag** and **Angle** edit boxes. The resulting location of the selected pole or zero is

$$z = Me^{j\theta}$$

where M is the magnitude specified by **Mag**, and θ is the angle in radians specified by **Angle**.

When **Rectangular** is selected, you can specify the coordinates of the currently selected pole or zero by entering values in the **X** and **Y** edit boxes. The resulting location of the selected pole or zero is

$$z = X + jY$$

The **Conjugate pair** checkbox controls whether the current selection is a complex conjugate pair of poles or zeros, or a lone pole or zero. When a lone pole or zero is selected on the pole-zero plot, the **Conjugate pair** checkbox is inactive. If you now enable the **Conjugate pair** checkbox by clicking it, a new conjugate pole or zero (as appropriate) is created to complete the pair.

When a conjugate pair of poles or zeros is selected on the plot (by clicking one of the pair), the **Conjugate pair** checkbox is active. If you now disable the **Conjugate pair** checkbox by clicking it, the conjugate pair is broken into two lone poles or zeros at the same positions, and only one of these remains selected.

When the **Conjugate pair** checkbox is selected, the coordinates you specify (in either angular or rectangular form) control the location of the currently selected complex conjugate pair of poles or zeros, with locations

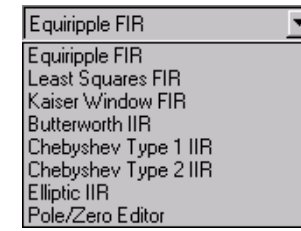
$$z = X \pm jY$$

or

$$z = Me^{\pm j\theta}$$

Designing a New Filter with the Pole/Zero Editor. In general, follow these steps to design a new filter using the **Specifications** panel parameters:

Meta Platforms, Inc. - Exhibit 1009
Page 247 of 720



1 Click-and-drag to select the **Pole/Zero Editor** option from the **Algorithm** pop-up menu.

2 Enter the desired filter gain in the **Gain** edit box.

× Pole
○ Zero

3 Select a pole or zero (or conjugate pair) by clicking on one of the × or ○ symbols on the plot.

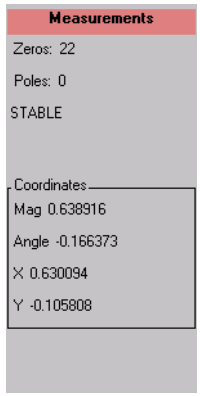
4 Choose the coordinates to work in by specifying **Polar** or **Rectangular** from the **Coordinates** pop-up menu.

5 Specify the new location(s) of the selected pole, zero, or conjugate pair by typing values into the **Mag** and **Angle** (for angular coordinates) or **X** and **Y** (for rectangular coordinates) edit boxes.

6 Use the **Conjugate pair** checkbox to create a conjugate pair from a lone pole or zero or to break a conjugate pair into two individual poles or zeros.

Edit a new filter or an existing filter in the same way.

TIP Keep the Filter Viewer open while designing a filter with the Pole/Zero Editor. Any changes that you make to the filter transfer function in the Pole/Zero Editor are then simultaneously reflected in the response plots of the Filter Viewer.



Filter Measurements Panel—Pole/Zero Editor

When you design a filter, the **Measurements** panel (shown at left) displays useful information about the filter being designed. The panel always shows the number of zeros and poles (the larger of which is the order of the filter), and indicates whether the filter is stable or unstable (poles outside the unit circle).

The panel also shows the phase characteristics of the filter. Where m is the total number of zeros, the four possible phase conditions are:

- **Minimum Phase**, which indicates that between 1 and m-1 zeros are on the unit circle, while the rest of the zeros are inside the unit circle

$$z_i \leq 1 \qquad i = 1, \ldots, m$$

Meta Platforms, Inc. - Exhibit 1009
Page 248 of 720

- **Strictly Minimum Phase**, which indicates that all of the zeros are inside the unit circle

  $z_i < 1 \qquad i = 1, \ldots, m$

- **Maximum Phase**, which indicates that between 1 and *m* zeros are on the unit circle, while the rest of the zeros are outside the unit circle

  $z_i \geq 1 \qquad i = 1, \ldots, m$

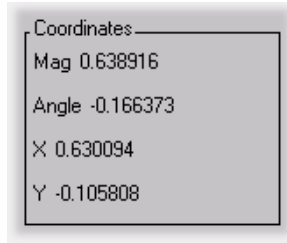
- **Strictly Maximum Phase**, which indicates that all of the zeros are outside the unit circle

  $z_i > 1 \qquad i = 1, \ldots, m$

When the **Measurements** panel does not indicate any of the above conditions, zeros are both inside and outside the unit circle.

Add Poles Pointer
Add Zeros Pointer

When a new pole or zero is being placed on the plot (using the **Add Poles** or **Add Zeros** tools, described on page 5-72) the **Measurements** panel shows additional positioning information. After pressing either the **Add Poles** or **Add Zeros** button, and before clicking on the plot to place the new pole(s) or zero(s), the **Measurements** panel displays the precise position of the pointer over the plot area.



All values in the **Measurements** panel are updated continuously as a pole or zero is repositioned.

Magnitude Plot (Display) Area—Design Methods

The response of the filter selected in the **Filter** pop-up menu is displayed graphically in the magnitude response plot area of the Filter Designer, and its characteristics are shown in the **Specifications** and **Measurements** panels.

Meta Platforms, Inc. - Exhibit 1009
Page 249 of 720

You can zoom in on the displayed filter in the magnitude plot (see "Zoom Controls" on page 5-31) and you can drag the specification lines to redesign the displayed filter visually (see the next section).

Specification Lines. You can redesign a filter by manipulating the green specification lines in the magnitude plot. Using the specification lines, you can change the passband ripple, stopband attenuation, and edge frequencies of a filter:

- Set passband ripple or stopband attenuation by clicking on a green line and dragging it up or down. The Rp and Rs values displayed in the **Specifications** panel change as you drag. If **Auto Design** is checked, the response plot also updates as you drag the lines. If **Auto Design** is disabled, the response plot updates when you release the mouse.

NOTE With IIR filters you can only drag the lower passband bar and the stopband bar. The upper passband bar is fixed at 0.

- Set edge frequencies either by clicking on the edge of a *horizontal* green line (the mouse pointer changes to ⇹) and dragging the edge to a new frequency, or by clicking anywhere on a *vertical* green line (if the Filter Designer provides one) and dragging it horizontally to a new frequency. The Fp and Fs values displayed in the **Specifications** panel change as you drag. If **Auto Design** is checked, the response plot also updates as you drag the lines. If **Auto Design** is disabled, the response plot updates when you release the mouse.

See "Redesigning a Filter Using the Magnitude Plot" on page 5-78 for details.

Measurement Lines. A number of the filter designs provide rulers on the response plot that allow you to measure response magnitude levels. These measurement lines, which appear in red on the plot, are available for the Kaiser window, Butterworth, Chebyshev type I, and Chebyshev type II filters when the **Minimum Order** check box is *not* selected. As you drag a measurement line, the corresponding values in the **Measurements** panel change to reflect the measurement line's current position.

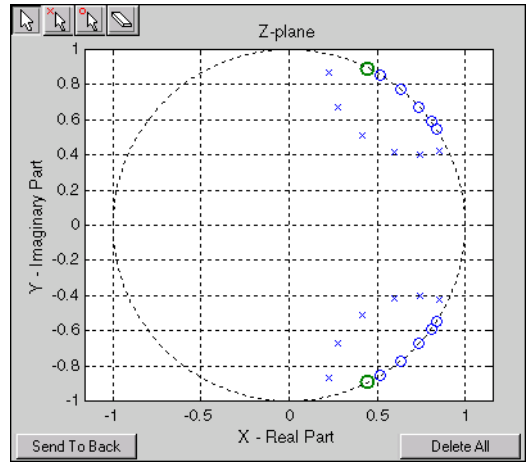
Meta Platforms, Inc. - Exhibit 1009
Page 250 of 720

Magnitude Plot (Display) Area—Pole/Zero Editor

The plot area of the Pole/Zero Editor displays the filter transfer function's pole and zeros on the complex *z*-plane. The unit circle is shown in dashed lines, along with the vertical and horizontal grid lines, to provide a visual reference for pole-zero placement.

You can zoom in on the *z*-plane (see "Zoom Controls" on page 5-31) and you can drag the pole and zero symbols to redesign the filter transfer function visually (see the next section).

Positioning Poles and Zeros. You can redesign the transfer function by simply clicking and dragging on one of the pole or zero symbols. Poles or zeros appear green (rather than the usual blue) when selected. In a conjugate pair of poles or zeros, both conjugates are selected when either is clicked; conjugate pairs move together when dragged, maintaining equal distance from the real axis at all times. To ungroup conjugates, select the desired pair and uncheck **Conjugate pair** in the **Specifications** panel.



When two or more pole symbols (or two or more zero symbols) are placed directly on top of each other, a number is displayed next to the symbols (on the left for poles, and on the right for zeros) indicating the number of poles or zeros at that location (e.g., 3). This number makes it easy to keep track of all the poles and zeros in the plot area, even when several are superimposed on each other and are not visually differentiable. Note, however, that this number does not indicate the *multiplicity* of the poles or zeros to which it is attached.

Meta Platforms, Inc. - Exhibit 1009
Page 251 of 720

To help detect whether a set of poles or zeros are truly multiples, use the zoom tools to magnify the region around the poles or zeros in question. Because numerical limitations usually prevent any set of poles or zeros from sharing *exactly* the same value, at a high enough zoom level even truly multiple poles or zeros appear distinct from each other. A common way to assess whether a particular group of poles or zeros contains multiples is by comparing the mutual proximity of the group members against a selected threshold value. As an example, the `residuez` function defines a pole or zero as being a multiple of another pole or zero if the absolute distance separating them is less than 0.1% of the larger pole or zero's magnitude.

Tools. The Pole/Zero Editor provides several tools to assist with pole-zero placement. The tools are located at the top left of the pole-zero plot area.

Icon	Name	Description
	Drag Poles/Zeros Tool	Use this tool to select and drag poles and zeros on the plot.
	Add Poles Tool	Use this tool to add poles to the plot. Simply click on the plot where you want to place the new pole (or conjugate pair, when the **Conjugate pair** checkbox is selected).
	Add Zeros Tool	Use this tool to add zeros to the plot. Simply click on the plot where you want to place the new zero (or conjugate pair, when the **Conjugate pair** checkbox is selected).
	Delete Poles/Zeros Tool	Use this tool to delete poles and zeros from the plot. Simply click on any pole or zero to delete it. When you click on a set of poles or zero occupying the same location, only the top-most pole or zero is removed. You can also use the **Delete** or **Backspace** key on the keyboard to delete the currently selected pole or zero.

At the bottom of the plot area, there are two additional buttons, **Send To Back** and **Delete All**. Use **Send To Back** when you are unable to select a pole or zero because another pole or zero is in the way. Select the interfering pole or zero on the plot, and press **Send To Back** to move that pole or zero behind all other symbols on the plot (i.e., last in the stacking order). You can then select the desired pole or zero without the interference of the other symbol.

Meta Platforms, Inc. - Exhibit 1009
Page 252 of 720

The **Delete All** button deletes every pole and zero in the plot area.

NOTE The **Delete All** operation cannot be undone. Once all the poles and zeros are removed, they cannot be restored.

Designing Finite Impulse Response (FIR) Filters

The Filter Designer provides three options for basic FIR filter design. These options allow you to create FIR filters with standard band configurations (lowpass, highpass, bandpass, or bandstop configurations only). The three options for FIR filter design in the **Algorithm** pop-up menu are:

- **Equiripple FIR**, which accesses the toolbox function `remez` to create an equiripple FIR filter.
- **Least Squares FIR**, which accesses the toolbox function `firls` to create an FIR filter using the least square design method.
- **Kaiser Window FIR**, which accesses the `fir1` function to create an FIR filter using a Kaiser window.

Example: FIR Filter Design, Standard Band Configuration

In the following example, use the Kaiser window filter design option:

1. Select **Kaiser Window FIR** as the filter design from the **Algorithm** pop-up menu.
2. Select **bandpass** from the **Type** pop-up menu as the configuration.
3. Set the filter's sampling frequency to 2000 Hz by entering this value in the **Sampling Frequency** text box.
4. Press **Apply** to redraw the response with these settings.

NOTE You must press **Apply** before you change the following parameters.

---

5. Check the **Minimum Order** check box to enable automatic order selection.

Meta Platforms, Inc. - Exhibit 1009
Page 253 of 720

6 Set **Fp1** to 290 and **Fp2** to 525.

These fields respectively define the lower and upper passband edge frequencies in Hertz.

7 Set **Fs1** to 200 and **Fs2** to 625.

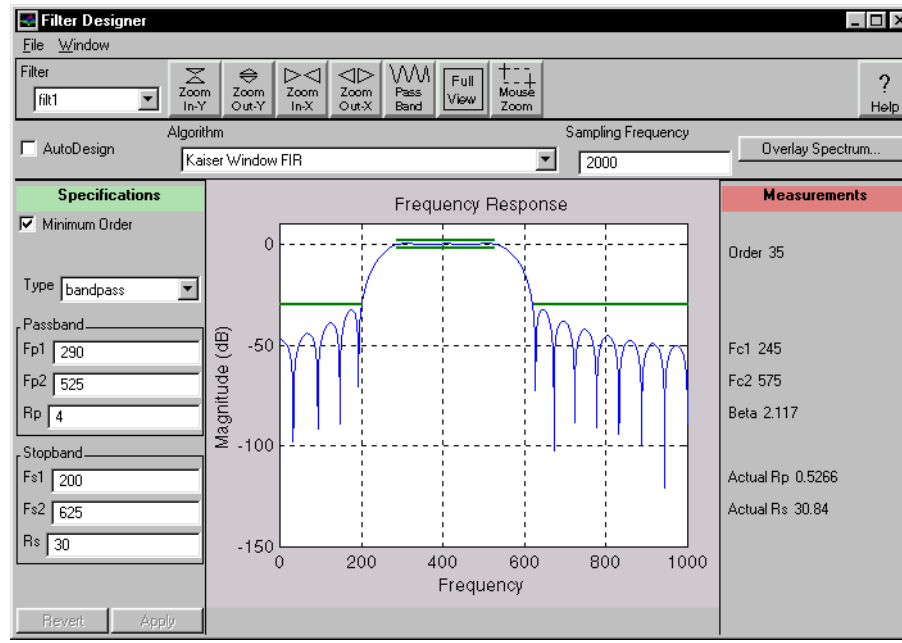
These fields respectively define the lower and upper stopband edge frequencies in Hertz.

8 Set **Rp** (passband ripple) to 4 and **Rs** (stopband attenuation) to 30.

Rp and **Rs** are specified in dB.

9 Press the **Apply** button.

The Filter Designer calls `fir1` to create the filter using a Kaiser window. The Filter Designer updates the magnitude plot to show the new filter's magnitude response.



Meta Platforms, Inc. - Exhibit 1009
Page 254 of 720

Filter Design Options

When the **Minimum Order** option is disabled, you can specify parameters that define characteristics unique to certain filter types:

- For equiripple and least squares filters: the weights for error minimization
- For Kaiser window filters: the cutoff frequency and β parameter of the Kaiser window

Order Selection for FIR Filter Design

As described earlier, the FIR filter design options available through the Filter Designer call the toolbox functions `remez`, `firls`, and `fir1`. In calculating filter order, the Filter Designer uses the same guidelines as the toolbox functions:

- The **Equiripple FIR** design option calls the `remezord` order estimation function to determine a filter order that meets a set of specifications. In some cases, `remezord` underestimates the filter order `n`. If the filter does not appear to meet the given specifications using **Minimum Order** order selection, deselect **Minimum Order** and manually specify a slightly larger order (`n+1` or `n+2`).
- The **Least Squares FIR** design option calls the toolbox function `firls`. Because the toolbox does not provide an order estimation function for use with `firls`, you cannot use the **Minimum Order** option with the **Least Squares FIR** method.
- The **Kaiser Window FIR** design option calls `kaiserord`, the order estimation function, which sometimes underestimates the filter order `n`. If the filter does not appear to meet the given specifications using **Minimum Order** order selection, deselect **Minimum Order** and manually specify a slightly larger order (`n+1` or `n+2`).

All of the FIR filter design options in the Filter Designer require an even filter order for the highpass and bandstop configurations. For more information on order selection with the FIR filter design options, see the reference descriptions of `remez`, `remezord`, `kaiserord`, `firls`, and `fir1` in Chapter 6.

Meta Platforms, Inc. - Exhibit 1009
Page 255 of 720

Designing Infinite Impulse Response (IIR) Filters

The Filter Designer lets you design a number of classical IIR filters, including Butterworth, Chebyshev type I, Chebyshev type II, and elliptic filters.

Example: Classical IIR Filter Design

In the following example, design a simple Chebyshev type I filter:

1. Select **Chebyshev Type I IIR** as the filter design from the **Algorithm** pop-up menu.
2. Select **highpass** from the **Type** menu as the configuration.
3. Set the filter's sample frequency to 2000 Hz by entering this value in the **Sampling Frequency** text box.
4. Press **Apply** to redraw the response with these settings.

NOTE You must press **Apply** before you change the following parameters.

---

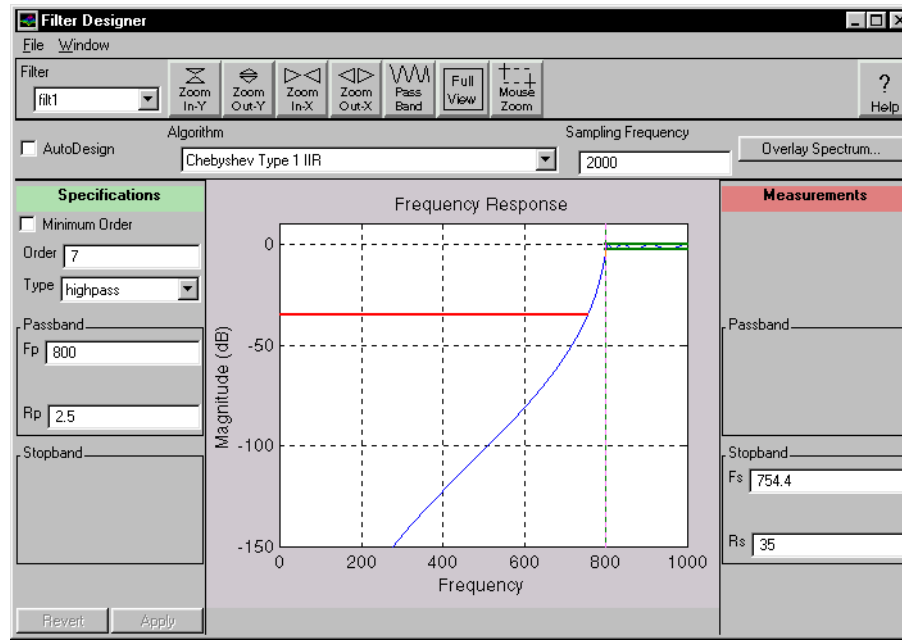
5. Check the **Minimum Order** check box.
6. Set **Fp** (passband edge frequency) to 800 and **Fs** (stopband edge frequency) to 700.

   Fp and **Fs** are specified in Hertz.
7. Set **Rp** (passband ripple) to 2.5 and **Rs** (stopband attenuation) to 35.

   Rp and **Rs** are specified in dB.
8. Press **Apply** to draw the magnitude response.
9. Deselect **Minimum Order** and specify a filter order of 7. Press **Apply** to redraw the magnitude response.

Meta Platforms, Inc. - Exhibit 1009
Page 256 of 720

The Filter Designer calls the appropriate filter design function to create the filter.



To zoom in on plot details, use the zoom control buttons as described in “Zoom Controls” on page 5-31.

Filter Design Options

When the **Minimum Order** option is disabled, you can specify parameters that define characteristics unique to certain filter types:

- For Butterworth filters: the -3 dB frequencies
- For Chebyshev type I filters: the passband edge frequencies
- For Chebyshev type II filters: the stopband edge frequencies
- For elliptic filters: the passband edge frequencies

In the following example, redesign the Chebyshev type I filter from the previous example as a Butterworth filter, using a -3 dB frequency of 800 Hz:

Meta Platforms, Inc. - Exhibit 1009
Page 257 of 720

1 Select the **Butterworth IIR** filter design from the **Algorithm** menu.

   The magnitude response plot changes to reflect the new design.

2 Type a value of 800 in the **F3db** text field.

3 Press **Apply** to update the response plot.

Order Selection for IIR Filter Design

The IIR filter design options available through the Filter Designer call the toolbox filter design functions. In calculating the order for a given filter, the Filter Designer uses the corresponding order estimation function if the **Minimum Order** check box is selected.

For details on order selection with the IIR filter design options, see the reference descriptions of `buttord`, `cheb1ord`, `cheb2ord`, and `ellipord` in Chapter 6.

Redesigning a Filter Using the Magnitude Plot

After designing a filter in the Filter Designer, you can redesign it by dragging the specification lines in the magnitude plot. Use the specification lines to change passband ripple, stopband attenuation, and edge frequencies (see "Specification Lines" on page 5-70 for details). In the following example, create a Chebyshev filter and modify it by dragging the specification lines:

1 Create a Chebyshev type I highpass filter with a sample frequency of 2000 Hz. Set the following parameters:

   Fp = 800
   Fs = 700
   Rp = 2.5
   Rs = 35

2 Check **Minimum Order** so the Filter Designer can calculate the lowest filter order that produces the desired characteristics.

3 Press **Apply** to update the response plot.

4 Position the cursor over the green line specifying the stopband.

   The cursor changes to the up/down drag indicator.

Meta Platforms, Inc. - Exhibit 1009
Page 258 of 720

5 Drag the line until the **Rs** (stopband attenuation) field reads 100.

   Note that the **Order** value in the **Measurements** panel changes because a higher filter order is needed to meet the new specifications.

Saving Filter Data

After designing a filter in the Filter Designer, you can export the filter information to the workspace or to disk using **Export...** from the **File** menu in SPTool. The filter information is stored in a structure that you can access to retrieve the coefficients and design parameters of the filter you created. The filter structure also contains a number of fields that are used internally by SPTool.

To see the fields of the filter structure, first export a filter to the workspace:

1 Create a new filter by pressing **New Design** in SPTool. The new filter is called `filt1`.

2 Select **Export...** from the **File** menu.

3 In the **Export from SPTool** dialog box, select `filt1` and press the **Export to Workspace** button.

4 Type `who` at the MATLAB command line to look at the variables in the workspace. The variable called `filt1` is the filter structure you exported from SPTool.

5 At the command line type `filt1` to list the fields of the filter structure.

The `tf`, `Fs`, and `specs` fields of the filter structure contain the information that describes the filter. These fields are discussed below. The other fields in the structure are used internally by SPTool, and are subject to change in future releases.

tf. The `tf` field is a structure containing the transfer function representation of the filter:

- `tf.num` contains the numerator coefficients
- `tf.den` contains the denominator coefficients

Meta Platforms, Inc. - Exhibit 1009
Page 259 of 720

both in descending powers of z

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + \cdots + b(nb + 1)z^{-m}}{a(1) + a(2)z^{-1} + \cdots + a(na + 1)z^{-n}}$$

where b is a vector containing the coefficients from the `tf.num` field, a is a vector containing the coefficients from the `tf.den` field, m is the numerator order, and n is the denominator order. You can change the filter representation from the default transfer function to another form by using the `tf2ss` or `tf2zp` functions.

Fs. The `Fs` field contains the sampling frequency of the filter in Hertz.

specs. The `specs` field is a structure containing information about the filter design. The first field, `specs.currentModule`, contains a string representing the design selected for the filter in the Filter Designer's **Algorithm** pop-up menu. The possible contents of the `currentModule` field, and the corresponding designs, are shown below.

currentModule	Algorithm
`fdbutter`	Butterworth IIR
`fdcheby1`	Chebyshev Type I IIR
`fdcheby2`	Chebyshev Type II IIR
`fdellip`	Elliptic IIR
`fdfirls`	Least Squares FIR
`fdkaiser`	Kaiser Window FIR
`fdremez`	Equiripple FIR

Following the `specs.currentModule` field, there may be up to seven additional fields, with labels such as `specs.fdremez`, `specs.fdfirls`, etc. The design specifications for the most recently exported filter are contained in the field

Meta Platforms, Inc. - Exhibit 1009
Page 260 of 720

whose label matches the `currentModule` string. For example, if the `specs` structure is

```
currentModule: 'fdkaiser'
fdremez: [1x1 struct]
fdfirls: [1x1 struct]
fdkaiser: [1x1 struct]
```

the filter specifications are contained in the `fdkaiser` field, which is itself a data structure.

The specifications include the parameter values from the **Specifications** panel of the Filter Designer, such as band edges and filter order. For example, the Kaiser window filter above has the following specifications stored in `specs.fdkaiser`.

```
setOrderFlag: 0
type: 1
f: [0 0.1000 0.1500 1]
Rp: 3
Rs: 20
Wn: 0.1250
order: 34
Beta: 0
wind: [35x1 double]
```

Because certain filter parameters are unique to a particular design, this structure has a different set of fields for each filter design. For example, the `Beta` field above only appears in the `specs` structure if the design is a Kaiser window FIR filter.

The table below lists the possible specifications fields that can appear in the export structure, and describes their contents.

Parameter	Description
`Beta`	Kaiser window β parameter.
`f`	Contains a vector of band-edge frequencies, normalized to 1 (i.e., 1 = Nyquist).

Meta Platforms, Inc. - Exhibit 1009
Page 261 of 720

Parameter	Description
`Fpass`	Passband cutoff frequencies. Scalar for lowpass and highpass designs, two-element vector for bandpass and bandstop designs.
`Fstop`	Stopband cutoff frequencies. Scalar for lowpass and highpass designs, two-element vector for bandpass and bandstop designs.
`m`	The response magnitudes corresponding to the band-edge frequencies in `f`.
`order`	Filter order.
`Rp`	Passband ripple (dB)
`Rs`	Stopband attenuation (dB)
`setOrderFlag`	Contains `1` if the filter order was specified manually (i.e., the **Minimum Order** box in the **Specifications** panel was not checked). Contains `0` if the filter order was computed automatically.
`type`	Contains `1` for lowpass, `2` for highpass, `3` for bandpass, or `4` for bandstop.
`w3db`	-3 dB frequency for Butterworth IIR designs.
`wind`	Vector of Kaiser window coefficients.
`Wn`	Cutoff frequency for the Kaiser window FIR filter when `setOrderFlag = 1`.
`wt`	Vector of weights, one weight per frequency band.

Viewing Frequency Response Plots

It is often useful to view a filter's frequency response, impulse response, and step response during the filter design process. You can use the Filter Viewer to view frequency-domain information about filters in the Filter Designer:

Meta Platforms, Inc. - Exhibit 1009
Page 262 of 720

1. Activate SPTool from the **Window** menu.
2. Make sure the filters you want to analyze are selected in the **Filters** list.
3. Press **View** in the Filter panel.

   The Filter Viewer is activated with the selected filters displayed.
4. To edit one of the filters you're viewing, you can reactivate the Filter Designer from the **Window** menu in the Filter Viewer.
5. When you want to review a filter's characteristics after you've edited it, reactivate the Filter Viewer from the **Window** menu in the Filter Designer.

When the Filter Viewer is open at the same time that the Filter Designer is open, they both display the same filter. You can move back and forth between the Filter Designer and the Filter Viewer until the filter design is finished.

You can apply the filter to a signal by activating SPTool, selecting the filter in the **Filters** list, and the signal to apply it to from the **Signals** list, and pressing **Apply**. See "Applying a Filter" on page 5-18 for details.

See "Using the Filter Viewer: Interactive Filter Analysis" on page 5-84 for more information on the Filter Viewer.

Meta Platforms, Inc. - Exhibit 1009
Page 263 of 720

Using the Filter Viewer: Interactive Filter Analysis

An important aspect of filter design is filter analysis, which encompasses both frequency and time-domain analysis of a filter. The Filter Viewer is a GUI-based frequency analysis tool that provides an interactive environment for the graphical display of digital filter characteristics.

The Filter Viewer can display six different characteristics subplots of a selected filter. Any combination of the six subplots may be displayed.

Using the Filter Viewer you can:

- View magnitude-response plots for one or more filters
- View phase-response plots for one or more filters
- View group-delay plots for one or more filters
- View zero-pole plots for one or more filters
- View impulse-response plots for one or more filters
- View step-response plots for one or more filters
- Zoom in to explore filter response details
- Modify selected plot parameters and display characteristics
- Measure a variety of characteristics of the filter response

For information on frequency analysis using toolbox functions from the command line or from M-files, see “Frequency Response” in Chapter 1 of this manual.

Opening the Filter Viewer

Open or activate the Filter Viewer from SPTool:

1. Select one or more filters from the **Filters** list in SPTool.
2. Press **View** in the **Filters** panel in SPTool.

   The Filter Viewer is activated and the selected filters are loaded into the Filter Viewer and displayed.

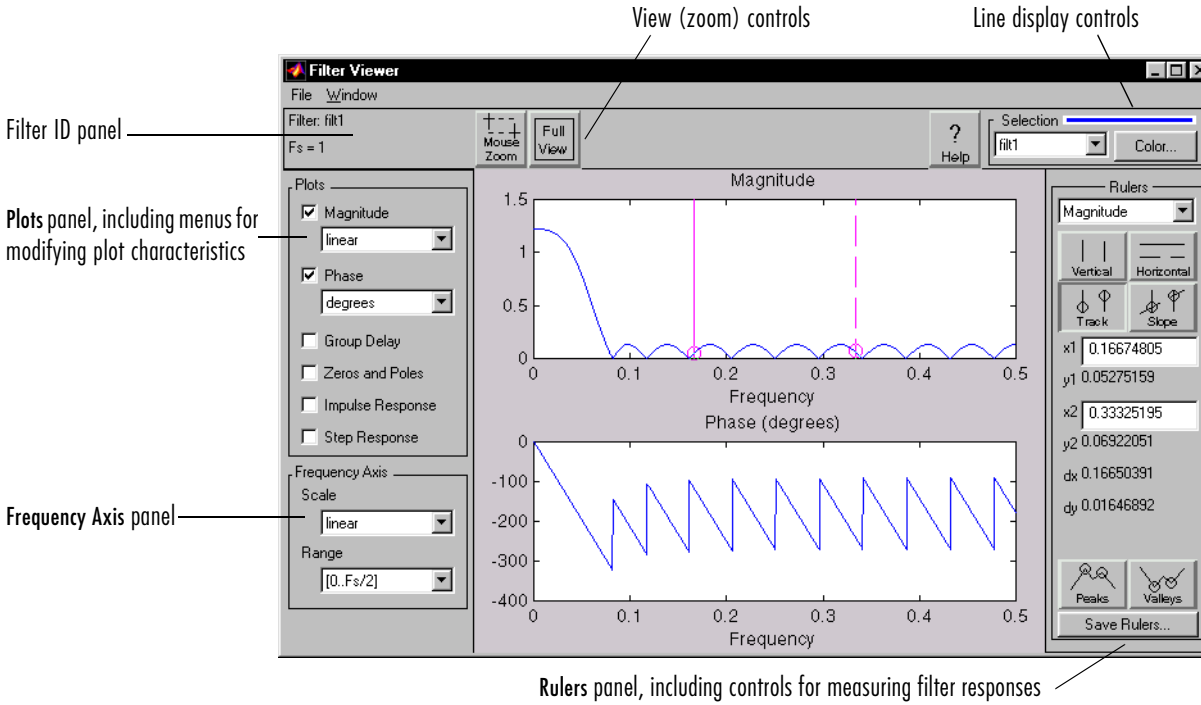
Basic Filter Viewer Functions

The Filter Viewer has the following components:

Meta Platforms, Inc. - Exhibit 1009
Page 264 of 720

- A **Plots** panel for selecting which subplots display in the main plots window
- A **Rulers** panel and line display controls for making signal measurements and comparisons
- A **Frequency Axis** panel for specifying *x*-axis scaling in the main plots window
- A filter identification panel that displays information about the currently selected filter(s)
- A main plots (display) area for viewing one or more frequency-domain plots for the selected filter(s)
- Zoom controls for getting a closer look at filter response characteristics

When you first open or activate the Filter Viewer, it displays the default plot configuration for the selected filter(s):



The filters' magnitude and phase plots are displayed. The frequency axis of the plots is set to **linear**, and the frequency axis range is set to **[0,Fs/2]**.

Meta Platforms, Inc. - Exhibit 1009
Page 265 of 720

You can choose to display one or any combination of the six available subplots by using the check boxes in the **Plots** panel, and you can modify many of the plot display characteristics using the pop-up menus in the **Plots** panel and the **Frequency Axis** panel.

Menus

File Menu. Use **Close** from the **File** menu to close the Filter Viewer. Settings you changed and saved using the **Preferences** dialog box in SPTool are saved and used the next time you open a Filter Viewer.

Window Menu. Use the **Window** menu to select a currently open MATLAB figure window.

Filter Identification Panel

This panel displays the variable names and the highest sampling frequency of the currently selected filters. To change names or sampling frequencies, use **Name...** or **Sampling Frequency...** from the **Edit** menu in SPTool.

Plots Panel

The check boxes in this panel select the subplots to display in the main plots area. Any combination of subplots may be displayed.

To display a subplot, check the box at the left of the plot description.

There are six available subplots:

- **Magnitude**: displays the magnitude of the frequency response of the currently selected filter(s).
- **Phase**: displays the phase of the frequency response.
- **Group Delay**: displays the negative of the derivative of the phase response.
- **Zeros and Poles**: displays the poles and zeros of the filter transfer function(s) and their proximity to the unit circle.
- **Impulse Response**: displays the response of the currently selected filter(s) to a discrete-time unit-height impulse at t=0.
- **Step Response**: displays the response of the currently selected filter(s) to a discrete-time unit-height step function.

Meta Platforms, Inc. - Exhibit 1009
Page 266 of 720

You can customize the display characteristics of the magnitude and phase subplots using the **Magnitude** and **Phase** pop-up menus. The options include:

- **Magnitude**: Scaling for the magnitude plot may be **linear**, **log**, or **decibels**.
- **Phase**: Phase units may be **degrees** or **radians**.

You can also change the magnitude and phase display characteristics for the Filter Viewer using the Filter settings panel of the **Preferences** dialog box in SPTool.

Frequency Axis Settings

You can change frequency axis scaling and range parameters for plots in the Filter Viewer.

Click on the option in the **Frequency Axis** panel you want to edit and drag to select a value. The options include:

- **Scale**: Scaling for the frequency axis may be **linear** or **log**.
- **Range**: The range for the frequency axis may be **[0,Fs/2]**, **[0,Fs]**, or **[-Fs/2,Fs/2]**, where **Fs** represents the filter's sampling frequency.

  The frequency range cannot be negative if **Scale** is set to **log**.

You can also change the frequency axis display characteristics for the Filter Viewer using the **Filter Viewer** settings panel of the **Preferences** dialog box in SPTool.

Zoom Controls

The available zoom controls in the Filter Viewer are **Mouse Zoom** and **Full View**. You can zoom independently in each displayed subplot.

By default, persistent zooming is disabled in the Filter Viewer. You can turn persistent zooming on from the **Filter Viewer** settings panel of the **Preferences** dialog box in SPTool.

See "Zoom Controls" on page 5-31 for details on using the zoom controls in the Filter Viewer.

Help Button

To use context-sensitive help, press the **Help** button. The mouse pointer becomes an arrow with a question mark symbol. You can then click on anything

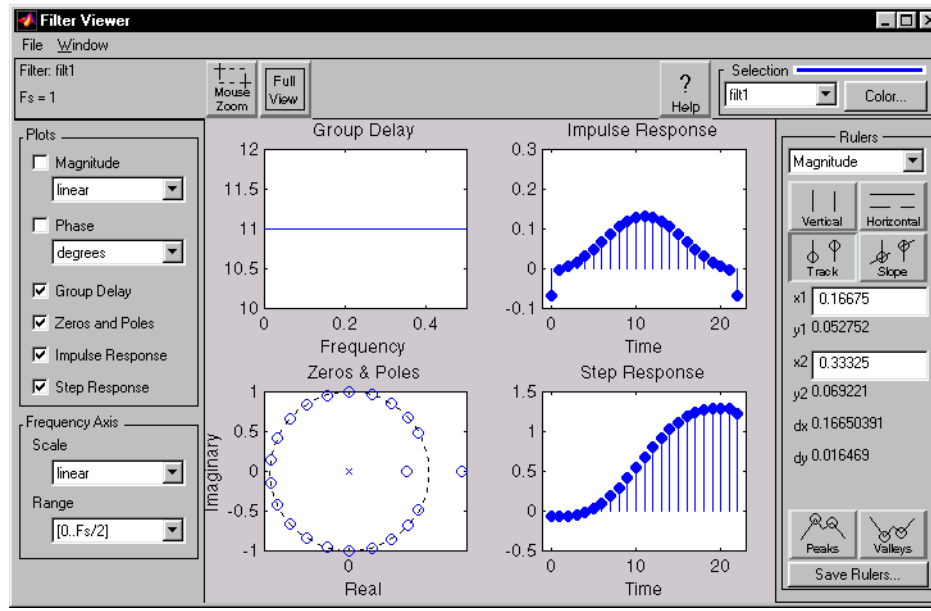
Meta Platforms, Inc. - Exhibit 1009
Page 267 of 720

in the Filter Viewer, including menu items, to find out what it is and how to use it.

Main Plots Area

One or more of the six filter response subplots may be displayed graphically in the main plots area of the Filter Viewer. You can specify how the subplots are arranged by selecting **Filter Viewer Tiling** from the **Preferences** dialog box in SPTool. The options are **2-by-3 Grid**, **3-by-2 Grid**, **Vertical (6-by-1 Grid)**, and **Horizontal (1-by-6 Grid)**.

The following figure shows the Filter Viewer when four subplots are turned on and the 2-by-3 grid option is selected.



You can experiment to find the tiling option that works best for each specific combination and number of subplots.

You can zoom in on a subplot by pressing **Mouse Zoom** and then clicking on or dragging over a selected area of the subplot. By default, mouse zooming in the Filter Viewer is not persistent; after you click once, the zoom mode is turned off. You can make zooming persistent by checking **Stay in Zoom-mode after**

Meta Platforms, Inc. - Exhibit 1009
Page 268 of 720

Zoom in the SPTool **Preferences** dialog box. This allows you to click repeatedly in a subplot to continue to zoom in on a particular feature of the display.

After you zoom in on a subplot, you can click and drag to pan around the subplot:

1. Press **Mouse Zoom** to turn on mouse zoom mode.
2. Click on a feature of a subplot to zoom in on it.
3. If persistent zooming is enabled, press **Mouse Zoom** again to turn off mouse zoom mode.
4. Click again in the same subplot, hold down the mouse button until the hand cursor is displayed, and drag the mouse to pan around the subplot.

Viewing Filter Plots

This section has a brief description and picture of each of the six filter response plots available in the Filter Viewer. A sequence of connected examples shows you how to display each plot on its own; you can also display any combination of plots, as needed.

Each plot in the example sequence displays the response of an order 22 equiripple lowpass filter with a sampling frequency of 1 Hz.

Regardless of how many or what combination of plots is displayed, you can zoom in on and pan each subplot independently.

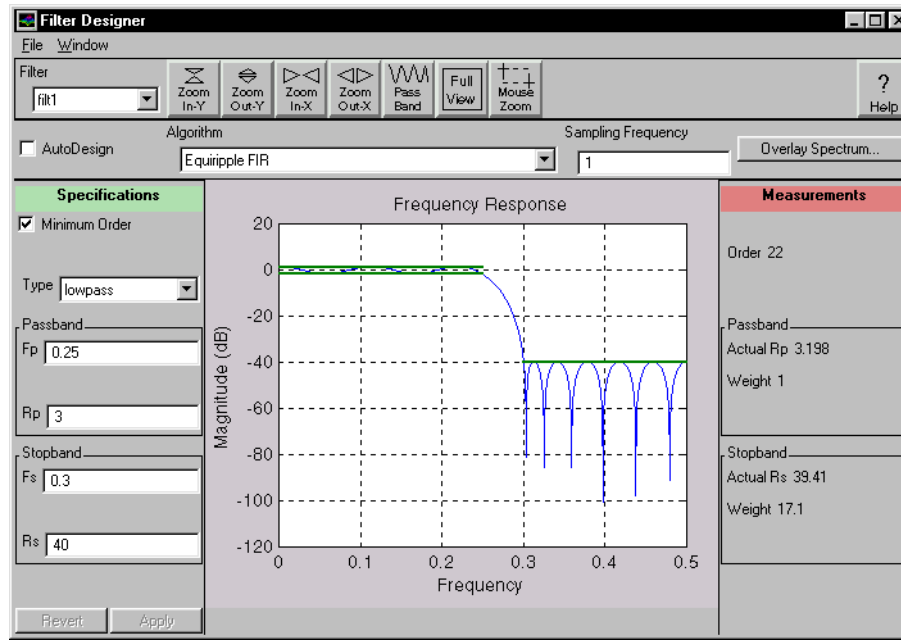
Viewing Magnitude Response

A magnitude response plot is generally the simplest way to obtain a high-level view of a filter's shape and fit to specifications. In the following example, use the Filter Designer to create a standard default filter and then view its magnitude response plot in the Filter Viewer:

Meta Platforms, Inc. - Exhibit 1009
Page 269 of 720

1 From SPTool, press **Create**.

The Filter Designer is activated and a standard default filter is created and displayed.



This is an order 22 equiripple lowpass filter with a sampling frequency of 1 Hz.

2 Use the **Window** menu in the Filter Designer to activate SPTool.

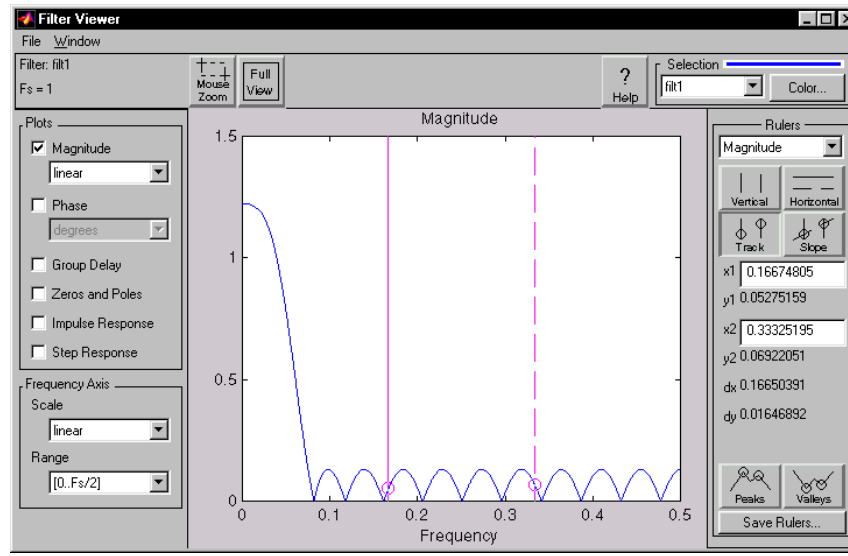
3 Press **View** from the **Filters** panel in SPTool to activate the Filter Viewer.

The Filter Viewer is displayed with a magnitude response plot and a phase response plot.

Meta Platforms, Inc. - Exhibit 1009
Page 270 of 720

4 Click the check box next to the **Phase** option to turn off the phase plot.

The magnitude plot for the default filter is displayed.



By default, this plot uses the default scaling (**linear**) for both axes and the default range for the frequency axis.

You can change the following display characteristics of the magnitude plot:

- Use the **Magnitude** pop-up menu to choose between **linear**, **log**, or **decibels** scaling of the *y*-axis.
- Use the **Scale** pop-up menu to choose between **linear** and **log** scaling of the *x*-axis.
- Use the **Range** pop-up menu to choose between the following ranges for the *x*-axis: **[0,Fs/2]**, **[0,Fs]**, or **[-Fs/2,Fs/2]**, where **Fs** represents the filter's sampling frequency.

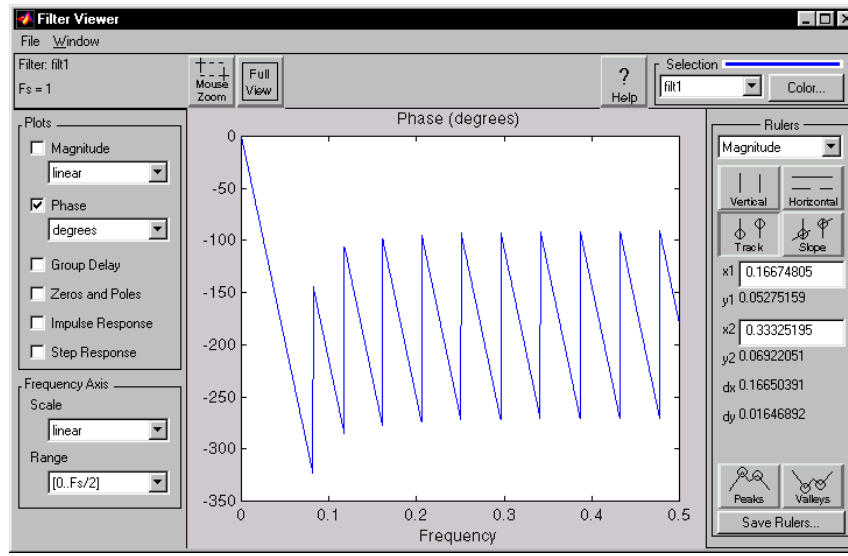
Viewing Phase Response

In addition to displaying magnitude response, the Filter Viewer can calculate and plot the filter's phase response. *Phase response* is the angular component of a filter's frequency response. To display only a phase response plot for the current filter:

Meta Platforms, Inc. - Exhibit 1009
Page 271 of 720

1 Click the check box next to the **Magnitude** option to turn off the magnitude plot.

2 Click the check box next to the **Phase** option to turn on the phase plot and update the display.



By default, this plot uses the default phase (**degrees**) and the default scaling and range for the frequency axis.

You can change the following display characteristics of the phase plot:

- Use the **Phase** pop-up menu to choose between displaying phase in **degrees** or **radians**.
- Use the **Scale** pop-up menu to choose between **linear** and **log** scaling of the *x*-axis.
- Use the **Range** pop-up menu to choose between the following ranges for the *x*-axis: **[0,Fs/2]**, **[0,Fs]**, or **[-Fs/2,Fs/2]**, where **Fs** represents the filter's sampling frequency.

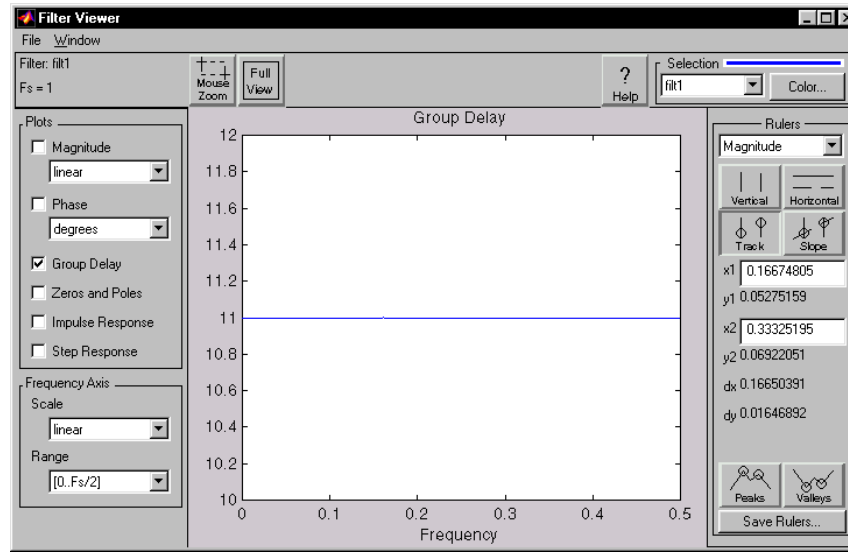
Meta Platforms, Inc. - Exhibit 1009
Page 272 of 720

Viewing Group Delay

Group delay is a measure of the average delay of a filter as a function of frequency. To display only a group delay plot for the currently selected filter(s):

1. Click the check box next to the **Phase** option to turn off the phase plot.

2. Click the check box next to the **Group Delay** option to turn on the group delay plot and update the display.



By default, this plot uses the default scaling and range for the frequency axis.

You can change the following display characteristics of the group delay plot:

- Use the **Scale** pop-up menu to choose between **linear** and **log** scaling of the *x*-axis.
- Use the **Range** pop-up menu to choose between the following ranges for the *x*-axis: **[0,Fs/2]**, **[0,Fs]**, or **[-Fs/2,Fs/2]**, where **Fs** represents the highest sampling frequency of the currently selected filters.

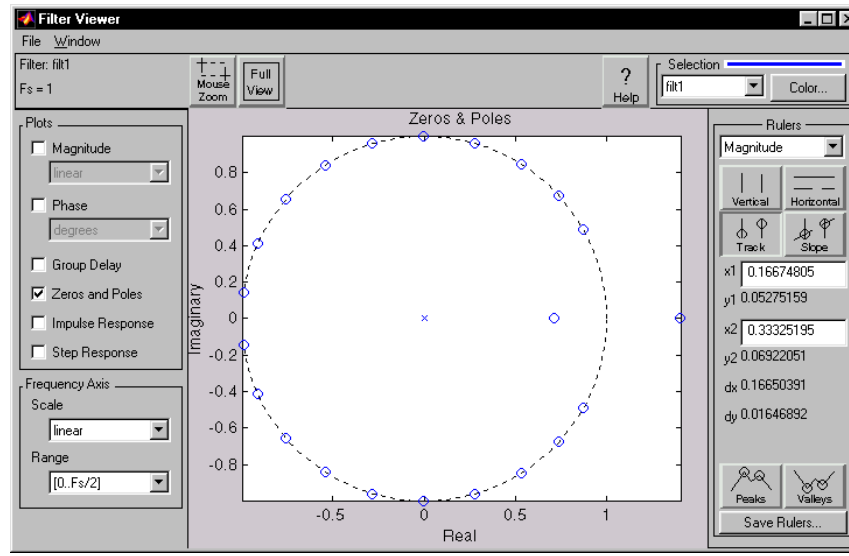
Meta Platforms, Inc. - Exhibit 1009
Page 273 of 720

Viewing a Zero-Pole Plot

The *zero-pole plot* displays the poles and zeros of the transfer function and their proximity to the unit circle. An × represents a pole of the transfer function; a ○ represents a zero of the transfer function. To display only a zero-pole plot for the currently selected filter(s):

1. Click the check box next to the **Group Delay** option to turn off the group delay plot.

2. Click the check box next to the **Zeros and Poles** option to turn on the zero-pole plot and update the display.



Viewing Impulse Response

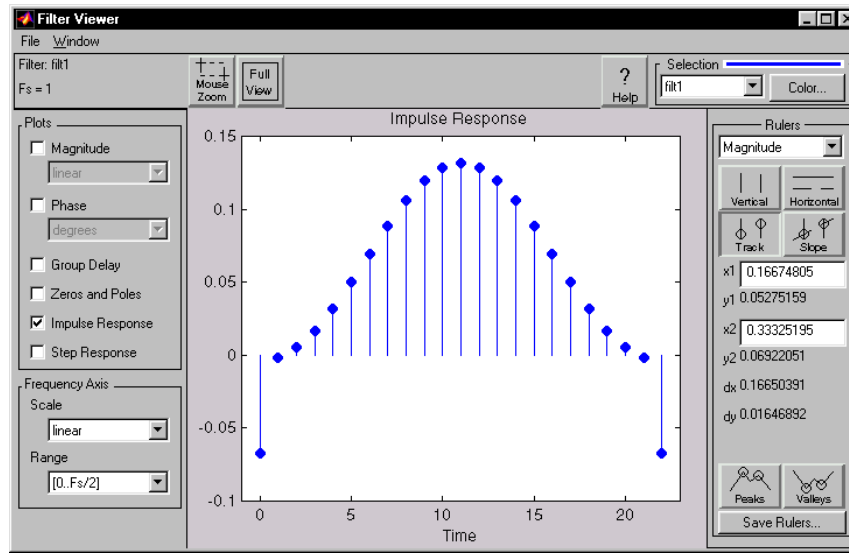
The *impulse response plot* displays the response of the current filter(s) to a discrete-time unit-height impulse at t=0.

To display only an impulse response plot for the currently selected filter(s):

Meta Platforms, Inc. - Exhibit 1009
Page 274 of 720

1 Click the check box next to the **Zeros and Poles** option to turn off the zero-pole plot.

2 Click the check box next to the **Impulse Response** option to turn on the impulse response plot and update the display.



You can change the following display characteristics of the impulse response plot:

Edit the **Time Response Length** field in the **Filter Viewer Preferences** panel to set the number of samples used to display the impulse response.

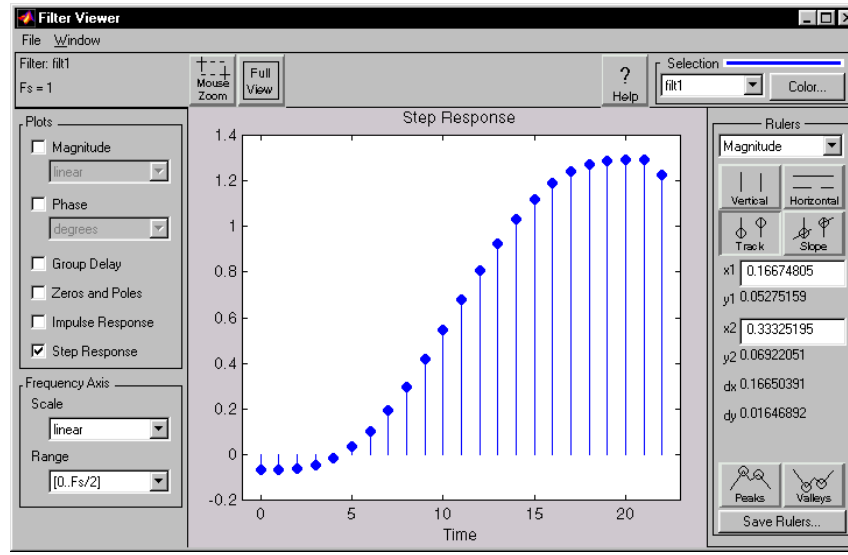
Viewing Step Response

The *step response plot* displays the response of the current filter(s) to a discrete-time unit-height step function. To display only a step response plot for the currently selected filter(s):

Meta Platforms, Inc. - Exhibit 1009
Page 275 of 720

1 Click the check box next to the **Impulse Response** option to turn off the impulse response plot.

2 Click the check box next to the **Step Response** option to turn on the step response plot and update the display.



You can change the following display characteristics of the step response plot:

Edit the **Time Response Length** field in the **Filter Viewer Preferences** panel to set the number of samples used to display the step response.

Meta Platforms, Inc. - Exhibit 1009
Page 276 of 720

Using the Spectrum Viewer: Interactive PSD Analysis

The Spectrum Viewer provides an interactive environment for the estimation of power spectral density for one data channel. It allows you to view and modify spectra created in SPTool.

Using the Spectrum Viewer you can:

- View and compare spectral density plots
- Use different estimation methods, including Burg, FFT, MTM, MUSIC, Welch, and Yule-Walker AR
- Modify spectrum parameters such as FFT length, window type, and sample frequency
- Print spectrum data

For information on spectral analysis using toolbox functions from the command line or from M-files, see Chapter 3 of this manual.

Opening the Spectrum Viewer

You can open or activate the Spectrum Viewer from SPTool by pressing one of the following buttons: **Create**, **View**, and **Update**. See "Creating a Spectrum" on page 5-19, "Viewing a Spectrum" on page 5-19, and "Updating a Spectrum" on page 5-19 for complete details.

Here is a brief summary of each method of activating the Spectrum Viewer:

- To create a spectrum, select a signal in SPTool and press **Create**. Press **Apply** in the Spectrum Viewer.

  A default spectrum of the selected signal is generated and displayed. You can view it in a variety of ways, measure it, and modify it in the Spectrum Viewer.
- To view a spectrum, select one or more spectra in SPTool and press **View** in the **Spectra** panel.
- To update a spectrum, select exactly one signal and one spectrum in SPTool and press **Update**. Press **Apply** in the Spectrum Viewer.

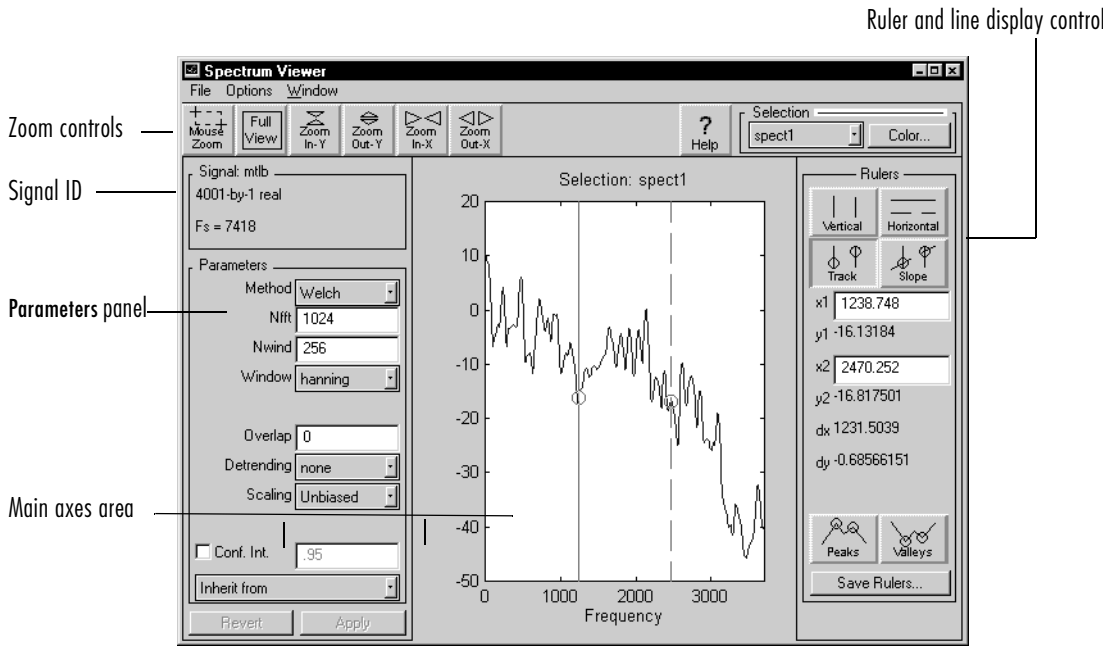
  The spectrum is updated to reflect the data in the currently selected signal.

Meta Platforms, Inc. - Exhibit 1009
Page 277 of 720

Basic Spectrum Viewer Functions

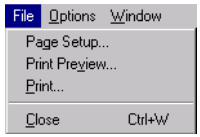
The Spectrum Viewer window has the following components:

- A **Parameters** panel for viewing and modifying the parameters or method of the current spectrum
- A signal identification panel that displays information about the signal linked to the current spectrum
- A main axes (display) area for viewing spectra graphically
- Zoom controls for getting a closer look at spectral features
- Rulers and line-display controls for making spectral measurements and comparisons
- Spectrum management controls: **Inherit from...**, **Revert**, and **Apply**
- Menu options for modifying plot display characteristics
- Menu options for printing spectrum data

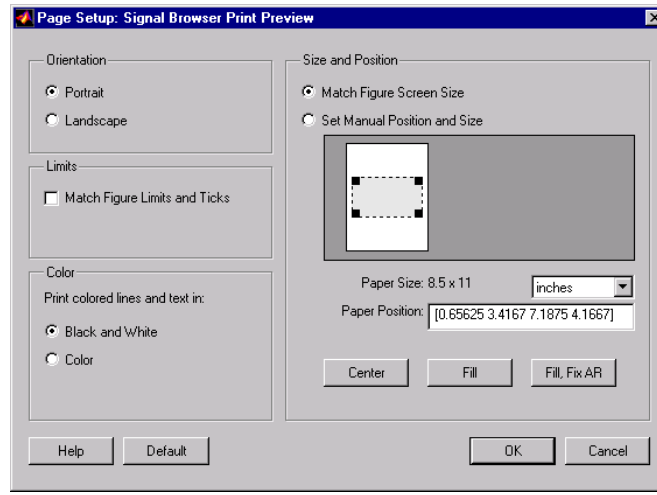


Meta Platforms, Inc. - Exhibit 1009
Page 278 of 720

Menus



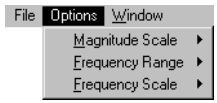
File Menu. Use **Page Setup...** from the **File** menu to open the standard MATLAB **Page Setup** dialog box (see *Using MATLAB Graphics*), in which you can set the orientation, size and position, limits, and color of your printout.



Use **Print Preview...** from the **File** menu to open a MATLAB figure window with a preview of the information from the Spectrum Viewer window that will appear in your printout. See “Spectrum Viewer Settings” on page 5-24 and “Printing Spectrum Data” on page 5-107 for details of printing from the Spectrum Viewer.

Use **Print...** from the **File** menu to open the standard operating system print dialog box, from which you can print your spectrum data.

Use **Close** from the **File** menu to close the Spectrum Viewer. All spectrum selection and ruler information will be lost. Settings you changed and saved using the **Preferences** dialog box in SPTool are saved and used the next time you open the Spectrum Viewer.



Options Menu. Use these options to change scaling and range parameters for plots in the Spectrum Viewer.

Click on the option you want to edit and drag to select a value.

Meta Platforms, Inc. - Exhibit 1009
Page 279 of 720

The options include:

- **Magnitude Scale**: Scaling for the magnitude plot may be **dB** or **linear**.
- **Frequency Range**: The range for the frequency axis may be **[0, Fs/2]**, **[0, Fs]**, or **[-Fs/2, Fs/2]**, where **Fs** is the sampling frequency. If multiple spectra are displayed, the value of **Fs** is the maximum of all the sampling frequencies.

  `Fs` is not defined for the case of a spectrum whose signal is `<None>`, that is, a spectrum whose associated signal has been deleted from SPTool. In this case, a value twice the highest frequency in the spectrum's frequency vector is chosen.

  The frequency range cannot be negative if **Frequency Scale** is set to **log**.
- **Frequency Scale**: Scaling for the frequency axis may be **linear** or **log**.

Window Menu. Use the **Window** menu to select a currently open MATLAB figure window.

Signal ID Panel

This panel displays information about the signal linked to the currently selected spectrum. The information includes the signal's name, size, data type (`real` or `complex`), and sampling frequency. To change any of these signal properties, use SPTool.

To associate a completely new signal with a displayed spectrum, select the signal in SPTool and press **Update** in the **Spectra** panel.

Spectrum Management Buttons

Inherit from... Choose a spectrum from this menu to let the active spectrum inherit its parameters (not including the associated signal).

Press **Inherit from...** and drag to select the spectrum from which you want to inherit parameters.

Revert. Restores the properties of the current spectrum to what they were the last time **Apply** was pressed.

Apply. Compute and display the active spectrum using the parameters set in the **Parameters** panel.

Meta Platforms, Inc. - Exhibit 1009
Page 280 of 720

Zoom Controls

The available zoom controls in the Spectrum Viewer are **Mouse Zoom**, **Full View**, **Zoom In-Y**, **Zoom Out-Y**, **Zoom In-X**, and **Zoom Out-X**. See "Zoom Controls" on page 5-31 for details on using the zoom controls in the Spectrum Viewer.

Ruler and Line Display Controls

Using the rulers and line-display controls, you can measure a variety of characteristics of spectra in the Spectrum Viewer. See "Ruler Controls" on page 5-33 for details on using rulers and modifying line displays in the Spectrum Viewer.

Help Button

To use context-sensitive help, press the **Help** button. The mouse pointer becomes an arrow with a question mark symbol. You can then click on anything in the Spectrum Viewer, including menu items, to find out what it is and how to use it.

Main Axes Display Area

The **Spectra** list in SPTool shows all spectra in the current SPTool session. One or more spectra may be selected. The spectral data of all selected spectra are displayed graphically in the main axes display area of the Spectrum Viewer.

NOTE If a spectrum is not displayed, or if it is displayed with the wrong signal information, press **Apply** to recompute the spectral data.

When there is only one spectrum displayed, its properties are displayed in the **Parameters** panel and its measurements are displayed in the **Rulers** panel. When more than one spectrum is displayed, select the line you want to focus on.

When a spectrum is selected, you can use the ruler controls on the selected line (see "Making Signal Measurements" on page 5-37) and you can modify its parameters (see below). The label of the selected spectrum (line) is displayed in the **Selection** pop-up menu.

Meta Platforms, Inc. - Exhibit 1009
Page 281 of 720

Select a spectrum (line) in one of two ways:

- Click on the **Selection** pop-up menu and drag to select the line to measure
- Move the mouse pointer over any point in the line you want to select and click on it

See "Selecting a Line to Measure" on page 5-34 for details.

Click-and-Drag Panning. You can use the mouse to pan around the main axes display:

Click on a line in the main axes, hold down the mouse button, and drag the mouse.

Click-and-drag panning is not enabled in mouse zoom mode.

Making Spectrum Measurements

Use the rulers to make a variety of measurements on the selected spectrum. See "Making Signal Measurements" on page 5-37 for details.

Viewing Spectral Density Plots

Spectral density estimation is a technique that finds the approximate frequency content of a signal. The Spectrum Viewer calculates single-channel power spectral density (PSD). When you first generate a spectrum, the Spectrum Viewer shows a default power spectral density function of the input data. By default, the Spectrum Viewer uses the Welch method of PSD estimation with a length-256 Hanning window and an FFT length of 1024.

You can change plot properties and computation parameters for a displayed spectrum, and you can set confidence intervals.

Controlling and Manipulating Plots

Changing Plot Properties

You can control the axes units and scaling properties that affect the Spectrum Viewer's plots.

Meta Platforms, Inc. - Exhibit 1009
Page 282 of 720

Use the **Options** menu to select:

- Linear or decibel scaling for the magnitude axis
- Linear or logarithmic scaling for the frequency axis
- The frequency range to view

See “Options Menu” on page 5-99 for details.

You can also zoom in on any of the Spectrum Viewer’s plots. See “Zoom Controls” on page 5-31 for details.

You can set other scaling properties in the **Parameters** panel, depending on the PSD method computation parameters you choose.

Choosing Computation Parameters

The Spectrum Viewer lets you control the PSD estimation parameters of the selected spectrum. Different parameters are available, depending on which method of PSD computation you choose. Set these parameters from the **Parameters** panel, as illustrated in the following steps:

1. Click on the **Method** pop-up menu and drag to select one of the following methods:
   - Burg
   - Covariance
   - FFT
   - Modified Covariance
   - MTM
   - MUSIC
   - Welch
   - Yule AR

   Appropriate parameter selections are displayed for each method you choose.

Meta Platforms, Inc. - Exhibit 1009
Page 283 of 720

2 Modify the appropriate parameters.

- When a parameter is in a pop-up menu, click on the parameter label and drag to select a value from the menu.
- When a parameter is in an edit box, type the value or variable into the box.

You can also modify the parameters by using **Inherit from** to copy the parameters of another spectrum in SPTool. See "Inherit from..." on page 5-100 for details.

3 If you change your mind, you can discard changes you make by pressing **Revert**.

4 To apply the modified parameters, press **Apply**.

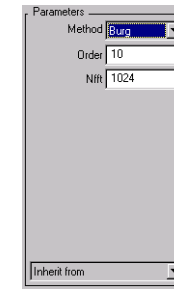
The new parameters are applied to the selected spectrum; the Spectrum Viewer recalculates the spectral density function and displays the modified spectrum.

Computation Methods and Parameters

You can choose from seven PSD computation methods. Each method has its own set of parameters.

This section shows the **Parameters** panel for each of the PSD computation methods. For detailed definitions and values for each parameter, use context-sensitive help (see "Help Button" on page 5-101).

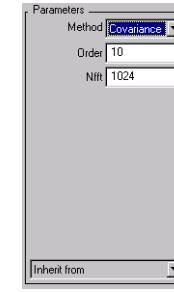
Burg. For the Burg method, you can specify the following parameters:



- **Order**
- **Nfft**

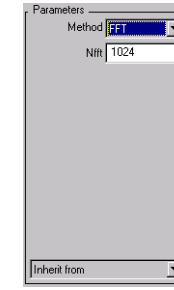
Meta Platforms, Inc. - Exhibit 1009
Page 284 of 720

Covariance. For the Covariance method, you can specify the following parameters:



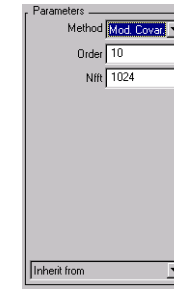
- **Order**
- **Nfft**

FFT. For the FFT method, you can specify the following parameter:



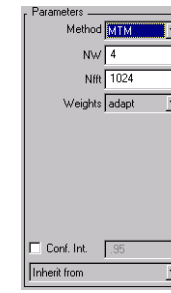
- **Nfft**

Mod. Covar. For the Modified Covariance method, you can specify the following parameters:



- **Order**
- **Nfft**

MTM. For the MTM method, you can specify the following parameters:



- **NW**
- **Nfft**

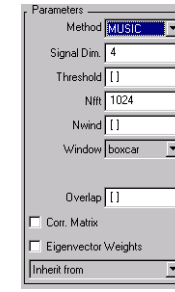
Meta Platforms, Inc. - Exhibit 1009
Page 285 of 720

- **Weights**

  Select one of the following from the pop-up menu:

  - **adapt**
  - **unity**
  - **eigen**
- **Conf. Int.**

  Check to compute a confidence interval and type in a value (see "Setting Confidence Intervals" on page 5-107).

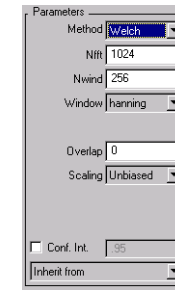


MUSIC. For the MUSIC method, you can specify the following parameters:

- **Signal Dim.**
- **Threshold**
- **Nfft**
- **Nwind**
- **Window**
- **Overlap**
- **Corr. Matrix**

  Check if selected signal is a correlation matrix.
- **Eigenvector Weights**

  Check to select eigenvector weights.



Welch. For Welch's method, you can specify the following parameters:

- **NFFT**
- **Nwind**
- **Window**
- **Overlap**
- **Scaling**

  Select one of the following from the pop-up menu:

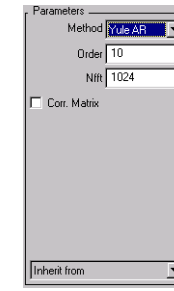
  - **Unbiased**
  - **Peaks**

Meta Platforms, Inc. - Exhibit 1009
Page 286 of 720

- **Conf. Int.**

  Check to compute a confidence interval and type in a value (see “Setting Confidence Intervals” on page 5-107).

Yule AR. For the Yule AR method, you can specify the following parameters:



- **Order**
- **Nfft**
- **Corr. Matrix**

  Check if selected signal is a correlation matrix.

Setting Confidence Intervals

By default, the Spectrum Viewer does not compute confidence intervals for spectral density. You can enable the computation of confidence intervals for the Welch and MTM methods by following these steps:

1. Click the **Conf. Int.** check box so that it is selected.
2. Type a value for the confidence level in the **Conf. Int.** edit box.

   This value must be a scalar between 0 and 1.
3. Press **Apply**.

NOTE Confidence intervals are reliable only for nonoverlapping sections.

Printing Spectrum Data

You can print all of the spectrum data that you can display and manipulate in the Spectrum Viewer. Your printout will always contain:

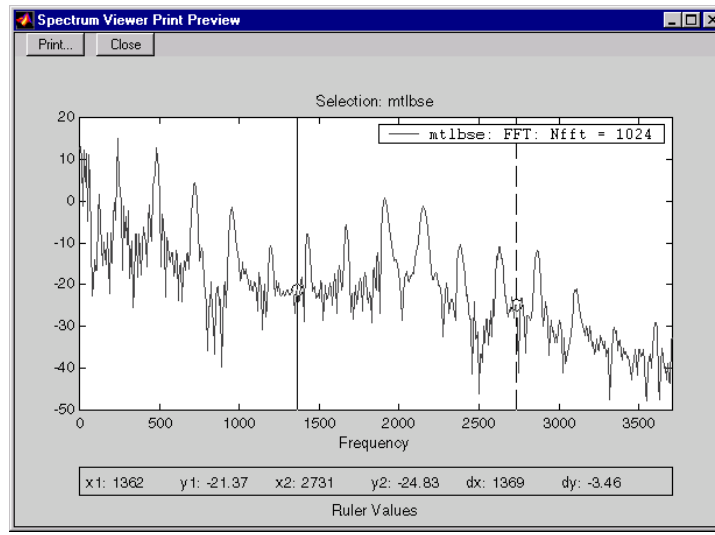
- The plots of one or more spectra, as displayed in the main axes area
- A legend displayed in the main axes area to label each spectrum by name, line color/style, method, and FFT length

The ruler measurements of the currently selected spectrum will be displayed at the bottom of the figure, unless you suppress them with the **Rulers** check

Meta Platforms, Inc. - Exhibit 1009
Page 287 of 720

box in the **Spectrum Viewer** preferences panel of the **Preferences** dialog box in SPTool (see "Spectrum Viewer Settings" on page 5-24).

You can preview your printout by choosing **Print Preview...** from the **File** menu in the Spectrum Viewer window. This opens the **Spectrum Viewer Print Preview** window, shown below with default settings for the spectrum `mtlbse`.



The **Close** button on the **Spectrum Viewer Print Preview** window closes the print preview window and returns you to the Spectrum Viewer.

The **Print...** button on the **Spectrum Viewer Print Preview** window opens the standard print dialog box, from which you can print the contents of your print preview window.

If you choose to print from the print dialog, both that dialog and the **Spectrum Viewer Print Preview** window will close, and the contents of the print preview will print.

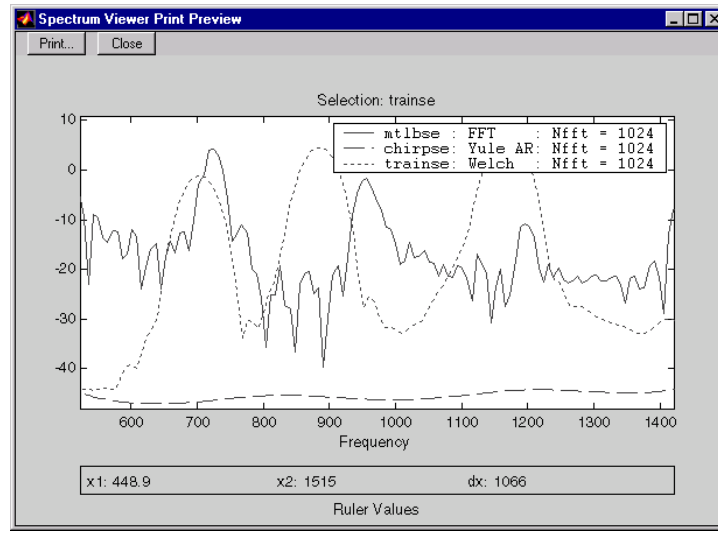
If you choose not to print, the print dialog and the **Spectrum Viewer Print Preview** window will close, and you will return to the Spectrum Viewer.

Any changes you make, prior to opening the print preview window, that affect the display of data in the Spectrum Viewer (see "Controlling and Manipulating Plots" on page 5-102) will affect the display of data in the **Spectrum Viewer**

Meta Platforms, Inc. - Exhibit 1009
Page 288 of 720

Print Preview window in exactly the same way. For example, in the print preview shown below:

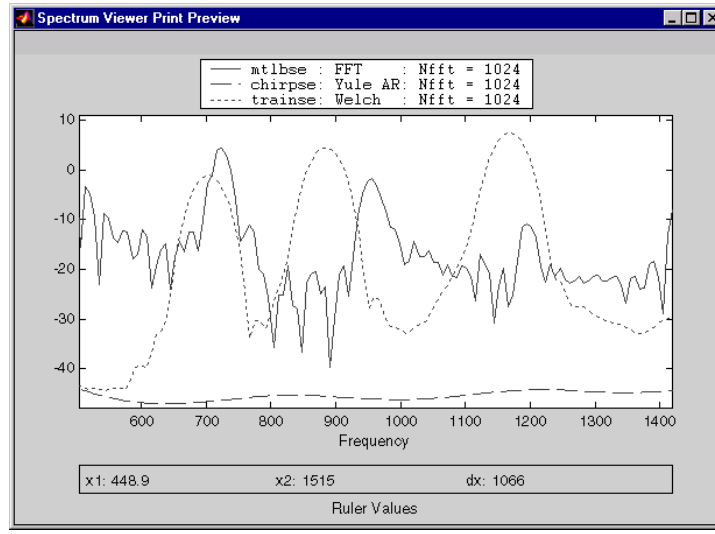
- Data from three different sample spectra have been selected for display in the same main axes area (see “Opening the Filter Viewer” on page 5-84).
- Using the tools in the **Edit Line** pop-up menu (see “Ruler and Line Display Controls” on page 5-101), the spectrum lines have all been changed to black and their line styles have been differentiated, as is reflected in the legend.
- The plots of all three spectra have been zoomed to an interval between their rulers (see “Zoom Controls” on page 5-101). Note that the ruler measurements displayed are for the `trainse` spectrum, which is the currently selected spectrum.



Changes you make in the **Preferences** dialog box in SPTool will also affect the **Spectrum Viewer Print Preview** window in the same way that they affect the Spectrum Viewer itself. For example, in the print preview shown below, the

Meta Platforms, Inc. - Exhibit 1009
Page 289 of 720

ruler measurements have been suppressed by changes in the Spectrum Viewer preferences (see “Spectrum Viewer Settings” on page 5-24).



NOTE The `HandleVisibility` property of the **Print Preview** figure window is set to `'on'`. This allows you to modify the preview using Handle Graphics commands from the MATLAB command line or the Plot Editor tools (see *Using MATLAB Graphics* for more information). For example, you can drag the legend to a different position within the figure, as has been done in the figure shown above

Changes that you make in the preview window print exactly as they appear on the screen; however, they are *not* saved when the **Print Preview** figure window is closed.

Saving Spectrum Data

After creating a spectrum in SPTool, you can export spectrum information to the workspace or to disk using **Export...** from the **File** menu in SPTool. The spectrum information is stored in a structure that you can access to retrieve the spectral power and frequency data. The spectrum structure also contains a number of fields that are used internally by SPTool.

Meta Platforms, Inc. - Exhibit 1009
Page 290 of 720

To see the fields of the spectrum structure, try exporting a spectrum to the workspace:

1. Create a new spectrum if none are currently loaded. Label the spectrum `spect1`.
2. In SPTool, select **Export...** from the **File** menu.
3. In the **Export** dialog box, select `spect1` and press the **Export to Workspace** button.
4. Type `who` at the MATLAB command line to look at the variables in the workspace. The variable called `spect1` is the spectrum structure you exported from SPTool.
5. Type `spect1` to list the fields of the spectrum structure.

The following structure fields describe the spectrum.

Field	Description
`P`	The spectral power vector.
`f`	The spectral frequency vector.
`confid`	A structure containing the confidence intervals data: • The `confid.level` field contains the chosen confidence level. • The `confid.Pc` field contains the spectral power data for the confidence intervals. • The `confid.enable` field contains a `1` if confidence levels are enabled for the spectrum.
`signalLabel`	The name of the signal from which the spectrum was generated.
`Fs`	The associated signal's sample rate.

Meta Platforms, Inc. - Exhibit 1009
Page 291 of 720

The other fields are used internally by SPTool, and are subject to change in future releases.

Meta Platforms, Inc. - Exhibit 1009
Page 292 of 720

Example: Generation of Bandlimited Noise

This section provides a complete example of using the GUI-based interactive tools to design and implement an FIR digital filter, apply it to a signal, and display signals and spectra. The steps include:

- Importing and naming a signal using SPTool
- Designing a filter using the Filter Designer
- In SPTool, applying the filter to the signal to create another signal
- Viewing the time domain information of the original and filtered signals using the Signal Browser
- Comparing the spectra of both signals using the Spectrum Viewer

Create, Import, and Name a Signal

You can import an existing signal into SPTool, or you can create a new signal and edit and name it in SPTool. In this step, you'll create a new signal at the command line and then import it into SPTool.

1. At the command line, create a random signal by typing:

   ```
   x = randn(5000,1);
   ```

2. Activate SPTool by typing:

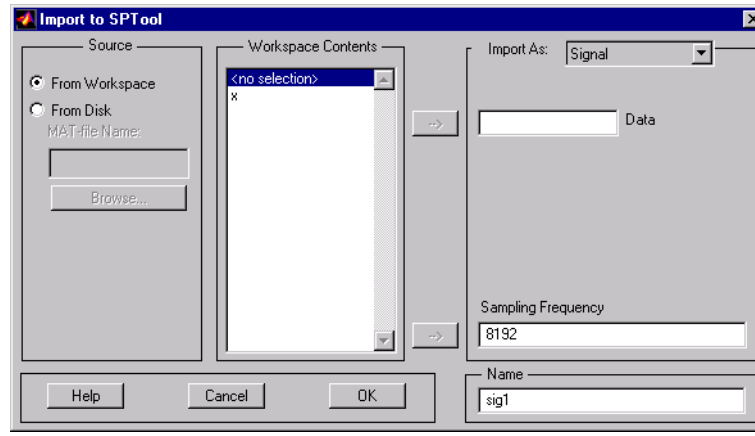
   ```
   sptool
   ```

   The SPTool window is displayed.

Meta Platforms, Inc. - Exhibit 1009
Page 293 of 720

3 Select **Import...** from the **File** menu:

The **Import to SPTool** window is displayed.



Notice that the variable `x` is displayed in the **Workspace Contents** list. (If it is not, click the **From Workspace** radio button to display the contents of the workspace.)

4 Name the signal and import it into SPTool:

a Make sure that **Signal** is selected in the **Import As** pop-up menu.

b Click in the **Data** field and type `x`.

You can also move the variable `x` into the **Data** field by clicking on **x** in the **Workspace Contents** list and then clicking on the arrow to the left of the **Data** field.

c Click in the **Sampling Frequency** field and type `5000`.

d Name the signal by clicking in the **Name** field and typing `noise`.

e Press **OK**.

The SPTool window is reactivated, and the signal `noise[vector]` is selected in the **Signals** list.

Meta Platforms, Inc. - Exhibit 1009
Page 294 of 720

Design a Filter

You can import an existing filter into SPTool, or you can design and edit a new filter using the Filter Designer. In this step, you'll create a default filter and customize it in the Filter Designer.

1 Press **New Design** in SPTool to activate the Filter Designer and generate a default filter.

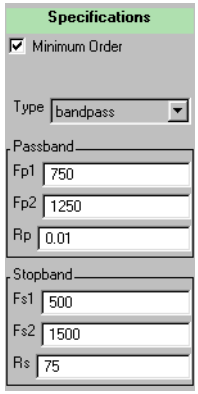
   The Filter Designer window is displayed with the default filter `filt1`.

2 Change the filter sampling frequency to 5000 by entering this value in the **Sampling Frequency** text box in the Filter Designer.

3 Specify the filter parameters shown at left:

   a Make sure **Equiripple FIR** is selected in the **Algorithm** pop-up menu.

   b Select **bandpass** from the **Type** pop-up menu.



   c Set the passband edge frequencies by entering 750 for **Fp1** and 1250 for **Fp2**.

   d Set the stopband edge frequencies by entering 500 for **Fs1** and 1500 for **Fs2**.

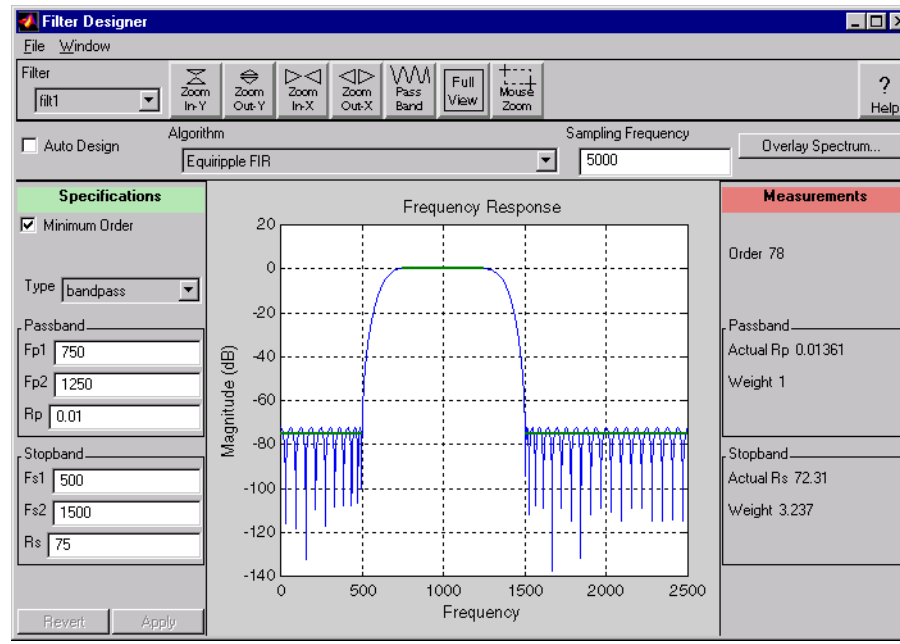
   e Type `.01` into the **Rp** field and 75 into the **Rs** field.

   Rp sets the maximum passband ripple and **Rs** sets the stopband attenuation for the filter.

   f Press the **Apply** button to compute the new filter.

Meta Platforms, Inc. - Exhibit 1009
Page 295 of 720

When the new filter is computed, the magnitude response of the filter is displayed with a solid line in the main axes display area.



The resulting filter is an order 78 bandpass equiripple filter.

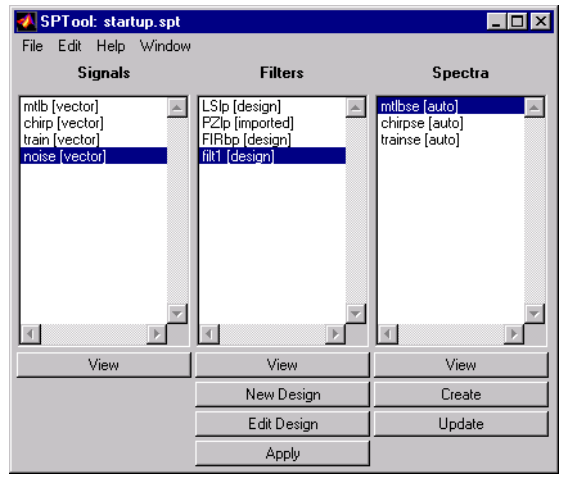
Apply the Filter to a Signal

In this step, you apply the filter to the signal in SPTool. The new, filtered signal is automatically created in SPTool.

Meta Platforms, Inc. - Exhibit 1009
Page 296 of 720

1 Activate SPTool from the **Window** menu in the Filter Designer.

2 Click to select the signal `noise[vector]` from the **Signals** list and click to select the filter (named `filt1[design]`) from the **Filters** list, as shown below.



3 Press **Apply** to apply the filter `filt1` to the signal `noise`.

The **Apply Filter** dialog box is displayed.

4 Name the new signal by clicking in the **Output Signal** field and typing `blnoise`.

5 Press **OK**.

The filter is applied to the selected signal. The new, filtered signal `blnoise[vector]` is displayed in the **Signals** list.

View, Play, and Print the Signals

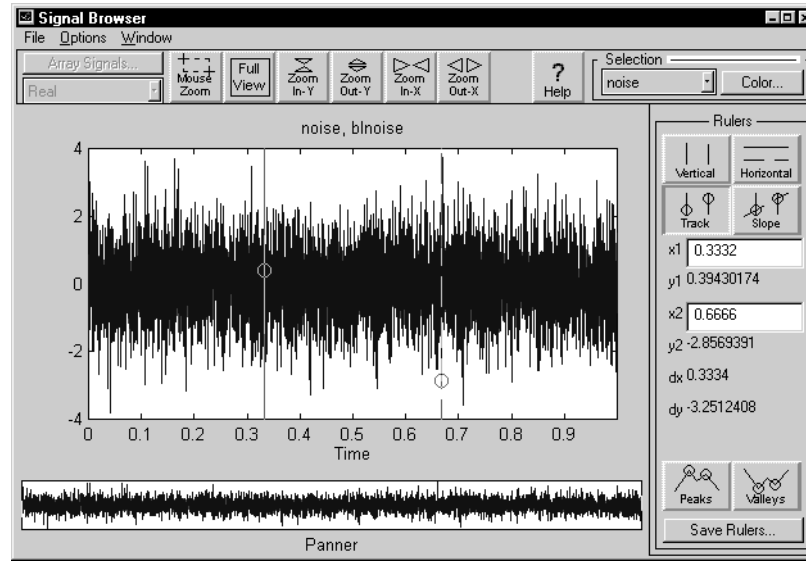
You can view and print the time domain information of the signals using the Signal Browser. You can also play the signals, if your computer has audio output capabilities. In this step, you'll display both signals in the Signal Browser, select and play each signal, and print both signals.

Meta Platforms, Inc. - Exhibit 1009
Page 297 of 720

1 **Shift**-click on the `noise` and `blnoise` signals in the **Signals** list of SPTool to select both signals.

2 Press **View** in the **Signals** panel.

The Signal Browser is activated and both signals are displayed in the main axes display area. Initially, the `noise` signal covers up the bandlimited `blnoise` signal, but you can see that both signals are displayed because the names of both signals are shown above the main axes display area.

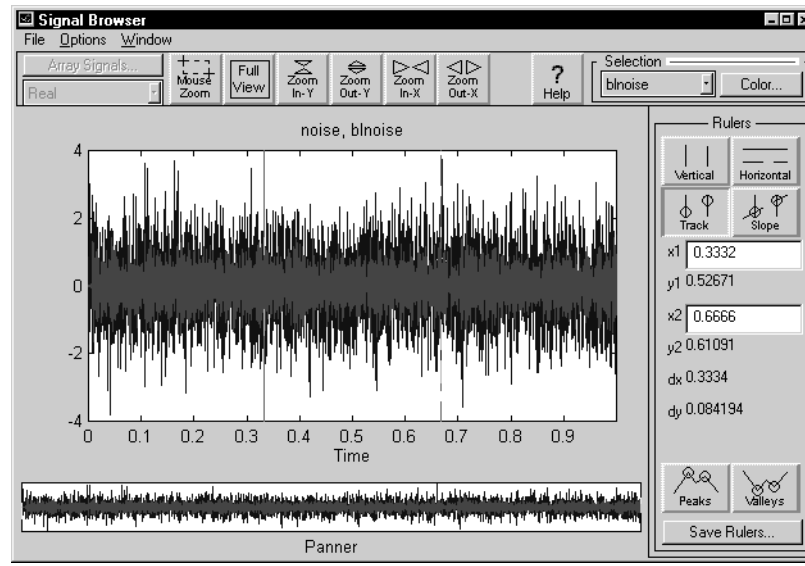


3 Click-and-drag in the **Selection** pop-up menu to select the `blnoise` signal.

The main axes display area is redisplayed. Now you can see the `blnoise` signal superimposed on top of the `noise` signal. The signals are displayed in different colors in both the main axes display area and the panner. Notice

Meta Platforms, Inc. - Exhibit 1009
Page 298 of 720

that the color of the line in the **Selection** display changes to correspond to the color of the signal that you've selected.



The signal that's displayed in the **Selection** pop-up menu and in the **Selection** display is the active signal. When you select **Play**, or use the rulers, the active signal is the one that is played or measured.

4 To hear the active signal, select **Play** from the **Options** menu.

5 To hear the other signal, select it as in step 3 above (you can also select the signal by clicking on it in the main axes display area). Select **Play** from the **Options** menu again.

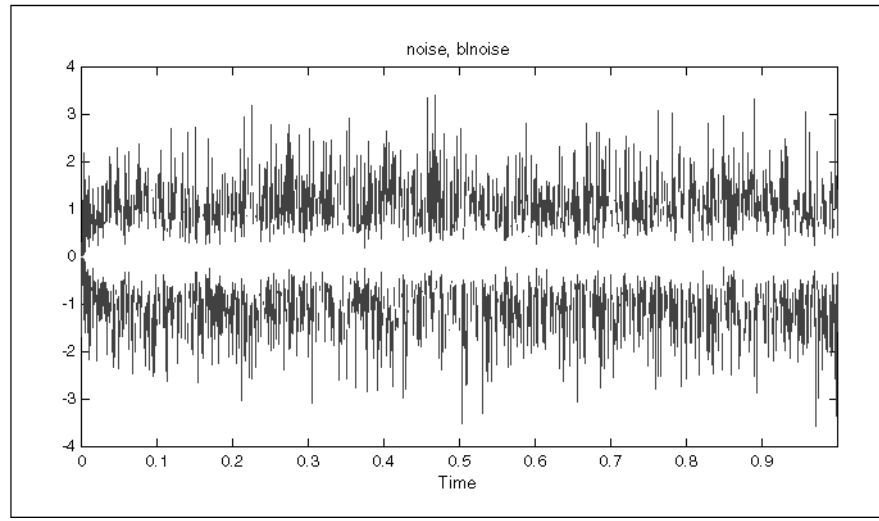
6 Before printing the two signals together, use the line controls to maximize the visual contrast between the signals by setting the line color for `noise` to

Meta Platforms, Inc. - Exhibit 1009
Page 299 of 720

gray and the line color for `blnoise` to white. (See "Ruler Controls" on page 5-33 for details on using these controls.)

Use the Signal Browser panel in the **Preferences** dialog box in SPTool to suppress printing of both the panner and the ruler settings. (See "Signal Browser Settings" on page 5-23 for details on these preferences.)

To print both signals, as seen in the picture below, select **Print...** from the **File** menu in the Signal Browser.



Compare Spectra of Both Signals

You can get an idea of the frequency content of the two signals by displaying their power spectra using the Spectrum Viewer, as described below.

1. Reactivate SPTool by selecting it from the **Window** menu of the Signal Browser.

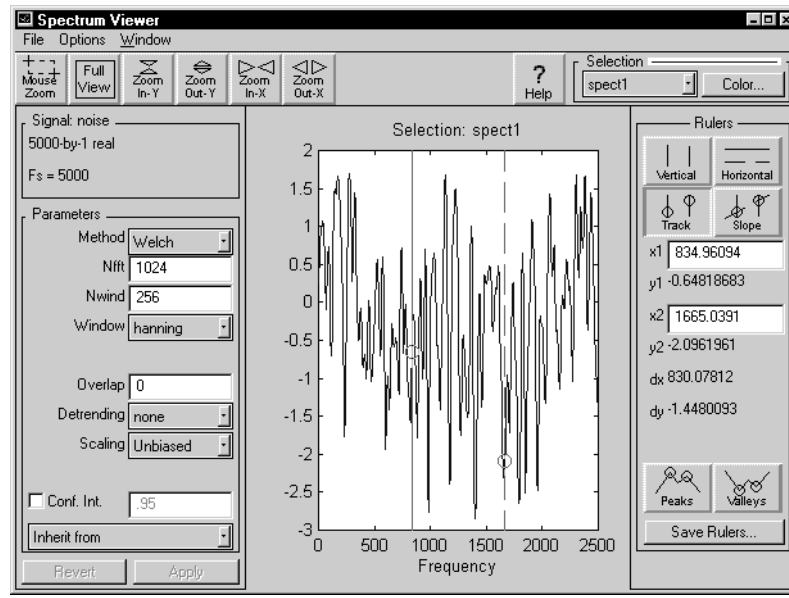
2. Click on the `noise[vector]` signal in the **Signals** list of SPTool to select it.

Meta Platforms, Inc. - Exhibit 1009
Page 300 of 720

3 Press **Create** in the **Spectra** panel.

The Spectrum Viewer is activated, and a spectrum object (`spect1`) corresponding to the `noise` signal is created in the **Spectra** list. The spectrum is not computed or displayed yet.

4 Press **Apply** in the Spectrum Viewer to compute and display `spect1`. The spectrum of the `noise` signal is displayed in the main axes display area.



Notice that the spectrum's signal identification information – including its name, its type, and its sampling frequency – is displayed above the **Parameters** panel, and the spectrum's name is displayed both above the main axes display area and in the **Selection** pop-up menu.

The spectrum estimate is within 2 or 3 dB of 0, so the noise has a fairly "flat" spectrum.

5 Reactivate SPTool by selecting it from the **Window** menu in the Spectrum Viewer.

6 Click on the `blnoise` signal in the **Signals** list of SPTool to select it.

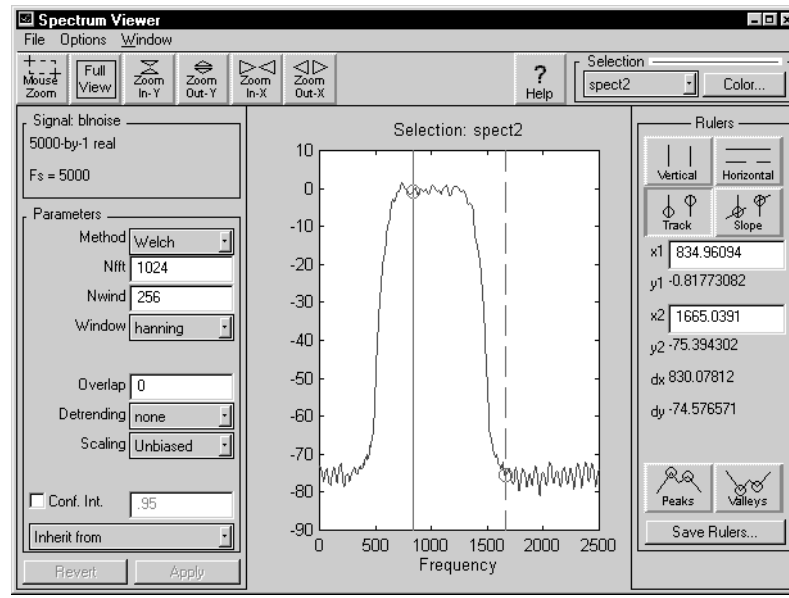
Meta Platforms, Inc. - Exhibit 1009
Page 301 of 720

7 Press **Create** in the **Spectra** panel.

The Spectrum Viewer is again activated, and a spectrum object (`spect2`) corresponding to the `blnoise` signal is created in the **Spectra** list. The spectrum is not computed or displayed yet.

8 Press **Apply** in the Spectrum Viewer to display `spect2`.

The spectrum of the `blnoise` signal is displayed in the main axes display area.



The new spectrum's signal identification information – including its name, its type, and its sampling frequency – is displayed above the **Parameters** panel, and the spectrum's name is displayed both above the main axes display area and in the **Selection** pop-up menu.

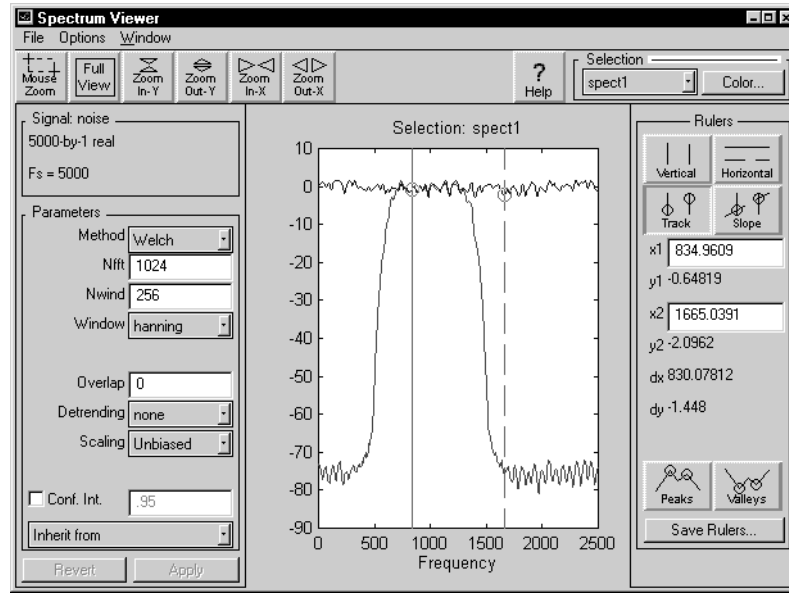
This spectrum is flat between 750 and 1250 Hz and has 75 dB less power in the stopband regions of `filt1`.

9 Reactivate SPTool again, as in step 5 above.

Meta Platforms, Inc. - Exhibit 1009
Page 302 of 720

10 **Shift**-click on `spect1` and `spect2` in the **Spectra** list to select them both.

11 Press **View** in the **Spectra** panel to reactivate the Spectrum Viewer and display both spectra together.



12 To select one of the spectra for measuring or editing, use the **Selection** pop-up menu, or click on the spectrum in the main axes display area.

The color of the line in the **Selection** display changes to correspond to the color of the spectrum that you've selected.

The spectrum that's displayed in the **Selection** pop-up menu and in the **Selection** display is the active spectrum. When you use the rulers or change parameters, the active spectrum is the one that is measured or modified.

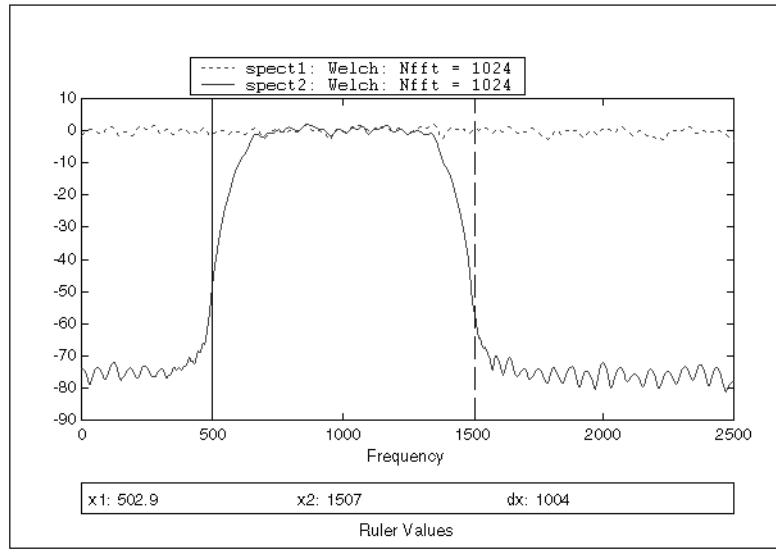
13 Before printing the two spectra together, use the selection and ruler controls to differentiate the two plots by line style, rather than by color. Drag the rulers to demark the stopband edge frequencies and change the rulers from

Meta Platforms, Inc. - Exhibit 1009
Page 303 of 720

Track to **Vertical**. (See “Ruler Controls” on page 5-33 for details on using these controls.)

Select **Print Preview...** from the **File** menu in the Spectrum Viewer. From the **Spectrum Viewer Print Preview** window, drag the legend out of the axes display area so that it doesn’t obscure part of the plot.

To print both signals, as seen in the picture below, select the **Print...** button on the **Spectrum Viewer Print Preview** window.



Meta Platforms, Inc. - Exhibit 1009
Page 304 of 720

6

Reference

This chapter contains detailed descriptions of all Signal Processing Toolbox functions. It begins with a list of functions grouped by subject area and continues with the reference entries in alphabetical order. For more information, see the online *MATLAB Function Reference*.

Waveform Generation and Plotting	
`chirp`	Swept-frequency cosine generator.
`diric`	Dirichlet or periodic sinc function.
`gauspuls`	Gaussian-modulated sinusoidal pulse generator.
`pulstran`	Pulse train generator.
`rectpuls`	Sampled aperiodic rectangle generator.
`sawtooth`	Sawtooth or triangle wave generator.
`sinc`	Sinc function.
`square`	Square wave generator.
`strips`	Strip plot.
`tripuls`	Sampled aperiodic triangle generator.

Filter Analysis and Implementation	
`abs`	Absolute value (magnitude).
`angle`	Phase angle.
`conv`	Convolution and polynomial multiplication.
`conv2`	Two-dimensional convolution.
`fftfilt`	FFT-based FIR filtering using the overlap-add method.
`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.

Meta Platforms, Inc. - Exhibit 1009
Page 306 of 720

Filter Analysis and Implementation	
`filter2`	Two-dimensional digital filtering.
`filtfilt`	Zero-phase digital filtering.
`filtic`	Find initial conditions for a transposed direct form II filter implementation.
`freqs`	Frequency response of analog filters.
`freqspace`	Frequency spacing for frequency response.
`freqz`	Frequency response of digital filters.
`grpdelay`	Average filter delay (group delay).
`impz`	Impulse response of digital filters.
`latcfilt`	Lattice and lattice-ladder filter implementation.
`sgolayfilt`	Savitzky-Golay filtering.
`sosfilt`	Second-order (biquadratic) IIR filtering.
`unwrap`	Unwrap phase angles.
`zplane`	Zero-pole plot.

Linear System Transformations	
`ac2poly`	Conversion of autocorrelation sequence to prediction polynomial.
`ac2rc`	Conversion of autocorrelation sequence to reflection coefficients.
`convmtx`	Convolution matrix.
`latc2tf`	Lattice filter to transfer function conversion.

Meta Platforms, Inc. - Exhibit 1009
Page 307 of 720

Linear System Transformations	
poly2ac	Conversion of prediction polynomial to autocorrelation sequence.
poly2rc	Conversion of prediction polynomial to reflection coefficients.
rc2ac	Conversion of reflection coefficients to autocorrelation sequence.
rc2poly	Conversion of reflection coefficients to prediction polynomial.
residuez	z-transform partial-fraction expansion.
sos2ss	Conversion of second-order sections to state-space.
sos2tf	Conversion of second-order sections to transfer function.
sos2zp	Conversion of second-order sections to zero-pole-gain.
ss2sos	Conversion of state-space to second-order sections.
ss2tf	Conversion of state-space to transfer function.
ss2zp	Conversion of state-space to zero-pole-gain.
tf2latc	Conversion of transfer function to lattice filter.
tf2sos	Conversion of transfer function to second-order sections.
tf2ss	Conversion of transfer function to state-space.
tf2zp	Conversion of transfer function to zero-pole-gain.
zp2sos	Conversion of zero-pole-gain to second-order sections.
zp2ss	Conversion of zero-pole-gain to state-space.
zp2tf	Conversion of zero-pole-gain to transfer function.

Meta Platforms, Inc. - Exhibit 1009
Page 308 of 720

IIR Filter Design—Classical and Direct	
`besself`	Bessel analog filter design.
`butter`	Butterworth analog and digital filter design.
`cheby1`	Chebyshev type I filter design (passband ripple).
`cheby2`	Chebyshev type II filter design (stopband ripple).
`ellip`	Elliptic (Cauer) filter design.
`maxflat`	Generalized digital Butterworth filter design.
`yulewalk`	Recursive digital filter design.

IIR Filter Order Selection	
`buttord`	Butterworth filter order selection.
`cheb1ord`	Chebyshev type I filter order selection.
`cheb2ord`	Chebyshev type II filter order selection.
`ellipord`	Elliptic filter order selection.

FIR Filter Design	
`cremez`	Complex and nonlinear-phase equiripple FIR filter design.
`fir1`	Window-based finite impulse response filter design – standard response.
`fir2`	Window-based finite impulse response filter design – arbitrary response.
`fircls`	Constrained least square FIR filter design for multiband filters.

Meta Platforms, Inc. - Exhibit 1009
Page 309 of 720

FIR Filter Design	
`fircls1`	Constrained least square filter design for lowpass and highpass linear phase FIR filters.
`firls`	Least square linear-phase FIR filter design.
`firrcos`	Raised cosine FIR filter design.
`intfilt`	Interpolation FIR filter design.
`kaiserord`	Estimate parameters for an FIR filter design with Kaiser window.
`remez`	Parks-McClellan optimal FIR filter design.
`remezord`	Parks-McClellan optimal FIR filter order estimation.
`sgolay`	Savitzky-Golay filter design.

Transforms	
`czt`	Chirp z-transform.
`dct`	Discrete cosine transform (DCT).
`dftmtx`	Discrete Fourier transform matrix.
`fft`	One-dimensional fast Fourier transform.
`fft2`	Two-dimensional fast Fourier transform.
`fftshift`	Rearrange the outputs of the FFT functions.
`hilbert`	Hilbert transform.
`idct`	Inverse discrete cosine transform.
`ifft`	One-dimensional inverse fast Fourier transform.
`ifft2`	Two-dimensional inverse fast Fourier transform.

Meta Platforms, Inc. - Exhibit 1009
Page 310 of 720

Statistical Signal Processing	
`cohere`	Estimate magnitude squared coherence function between two signals.
`corrcoef`	Correlation coefficient matrix.
`cov`	Covariance matrix.
`csd`	Estimate the cross spectral density (CSD) of two signals.
`pburg`	Power spectrum estimate using the Burg method.
`pcov`	Power spectrum estimate using the covariance method.
`pmcov`	Power spectrum estimate using the modified covariance method.
`pmtm`	Power spectrum estimate using the multitaper method (MTM).
`pmusic`	Power spectrum estimate using MUSIC eigenvector method.
`pwelch`	Estimate the power spectral density (PSD) of a signal using Welch's method.
`pyulear`	Power spectrum estimate using Yule-Walker AR method.
`tfe`	Transfer function estimate from input and output.
`xcorr`	Cross-correlation function estimate.
`xcorr2`	Two-dimensional cross-correlation.
`xcov`	Cross-covariance function estimate (equal to mean-removed cross-correlation).

Meta Platforms, Inc. - Exhibit 1009
Page 311 of 720

Windows	
bartlett	Bartlett window.
blackman	Blackman window.
boxcar	Rectangular window.
chebwin	Chebyshev window.
hamming	Hamming window.
hanning	Hanning window.
kaiser	Kaiser window.
triang	Triangular window.

Parametric Modeling	
arburg	Compute an estimate of AR model parameters using the Burg method.
arcov	Compute an estimate of AR model parameters using the covariance method.
armcov	Compute an estimate of AR model parameters using the modified covariance method.
aryule	Compute an estimate of AR model parameters using the Yule-Walker method.
invfreqs	Continuous-time (analog) filter identification from frequency data.
invfreqz	Discrete-time filter identification from frequency data.
levinson	Levinson-Durbin recursion.
lpc	Linear prediction coefficients.
prony	Prony's method for time domain IIR filter design.

Meta Platforms, Inc. - Exhibit 1009
Page 312 of 720

Parametric Modeling	
`rlevinson`	Reverse Levinson-Durbin recursion.
`stmcb`	Linear model using Steiglitz-McBride iteration.

Specialized Operations	
`buffer`	Buffer a signal vector into a matrix of data frames.
`cceps`	Complex cepstral analysis.
`cplxpair`	Group complex numbers into complex conjugate pairs.
`decimate`	Decrease the sampling rate for a sequence (decimation).
`deconv`	Deconvolution and polynomial division.
`demod`	Demodulation for communications simulation.
`dpss`	Discrete prolate spheroidal sequences (Slepian sequences).
`dpssclear`	Remove discrete prolate spheroidal sequences from database.
`dpssdir`	Discrete prolate spheroidal sequences database directory.
`dpssload`	Load discrete prolate spheroidal sequences from database.
`dpsssave`	Save discrete prolate spheroidal sequences in database.
`icceps`	Inverse complex cepstrum.
`interp`	Increase sampling rate by an integer factor (interpolation).
`medfilt1`	One-dimensional median filtering.
`modulate`	Modulation for communications simulation.
`polystab`	Stabilize polynomial.
`rceps`	Real cepstrum and minimum phase reconstruction.

Meta Platforms, Inc. - Exhibit 1009
Page 313 of 720

Specialized Operations	
`resample`	Change sampling rate by any rational factor.
`specgram`	Time-dependent frequency analysis (spectrogram).
`upfirdn`	Upsample, apply an FIR filter, and downsample.
`vco`	Voltage controlled oscillator.

Analog Prototype Design	
`besselap`	Bessel analog lowpass filter prototype.
`buttap`	Butterworth analog lowpass filter prototype.
`cheb1ap`	Chebyshev type I analog lowpass filter prototype.
`cheb2ap`	Chebyshev type II analog lowpass filter prototype.
`ellipap`	Elliptic analog lowpass filter prototype.

Frequency Translation	
`lp2bp`	Lowpass to bandpass analog filter transformation.
`lp2bs`	Lowpass to bandstop analog filter transformation.
`lp2hp`	Lowpass to highpass analog filter transformation.
`lp2lp`	Lowpass to lowpass analog filter transformation.

Meta Platforms, Inc. - Exhibit 1009
Page 314 of 720

Filter Discretization	
`bilinear`	Bilinear transformation method of analog-to-digital filter conversion.
`impinvar`	Impulse invariance method of analog-to-digital filter conversion.

Interactive Tools	
`sptool`	Interactive digital signal processing tool (SPTool).

Meta Platforms, Inc. - Exhibit 1009
Page 315 of 720

Purpose Absolute value (magnitude).

Syntax

```
y = abs(x)
```

Description `y = abs(x)` returns the absolute value of the elements of `x`. If `x` is complex, `abs` returns the complex modulus (magnitude):

```
abs(x) = sqrt(real(x).^2 + imag(x).^2)
```

If `x` is a MATLAB string, `abs` returns the numeric values of the ASCII characters in the string. The display format of the string changes; the internal representation does not.

The `abs` function is part of the standard MATLAB language.

Example Calculate the magnitude of the FFT of a sequence:

```
t = (0:99)/100;                      % time vector
x = sin(2*pi*15*t) + sin(2*pi*40*t); % signal
y = fft(x);                          % compute DFT of x
m = abs(y);                          % magnitude
```

Plot the magnitude:

```
f = (0:length(y)-1)'/length(y)*100;  % frequency vector
plot(f,m)
```

See Also

`angle`	Phase angle.

Meta Platforms, Inc. - Exhibit 1009
Page 316 of 720

Purpose

Conversion of autocorrelation sequence to prediction polynomial.

Syntax

```
a = ac2poly(r)
[a,efinal] = ac2poly(r)
```

Description

`a = ac2poly(r)` finds the prediction polynomial, `a`, corresponding to the autocorrelation sequence `r`. `a` is the same length as `r`, and `a(1) = 1`.

`[a,efinal] = ac2poly(r)` returns the final prediction error, `efinal`.

Example

Consider the autocorrelation sequence

```
r = [5.0000   -1.5450   -3.9547   3.9331   1.4681   -4.7500];
```

The equivalent prediction polynomial is

```
a = ac2poly(r)

a =

1.0000    0.6147    0.9898    0.0004    0.0034   -0.0077
```

See Also

`ac2rc`	Conversion of autocorrelation sequence to reflection coefficients.
`poly2ac`	Conversion of prediction polynomial to autocorrelation sequence.
`rc2poly`	Conversion of prediction polynomial to reflection coefficients.

References

[1] Kay, S.M. *Modern Spectral Estimation*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

Meta Platforms, Inc. - Exhibit 1009
Page 317 of 720

Purpose Conversion of autocorrelation sequence to reflection coefficients.

Syntax

```
[k,r0] = ac2rc(r)
```

Description `[k,r0] = ac2rc(r)` finds the reflection coefficients, `k`, corresponding to the autocorrelation sequence `r`. `r0` contains the zero-lag autocorrelation.

See Also

`ac2poly`	Conversion of autocorrelation sequence to prediction polynomial.
`poly2rc`	Conversion of prediction polynomial to reflection coefficients.
`rc2ac`	Conversion of reflection coefficients to autocorrelation sequence.

References

[1] Kay, S.M. *Modern Spectral Estimation*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

Meta Platforms, Inc. - Exhibit 1009
Page 318 of 720

Purpose

Phase angle.

Syntax

```
p = angle(h)
```

Description

`p = angle(h)` returns the phase angles, in radians, of the elements of complex vector or array `h`. The phase angles lie between -π and π.

For complex sequence $h = x + iy = me^{ip}$, the magnitude and phase are given by

```
m = abs(h)
p = angle(h)
```

To convert to the original `h` from its magnitude and phase:

```
i = sqrt(-1)
h = m.*exp(i*p)
```

The `angle` function is part of the standard MATLAB language.

Example

Calculate the phase of the FFT of a sequence:

```
t = (0:99)/100;                      % time vector
x = sin(2*pi*15*t) + sin(2*pi*40*t); % signal
y = fft(x);                          % compute DFT of x
p = unwrap(angle(y));                % phase
```

Plot the phase:

```
f = (0:length(y)-1)'/length(y)*100;  % frequency vector
plot(f,p)
```

Algorithm

`angle` can be expressed as:

```
angle(x) = imag(log(x)) = atan2(imag(x),real(x))
```

See Also

`abs`	Absolute value (magnitude).

Meta Platforms, Inc. - Exhibit 1009
Page 319 of 720

Purpose

Compute an estimate of AR model parameters using the Burg method.

Syntax

```
a = arburg(x,p)
[a,e] = arburg(x,p)
[a,e,k] = arburg(x,p)
```

Description

`a = arburg(x,p)` uses the Burg method to fit a `p`-th order autoregressive (AR) model to the input signal, `x`, by minimizing (least squares) the forward and backward prediction errors while constraining the AR parameters to satisfy the Levinson-Durbin recursion. `x` is assumed to be the output of an AR system driven by white noise. Vector `a` contains the normalized estimate of the AR system parameters, $A(z)$, in descending powers of z.

$$H(z) = \frac{\sqrt{e}}{A(z)} = \frac{\sqrt{e}}{1 + a_2 z^{-1} + \ldots + a_{(p+1)} z^{-p}}$$

Since the method characterizes the input data using an all-pole model, the correct choice of the model order `p` is important.

`[a,e] = arburg(x,p)` returns the variance estimate, `e`, of the white noise input to the AR model.

`[a,e,k] = arburg(x,p)` returns a vector, `k`, of reflection coefficients.

See Also

`arcov`	Compute an estimate of AR model parameters using the covariance method.
`armcov`	Compute an estimate of AR model parameters using the modified covariance method.
`aryule`	Compute an estimate of AR model parameters using the Yule-Walker method.
`lpc`	Linear prediction coefficients.
`pburg`	Power spectrum estimate using the Burg method.
`prony`	Prony's method for time domain IIR filter design.

Meta Platforms, Inc. - Exhibit 1009
Page 320 of 720

Purpose

Compute an estimate of AR model parameters using the covariance method.

Syntax

```
a = arcov(x,p)
[a,e] = arcov(x,p)
```

Description

`a = arcov(x,p)` uses the covariance method to fit a `p`-th order autoregressive (AR) model to the input signal, `x`, which is assumed to be the output of an AR system driven by white noise. This method minimizes the forward prediction error in the least-squares sense. Vector `a` contains the normalized estimate of the AR system parameters, $A(z)$, in descending powers of z.

$$H(z) = \frac{\sqrt{e}}{A(z)} = \frac{\sqrt{e}}{1 + a_2 z^{-1} + \ldots + a_{(p+1)} z^{-p}}$$

Because the method characterizes the input data using an all-pole model, the correct choice of the model order `p` is important.

`[a,e] = arcov(x,p)` returns the variance estimate, `e`, of the white noise input to the AR model.

See Also

`arburg`	Compute an estimate of AR model parameters using the Burg method.
`armcov`	Compute an estimate of AR model parameters using the modified covariance method.
`aryule`	Compute an estimate of AR model parameters using the Yule-Walker method.
`lpc`	Linear prediction coefficients.
`pcov`	Power spectrum estimate using the covariance method.
`prony`	Prony's method for time domain IIR filter design.

Meta Platforms, Inc. - Exhibit 1009
Page 321 of 720

Purpose

Compute an estimate of AR model parameters using the modified covariance method.

Syntax

```
a = armcov(x,p)
[a,e] = armcov(x,p)
```

Description

`a = armcov(x,p)` uses the modified covariance method to fit a `p`-th order autoregressive (AR) model to the input signal, `x`, which is assumed to be the output of an AR system driven by white noise. This method minimizes the forward and backward prediction errors in the least-squares sense. Vector `a` contains the normalized estimate of the AR system parameters, $A(z)$, in descending powers of z.

$$H(z) = \frac{\sqrt{e}}{A(z)} = \frac{\sqrt{e}}{1 + a_2 z^{-1} + \ldots + a_{(p+1)} z^{-p}}$$

Because the method characterizes the input data using an all-pole model, the correct choice of the model order `p` is important.

`[a,e] = armcov(x,p)` returns the variance estimate, `e`, of the white noise input to the AR model.

See Also

`arburg`	Compute an estimate of AR model parameters using the Burg method.
`arcov`	Compute an estimate of AR model parameters using the covariance method.
`aryule`	Compute an estimate of AR model parameters using the Yule-Walker method.
`lpc`	Linear prediction coefficients.
`pmcov`	Power spectrum estimate using the modified covariance method.
`prony`	Prony's method for time domain IIR filter design.

Meta Platforms, Inc. - Exhibit 1009
Page 322 of 720

Purpose

Compute an estimate of AR model parameters using the Yule-Walker method.

Syntax

```
a = aryule(x,p)
[a,e] = aryule(x,p)
[a,e,k] = aryule(x,p)
```

Description

`a = aryule(x,p)` uses the Yule-Walker method, also called the autocorrelation method, to fit a `p`-th order autoregressive (AR) model to the windowed input signal, `x`, 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. `x` is assumed to be the output of an AR system driven by white noise. Vector `a` contains the normalized estimate of the AR system parameters, $A(z)$, in descending powers of z.

$$H(z) = \frac{\sqrt{e}}{A(z)} = \frac{\sqrt{e}}{1 + a_2 z^{-1} + \ldots + a_{(p+1)} z^{-p}}$$

Because the method characterizes the input data using an all-pole model, the correct choice of the model order `p` is important.

`[a,e] = aryule(x,p)` returns the variance estimate, `e`, of the white noise input to the AR model.

`[a,e,k] = aryule(x,p)` returns a vector, `k`, of reflection coefficients.

See Also

`arburg`	Compute an estimate of AR model parameters using the Burg method.
`arcov`	Compute an estimate of AR model parameters using the covariance method.
`armcov`	Compute an estimate of AR model parameters using the modified covariance method.
`lpc`	Linear prediction coefficients.
`prony`	Prony's method for time domain IIR filter design.
`pyulear`	Power spectrum estimate using Yule-Walker AR method.

Meta Platforms, Inc. - Exhibit 1009
Page 323 of 720

Purpose Bartlett window.

Syntax

```
w = bartlett(n)
```

Description `w = bartlett(n)` returns an `n`-point Bartlett window in the column vector `w`. The coefficients of a Bartlett window are:

- For `n` odd

$$w[k] = \begin{cases} \dfrac{2(k-1)}{n-1}, & 1 \le k \le \dfrac{n+1}{2} \\ 2 - \dfrac{2(k-1)}{n-1}, & \dfrac{n+1}{2} \le k \le n \end{cases}$$

- For `n` even

$$w[k] = \begin{cases} \dfrac{2(k-1)}{n-1}, & 1 \le k \le \dfrac{n}{2} \\ \dfrac{2(n-k)}{n-1}, & \dfrac{n}{2} + 1 \le k \le n \end{cases}$$

The Bartlett window is very similar to a triangular window as returned by the `triang` function. The Bartlett window always ends with zeros at samples 1 and `n`, however, while the triangular window is nonzero at those points. For `n` odd, the center `n–2` points of `bartle9tt(n)` are equivalent to `triang(n–2)`.

See Also

`blackman`	Blackman window.
`boxcar`	Rectangular window.
`chebwin`	Chebyshev window.
`hamming`	Hamming window.
`hanning`	Hanning window.
`kaiser`	Kaiser window.
`triang`	Triangular window.

References

[1] Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

Meta Platforms, Inc. - Exhibit 1009
Page 324 of 720

Purpose

Bessel analog lowpass filter prototype.

Syntax

```
[z,p,k] = besselap(n)
```

Description

`[z,p,k] = besselap(n)` returns the zeros, poles, and gain of an order `n` Bessel analog lowpass filter prototype. It returns the poles in the length `n` column vector `p` and the gain in scalar `k`. `z` is an empty matrix, because there are no zeros. `n` must be less than or equal to 25. The transfer function is

$$H(s) = \frac{k}{(s-p(1))(s-p(2))\cdots(s-p(n))}$$

`besselap` normalizes the poles and gain so that at low frequency and high frequency the Bessel prototype is asymptotically equivalent to the Butterworth prototype of the same order [1]. The magnitude of the filter is less than `sqrt(1/2)` at the unity cutoff frequency $\Omega_c = 1$.

Analog Bessel filters are characterized by a group delay that is maximally flat at zero frequency and almost constant throughout the passband. The group delay at zero frequency is

$$\left(\frac{(2n)!}{2^n n!}\right)^{1/n}$$

Algorithm

`besselap` finds the filter roots from a look-up table constructed using the Symbolic Math Toolbox.

See Also

`besself`	Bessel analog filter design.
`buttap`	Butterworth analog lowpass filter prototype.
`cheb1ap`	Chebyshev type I analog lowpass filter prototype.
`cheb2ap`	Chebyshev type II analog lowpass filter prototype.
`ellipap`	Elliptic analog lowpass filter prototype.

Also see the *Symbolic Math Toolbox User's Guide*.

References

[1] Rabiner, L.R., and B. Gold. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975. Pgs. 228-230.

Meta Platforms, Inc. - Exhibit 1009
Page 325 of 720

Purpose

Bessel analog filter design.

Syntax

```
[b,a] = besself(n,Wn)
[b,a] = besself(n,Wn,'ftype')
[z,p,k] = besself(...)
[A,B,C,D] = besself(...)
```

Description

`besself` designs lowpass, bandpass, highpass, and bandstop analog Bessel filters. Analog Bessel filters are characterized by almost constant group delay across the entire passband, thus preserving the wave shape of filtered signals in the passband. Digital Bessel filters do not retain this quality, and `besself` therefore does not support the design of digital Bessel filters.

`[b,a] = besself(n,Wn)` designs an order `n` lowpass analog filter with cutoff frequency `Wn`. It returns the filter coefficients in the length `n+1` row vectors `b` and `a`, with coefficients in descending powers of *s*:

$$H(s) = \frac{B(s)}{A(s)} = \frac{b(1)s^n + b(2)s^{n-1} + \cdots + b(n+1)}{s^n + a(2)s^{n-1} + \cdots + a(n+1)}$$

Cutoff frequency is the frequency at which the magnitude response of the filter begins to decrease significantly. For `besself`, the cutoff frequency `Wn` must be greater than 0. The magnitude response of a Bessel filter designed by `besself` is always less than `sqrt(1/2)` at the cutoff frequency, and it decreases as the order `n` increases.

If `Wn` is a two-element vector, `Wn = [w1 w2]` with `w1 < w2`, `besself(n,Wn)` returns an order `2*n` bandpass analog filter with passband `w1` < ω < `w2`.

`[b,a] = besself(n,Wn,'ftype')` designs a highpass or bandstop filter, where *`ftype`* is

- `high` for a highpass analog filter with cutoff frequency `Wn`
- `stop` for an order `2*n` bandstop analog filter if `Wn` is a two-element vector, `Wn = [w1 w2]`

  The stopband is `w1` < ω < `w2`.

With different numbers of output arguments, `besself` directly obtains other realizations of the analog filter. To obtain zero-pole-gain form, use three output arguments:

Meta Platforms, Inc. - Exhibit 1009
Page 326 of 720

`[z,p,k] = besself(n,Wn)` or

`[z,p,k] = besself(n,Wn,'ftype')`

`besself` returns the zeros and poles in length `n` or `2*n` column vectors `z` and `p` and the gain in the scalar `k`.

To obtain state-space form, use four output arguments:

`[A,B,C,D] = besself(n,Wn)` or

`[A,B,C,D] = besself(n,Wn,'ftype')` where `A`, `B`, `C`, and `D` are

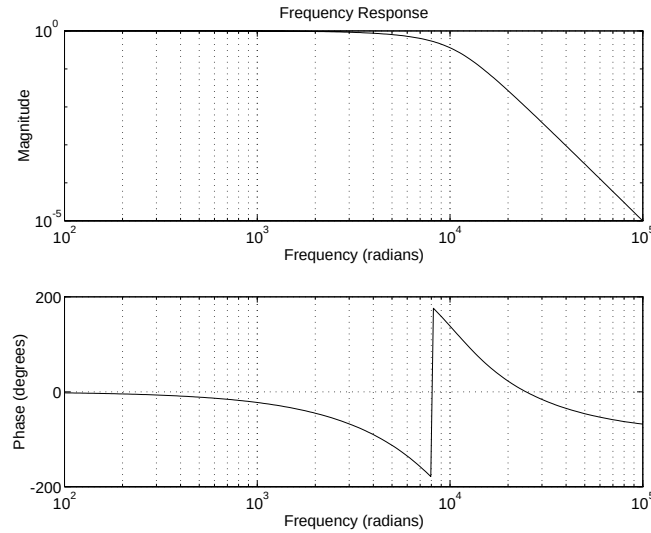
$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

and u is the input, x is the state vector, and y is the output.

Example

Design a fifth-order analog lowpass Bessel filter that suppresses frequencies greater than 10,000 rad/sec and plot the frequency response of the filter using `freqs`:

```
[b,a] = besself(5,10000);
freqs(b,a)                    % plot frequency response
```



Meta Platforms, Inc. - Exhibit 1009
Page 327 of 720

Limitations

Lowpass Bessel filters have a monotonically decreasing magnitude response, as do lowpass Butterworth filters. Compared to the Butterworth, Chebyshev, and elliptic filters, the Bessel filter has the slowest rolloff and requires the highest order to meet an attenuation specification.

For high order filters, the state-space form is the most numerically accurate, followed by the zero-pole-gain form. The transfer function coefficient form is the least accurate; numerical problems can arise for filter orders as low as 15.

Algorithm

`besself` performs a four-step algorithm:

1. It finds lowpass analog prototype poles, zeros, and gain using the `besselap` function.
2. It converts the poles, zeros, and gain into state-space form.
3. It transforms the lowpass filter into a bandpass, highpass, or bandstop filter with desired cutoff frequencies, using a state-space transformation.
4. It converts the state-space filter back to transfer function or zero-pole-gain form, as required.

See Also

`besselap`	Bessel analog lowpass filter prototype.
`butter`	Butterworth analog and digital filter design.
`cheby1`	Chebyshev type I filter design (passband ripple).
`cheby2`	Chebyshev type II filter design (stopband ripple).
`ellip`	Elliptic (Cauer) filter design.

Meta Platforms, Inc. - Exhibit 1009
Page 328 of 720

Purpose

Bilinear transformation method of analog-to-digital filter conversion.

Syntax

```
[zd,pd,kd] = bilinear(z,p,k,Fs)
[zd,pd,kd] = bilinear(z,p,k,Fs,Fp)
[numd,dend] = bilinear(num,den,Fs)
[numd,dend] = bilinear(num,den,Fs,Fp)
[Ad,Bd,Cd,Dd] = bilinear(A,B,C,D,Fs)
[Ad,Bd,Cd,Dd] = bilinear(A,B,C,D,Fs,Fp)
```

Description

The *bilinear transformation* is a mathematical mapping of variables. In digital filtering, it is a standard method of mapping the *s* or analog plane into the *z* or digital plane. It transforms analog filters, designed using classical filter design techniques, into their discrete equivalents.

The bilinear transformation maps the *s*-plane into the *z*-plane by

$$H(z) = H(s)\Big|_{s = 2f_s\frac{z-1}{z+1}}$$

This transformation maps the $j\Omega$ axis (from $\Omega = -\infty$ to $+\infty$) repeatedly around the unit circle ($\exp(j\omega)$, from $\omega = -\pi$ to π) by

$$\omega = 2\tan^{-1}\left(\frac{\Omega}{2f_s}\right)$$

`bilinear` can accept an optional parameter `Fp` that specifies prewarping. `Fp`, in Hertz, indicates a “match” frequency, that is, a frequency for which the frequency responses before and after mapping match exactly. In prewarped mode, the bilinear transformation maps the *s*-plane into the *z*-plane with

$$H(z) = H(s)\Big|_{s = \frac{2\pi f_p}{\tan\left(\pi\frac{f_p}{f_s}\right)}\frac{(z-1)}{(z+1)}}$$

Meta Platforms, Inc. - Exhibit 1009
Page 329 of 720

With the prewarping option, `bilinear` maps the $j\Omega$ axis (from $\Omega = -\infty$ to $+\infty$) repeatedly around the unit circle ($\exp(j\omega)$, from $\omega = -\pi$ to π) by

$$\omega = 2\tan^{-1}\left(\frac{\Omega\tan\left(\pi\frac{f_p}{f_s}\right)}{2\pi f_p}\right)$$

In prewarped mode, `bilinear` matches the frequency $2\pi f_p$ (in radians per second) in the s-plane to the normalized frequency $2\pi f_p/f_s$ (in radians per second) in the z-plane.

The `bilinear` function works with three different linear system representations: zero-pole-gain, transfer function, and state-space form.

Zero-Pole-Gain

`[zd,pd,kd] = bilinear(z,p,k,Fs)` and

`[zd,pd,kd] = bilinear(z,p,k,Fs,Fp)` convert the s-domain transfer function specified by `z`, `p`, and `k` to a discrete equivalent. Inputs `z` and `p` are column vectors containing the zeros and poles, and `k` is a scalar gain. `Fs` is the sampling frequency in Hertz. `bilinear` returns the discrete equivalent in column vectors `zd` and `pd` and scalar `kd`. `Fp` is the optional match frequency, in Hertz, for prewarping.

Transfer Function

`[numd,dend] = bilinear(num,den,Fs)` and

`[numd,dend] = bilinear(num,den,Fs,Fp)` convert an s-domain transfer function given by `num` and `den` to a discrete equivalent. Row vectors `num` and `den` specify the coefficients of the numerator and denominator, respectively, in descending powers of s

$$\frac{num(s)}{den(s)} = \frac{num(1)s^{nn} + \cdots + num(nn)s + num(nn+1)}{den(1)s^{nd} + \cdots + den(nd)s + den(nd+1)}$$

Meta Platforms, Inc. - Exhibit 1009
Page 330 of 720

Fs is the sampling frequency in Hertz. bilinear returns the discrete equivalent in row vectors numd and dend in descending powers of z (ascending powers of z^{-1}). Fp is the optional match frequency, in Hertz, for prewarping.

State-Space

[Ad,Bd,Cd,Dd] = bilinear(A,B,C,D,Fs) and

[Ad,Bd,Cd,Dd] = bilinear(A,B,C,D,Fs,Fp) convert the continuous-time state-space system in matrices A, B, C, D,

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

to the discrete-time system

$$x[n+1] = A_d x[n] + B_d u[n]$$
$$y[n] = C_d x[n] + D_d u[n]$$

Fs is the sampling frequency in Hertz. bilinear returns the discrete equivalent in matrices Ad, Bd, Cd, Dd. Fp is the optional match frequency, in Hertz, for prewarping.

Algorithm

bilinear uses one of two algorithms, depending on the format of the input linear system you supply. One algorithm works on the zero-pole-gain format and the other on the state-space format. For transfer function representations, bilinear converts to state-space form, performs the transformation, and converts the resulting state-space system back to transfer function form.

Zero-Pole-Gain Algorithm

For a system in zero-pole-gain form, bilinear performs four steps:

1. If Fp is present, k = 2*pi*Fp/tan(pi*Fp/Fs); otherwise k = 2*Fs.
2. It strips any zeros at plus or minus infinity using

```
z = z(find(finite(z)));
```


Meta Platforms, Inc. - Exhibit 1009
Page 331 of 720

3 It transforms the zeros, poles, and gain using

```
pd = (1+p/k)./(1-p/k);
zd = (1+z/k)./(1-z/k);
kd = real(k*prod(fs-z)./prod(fs-p));
```

4 It adds extra zeros at -1 so the resulting system has equivalent numerator and denominator order.

State-Space Algorithm

For a system in state-space form, `bilinear` performs two steps:

1 If `Fp` is present, `k = 2*pi*Fp/tan(pi*Fp/Fs)`; else `k = 2*Fs`.

2 It computes `Ad`, `Bd`, `Cd`, and `Dd` in terms of `A`, `B`, `C`, and `D` using

$$A_d = \left(I + \left(\frac{1}{k}\right)A\right)\left(I - \left(\frac{1}{k}\right)A\right)^{-1}$$

$$B_d = \frac{2k}{r}\left(I - \left(\frac{1}{k}\right)A\right)^{-1} B$$

$$C_d = rC\left(I - \left(\frac{1}{k}\right)A\right)^{-1}$$

$$D_d = \left(\frac{1}{k}\right)C\left(I - \left(\frac{1}{k}\right)A\right)^{-1} B + D$$

`bilinear` implements these relations using conventional MATLAB statements. The scalar `r` is arbitrary; `bilinear` uses `sqrt(2/k)` to ensure good quantization noise properties in the resulting system.

Diagnostics

`bilinear` requires that the numerator order be no greater than the denominator order. If this is not the case, `bilinear` displays:

```
Numerator cannot be higher order than denominator.
```

For `bilinear` to distinguish between the zero-pole-gain and transfer function linear system formats, the first two input parameters must be vectors with the same orientation in these cases. If this is not the case, `bilinear` displays:

```
First two arguments must have the same orientation.
```


Meta Platforms, Inc. - Exhibit 1009
Page 332 of 720

See Also

`impinvar`	Impulse invariance method of analog-to-digital filter conversion.
`lp2bp`	Lowpass to bandpass analog filter transformation.
`lp2bs`	Lowpass to bandstop analog filter transformation.
`lp2hp`	Lowpass to highpass analog filter transformation.
`lp2lp`	Lowpass to lowpass analog filter transformation.

References

[1] Parks, T.W., and C.S. Burrus. *Digital Filter Design*. New York: John Wiley & Sons, 1987. Pgs. 209-213.

[2] Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989. Pgs. 415-430.

Meta Platforms, Inc. - Exhibit 1009
Page 333 of 720

Purpose Blackman window.

Syntax

```
w = blackman(n)
w = blackman(n,sflag)
```

Description `w = blackman(n)` returns the `n`-point symmetrically sampled Blackman window in the column vector `w`. `n` should be a nonnegative integer. The equation for a Blackman window is

$$w[k] = 0.42 - 0.5\cos\left(2\pi\frac{k-1}{n-1}\right) + 0.08\cos\left(4\pi\frac{k-1}{n-1}\right), \quad k = 1, \ldots, n$$

Blackman windows have slightly wider central lobes and less sideband leakage than equivalent length Hamming and Hanning windows.

`w = blackman(n,sflag)` returns an `n`-point Blackman window using the window sampling specified by `sflag`, which can be either `'periodic'` or `'symmetric'` (the default). When `'periodic'` is specified, `blackman` computes a length `n+1` window and returns the first `n` points.

Algorithm

```
w = (0.42 - 0.5*cos(2*pi*(0:N-1)/(N-1)) + ...
0.08*cos(4*pi*(0:N-1)/(N-1)))';
```

Diagnostics An error message is displayed when incorrect arguments are used:

```
Order cannot be less than zero.
Sampling must be either 'symmetric' or 'periodic'.
```

A warning message is displayed for noninteger `n`:

```
Warning: Rounding order to nearest integer.
```


Meta Platforms, Inc. - Exhibit 1009
Page 334 of 720

See Also

`bartlett`	Bartlett window.
`boxcar`	Rectangular window.
`chebwin`	Chebyshev window.
`hamming`	Hamming window.
`hanning`	Hanning window.
`kaiser`	Kaiser window.
`triang`	Triangular window.

References

[1] Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

Meta Platforms, Inc. - Exhibit 1009
Page 335 of 720

boxcar

Purpose Rectangular window.

Syntax

```
w = boxcar(n)
```

Description `w = boxcar(n)` returns a rectangular window of length `n` in the column vector `w`. This function is provided for completeness; a rectangular window is equivalent to no window at all.

Algorithm

```
w = ones(n,1);
```

See Also

`bartlett`	Bartlett window.
`blackman`	Blackman window.
`chebwin`	Chebyshev window.
`hamming`	Hamming window.
`hanning`	Hanning window.
`kaiser`	Kaiser window.
`triang`	Triangular window.

References

[1] Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

Meta Platforms, Inc. - Exhibit 1009
Page 336 of 720

Purpose

Buffer a signal vector into a matrix of data frames.

Syntax

```
y = buffer(x,n)
y = buffer(x,n,p)
y = buffer(x,n,p,opt)
[y,z] = buffer(...)
[y,z,opt] = buffer(...)
```

Description

`y = buffer(x,n)` partitions a length-`L` signal vector `x` into nonoverlapping data segments (frames) of length `n`. Each data frame occupies one column of matrix output `y`, which has `n` rows and `ceil(L/n)` columns. If `L` is not evenly divisible by `n`, the last column is zero-padded to length `n`.

`y = buffer(x,n,p)` overlaps or underlaps successive frames in the output matrix by `p` samples:

- For `0<p<n` (overlap), `buffer` repeats the final `p` samples of each frame at the beginning of the following frame. For example, if `x=1:30` and `n=7`, an overlap of `p=3` looks like this:

```
y =
     0     2     6    10    14    18    22    26
     0     3     7    11    15    19    23    27
     0     4     8    12    16    20    24    28
     1     5     9    13    17    21    25    29
     2     6    10    14    18    22    26    30
     3     7    11    15    19    23    27     0
     4     8    12    16    20    24    28     0
```

  The first frame starts with `p` zeros (the default initial condition), and the number of columns in `y` is `ceil(L/(n–p))`.

Meta Platforms, Inc. - Exhibit 1009
Page 337 of 720

- For `p<0` (underlap), `buffer` skips `p` samples between consecutive frames. For example, if `x=1:30` and `n=7`, a buffer with underlap of `p=-3` looks like this:

```
y =
     1    11    21
     2    12    22
     3    13    23
     4    14    24
     5    15    25
     6    16    26
     7    17    27
```

skipped { 8 18 28 / 9 19 29 / 10 20 30 }

  The number of columns in `y` is `ceil(L/(n-p))`.

`y = buffer(x,n,p,opt)` specifies a vector of samples to precede `x(1)` in an overlapping buffer, or the number of initial samples to skip in an underlapping buffer:

- For `0<p<n` (overlap), `opt` specifies a length-`p` vector to insert before `x(1)` in the buffer. This vector can be considered an *initial condition*, which is needed when the current buffering operation is one in a sequence of consecutive buffering operations. To maintain the desired frame overlap from one buffer to the next, `opt` should contain the final `p` samples of the previous buffer in the sequence. See "Continuous Buffering" below.
- By default, `opt` is `zeros(p,1)` for an overlapping buffer. Set `opt` to `'nodelay'` to skip the initial condition and begin filling the buffer immediately with `x(1)`. In this case, `L` must be `length(p)` or longer. For example, if `x=1:30` and `n=7`, a buffer with overlap of `p=3` looks like this:

```
y =
     1     5     9    13    17    21    25
     2     6    10    14    18    22    26
     3     7    11    15    19    23    27
     4     8    12    16    20    24    28
     5     9    13    17    21    25    29
     6    10    14    18    22    26    30
     7    11    15    19    23    27     0
```

- For `p<0` (underlap), `opt` is an integer value in the range `[0,-p]` specifying the number of initial input samples, `x(1:opt)`, to skip before adding samples to

Meta Platforms, Inc. - Exhibit 1009
Page 338 of 720

the buffer. The first value in the buffer is therefore `x(opt+1)`. By default, `opt` is zero for an underlapping buffer.

This option is especially useful when the current buffering operation is one in a sequence of consecutive buffering operations. To maintain the desired frame underlap from one buffer to the next, `opt` should equal the difference between the total number of points to skip between frames (`p`) and the number of points that were *available* to be skipped in the previous input to `buffer`. If the previous input had fewer than `p` points that could be skipped after filling the final frame of that buffer, the remaining `opt` points need to be removed from the first frame of the current buffer. See "Continuous Buffering" below for an example of how this works in practice.

`[y,z] = buffer(...)` partitions the length-`L` signal vector `x` into frames of length `n`, and outputs only the *full* frames in `y`. If `y` is an overlapping buffer, it has `n` rows and `m` columns, where

```
m = floor(L/(n-p))          % when length(opt) = p
```

or

```
m = floor((L-n)/(n-p))+1    % when opt = 'nodelay'
```

If `y` is an underlapping buffer, it has `n` rows and `m` columns, where

```
m = floor((L-opt)/(n-p)) + (rem((L-opt),(n-p)) >= n)
```

If the number of samples in the input vector (after the appropriate overlapping or underlapping operations) exceeds the number of places available in the `n`-by-`m` buffer, the remaining samples in `x` are output in vector `z`, which for an overlapping buffer has length

```
length(z) = L - m*(n-p)            % when length(opt) = p
```

or

```
length(z) = L - ((m-1)*(n-p)+n)    % when opt = 'nodelay'
```

and for an underlapping buffer has length

```
length(z) = (L-opt) - m*(n-p)
```


Meta Platforms, Inc. - Exhibit 1009
Page 339 of 720

Output z shares the same orientation (row or column) as x. If there are no remaining samples in the input after the buffer with the specified overlap or underlap is filled, z is an empty vector.

[y,z,opt] = buffer(...) returns the last p samples of a overlapping buffer in output opt. In an underlapping buffer, opt is the difference between the total number of points to skip between frames (–p) and the number of points in x that were *available* to be skipped after filling the last frame:

- For 0<p<n (overlap), opt (as an output) contains the final p samples in the last frame of the buffer. This vector can be used as the *initial condition* for a subsequent buffering operation in a sequence of consecutive buffering operations. This allows the desired frame overlap to be maintained from one buffer to the next. See "Continuous Buffering" below.
- For p<0 (underlap), opt (as an output) is the difference between the total number of points to skip between frames (–p) and the number of points in x that were *available* to be skipped after filling the last frame.

  ```
  opt = m*(n–p) + opt – L    % for z = empty vector
  ```

  where opt on the right-hand side is the input argument to buffer, and opt on the left-hand side is the output argument. Here m is the number of columns in the buffer, which is

  ```
  m = floor((L–opt)/(n–p)) + (rem((L–opt),(n–p)) >= n)
  ```

  Note that for an underlapping buffer output opt is always zero when output z contains data.

  The opt output for an underlapping buffer is especially useful when the current buffering operation is one in a sequence of consecutive buffering operations. The opt output from each buffering operation specifies the number of samples that need to be skipped at the start of the next buffering operation to maintain the desired frame underlap from one buffer to the next. If fewer than p points were available to be skipped after filling the final frame of the current buffer, the remaining opt points need to be removed from the first frame of the next buffer.

In a sequence of buffering operations, the opt output from each operation should be used as the opt input to the subsequent buffering operation. This ensures that the desired frame overlap or underlap is maintained from buffer

Meta Platforms, Inc. - Exhibit 1009
Page 340 of 720

to buffer, as well as from frame to frame within the same buffer. See "Continuous Buffering" below for an example of how this works in practice.

Continuous Buffering

In a continuous buffering operation, the vector input to the `buffer` function represents one frame in a sequence of frames that make up a discrete signal. These signal frames can originate in a frame-based data acquisition process, or within a frame-based algorithm like the FFT.

As an example, you might acquire data from an A/D card in frames of 64 samples. In the simplest case, you could rebuffer the data into frames of 16 samples; `buffer` with `n=16` creates a buffer of four frames from each 64-element input frame. The result is that the signal of frame size 64 has been converted to a signal of frame size 16; no samples were added or removed.

In the general case where the original signal frame size, `L`, is not equally divisible by the new frame size, `n`, the overflow from the last frame needs to be captured and recycled into the following buffer. You can do this by iteratively calling `buffer` on input `x` with the two-output-argument syntax:

```
[y,z] = buffer([z;x],n)    % for column vector x

[y,z] = buffer([z,x],n)    % for row vector x
```

This simply captures any buffer overflow in `z`, and prepends the data to the subsequent input in the next call to `buffer`. Again, the input signal, `x`, of frame size `L`, has been converted to a signal of frame size `n` without any insertion or deletion of samples.

Note that continuous buffering cannot be done with the single-output syntax `y = buffer(...)`, because the last frame of `y` in this case is zero padded, which adds new samples to the signal.

Continuous buffering in the presence of overlap and underlap is handled with the `opt` parameter, which is used as both an input and output to `buffer`. The following two examples demonstrate how the `opt` parameter should be used.

Examples

Example 1: Continuous Overlapping Buffers

First create a buffer containing 100 frames, each with 11 samples.

```
data = buffer(1:1100,11); % 11 samples per frame
```


Meta Platforms, Inc. - Exhibit 1009
Page 341 of 720

Imagine that the frames (columns) in the matrix called `data` are the sequential outputs of a data acquisition board sampling a physical signal: `data(:,1)` is the first D/A output, containing the first 11 signal samples; `data(:,2)` is the second output, containing the next 11 signal samples, and so on.

You want to rebuffer this signal from the acquired frame size of 11 to a frame size of 4 with an overlap of 1. To do this, you will repeatedly call `buffer` to operate on each successive input frame, using the `opt` parameter to maintain consistency in the overlap from one buffer to the next.

Set the buffer parameters.

```
n = 4;      % new frame size
p = 1;      % overlap
opt = -5;   % value of y(1)
z = [];     % initialize the carry-over vector
```

Now repeatedly call `buffer`, each time passing in a new signal frame from `data`. Note that overflow samples (returned in `z`) are carried over and prepended to the input in the subsequent call to `buffer`.

```
for i=1:size(data,2),    % Loop over each source frame (column)
   x = data(:,i);        % A single frame of the D/A output

   [y,z,opt] = buffer([z;x],n,p,opt);

   disp(y);            % Do something with the buffer of data
   pause
end
```


Meta Platforms, Inc. - Exhibit 1009
Page 342 of 720

Here's what happens during the first four iterations.

Iteration	Input frame `[z;x]'`	opt (input)	opt (output)	Output buffer `(y)`	Overflow `(z)`
i=1	`[1:11]`	–5	9	$\begin{bmatrix} -5 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$	`[10 11]`
i=2	`[10 11 12:22]`	9	21	$\begin{bmatrix} 9 & 12 & 15 & 18 \\ 10 & 13 & 16 & 19 \\ 11 & 14 & 17 & 20 \\ 12 & 15 & 18 & 21 \end{bmatrix}$	`[22]`
i=3	`[22 23:33]`	21	33	$\begin{bmatrix} 21 & 24 & 27 & 30 \\ 22 & 25 & 28 & 31 \\ 23 & 26 & 29 & 32 \\ 24 & 27 & 30 & 33 \end{bmatrix}$	`[]`
i=4	`[34:44]`	33	42	$\begin{bmatrix} 33 & 36 & 39 \\ 34 & 37 & 40 \\ 35 & 38 & 41 \\ 36 & 39 & 42 \end{bmatrix}$	`[43 44]`

Note that the size of the output matrix, `y`, can vary by a single column from one iteration to the next. This is typical for buffering operations with overlap or underlap.

Example 2: Continuous Underlapping Buffers

Again create a buffer containing 100 frames, each with 11 samples.

```
data = buffer(1:1100,11); % 11 samples per frame
```

Again, imagine that `data(:,1)` is the first D/A output, containing the first 11 signal samples; `data(:,2)` is the second output, containing the next 11 signal samples, and so on.

You want to rebuffer this signal from the acquired frame size of 11 to a frame size of 4 with an underlap of 2. To do this, you will repeatedly call `buffer` to operate on each successive input frame, using the `opt` parameter to maintain consistency in the underlap from one buffer to the next.

Meta Platforms, Inc. - Exhibit 1009
Page 343 of 720

Set the buffer parameters:

```
n = 4;     % new frame size
p = -2;    % underlap
opt = 1;   % skip the first input element, x(1)
z = [];    % initialize the carry-over vector
```

Now repeatedly call `buffer`, each time passing in a new signal frame from `data`. Note that overflow samples (returned in `z`) are carried over and prepended to the input in the subsequent call to `buffer`.

```
for i=1:size(data,2),    % Loop over each source frame (column)
   x = data(:,i);        % A single frame of the D/A output

   [y,z,opt] = buffer([z;x],n,p,opt);

   disp(y);            % Do something with the buffer of data
   pause
end
```


Meta Platforms, Inc. - Exhibit 1009
Page 344 of 720

Here's what happens during the first three iterations.

Iteration	Input frame `[z;x]'`	opt (input)	opt (output)	Output buffer `(y)`	Overflow `(z)`
i=1	[1:11]	1	2	1 – } skip 2 8 3 9 4 10 5 11 6 – } skip 7 –	[]
i=2	[12:22]	2	0	12 13 } skip 14 15 16 17 18 19 } skip	[20 21 22]
i=3	[20 21 22 23:33]	0	0	– – } skip 20 26 21 27 22 28 23 29 24 30 25 31 } skip	[32 33]

Diagnostics

Error messages are displayed when p≥n or `length(opt)`≠`length(p)` in an overlapping buffer case:

```
Frame overlap P must be less than the buffer size N.
Initial conditions must be specified as a length-P vector.
```

See Also

`reshape` Reshape array.

Purpose

Butterworth analog lowpass filter prototype.

Syntax

```
[z,p,k] = buttap(n)
```

Description

`[z,p,k] = buttap(n)` returns the zeros, poles, and gain of an order `n` Butterworth analog lowpass filter prototype. It returns the poles in the length `n` column vector `p` and the gain in scalar `k`. `z` is an empty matrix, because there are no zeros. The transfer function is

$$H(s) = \frac{z(s)}{p(s)} = \frac{k}{(s-p(1))(s-p(2))\cdots(s-p(n))}$$

Butterworth filters are characterized by a magnitude response that is maximally flat in the passband and monotonic overall. In the lowpass case, the first 2n–1 derivatives of the squared magnitude response are zero at $\omega = 0$. The squared magnitude response function is

$$|H(\omega)|^2 = \frac{1}{1 + (\omega/\omega_0)^{2n}}$$

corresponding to a transfer function with poles equally spaced around a circle in the left half plane. The magnitude response at the cutoff frequency ω_0 is always `1/sqrt(2)`, regardless of the filter order. `buttap` sets ω_0 to 1 for a normalized result.

Algorithm

```
z = [];
p = exp(sqrt(-1)*(pi*(1:2:2*n-1)/(2*n)+pi/2)).';
k = real(prod(-p));
```

See Also

`besselap`	Bessel analog lowpass filter prototype.
`butter`	Butterworth analog and digital filter design.
`cheb1ap`	Chebyshev type I analog lowpass filter prototype.
`cheb2ap`	Chebyshev type II analog lowpass filter prototype.
`ellipap`	Elliptic analog lowpass filter prototype.

References

[1] Parks, T.W., and C.S. Burrus. *Digital Filter Design*. New York: John Wiley & Sons, 1987. Chapter 7.

Meta Platforms, Inc. - Exhibit 1009
Page 346 of 720

Purpose

Butterworth analog and digital filter design.

Syntax

```
[b,a] = butter(n,Wn)
[b,a] = butter(n,Wn,'ftype')
[b,a] = butter(n,Wn,'s')
[b,a] = butter(n,Wn,'ftype','s')
[z,p,k] = butter(...)
[A,B,C,D] = butter(...)
```

Description

`butter` designs lowpass, bandpass, highpass, and bandstop digital and analog Butterworth filters. Butterworth filters are characterized by a magnitude response that is maximally flat in the passband and monotonic overall.

Butterworth filters sacrifice rolloff steepness for monotonicity in the pass- and stopbands. Unless the smoothness of the Butterworth filter is needed, an elliptic or Chebyshev filter can generally provide steeper rolloff characteristics with a lower filter order.

Digital Domain

`[b,a] = butter(n,Wn)` designs an order `n` lowpass digital Butterworth filter with cutoff frequency `Wn`. It returns the filter coefficients in length `n + 1` row vectors `b` and `a`, with coefficients in descending powers of z:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + \cdots + b(n+1)z^{-n}}{1 + a(2)z^{-1} + \cdots + a(n+1)z^{-n}}$$

Cutoff frequency is that frequency where the magnitude response of the filter is `sqrt(1/2)`. For `butter`, the cutoff frequency `Wn` must be a number between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency).

If `Wn` is a two-element vector, `Wn = [w1 w2]`, `butter` returns an order `2*n` digital bandpass filter with passband `w1 < ω < w2`.

`[b,a] = butter(n,Wn,'ftype')` designs a highpass or bandstop filter, where *`ftype`* is

Meta Platforms, Inc. - Exhibit 1009
Page 347 of 720

- `high` for a highpass digital filter with cutoff frequency `Wn`
- `stop` for an order `2*n` bandstop digital filter if `Wn` is a two-element vector, `Wn = [w1 w2]`

  The stopband is `w1` < ω < `w2`.

With different numbers of output arguments, `butter` directly obtains other realizations of the filter. To obtain zero-pole-gain form, use three output arguments:

`[z,p,k] = butter(n,Wn)` or

`[z,p,k] = butter(n,Wn,'ftype')`

`butter` returns the zeros and poles in length `n` column vectors `z` and `p`, and the gain in the scalar `k`.

To obtain state-space form, use four output arguments:

`[A,B,C,D] = butter(n,Wn)` or

`[A,B,C,D] = butter(n,Wn,'ftype')` where `A`, `B`, `C`, and `D` are

$$\begin{aligned} x[n+1] &= Ax[n] + Bu[n] \\ y[n] &= Cx[n] + Du[n] \end{aligned}$$

and u is the input, x is the state vector, and y is the output.

Analog Domain

`[b,a] = butter(n,Wn,'s')` designs an order `n` lowpass analog Butterworth filter with cutoff frequency `Wn`. It returns the filter coefficients in the length `n + 1` row vectors `b` and `a`, in descending powers of s:

$$H(s) = \frac{B(s)}{A(s)} = \frac{b(1)s^n + b(2)s^{n-1} + \cdots + b(n+1)}{s^n + a(2)s^{n-1} + \cdots + a(n+1)}$$

`butter`'s cutoff frequency `Wn` must be greater than 0.

If `Wn` is a two-element vector with `w1 < w2`, `butter(n,Wn,'s')` returns an order `2*n` bandpass analog filter with passband `w1` < ω < `w2`.

Meta Platforms, Inc. - Exhibit 1009
Page 348 of 720

`[b,a] = butter(n,Wn,'ftype','s')` designs a highpass or bandstop filter, where *ftype* is

- `high` for a highpass analog filter with cutoff frequency `Wn`
- `stop` for an order `2*n` bandstop analog filter if `Wn` is a two-element vector, `Wn = [w1 w2]`

  The stopband is `w1` < ω < `w2`.

With different numbers of output arguments, `butter` directly obtains other realizations of the analog filter. To obtain zero-pole-gain form, use three output arguments:

`[z,p,k] = butter(n,Wn,'s')` or

`[z,p,k] = butter(n,Wn,'ftype','s')` returns the zeros and poles in length `n` or `2*n` column vectors `z` and `p` and the gain in the scalar `k`.

To obtain state-space form, use four output arguments:

`[A,B,C,D] = butter(n,Wn,'s')` or

`[A,B,C,D] = butter(n,Wn,'ftype','s')` where `A`, `B`, `C`, and `D` are

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

and u is the input, x is the state vector, and y is the output.

Examples

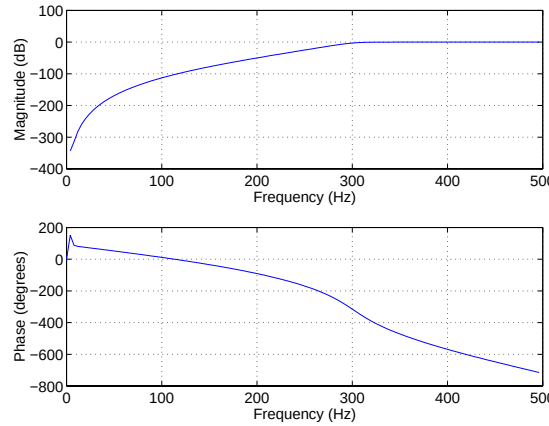
For data sampled at 1000 Hz, design a 9th-order highpass Butterworth filter with cutoff frequency of 300 Hz:

```
[b,a] = butter(9,300/500,'high');
```


Meta Platforms, Inc. - Exhibit 1009
Page 349 of 720

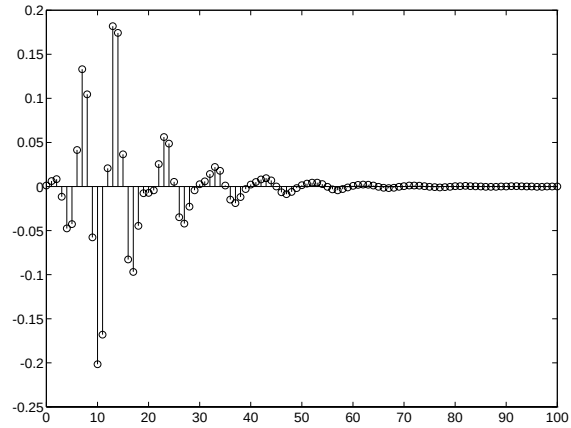
The filter's frequency response is

```
freqz(b,a,128,1000)
```



Design a 10th-order bandpass Butterworth filter with a passband from 100 to 200 Hz and plot its impulse response, or *unit sample response*:

```
n = 5; Wn = [100 200]/500;
[b,a] = butter(n,Wn);
[y,t] = impz(b,a,101);
stem(t,y)
```



Meta Platforms, Inc. - Exhibit 1009
Page 350 of 720

Limitations

For high order filters, the state-space form is the most numerically accurate, followed by the zero-pole-gain form. The transfer function coefficient form is the least accurate; numerical problems can arise for filter orders as low as 15.

Algorithm

`butter` uses a five-step algorithm:

1. It finds the lowpass analog prototype poles, zeros, and gain using the `buttap` function.
2. It converts the poles, zeros, and gain into state-space form.
3. It transforms the lowpass filter into a bandpass, highpass, or bandstop filter with desired cutoff frequencies, using a state-space transformation.
4. For digital filter design, `butter` uses `bilinear` to convert the analog filter into a digital filter through a bilinear transformation with frequency prewarping. Careful frequency adjustment guarantees that the analog filters and the digital filters will have the same frequency response magnitude at `Wn` or `w1` and `w2`.
5. It converts the state-space filter back to transfer function or zero-pole-gain form, as required.

See Also

`besself`	Bessel analog filter design.
`buttap`	Butterworth analog lowpass filter prototype.
`buttord`	Butterworth filter order selection.
`cheby1`	Chebyshev type I filter design (passband ripple).
`cheby2`	Chebyshev type II filter design (stopband ripple).
`ellip`	Elliptic (Cauer) filter design.
`maxflat`	Generalized digital Butterworth filter design.

Meta Platforms, Inc. - Exhibit 1009
Page 351 of 720

Purpose Butterworth filter order selection.

Syntax

```
[n,Wn] = buttord(Wp,Ws,Rp,Rs)
[n,Wn] = buttord(Wp,Ws,Rp,Rs,'s')
```

Description `buttord` selects the minimum order digital or analog Butterworth filter required to meet a set of filter design specifications:

`Wp`	Passband corner frequency. `Wp`, the cutoff frequency, has a value between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency).
`Ws`	Stopband corner frequency. `Ws` is in the same units as `Wp`; it has a value between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency).
`Rp`	Passband ripple, in decibels. This value is the maximum permissible passband loss in decibels. The passband is `0 < w < Wp`.
`Rs`	Stopband attenuation, in decibels. This value is the number of decibels the stopband is down from the passband. The stopband is `Ws < w < 1`.

Digital Domain

`[n,Wn] = buttord(Wp,Ws,Rp,Rs)` returns the order `n` of the lowest order digital Butterworth filter that loses no more than `Rp` dB in the passband and has at least `Rs` dB of attenuation in the stopband. The passband runs from 0 to `Wp` and the stopband runs from `Ws` to 1, the Nyquist frequency. `buttord` also returns `Wn`, the Butterworth cutoff frequency that allows `butter` to achieve the given specifications (the "-3 dB" frequency).

Use `buttord` for highpass, bandpass, and bandstop filters. For highpass filters, `Wp` is greater than `Ws`. For bandpass and bandstop filters, `Wp` and `Ws` are two-element vectors that specify the corner frequencies at both edges of the filter, lower frequency edge first. For the band filters, `buttord` returns `Wn` as a two-element row vector for input to `butter`.

If filter specifications call for a bandpass or bandstop filter with unequal ripple in each of the passbands or stopbands, design the filter as separate lowpass and highpass sections and cascade the two filters together.

Meta Platforms, Inc. - Exhibit 1009
Page 352 of 720

Analog Domain

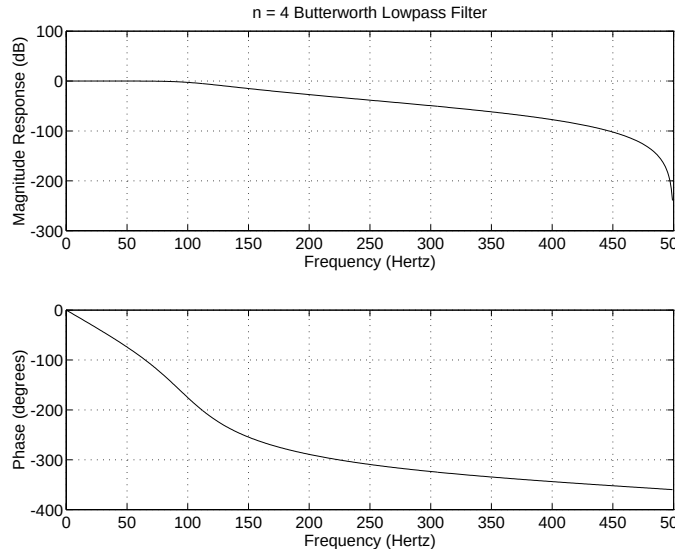
`[n,Wn] = buttord(Wp,Ws,Rp,Rs,'s')` finds the minimum order `n` and cutoff frequencies `Wn` for an analog filter. In this case the frequencies in `Wp` and `Ws` are in radians per second and may be greater than 1.

Use `buttord` for highpass, bandpass, and bandstop filters, as described under "Digital Domain."

Examples

For data sampled at 1000 Hz, design a lowpass filter with less than 3 dB of attenuation from 0 to 100 Hz, and attenuation at least 15 dB from 150 Hz to the Nyquist frequency. Plot the filter's frequency response:

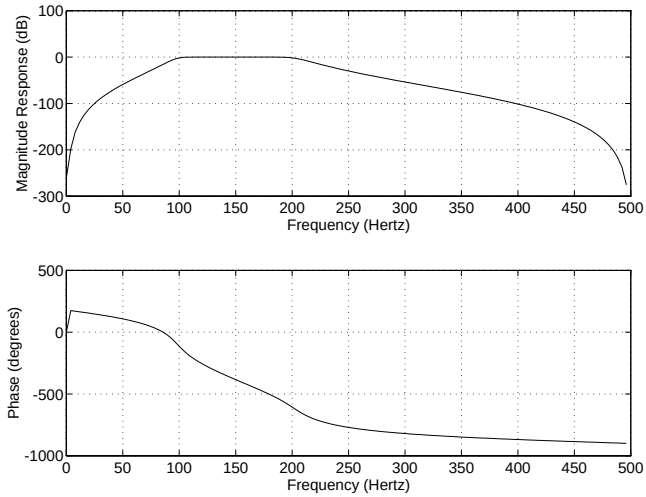
```
Wp = 100/500; Ws = 150/500;
[n,Wn] = buttord(Wp,Ws,3,15)

n =
     4

Wn =
    0.2042

[b,a] = butter(n,Wn);
freqz(b,a,512,1000); title('n=4 Butterworth Lowpass Filter')
```



Meta Platforms, Inc. - Exhibit 1009
Page 353 of 720

Next design a bandpass filter with passband of 100 Hz to 200 Hz, less than 3 dB of attenuation at the passband corners, and attenuation down 30 dB by 50 Hz out on both sides of the passband:

```
Wp = [100 200]/500; Ws = [50 250]/500;
Rp = 3; Rs = 30;
[n,Wn] = buttord(Wp,Ws,Rp,Rs); [b,a] = butter(n,Wn);
freqz(b,a,128,1000)
```



Algorithm

`buttord`'s order prediction formula is described in [1]. It operates in the analog domain for both analog and digital cases. For the digital case, it converts the frequency parameters to the *s*-domain before estimating the order and natural frequency, then converts back to the *z*-domain.

`buttord` initially develops a lowpass filter prototype by transforming the passband frequencies of the desired filter to 1 rad/sec (for low- and highpass filters) and to -1 and 1 rad/sec (for bandpass and bandstop filters). It then computes the minimum order required for a lowpass filter to meet the stopband specification.

Meta Platforms, Inc. - Exhibit 1009
Page 354 of 720

See Also

`butter`	Butterworth analog and digital filter design.
`cheb1ord`	Chebyshev type I filter order selection.
`cheb2ord`	Chebyshev type II filter order selection.
`ellipord`	Elliptic filter order selection.
`kaiserord`	Estimate parameters for an FIR filter design with Kaiser window.

References

[1] Rabiner, L.R., and B. Gold. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975. Pg. 227.

Meta Platforms, Inc. - Exhibit 1009
Page 355 of 720

Purpose

Complex cepstral analysis.

Syntax

```
xhat = cceps(x)
[xhat,nd] = cceps(x)
[xhat,nd,xhat1] = cceps(x)
[...] = cceps(x,n)
```

Description

Cepstral analysis is a nonlinear signal processing technique that is applied most commonly in speech processing and homomorphic filtering [1].

`xhat = cceps(x)` returns the complex cepstrum of the (assumed real) sequence `x`. The input is altered, by the application of a linear phase term, to have no phase discontinuity at $\pm\pi$ radians. That is, it is circularly shifted (after zero padding) by some samples, if necessary, to have zero phase at π radians.

`[xhat,nd] = cceps(x)` returns the number of samples `nd` of (circular) delay added to `x` prior to finding the complex cepstrum.

`[xhat,nd,xhat1] = cceps(x)` returns a second complex cepstrum, computed using an alternate rooting algorithm, in `xhat1`. The alternate method ([1] p.795) is useful for short sequences that can be rooted and do not have zeros on the unit circle. For these signals, `xhat1` can provide a verification of `xhat`.

`[...] = cceps(x,n)` zero pads `x` to length `n` and returns the length `n` complex cepstrum of `x`.

Algorithm

`cceps`, in its basic form, is an M-file implementation of algorithm 7.1 in [2]. A lengthy Fortran program reduces to three lines of MATLAB code:

```
h = fft(x);
logh = log(abs(h)) + sqrt(-1)*rcunwrap(angle(h));
y = real(ifft(logh));
```

`rcunwrap` is a special version of `unwrap` that subtracts a straight line from the phase.

Meta Platforms, Inc. - Exhibit 1009
Page 356 of 720

See Also

`icceps`	Inverse complex cepstrum.
`hilbert`	Hilbert transform.
`rceps`	Real cepstrum and minimum phase reconstruction.
`unwrap`	Unwrap phase angles.

References

[1] Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[2] IEEE. *Programs for Digital Signal Processing*. IEEE Press. New York: John Wiley & Sons, 1979.

Meta Platforms, Inc. - Exhibit 1009
Page 357 of 720

Purpose Chebyshev type I analog lowpass filter prototype.

Syntax

```
[z,p,k] = cheb1ap(n,Rp)
```

Description `[z,p,k] = cheb1ap(n,Rp)` returns the zeros, poles, and gain of an order `n` Chebyshev type I analog lowpass filter prototype with `Rp` dB of ripple in the passband. It returns the poles in the length `n` column vector `p` and the gain in scalar `k`. `z` is an empty matrix, because there are no zeros. The transfer function is

$$H(s) = \frac{z(s)}{p(s)} = \frac{k}{(s-p(1))(s-p(2))\cdots(s-p(n))}$$

Chebyshev type I filters are equiripple in the passband and monotonic in the stopband. The poles are evenly spaced about an ellipse in the left half plane. The Chebyshev type I cutoff frequency ω_0 is set to 1.0 for a normalized result. This is the frequency at which the passband ends and the filter has magnitude response of $10^{-Rp/20}$.

See Also

`besselap`	Bessel analog lowpass filter prototype.
`buttap`	Butterworth analog and digital filter design.
`cheb2ap`	Chebyshev type I analog lowpass filter prototype.
`cheby1`	Chebyshev type I filter design (passband ripple).
`ellipap`	Elliptic analog lowpass filter prototype.

References

[1] Parks, T.W., and C.S. Burrus. *Digital Filter Design*. New York: John Wiley & Sons, 1987. Chapter 7.

Meta Platforms, Inc. - Exhibit 1009
Page 358 of 720

Purpose Chebyshev type I filter order selection.

Syntax

```
[n,Wn] = cheb1ord(Wp,Ws,Rp,Rs)
[n,Wn] = cheb1ord(Wp,Ws,Rp,Rs,'s')
```

Description `cheb1ord` selects the minimum order digital or analog Chebyshev type I filter required to meet a set of filter design specifications:

`Wp`	Passband corner frequency. `Wp`, the cutoff frequency, has a value between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency).
`Ws`	Stopband corner frequency. `Ws` is in the same units as `Wp`; it has a value between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency).
`Rp`	Passband ripple, in decibels. This value is the maximum permissible passband loss in decibels. The passband is `0 < w < Wp`.
`Rs`	Stopband attenuation, in decibels. This value is the number of decibels the stopband is down from the passband. The stopband is `Ws < w < 1`.

Digital Domain

`[n,Wn] = cheb1ord(Wp,Ws,Rp,Rs)` returns the order `n` of the lowest order Chebyshev filter that loses no more than `Rp` dB in the passband and has at least `Rs` dB of attenuation in the stopband. The passband runs from 0 to `Wp` and the stopband runs from `Ws` to 1, the Nyquist frequency. `cheb1ord` also returns `Wn`, the Chebyshev type I cutoff frequency that allows `cheby1` to achieve the given specifications.

Use `cheb1ord` for lowpass, highpass, bandpass, and bandstop filters. For highpass filters, `Wp > Ws`. For bandpass and bandstop filters, `Wp` and `Ws` are two-element vectors that specify the corner frequencies at both edges of the filter, lower frequency edge first. For the band filters, `cheb1ord` returns `Wn` as a two-element row vector for input to `cheby1`.

If filter specifications call for a bandpass or bandstop filter with unequal ripple in each of the passbands or stopbands, design the filter as separate lowpass and highpass sections and cascade the two filters together.

Meta Platforms, Inc. - Exhibit 1009
Page 359 of 720

Analog Domain

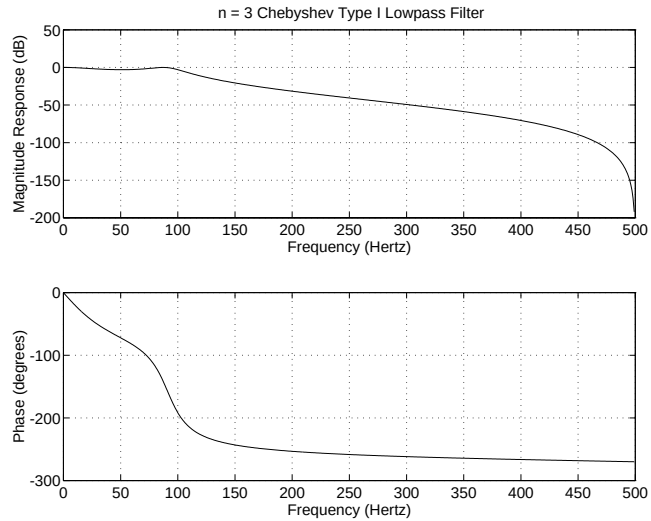
`[n,Wn] = cheb1ord(Wp,Ws,Rp,Rs,'s')` finds the minimum order `n` and cutoff frequencies `Wn` for an analog filter. In this case the frequencies in `Wp` and `Ws` are in radians per second and may be greater than 1.

Use `cheb1ord` for lowpass, highpass, bandpass, and bandstop filters, as described under "Digital Domain."

Examples

For data sampled at 1000 Hz, design a lowpass filter with less than 3 dB of attenuation from 0 to 100 Hz and attenuation at least 15 dB from 150 Hz to the Nyquist frequency:

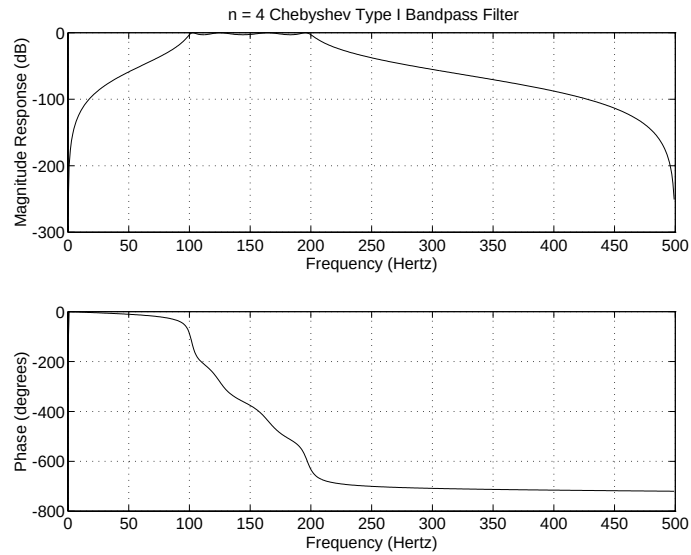
```
Wp = 100/500; Ws = 150/500;
Rp = 3; Rs = 15;
[n,Wn] = cheb1ord(Wp,Ws,Rp,Rs)

n =
     3

Wn =
    0.2000

[b,a] = cheby1(n,Rp,Wn);
freqz(b,a,512,1000); title('n=3 Chebyshev Type I Lowpass Filter')
```



Meta Platforms, Inc. - Exhibit 1009
Page 360 of 720

Next design a bandpass filter with a passband of 100 Hz to 200 Hz, less than 3 dB of attenuation throughout the passband, and 30 dB stopbands 50 Hz out on both sides of the passband:

```
Wp = [100 200]/500; Ws = [50 250]/500;
Rp = 3; Rs = 30;
[n,Wn] = cheb1ord(Wp,Ws,Rp,Rs)

n =
     4

Wn =
    0.2000    0.4000

[b,a] = cheby1(n,Rp,Wn);
freqz(b,a,512,1000);
title('n=4 Chebyshev Type I Bandpass Filter')
```



Algorithm

cheb1ord uses the Chebyshev lowpass filter order prediction formula described in [1]. The function performs its calculations in the analog domain for both analog and digital cases. For the digital case, it converts the frequency parameters to the *s*-domain before the order and natural frequency estimation process, then converts them back to the *z*-domain.

Meta Platforms, Inc. - Exhibit 1009
Page 361 of 720

`cheb1ord` initially develops a lowpass filter prototype by transforming the passband frequencies of the desired filter to 1 rad/sec (for low- or highpass filters) or to -1 and 1 rad/sec (for bandpass or bandstop filters). It then computes the minimum order required for a lowpass filter to meet the stopband specification.

See Also

`buttord`	Butterworth filter order selection.
`cheby1`	Chebyshev type I filter design (passband ripple).
`cheb2ord`	Chebyshev type II filter order selection.
`ellipord`	Elliptic filter order selection.
`kaiserord`	Estimate parameters for an FIR filter design with Kaiser window.

References

[1] Rabiner, L.R., and B. Gold. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975. Pg. 241.

Meta Platforms, Inc. - Exhibit 1009
Page 362 of 720

Purpose

Chebyshev type II analog lowpass filter prototype.

Syntax

```
[z,p,k] = cheb2ap(n,Rs)
```

Description

`[z,p,k] = cheb2ap(n,Rs)` finds the zeros, poles, and gain of an order `n` Chebyshev type II analog lowpass filter prototype with stopband ripple `Rs` dB down from the passband peak value. `cheb2ap` returns the zeros and poles in length `n` column vectors `z` and `p` and the gain in scalar `k`. If `n` is odd, `z` is length `n–1`. The transfer function is

$$H(s) = \frac{z(s)}{p(s)} = k\frac{(s-z(1))(s-z(2))\cdots(s-z(n))}{(s-p(1))(s-p(2))\cdots(s-p(n))}$$

Chebyshev type II filters are monotonic in the passband and equiripple in the stopband. The pole locations are the inverse of the pole locations of `cheb1ap`, whose poles are evenly spaced about an ellipse in the left half plane. The Chebyshev type II cutoff frequency ω_0 is set to 1 for a normalized result. This is the frequency at which the stopband begins and the filter has magnitude response of $10^{-Rs/20}$.

Algorithm

Chebyshev type II filters are sometimes called *inverse Chebyshev* filters because of their relationship to Chebyshev type I filters. The `cheb2ap` function is a modification of the Chebyshev type I prototype algorithm:

1. `cheb2ap` replaces the frequency variable ω with $1/\omega$, turning the lowpass filter into a highpass filter while preserving the performance at $\omega = 1$.
2. `cheb2ap` subtracts the filter transfer function from unity.

See Also

`besselap`	Bessel analog lowpass filter prototype.
`buttap`	Butterworth analog lowpass filter prototype.
`cheb1ap`	Chebyshev type I analog lowpass filter prototype.
`cheby2`	Chebyshev type II filter design (stopband ripple).
`ellipap`	Elliptic analog lowpass filter prototype.

References

[1] Parks, T.W., and C.S. Burrus. *Digital Filter Design*. New York: John Wiley & Sons, 1987. Chapter 7.

Meta Platforms, Inc. - Exhibit 1009
Page 363 of 720

Purpose Chebyshev type II filter order selection.

Syntax

```
[n,Wn] = cheb2ord(Wp,Ws,Rp,Rs)
[n,Wn] = cheb2ord(Wp,Ws,Rp,Rs,'s')
```

Description `cheb2ord` selects the minimum order digital or analog Chebyshev type II filter required to meet a set of filter design specifications:

`Wp`	Passband corner frequency. `Wp`, the cutoff frequency, has a value between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency).
`Ws`	Stopband corner frequency. `Ws` is in the same units as `Wp`; it has a value between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency).
`Rp`	Passband ripple, in decibels. This value is the maximum permissible passband loss in decibels. The passband is `0 < w < Wp`.
`Rs`	Stopband attenuation, in decibels. This value is the number of decibels the stopband is down from the passband. The stopband is `Ws < w < 1`.

Digital Domain

`[n,Wn] = cheb2ord(Wp,Ws,Rp,Rs)` returns the order `n` of the lowest order Chebyshev filter that loses no more than `Rp` dB in the passband and has at least `Rs` dB of attenuation in the stopband. The passband runs from 0 to `Wp` and the stopband runs from `Ws` to 1, the Nyquist frequency. `cheb2ord` also returns `Wn`, the Chebyshev type II cutoff frequency that allows `cheby2` to achieve the given specifications.

Use `cheb2ord` for lowpass, highpass, bandpass, and bandstop filters. For highpass filters, `Wp` is greater than `Ws`. For bandpass and bandstop filters, `Wp` and `Ws` are two-element vectors that specify the corner frequencies at both edges of the filter, lower frequency edge first. For the band filters, `cheb2ord` returns `Wn` as a two-element row vector for input to `cheby2`.

If filter specifications call for a bandpass or bandstop filter with unequal ripple in each of the passbands or stopbands, design the filter as separate lowpass and highpass sections and cascade the two filters together.

Meta Platforms, Inc. - Exhibit 1009
Page 364 of 720

Analog Domain

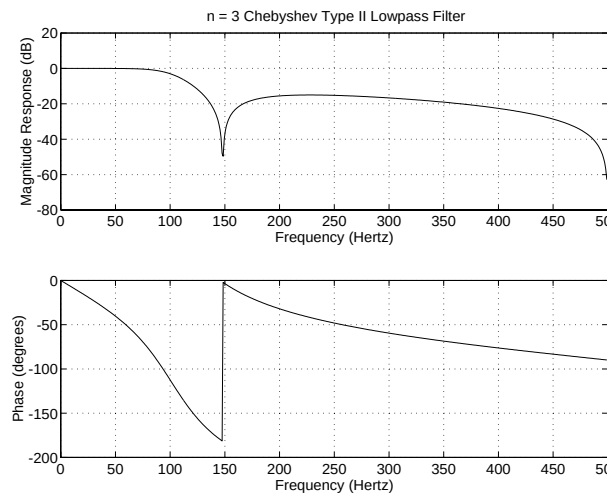
`[n,Wn] = cheb2ord(Wp,Ws,Rp,Rs,'s')` finds the minimum order `n` and cutoff frequencies `Wn` for an analog filter. In this case the frequencies in `Wp` and `Ws` are in radians per second and may be greater than 1.

Use `cheb2ord` for lowpass, highpass, bandpass, and bandstop filters, as described under "Digital Domain."

Examples

For data sampled at 1000 Hz, design a lowpass filter with less than 3 dB of attenuation from 0 to 100 Hz, and attenuation at least 15 dB from 150 Hz to the Nyquist frequency:

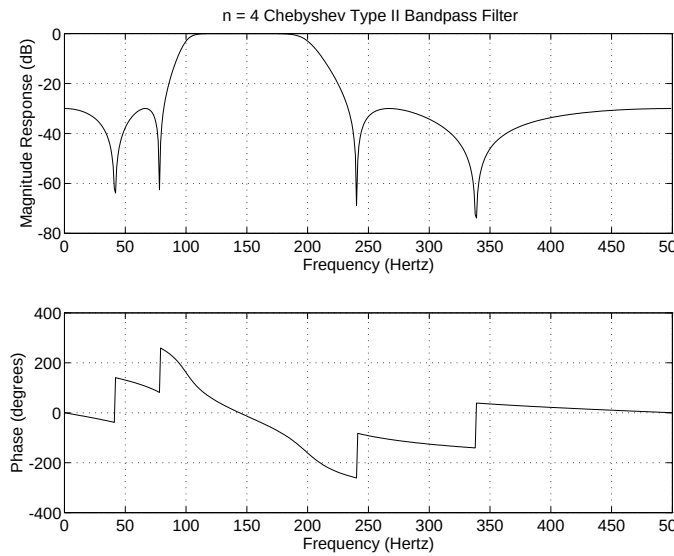
```
Wp = 100/500; Ws = 150/500;
Rp = 3; Rs = 15;
[n,Wn] = cheb2ord(Wp,Ws,Rp,Rs)

n =
     3

Wn =
    0.2609

[b,a] = cheby2(n,Rs,Wn);
freqz(b,a,512,1000);
title('n=3 Chebyshev Type II Lowpass Filter')
```



Meta Platforms, Inc. - Exhibit 1009
Page 365 of 720

Next design a bandpass filter with a passband of 100 Hz to 200 Hz, less than 3 dB of attenuation throughout the passband, and 30 dB stopbands 50 Hz out on both sides of the passband:

```
Wp = [100 200]/500; Ws = [50 250]/500;
Rp = 3; Rs = 30;
[n,Wn] = cheb2ord(Wp,Ws,Rp,Rs)

n =
     4

Wn =
    0.1633    0.4665

[b,a] = cheby2(n,Rs,Wn);
freqz(b,a,512,1000)
title('n=4 Chebyshev Type II Bandpass Filter')
```



Algorithm

`cheb2ord` uses the Chebyshev lowpass filter order prediction formula described in [1]. The function performs its calculations in the analog domain for both analog and digital cases. For the digital case, it converts the frequency parameters to the *s*-domain before the order and natural frequency estimation process, then converts them back to the *z*-domain.

Meta Platforms, Inc. - Exhibit 1009
Page 366 of 720

`cheb2ord` initially develops a lowpass filter prototype by transforming the stopband frequencies of the desired filter to 1 rad/sec (for low- and highpass filters) and to -1 and 1 rad/sec (for bandpass and bandstop filters). It then computes the minimum order required for a lowpass filter to meet the passband specification.

See Also

`buttord`	Butterworth filter order selection.
`cheb1ord`	Chebyshev type I filter order selection.
`cheby2`	Chebyshev type II filter design (stopband ripple).
`ellipord`	Elliptic filter order selection.
`kaiserord`	Estimate parameters for an FIR filter design with Kaiser window.

References

[1] Rabiner, L.R., and B. Gold. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975. Pg. 241.

Meta Platforms, Inc. - Exhibit 1009
Page 367 of 720

Purpose Chebyshev window.

Syntax

```
w = chebwin(n,r)
```

Description `w = chebwin(n,r)` returns the column vector `w`, containing the length `n` Chebyshev window whose Fourier transform magnitude sidelobe ripple is `r` dB below the mainlobe magnitude.

See Also

`bartlett`	Bartlett window.
`blackman`	Blackman window.
`boxcar`	Rectangular window.
`hamming`	Hamming window.
`hanning`	Hanning window.
`kaiser`	Kaiser window.
`triang`	Triangular window.

References

[1] IEEE. *Programs for Digital Signal Processing*. IEEE Press. New York: John Wiley & Sons, 1979. Program 5.2.

Meta Platforms, Inc. - Exhibit 1009
Page 368 of 720

Purpose

Chebyshev type I filter design (passband ripple).

Syntax

```
[b,a] = cheby1(n,Rp,Wn)
[b,a] = cheby1(n,Rp,Wn,'ftype')
[b,a] = cheby1(n,Rp,Wn,'s')
[b,a] = cheby1(n,Rp,Wn,'ftype','s')
[z,p,k] = cheby1(...)
[A,B,C,D] = cheby1(...)
```

Description

`cheby1` designs lowpass, bandpass, highpass, and bandstop digital and analog Chebyshev type I filters. Chebyshev type I filters are equiripple in the passband and monotonic in the stopband. Type I filters roll off faster than type II filters, but at the expense of greater deviation from unity in the passband.

Digital Domain

`[b,a] = cheby1(n,Rp,Wn)` designs an order `n` lowpass digital Chebyshev filter with cutoff frequency `Wn` and `Rp` dB of ripple in the passband. It returns the filter coefficients in the length `n+1` row vectors `b` and `a`, with coefficients in descending powers of z:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + \cdots + b(n+1)z^{-n}}{1 + a(2)z^{-1} + \cdots + a(n+1)z^{-n}}$$

Cutoff frequency is the frequency at which the magnitude response of the filter is equal to –`Rp` dB. For `cheby1`, the cutoff frequency `Wn` is a number between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency). Smaller values of passband ripple `Rp` lead to wider transition widths (shallower rolloff characteristics).

If `Wn` is a two-element vector, `Wn = [w1 w2]`, `cheby1` returns an order `2*n` bandpass filter with passband `w1 < ω < w2`.

`[b,a] = cheby1(n,Rp,Wn,'ftype')` designs a highpass or bandstop filter, where *`ftype`* is

Meta Platforms, Inc. - Exhibit 1009
Page 369 of 720

- `high` for a highpass digital filter with cutoff frequency `Wn`
- `stop` for an order `2*n` bandstop digital filter if `Wn` is a two-element vector, `Wn = [w1 w2]`

  The stopband is `w1` < ω < `w2`.

With different numbers of output arguments, `cheby1` directly obtains other realizations of the filter. To obtain zero-pole-gain form, use three output arguments:

`[z,p,k] = cheby1(n,Rp,Wn)` or

`[z,p,k] = cheby1(n,Rp,Wn,'ftype')` returns the zeros and poles in length `n` column vectors `z` and `p` and the gain in the scalar `k`.

To obtain state-space form, use four output arguments:

`[A,B,C,D] = cheby1(n,Rp,Wn)` or

`[A,B,C,D] = cheby1(n,Rp,Wn,'ftype')` where `A`, `B`, `C`, and `D` are

$$\begin{aligned} x[n+1] &= Ax[n] + Bu[n] \\ y[n] &= Cx[n] + Du[n] \end{aligned}$$

and u is the input, x is the state vector, and y is the output.

Analog Domain

`[b,a] = cheby1(n,Rp,Wn,'s')` designs an order `n` lowpass analog Chebyshev type I filter with cutoff frequency `Wn`. It returns the filter coefficients in length `n + 1` row vectors `b` and `a`, in descending powers of s:

$$H(s) = \frac{B(s)}{A(s)} = \frac{b(1)s^n + b(2)s^{n-1} + \cdots + b(n+1)}{s^n + a(2)s^{n-1} + \cdots + a(n+1)}$$

Cutoff frequency is the frequency at which the magnitude response of the filter is –`Rp` dB. For `cheby1`, the cutoff frequency `Wn` must be greater than 0.

If `Wn` is a two-element vector, `Wn = [w1 w2]`, with `w1 < w2`, then `cheby1(n,Rp,Wn,'s')` returns an order `2*n` bandpass analog filter with passband `w1` < ω < `w2`.

Meta Platforms, Inc. - Exhibit 1009
Page 370 of 720

`[b,a] = cheby1(n,Rp,Wn,'ftype','s')` designs a highpass or bandstop filter, where *ftype* is

- `high` for a highpass analog filter with cutoff frequency `Wn`
- `stop` for an order `2*n` bandstop analog filter if `Wn` is a two-element vector, `Wn = [w1 w2]`

  The stopband is `w1` < ω < `w2`.

You can supply different numbers of output arguments for `cheby1` to directly obtain other realizations of the analog filter. To obtain zero-pole-gain form, use three output arguments:

`[z,p,k] = cheby1(n,Rp,Wn,'s')` or

`[z,p,k] = cheby1(n,Rp,Wn,'ftype','s')` returns the zeros and poles in length `n` or `2*n` column vectors `z` and `p` and the gain in the scalar `k`.

To obtain state-space form, use four output arguments:

`[A,B,C,D] = cheby1(n,Rp,Wn,'s')` or

`[A,B,C,D] = cheby1(n,Rp,Wn,'ftype','s')` where `A`, `B`, `C`, and `D` are defined as

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

and u is the input, x is the state vector, and y is the output.

Examples

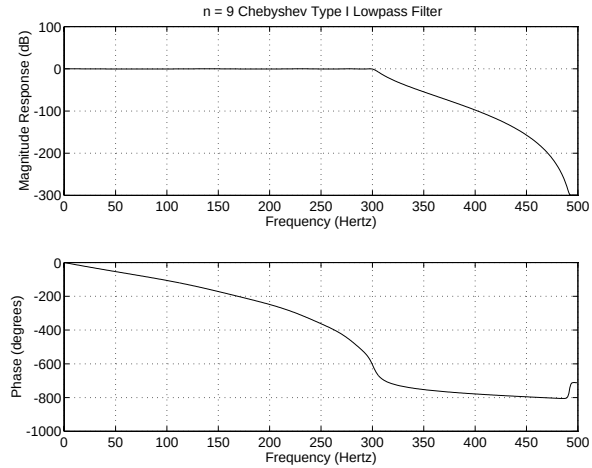
For data sampled at 1000 Hz, design a 9th-order lowpass Chebyshev type I filter with 0.5 dB of ripple in the passband and a cutoff frequency of 300 Hz:

```
[b,a] = cheby1(9,0.5,300/500);
```


Meta Platforms, Inc. - Exhibit 1009
Page 371 of 720

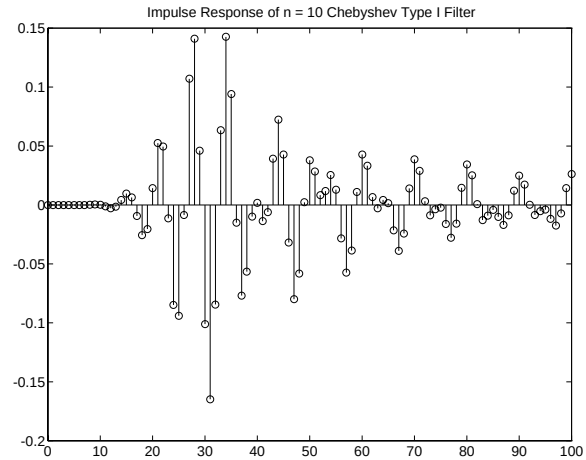
The frequency response of the filter is

```
freqz(b,a,512,1000)
```



Design a 10th-order bandpass Chebyshev type I filter with a passband from 100 to 200 Hz and plot its impulse response:

```
n = 10; Rp = 0.5;
Wn = [100 200]/500;
[b,a] = cheby1(n,Rp,Wn);
[y,t] = impz(b,a,101); stem(t,y)
```



Meta Platforms, Inc. - Exhibit 1009
Page 372 of 720

Limitations

For high order filters, the state-space form is the most numerically accurate, followed by the zero-pole-gain form. The transfer function form is the least accurate; numerical problems can arise for filter orders as low as 15.

Algorithm

`cheby1` uses a five-step algorithm:

1. It finds the lowpass analog prototype poles, zeros, and gain using the `cheb1ap` function.
2. It converts the poles, zeros, and gain into state-space form.
3. It transforms the lowpass filter into a bandpass, highpass, or bandstop filter with desired cutoff frequencies, using a state-space transformation.
4. For digital filter design, `cheby1` uses `bilinear` to convert the analog filter into a digital filter through a bilinear transformation with frequency prewarping. Careful frequency adjustment guarantees that the analog filters and the digital filters will have the same frequency response magnitude at `Wn` or `w1` and `w2`.
5. It converts the state-space filter back to transfer function or zero-pole-gain form, as required.

See Also

`besself`	Bessel analog filter design.
`butter`	Butterworth analog and digital filter design.
`cheb1ap`	Chebyshev type I analog lowpass filter prototype.
`cheb1ord`	Chebyshev type I filter order selection.
`cheby2`	Chebyshev type II filter design (stopband ripple).
`ellip`	Elliptic (Cauer) filter design.

Meta Platforms, Inc. - Exhibit 1009
Page 373 of 720

Purpose Chebyshev type II filter design (stopband ripple).

Syntax

```
[b,a] = cheby2(n,Rs,Wn)
[b,a] = cheby2(n,Rs,Wn,'ftype')
[b,a] = cheby2(n,Rs,Wn,'s')
[b,a] = cheby2(n,Rs,Wn,'ftype','s')
[z,p,k] = cheby2(...)
[A,B,C,D] = cheby2(...)
```

Description `cheby2` designs lowpass, highpass, bandpass, and bandstop digital and analog Chebyshev type II filters. Chebyshev type II filters are monotonic in the passband and equiripple in the stopband. Type II filters do not roll off as fast as type I filters, but are free of passband ripple.

Digital Domain

`[b,a] = cheby2(n,Rs,Wn)` designs an order `n` lowpass digital Chebyshev type II filter with cutoff frequency `Wn` and stopband ripple `Rs` dB down from the peak passband value. It returns the filter coefficients in the length `n + 1` row vectors `b` and `a`, with coefficients in descending powers of z:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + \cdots + b(n+1)z^{-n}}{1 + a(2)z^{-1} + \cdots + a(n+1)z^{-n}}$$

Cutoff frequency is the beginning of the stopband, where the magnitude response of the filter is equal to –`Rs` dB. For `cheby2`, the cutoff frequency `Wn` is a number between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency). Larger values of stopband attenuation `Rs` lead to wider transition widths (shallower rolloff characteristics).

If `Wn` is a two-element vector, `Wn = [w1 w2]`, `cheby2` returns an order `2*n` bandpass filter with passband `w1 < ω < w2`.

`[b,a] = cheby2(n,Rs,Wn,'ftype')` designs a highpass or bandstop filter, where *`ftype`* is

Meta Platforms, Inc. - Exhibit 1009
Page 374 of 720

- `high` for a highpass digital filter with cutoff frequency `Wn`
- `stop` for an order `2*n` bandstop digital filter if `Wn` is a two-element vector, `Wn = [w1 w2]`.

  The stopband is `w1` < ω < `w2`.

With different numbers of output arguments, `cheby2` directly obtains other realizations of the filter. To obtain zero-pole-gain form, use three output arguments:

`[z,p,k] = cheby2(n,Rs,Wn)` or

`[z,p,k] = cheby2(n,Rs,Wn,'ftype')` returns the zeros and poles in length `n` column vectors `z` and `p` and the gain in the scalar `k`.

To obtain state-space form, use four output arguments:

`[A,B,C,D] = cheby2(n,Rs,Wn)` or

`[A,B,C,D] = cheby2(n,Rs,Wn,'ftype')` where `A`, `B`, `C`, and `D` are

$$\begin{aligned} x[n+1] &= Ax[n] + Bu[n] \\ y[n] &= Cx[n] + Du[n] \end{aligned}$$

and u is the input, x is the state vector, and y is the output.

Analog Domain

`[b,a] = cheby2(n,Rs,Wn,'s')` designs an order `n` lowpass analog Chebyshev type II filter with cutoff frequency `Wn`. It returns the filter coefficients in the length `n + 1` row vectors `b` and `a`, with coefficients in descending powers of s:

$$H(s) = \frac{B(s)}{A(s)} = \frac{b(1)s^n + b(2)s^{n-1} + \cdots + b(n+1)}{s^n + a(2)s^{n-1} + \cdots + a(n+1)}$$

Cutoff frequency is the frequency at which the magnitude response of the filter is equal to –`Rs` dB. For `cheby2`, the cutoff frequency `Wn` must be greater than 0.

If `Wn` is a two-element vector, `Wn = [w1 w2]`, with `w1 < w2`, then `cheby2(n,Rs,Wn,'s')` returns an order `2*n` bandpass analog filter with passband `w1` < ω < `w2`.

Meta Platforms, Inc. - Exhibit 1009
Page 375 of 720

`[b,a] = cheby2(n,Rs,Wn,'ftype','s')` designs a highpass or bandstop filter, where *ftype* is

- `high` for a highpass analog filter with cutoff frequency `Wn`
- `stop` for an order `2*n` bandstop analog filter if `Wn` is a two-element vector, `Wn = [w1 w2]`

  The stopband is `w1` < ω < `w2`.

With different numbers of output arguments, `cheby2` directly obtains other realizations of the analog filter. To obtain zero-pole-gain form, use three output arguments:

`[z,p,k] = cheby2(n,Rs,Wn,'s')` or

`[z,p,k] = cheby2(n,Rs,Wn,'ftype','s')` returns the zeros and poles in length `n` or `2*n` column vectors `z` and `p` and the gain in the scalar `k`.

To obtain state-space form, use four output arguments:

`[A,B,C,D] = cheby2(n,Rs,Wn,'s')` or

`[A,B,C,D] = cheby2(n,Rs,Wn,'ftype','s')` where `A`, `B`, `C`, and `D` are

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

and u is the input, x is the state vector, and y is the output.

Examples

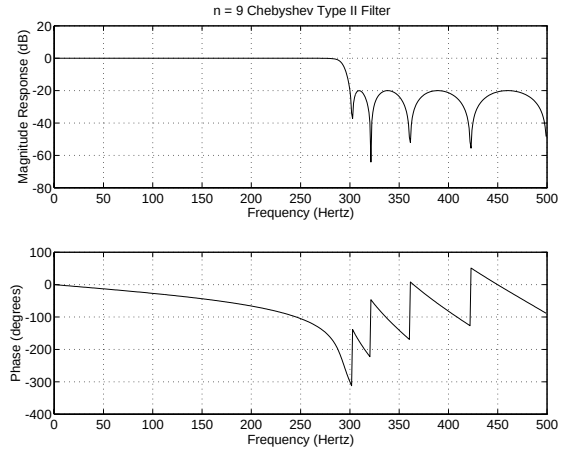
For data sampled at 1000 Hz, design a ninth-order lowpass Chebyshev type II filter with stopband attenuation 20 dB down from the passband and a cutoff frequency of 300 Hz:

```
[b,a] = cheby2(9,20,300/500);
```


Meta Platforms, Inc. - Exhibit 1009
Page 376 of 720

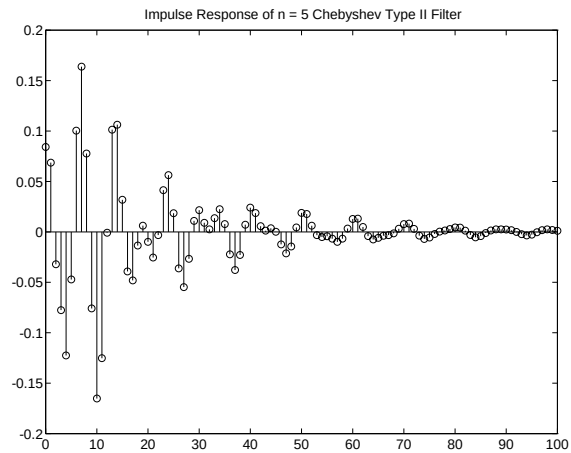
The frequency response of the filter is

```
freqz(b,a,512,1000)
```



Design a fifth-order bandpass Chebyshev type II filter with passband from 100 to 200 Hz and plot the impulse response of the filter:

```
n = 5; r = 20;
Wn = [100 200]/500;
[b,a] = cheby2(n,r,Wn);
[y,t] = impz(b,a,101); stem(t,y)
```



Meta Platforms, Inc. - Exhibit 1009
Page 377 of 720

Limitations

For high order filters, the state-space form is the most numerically accurate, followed by the zero-pole-gain form. The transfer function coefficient form is the least accurate; numerical problems can arise for filter orders as low as 15.

Algorithm

`cheby2` uses a five-step algorithm:

1. It finds the lowpass analog prototype poles, zeros, and gain using the `cheb2ap` function.
2. It converts poles, zeros, and gain into state-space form.
3. It transforms the lowpass filter into a bandpass, highpass, or bandstop filter with desired cutoff frequencies, using a state-space transformation.
4. For digital filter design, `cheby2` uses `bilinear` to convert the analog filter into a digital filter through a bilinear transformation with frequency prewarping. Careful frequency adjustment guarantees that the analog filters and the digital filters will have the same frequency response magnitude at `Wn` or `w1` and `w2`.
5. It converts the state-space filter back to transfer function or zero-pole-gain form, as required.

See Also

`besself`	Bessel analog filter design.
`butter`	Butterworth analog and digital filter design.
`cheb2ap`	Chebyshev type II analog lowpass filter prototype.
`cheb2ord`	Chebyshev type II filter order selection.
`cheby1`	Chebyshev type I filter design (passband ripple).
`ellip`	Elliptic (Cauer) filter design.

Meta Platforms, Inc. - Exhibit 1009
Page 378 of 720

Purpose

Swept-frequency cosine generator.

Syntax

```
y = chirp(t,f0,t1,f1)
y = chirp(t,f0,t1,f1,'method')
y = chirp(t,f0,t1,f1,'method',phi)
```

Description

`y = chirp(t,f0,t1,f1)` generates samples of a linear swept-frequency cosine signal at the time instances defined in array `t`, where `f0` is the instantaneous frequency at time 0, and `f1` is the instantaneous frequency at time `t1`. `f0` and `f1` are both in Hertz. If unspecified, `f0` is 0, `t1` is 1, and `f1` is 100.

`y = chirp(t,f0,t1,f1,'method')` specifies alternative sweep method options, where *`method`* can be

- `linear`, which specifies an instantaneous frequency sweep $f_i(t)$ given by

  $$f_i(t) = f_0 + \beta t$$

- where

  $$\beta = (f_1 - f_0)/t_1$$

- β ensures that the desired frequency breakpoint f_1 at time t_1 is maintained.
- `quadratic`, which specifies an instantaneous frequency sweep $f_i(t)$ given by

  $$f_i(t) = f_0 + \beta t^2$$

- where

  $$\beta = (f_1 - f_0)/t_1$$

- `logarithmic` specifies an instantaneous frequency sweep $f_i(t)$ given by

  $$f_i(t) = f_0 + 10^{\beta t}$$

  where

  $$\beta = [\log_{10}(f_1 - f_0)]/t_1$$

- For a log-sweep, `f1` must be greater than `f0`.

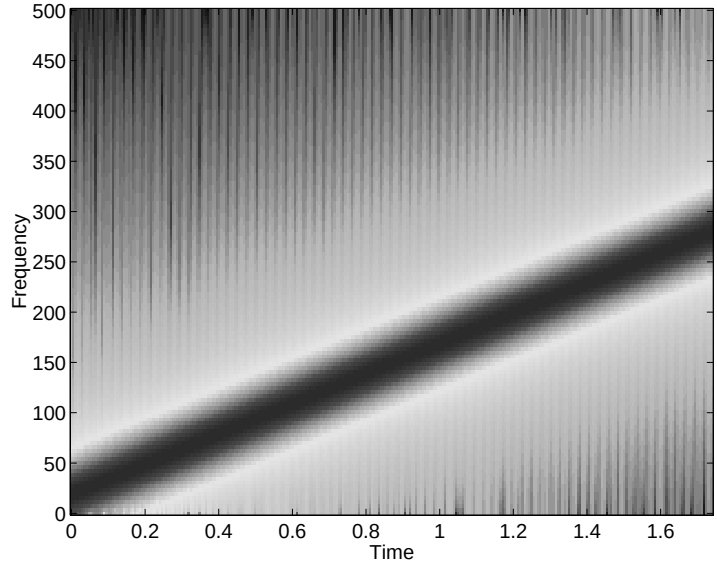
Meta Platforms, Inc. - Exhibit 1009
Page 379 of 720

`y = chirp(t,f0,t1,f1,'method',phi)` allows an initial phase `phi` to be specified in degrees. If unspecified, `phi` is 0. Default values are substituted for empty or omitted trailing input arguments.

Examples

Compute the spectrogram of a chirp with linear instantaneous frequency deviation:

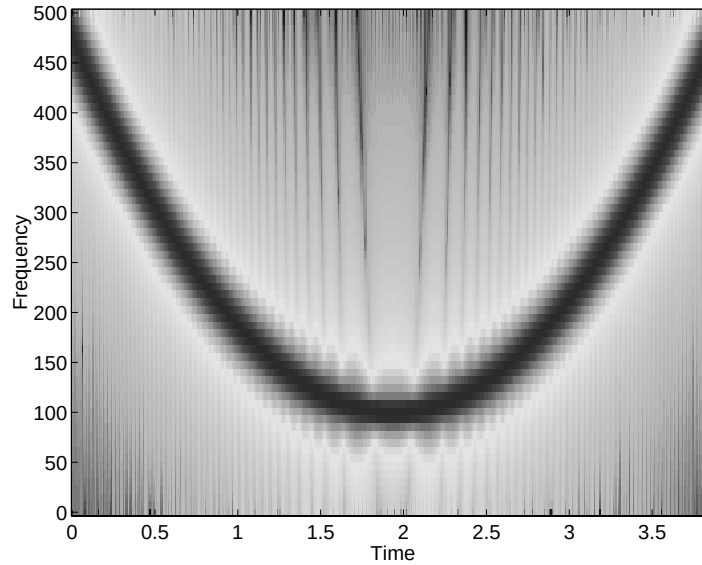
```
t = 0:0.001:2;              % 2 secs @ 1kHz sample rate
y = chirp(t,0,1,150);       % Start @ DC, cross 150Hz at t=1 sec
specgram(y,256,1e3,256,250) % Display the spectrogram
```



Meta Platforms, Inc. - Exhibit 1009
Page 380 of 720

Compute the spectrogram of a chirp with quadratic instantaneous frequency deviation:

```
t = -2:0.001:2;                      % ±2 secs @ 1kHz sample rate
y = chirp(t,100,1,200,'quadratic'); % Start @ 100Hz, cross 200Hz
                                     % at t=1 sec
specgram(y,128,1e3,128,120)          % Display the spectrogram
```



Meta Platforms, Inc. - Exhibit 1009
Page 381 of 720

See Also

`cos`	Cosine of vector/matrix elements (see the online *MATLAB Function Reference*).
`diric`	Dirichlet or periodic sinc function.
`gauspuls`	Gaussian-modulated sinusoidal pulse generator.
`pulstran`	Pulse train generator.
`rectpuls`	Sampled aperiodic rectangle generator.
`sawtooth`	Sawtooth or triangle wave generator.
`sin`	Sine of vector/matrix elements (see the online *MATLAB Function Reference*).
`sinc`	Sinc function.
`square`	Square wave generator.
`tripuls`	Sampled aperiodic triangle generator.

Meta Platforms, Inc. - Exhibit 1009
Page 382 of 720

Purpose

Estimate magnitude squared coherence function between two signals.

Syntax

```
Cxy = cohere(x,y)
Cxy = cohere(x,y,nfft)
[Cxy,f] = cohere(x,y,nfft,Fs)
Cxy = cohere(x,y,nfft,Fs,window)
Cxy = cohere(x,y,nfft,Fs,window,noverlap)
Cxy = cohere(x,y,...,'dflag')
cohere(x,y)
```

Description

`Cxy = cohere(x,y)` finds the magnitude squared coherence between length `n` signal vectors `x` and `y`. The coherence is a function of the power spectra of `x` and `y` and the cross spectrum of `x` and `y`:

$$C_{xy}(f) = \frac{|P_{xy}(f)|^2}{P_{xx}(f)P_{yy}(f)}$$

`x` and `y` must be the same length. `Cxy = cohere(x,y)` uses the following default values:

- `nfft = min(256,length(x))`
- `Fs = 2`
- `window = hanning(nfft)`
- `noverlap = 0`

`nfft` specifies the FFT length that `cohere` uses. This value determines the frequencies at which the coherence is estimated. `Fs` is a scalar that specifies the sampling frequency. `window` specifies a windowing function and the number of samples `cohere` uses in its sectioning of the `x` and `y` vectors. `noverlap` is the number of samples by which the sections overlap. Any arguments that you omit from the end of the parameter list use the default values shown above.

If `x` is real, `cohere` estimates the coherence function at positive frequencies only; in this case, the output `Cxy` is a column vector of length `nfft/2 + 1` for `nfft` even and `(nfft + 1)/2` for `n` odd. If `x` or `y` is complex, `cohere` estimates the coherence function at both positive and negative frequencies, and `Cxy` has length `nfft`.

Meta Platforms, Inc. - Exhibit 1009
Page 383 of 720

`Cxy = cohere(x,y,nfft)` uses the FFT length `nfft` in estimating the power spectrum for `x`. Specify `nfft` as a power of 2 for fastest execution.

`[Cxy,f] = cohere(x,y,nfft,Fs)` returns a vector `f` of frequencies at which the function evaluates the coherence. `Fs` is the sampling frequency. `f` is the same size as `Cxy`, so `plot(f,Cxy)` plots the coherence function versus properly scaled frequency. `Fs` has no effect on the output `Cxy`; it is a frequency scaling multiplier.

`Cxy = cohere(x,y,nfft,Fs,window)` specifies a windowing function and the number of samples per section of the vectors `x` and `y`. If you supply a scalar for `window`, `cohere` uses a Hanning window of that length. The length of the window must be less than or equal to `nfft`; `cohere` zero pads the sections if the window length exceeds `nfft`.

`Cxy = cohere(x,y,nfft,Fs,window,noverlap)` overlaps the sections of `x` by `noverlap` samples.

You can use the empty matrix `[]` to specify the default value for any input argument except `x` or `y`. For example,

```
Cxy = cohere(x,y,[],[],kaiser(128,5));
```

uses 256 as the value for `nfft` and 2 as the value for `Fs`.

`Cxy = cohere(x,y,...,'dflag')` specifies a detrend option, where *`dflag`* is

- `linear`, to remove the best straight-line fit from the prewindowed sections of `x` and `y`
- `mean`, to remove the mean from the prewindowed sections of `x` and `y`
- `none`, for no detrending (default)

The *`dflag`* parameter must appear last in the list of input arguments. `cohere` recognizes a *`dflag`* string no matter how many intermediate arguments are omitted.

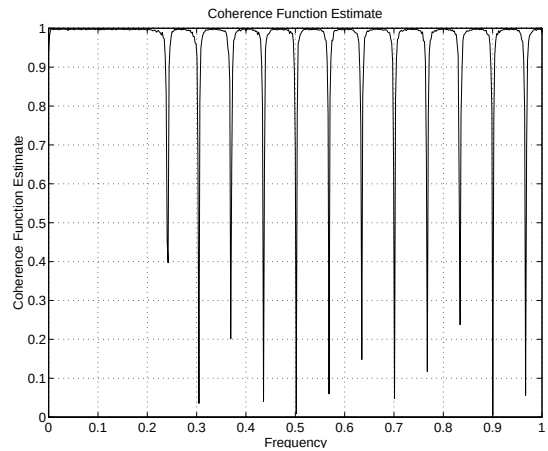
`cohere` with no output arguments plots the coherence estimate versus frequency in the current figure window.

Meta Platforms, Inc. - Exhibit 1009
Page 384 of 720

Example

Compute and plot the coherence estimate between two colored noise sequences x and y:

```
h = fir1(30,0.2,boxcar(31));
h1 = ones(1,10)/sqrt(10);
r = randn(16384,1);
x = filter(h1,1,r);
y = filter(h,1,x);
cohere(x,y,1024,[],[],512)
```



Diagnostics

An appropriate diagnostic message is displayed when incorrect arguments are used:

```
Requires window's length to be no greater than the FFT length.
Requires NOVERLAP to be strictly less than the window length.
Requires positive integer values for NFFT and NOVERLAP.
Requires vector (either row or column) input.
Requires inputs X and Y to have the same length.
```

Algorithm

cohere estimates the magnitude squared coherence function [1] using Welch's method of power spectrum estimation (see references [2] and [3]), as follows:

Meta Platforms, Inc. - Exhibit 1009
Page 385 of 720

1. It divides the signals `x` and `y` into overlapping sections, detrends each section, and multiplies each section by `window`.
2. It calculates the length `nfft` fast Fourier transform of each section.
3. It averages the squares of the spectra of the `x` sections to form `Pxx`, averages the squares of the spectra of the `y` sections to form `Pyy`, and averages the products of the spectra of the `x` and `y` sections to form `Pxy`. It calculates `Cxy` by

```
Cxy = abs(Pxy).^2/(Pxx.*Pyy)
```

See Also

`csd`	Estimate the cross spectral density (CSD) of two signals.
`pwelch`	Estimate the power spectral density (PSD) of a signal using Welch's method.
`tfe`	Transfer function estimate from input and output.

References

[1] Kay, S.M. *Modern Spectral Estimation*. Englewood Cliffs, NJ: Prentice-Hall, 1988. Pg. 454.

[2] Rabiner, L.R., and B. Gold. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

[3] Welch, P.D. "The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified Periodograms." *IEEE Trans. Audio Electroacoust*. Vol. AU-15 (June 1967). Pgs. 70-73.

Meta Platforms, Inc. - Exhibit 1009
Page 386 of 720

Purpose

Convolution and polynomial multiplication.

Syntax

```
c = conv(a,b)
```

Description

`conv(a,b)` convolves vectors `a` and `b`. The convolution sum is

$$c(n+1) = \sum_{k=0}^{N-1} a(k+1)b(n-k)$$

where N is the maximum sequence length. The series is indexed from $n + 1$ and $k + 1$ instead of the usual n and k because MATLAB vectors run from 1 to n instead of from 0 to n-1.

The `conv` function is part of the standard MATLAB language.

Example

The convolution of `a = [1 2 3]` and `b = [4 5 6]` is

```
c = conv(a,b)

c =
     4    13    28    27    18
```

Algorithm

The `conv` function is an M-file that uses the `filter` primitive. `conv` computes the convolution operation as FIR filtering with an appropriate number of zeros appended to the input.

See Also

`conv2`	Two-dimensional convolution.
`convmtx`	Convolution matrix.
`convn`	*N*-dimensional convolution (see the online *MATLAB Function Reference*).
`deconv`	Deconvolution and polynomial division.
`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
`residuez`	*z*-transform partial fraction expansion.
`xcorr`	Cross-correlation function estimate.

Meta Platforms, Inc. - Exhibit 1009
Page 387 of 720

Purpose

Two-dimensional convolution.

Syntax

```
C = conv2(A,B)
C = conv2(A,B,'shape')
```

Description

`C = conv2(A,B)` computes the two-dimensional convolution of matrices `A` and `B`. If one of these matrices describes a two-dimensional FIR filter, the other matrix is filtered in two dimensions.

Each dimension of the output matrix `C` is equal in size to the sum of the corresponding dimensions of the input matrices minus 1. For `[ma,na] = size(A)` and `[mb,nb] = size(B)`, then

```
size(C) = [ma+mb–1,na+nb–1]
```

`C = conv2(A,B,'shape')` returns a subsection of the two-dimensional convolution with size specified by *shape*, where:

- `full` returns the full two-dimensional convolution (default)
- `same` returns the central part of the convolution that is the same size as `A`
- `valid` returns only those parts of the convolution that are computed without the zero-padded edges. Using this option, `size(C) = [ma–mb+1,na–nb+1]` when `size(A)` > `size(B)`

`conv2` executes most quickly when `size(A)` > `size(B)`.

The `conv2` function is part of the standard MATLAB language.

Examples

In image processing, the Sobel edge-finding operation is a two-dimensional convolution of an input array with the special matrix

```
s = [1 2 1; 0 0 0; –1 –2 –1];
```

Given any image, the following line extracts the horizontal edges:

```
h = conv2(I,s);
```

The lines below extract first the vertical edges, then both horizontal and vertical edges combined:

```
v = conv2(I,s');
v2 = (sqrt(h.^2 + v.^2))
```


Meta Platforms, Inc. - Exhibit 1009
Page 388 of 720

See Also

`conv`	Convolution and polynomial multiplication.
`convn`	*N*-dimensional convolution (see the online *MATLAB Function Reference*).
`deconv`	Deconvolution and polynomial division.
`filter2`	Two-dimensional digital filtering.
`xcorr`	Cross-correlation function estimate.
`xcorr2`	Two-dimensional cross-correlation.

Meta Platforms, Inc. - Exhibit 1009
Page 389 of 720

Purpose

Convolution matrix.

Syntax

```
A = convmtx(c,n)
A = convmtx(r,n)
```

Description

A *convolution matrix* is a matrix, formed from a vector, whose inner product with another vector is the convolution of the two vectors.

`A = convmtx(c,n)` where `c` is a length `m` column vector returns a matrix `A` of size `(m + n–1)`-by-`n`. The product of `A` and another column vector `x` of length `n` is the convolution of `c` with `x`.

`A = convmtx(r,n)` where `r` is a length `m` row vector returns a matrix `A` of size `n`-by-`(m + n–1)`. The product of `A` and another row vector `x` of length `n` is the convolution of `r` with `x`.

Example

Generate a simple convolution matrix:

```
h = [1 2 3 2 1];
convmtx(h,7)
ans =

 1     2     3     2     1     0     0     0     0     0     0
 0     1     2     3     2     1     0     0     0     0     0
 0     0     1     2     3     2     1     0     0     0     0
 0     0     0     1     2     3     2     1     0     0     0
 0     0     0     0     1     2     3     2     1     0     0
 0     0     0     0     0     1     2     3     2     1     0
 0     0     0     0     0     0     1     2     3     2     1
```

Note that `convmtx` handles edge conditions by zero padding.

In practice, it is more efficient to compute convolution using

```
y = conv(c,x)
```

than by using a convolution matrix:

```
n = length(x);
y = convmtx(c,n)*x
```

Algorithm

`convmtx` uses the function `toeplitz` to generate the convolution matrix.

Meta Platforms, Inc. - Exhibit 1009
Page 390 of 720

See Also

`conv`	Convolution and polynomial multiplication.
`convn`	*N*-dimensional convolution (see the online *MATLAB Function Reference*).
`conv2`	Two-dimensional convolution.
`dftmtx`	Discrete Fourier transform matrix.

Meta Platforms, Inc. - Exhibit 1009
Page 391 of 720

Purpose

Correlation coefficient matrix.

Syntax

```
C = corrcoef(X)
C = corrcoef(X,Y)
```

Description

`corrcoef` returns a matrix of correlation coefficients calculated from an input matrix whose rows are observations and whose columns are variables. If `C = cov(X)`, then `corrcoef(X)` is the matrix whose element (i, j) is

$$\text{corrcoef}(i, j) = \frac{C(i, j)}{\sqrt{C(i, i)C(j, j)}}$$

`C = corrcoef(X)` is the zeroth lag of the covariance function, that is, the zeroth lag of `xcov(x,'coeff')` packed into a square array.

`C = corrcoef(X,Y)` is the same as `corrcoef([X Y])`, that is, it concatenates `X` and `Y` in the row direction before its computation.

`corrcoef` removes the mean from each column before calculating the results. See the `xcorr` function for cross-correlation options.

The `corrcoef` function is part of the standard MATLAB language.

See Also

`cov`	Covariance matrix.
`mean`	Average value (see the online *MATLAB Function Reference*).
`median`	Median value (see the online *MATLAB Function Reference*).
`std`	Standard deviation (see the online *MATLAB Function Reference*).
`xcorr`	Cross-correlation function estimate.
`xcov`	Cross-covariance function estimate (equal to mean-removed cross-correlation).

Meta Platforms, Inc. - Exhibit 1009
Page 392 of 720

Purpose

Covariance matrix.

Syntax

```
c = cov(x)
c = cov(x,y)
```

Description

`cov` computes the covariance matrix. If `x` is a vector, `c` is a scalar containing the variance. For an array where each row is an observation and each column a variable, `cov(X)` is the covariance matrix. `diag(cov(X))` is a vector of variances for each column, and `sqrt(diag(cov(X)))` is a vector of standard deviations.

`cov(x)` is the zeroth lag of the covariance function, that is, the zeroth lag of `xcov(x)/(n-1)` packed into a square array.

`cov(x,y)` where `x` and `y` are column vectors of equal length is equivalent to `cov([x y]))`, that is, it concatenates `x` and `y` in the row direction before its computation.

`cov` removes the mean from each column before calculating the results.

The `cov` function is part of the standard MATLAB language.

Algorithm

```
[n,p] = size(x);
x = x-ones(n,1)*(sum(x)/n);
y = x'*x/(n-1);
```

See Also

`corrcoef`	Correlation coefficient matrix.
`mean`	Average value (see the online *MATLAB Function Reference*).
`median`	Median value (see the online *MATLAB Function Reference*).
`std`	Standard deviation (see the online *MATLAB Function Reference*).
`xcorr`	Cross-correlation function estimate.
`xcov`	Cross-covariance function estimate (equal to mean-removed cross-correlation).

Meta Platforms, Inc. - Exhibit 1009
Page 393 of 720

Purpose

Group complex numbers into complex conjugate pairs.

Syntax

```
y = cplxpair(x)
y = cplxpair(x,tol)
```

Description

`y = cplxpair(x)` returns `x` with complex conjugate pairs grouped together. `cplxpair` orders the conjugate pairs by increasing real part. Within a pair, the element with negative imaginary part comes first. The function returns all purely real values following all the complex pairs.

`y = cplxpair(x,tol)` includes a tolerance, `tol`, for determining which numbers are real and which are paired complex conjugates. By default, `cplxpair` uses a tolerance of `100*eps` relative to `abs(x(i))`. `cplxpair` forces the complex conjugate pairs to be exact complex conjugates.

The `cplxpair` function is part of the standard MATLAB language.

Example

Order five poles evenly spaced around the unit circle into complex pairs:

```
cplxpair(exp(2*pi*sqrt(-1)*(0:4)/5)')
```

```
ans =
    -0.8090 - 0.5878i
    -0.8090 + 0.5878i
     0.3090 - 0.9511i
     0.3090 + 0.9511i
     1.0000
```

Diagnostics

If there is an odd number of complex numbers, or if the complex numbers cannot be grouped into complex conjugate pairs within the tolerance, `cplxpair` generates the error message:

```
Complex numbers can't be paired.
```


Meta Platforms, Inc. - Exhibit 1009
Page 394 of 720

Purpose

Complex and nonlinear-phase equiripple FIR filter design.

Syntax

```
b = cremez(n,f,'fresp')
b = cremez(n,f,'fresp',w)
b = cremez(n,f,{'fresp',p1,p2,...},w)
b = cremez(n,f,a,w)
b = cremez(...,'sym')
b = cremez(...,'skip_stage2')
b = cremez(...,'debug')
b = cremez(...,{lgrid})
[b,delta,opt] = cremez(...)
```

Description

`cremez` allows arbitrary frequency-domain constraints to be specified for the design of a possibly complex FIR filter. The Chebyshev (or minimax) filter error is optimized, producing equiripple FIR filter designs.

`b = cremez(n,f,'fresp')` returns a length `n+1` FIR filter with the best approximation to the desired frequency response as returned by function `fresp`. `f` is a vector of frequency band edge pairs, specified in the range -1 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency). The frequencies must be in increasing order, and `f` must have even length. The frequency bands span `f(k)` to `f(k+1)` for `k` odd; the intervals `f(k+1)` to `f(k+2)` for `k` odd are “transition bands” or “don’t care” regions during optimization.

`b = cremez(n,f,'fresp',w)` uses the real, non-negative weights in vector `w` to weight the fit in each frequency band. The length of `w` is half the length of `f`, so there is exactly one weight per band.

`b = cremez(n,f,{'fresp',p1,p2,...},...)` supplies optional parameters `p1, p2, ...`, to the frequency response function *`fresp`*. Predefined `'fresp'` frequency response functions are included for a number of common filter designs, as described below. For all of the predefined frequency response functions, the symmetry option `'sym'` defaults to `'even'` if no negative frequencies are contained in `f` and `d = 0`; otherwise `'sym'` defaults to `'none'`. (See the `'sym'` option below for details.) For all of the predefined frequency response functions, `d` specifies a group-delay offset such that the filter response

Meta Platforms, Inc. - Exhibit 1009
Page 395 of 720

has a group delay of `n/2+d` in units of the sample interval. Negative values create less delay; positive values create more delay. By default, `d = 0`.

- `lowpass, highpass, bandpass, bandstop`

  These functions share a common syntax, exemplified here by `'lowpass'`:

  `b = cremez(n,f,'lowpass',...)` and

  `b = cremez(n,f,{'lowpass',d},...)` design a linear-phase (`n/2+d` delay) filter.

- `multiband` designs a linear-phase frequency response filter with arbitrary band amplitudes.

  `b = cremez(n,f,{'multiband',a},...)` and

  `b = cremez(n,f,{'multiband',a,d},...)` specify vector `a` containing the desired amplitudes at the band edges in `f`. The desired amplitude at frequencies between pairs of points `f(k)` and `f(k+1)` for `k` odd is the line segment connecting the points `(f(k),a(k))` and `(f(k+1),a(k+1))`.

- `differentiator` designs a linear-phase differentiator. For these designs, zero-frequency must be in a transition band, and band weighting is set to be inversely proportional to frequency.

  `b = cremez(n,f,{'differentiator',Fs},...)` and

  `b = cremez(n,f,{'differentiator',Fs,d},...)` specify the sample rate `Fs` used to determine the slope of the differentiator response. If omitted, `Fs` defaults to 1.

- `hilbfilt` designs a linear-phase Hilbert transform filter response. For Hilbert designs, zero-frequency must be in a transition band.

  `b = cremez(n,f,'hilbfilt',...)` and

  `b = cremez(N,F,{'hilbfilt',d},...)` design a linear-phase (`n/2+d` delay) Hilbert transform filter.

`b = cremez(n,f,a,w)` is a synonym for
`b = cremez(n,f,{'multiband',a},w)`.

`b = cremez(...,'sym')` imposes a symmetry constraint on the impulse response of the design, where `'sym'` may be one of the following:

Meta Platforms, Inc. - Exhibit 1009
Page 396 of 720

- `'none'` indicates no symmetry constraint

  This is the default if any negative band edge frequencies are passed, or if `'fresp'` does not supply a default.
- `'even'` indicates a real and even impulse response

  This is the default for highpass, lowpass, bandpass, bandstop, and multiband designs.
- `'odd'` indicates a real and odd impulse response

  This is the default for Hilbert and differentiator designs.
- `'real'` indicates conjugate symmetry for the frequency response

If any `'sym'` option other than `'none'` is specified, the band edges should only be specified over positive frequencies; the negative frequency region is filled in from symmetry. If a `'sym'` option is not specified, the `'fresp'` function is queried for a default setting.

`b = cremez(...,'skip_stage2')` disables the second-stage optimization algorithm, which executes only when `cremez` determines that an optimal solution has not been reached by the standard Remez error-exchange. Disabling this algorithm may increase the speed of computation, but may incur a reduction in accuracy. By default, the second-stage optimization is enabled.

`b = cremez(...,'debug')` enables the display of intermediate results during the filter design, where `'debug'` may be one of `'trace'`, `'plots'`, `'both'`, or `'off'`. By default, it is set to `'off'`.

`b = cremez(...,{lgrid})` uses the integer `lgrid` to control the density of the frequency grid, which has roughly `2^nextpow2(lgrid*n)` frequency points. The default value for `lgrid` is `25`. Note that the `{lgrid}` argument must be a 1-by-1 cell array.

Any combination of the `'sym'`, `'skip_stage2'`, `'debug'`, and `{lgrid}` options may be specified.

`[b,delta] = cremez(...)` returns the maximum ripple height `delta`.

Meta Platforms, Inc. - Exhibit 1009
Page 397 of 720

`[b,delta,opt] = cremez(...)` returns a structure `opt` of optional results computed by `cremez` and contains the following fields:

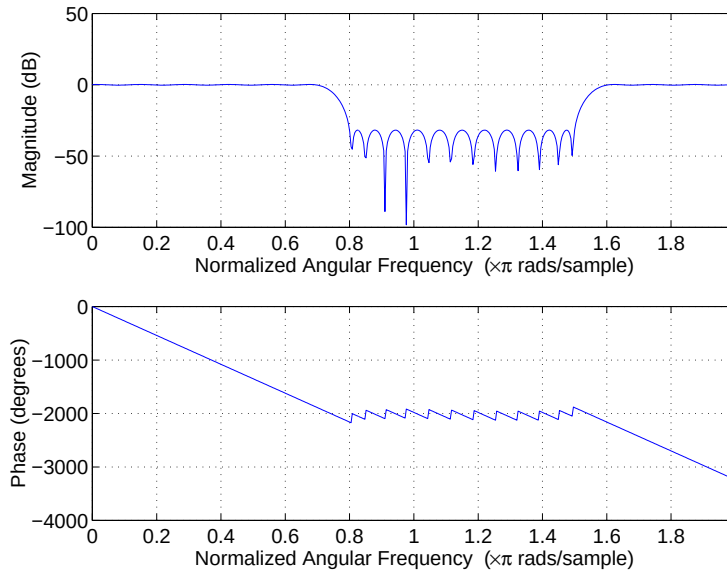
`opt.fgrid`	Frequency grid vector used for the filter design optimization
`opt.des`	Desired frequency response for each point in `opt.fgrid`
`opt.wt`	Weighting for each point in `opt.fgrid`
`opt.H`	Actual frequency response for each point in `opt.fgrid`
`opt.error`	Error at each point in `opt.fgrid`
`opt.iextr`	Vector of indices into `opt.fgrid` for extremal frequencies
`opt.fextr`	Vector of extremal frequencies

Examples

Example 1

Design a 31-tap, linear-phase, lowpass filter:

```
b = cremez(30,[-1 -0.5 -0.4 0.7 0.8 1],'lowpass');
freqz(b,1,512,'whole');
```



Meta Platforms, Inc. - Exhibit 1009
Page 398 of 720

Example 2

Design a nonlinear-phase allpass FIR filter.

First select (or create) the function *fresp* that returns the desired frequency response. For this example, *fresp* is the `allpass.m` function in the `signal/signal/private` directory which returns the frequency response of a nonlinear-phase allpass filter. Copy `allpass.m` to another location on the MATLAB path before trying the example.

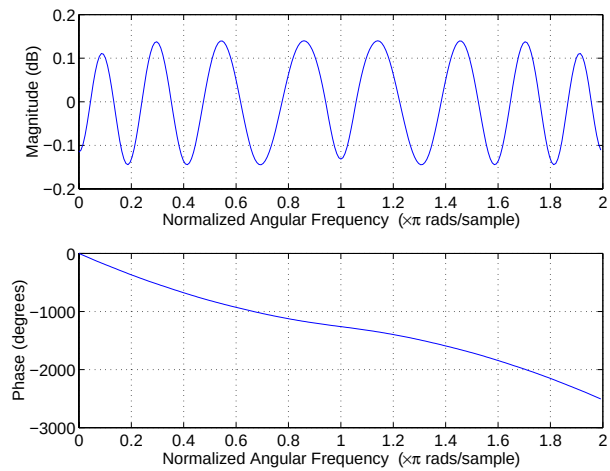
Before using `cremez` with `allpass.m` to generate the filter coefficients, call `allpass` alone to create the *desired* response.

```
n = 22;                     % Filter order
f = [-1 1];                 % Frequency band edges
w = [1 1];                  % Weights for optimization
gf = linspace(-1,1,256);    % Grid of frequency points
d = allpass(n,f,gf,w);      % Desired frequency response
```

Vector `d` now contains the complex frequency response that we desire for the FIR filter computed by `cremez`.

Now compute the FIR filter that best approximates this response:

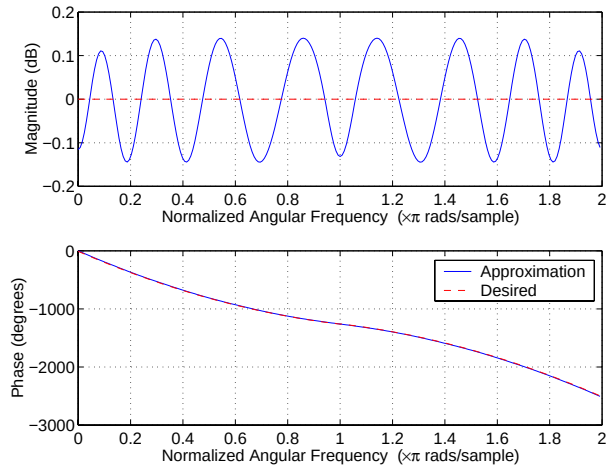
```
b = cremez(n,f,'allpass',w,'real' ); % Approximation
freqz(b,1,256,'whole');
```



Meta Platforms, Inc. - Exhibit 1009
Page 399 of 720

The freqz plot shows the frequency response of the filter computed by cremez to approximate the desired response. Check the accuracy of the approximation by overlaying the *desired* frequency response on the plot.

```
subplot(2,1,1); hold on
plot(pi*(gf+1),20*log10(abs(fftshift(d))),'r--')

subplot(2,1,2); hold on
plot(pi*(gf+1),unwrap(angle(fftshift(d)))*180/pi,'r--')
legend('Approximation','Desired')
```



Remarks

User-definable functions may be used, instead of the predefined frequency response functions for '*fresp*'. The function is called from within cremez using the following syntax:

[dh,dw] = *fresp*(n,f,gf,w,p1,p2,...) where

- n is the filter order.
- f is the vector of frequency band edges that appear monotonically between -1 and 1, where 1 is the Nyquist frequency.
- gf is a vector of grid points that have been linearly interpolated over each specified frequency band by cremez. gf determines the frequency grid at which the response function must be evaluated. This is the same data returned by cremez in the fgrid field of the opt structure.

Meta Platforms, Inc. - Exhibit 1009
Page 400 of 720

- `w` is a vector of real, positive weights, one per band, used during optimization. `w` is optional in the call to `cremez`; if not specified, it is set to unity weighting before being passed to `'fresp'`.
- `dh` and `dw` are the desired complex frequency response and band weight vectors, respectively, evaluated at each frequency in grid `gf`.
- `p1`, `p2`, ..., are optional parameters that may be passed to `'fresp'`.

Additionally, a preliminary call is made to `'fresp'` to determine the default symmetry property `'sym'`. This call is made using the syntax:

```
sym = fresp('defaults',{n,f,[],w,p1,p2,...})
```

The arguments may be used in determining an appropriate symmetry default as necessary. The function `private/lowpass.m` may be useful as a template for generating new frequency response functions.

Algorithm

An extended version of the Remez exchange method is implemented for the complex case. This exchange method obtains the optimal filter when the equiripple nature of the filter is restricted to have `n+2` extremals. When it does not converge, the algorithm switches to an ascent-descent algorithm that takes over to finish the convergence to the optimal solution. See the references for further details.

Meta Platforms, Inc. - Exhibit 1009
Page 401 of 720

Diagnostics

The following diagnostic messages arise from incorrect usage of `cremez`:

`Not enough input arguments.`

`F must contain an even number of band edge entries.`

`Band edges must be monotonically increasing.`

`Expecting a string argument.`

`Invalid argument` *`arg`* `specified.`

`Invalid default symmetry option` *`sym`* `returned from response function` *`fresp`*`. Must be one of 'none', 'real', 'even', or 'odd'.`

`Frequency band edges must be in the range [–1,+1] for designs with SYM = '`*`sym`*`'.`

`Frequency band edges must be in the range [0,+1] for designs with SYM = '`*`sym`*`'.`

`Incorrect size of results from response function` *`fresp`*`. Sizes must be the same size as the frequency grid GF.`

`Both –1 and 1 have been specified as frequencies in F, and the frequency spacing is too close to move either of them toward its neighbor.`

`Internal error: Grid frequencies out of range.`

`Internal error: domain must be "whole" or "half".`

`Internal error: obtained a negative bandwidth.`

`Internal error: two extremal frequecies at the same grid point.`

`Internal error: dBrange must be > 0.`

Meta Platforms, Inc. - Exhibit 1009
Page 402 of 720

See Also

`fir1`	Window-based finite impulse response filter design – standard response.
`fir2`	Window-based finite impulse response filter design – arbitrary response.
`firls`	Least square linear-phase FIR filter design.
`remez`	Parks-McClellan optimal FIR filter design.
`private/bandpass`	Bandpass filter design function.
`private/bandstop`	Bandstop filter design function.
`private/differentiator`	Differentiator filter design function.
`private/highpass`	Highpass filter design function.
`private/hilbfilt`	Hilbert filter design function.
`private/lowpass`	Lowpass filter design function.
`private/multiband`	Multiband filter design function.

References

[1] Karam, L.J., and J.H. McClellan. "Complex Chebyshev Approximation for FIR Filter Design." *IEEE Trans. on Circuits and Systems II*. March 1995. Pgs. 207-216.

[2] Karam, L.J. *Design of Complex Digital FIR Filters in the Chebyshev Sense*. Ph.D. Thesis, Georgia Institute of Technology, March 1995.

[3] Demjanjov, V.F., and V.N. Malozemov. *Introduction to Minimax*. New York: John Wiley & Sons, 1974.

Meta Platforms, Inc. - Exhibit 1009
Page 403 of 720

Purpose

Estimate the cross spectral density (CSD) of two signals.

Syntax

```
Pxy = csd(x,y)
Pxy = csd(x,y,nfft)
[Pxy,f] = csd(x,y,nfft,Fs)
Pxy = csd(x,y,nfft,Fs,window)
Pxy = csd(x,y,nfft,Fs,window,noverlap)
Pxy = csd(x,y,...,'dflag')
[Pxy,Pxyc,f] = csd(x,y,nfft,Fs,window,noverlap,p)
csd(x,y,...)
```

Description

`Pxy = csd(x,y)` estimates the cross spectral density of the length `n` sequences `x` and `y` using the Welch method of spectral estimation. `Pxy = csd(x,y)` uses the following default values:

- `nfft = min(256,length(x))`
- `Fs = 2`
- `window = hanning(nfft)`
- `noverlap = 0`

`nfft` specifies the FFT length that `csd` uses. This value determines the frequencies at which the cross spectrum is estimated. `Fs` is a scalar that specifies the sampling frequency. `window` specifies a windowing function and the number of samples `csd` uses in its sectioning of the `x` and `y` vectors. `noverlap` is the number of samples by which the sections overlap. Any arguments omitted from the end of the parameter list use the default values shown above.

If `x` and `y` are real, `csd` estimates the cross spectral density at positive frequencies only; in this case, the output `Pxy` is a column vector of length `nfft/2 + 1` for `nfft` even and `(nfft + 1)/2` for `nfft` odd. If `x` or `y` is complex, `csd` estimates the cross spectral density at both positive and negative frequencies and `Pxy` has length `nfft`.

`Pxy = csd(x,y,nfft)` uses the FFT length `nfft` in estimating the cross spectral density of `x` and `y`. Specify `nfft` as a power of 2 for fastest execution.

`[Pxy,f] = csd(x,y,nfft,Fs)` returns a vector `f` of frequencies at which the function evaluates the CSD. `f` is the same size as `Pxy`, so `plot(f,Pxy)` plots the

Meta Platforms, Inc. - Exhibit 1009
Page 404 of 720

spectrum versus properly scaled frequency. `Fs` has no effect on the output `Pxy`; it is a frequency scaling multiplier.

`Pxy = csd(x,y,nfft,Fs,window)` specifies a windowing function and the number of samples per section of the `x` vector. If you supply a scalar for `window`, `csd` uses a Hanning window of that length. The length of the window must be less than or equal to `nfft`; `csd` zero pads the sections if the length of the window is less than `nfft`. `csd` returns an error if the length of the window is greater than `nfft`.

`Pxy = csd(x,y,nfft,Fs,window,noverlap)` overlaps the sections of `x` and `y` by `noverlap` samples.

You can use the empty matrix `[]` to specify the default value for any input argument except `x` or `y`. For example,

```
csd(x,y,[],10000)
```

is equivalent to

```
csd(x)
```

but with a sampling frequency of 10,000 Hz instead of the default of 2 Hz.

`Pxy = csd(x,y,...,'dflag')` specifies a detrend option, where *`dflag`* is:

- `linear`, to remove the best straight-line fit from the prewindowed sections of `x` and `y`
- `mean`, to remove the mean from the prewindowed sections of `x` and `y`
- `none`, for no detrending (default)

The *`dflag`* parameter must appear last in the list of input arguments. `csd` recognizes a *`dflag`* string no matter how many intermediate arguments are omitted.

`[Pxy,Pxyc,f] = csd(x,y,nfft,Fs,window,noverlap,p)` where `p` is a positive scalar between 0 and 1 returns a vector `Pxyc` that contains an estimate of the `p*100` percent confidence interval for `Pxy`. `Pxyc` is a two-column matrix the same length as `Pxy`. The interval `[Pxyc(:,1), Pxyc(:,2)]` covers the true CSD with probability `p`. `plot(f,[Pxy Pxyc])` plots the cross spectrum inside the `p*100` percent confidence interval. If unspecified, `p` defaults to 0.95.

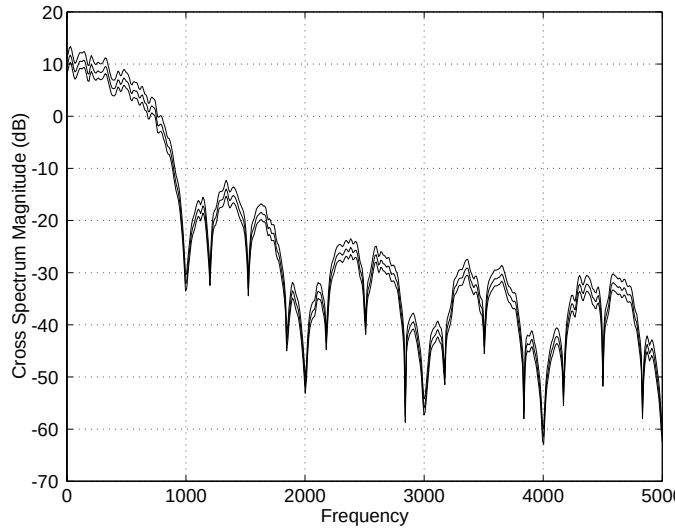
Meta Platforms, Inc. - Exhibit 1009
Page 405 of 720

`csd(x,y,...)` plots the CSD versus frequency in the current figure window. If the `p` parameter is specified, the plot includes the confidence interval.

Example

Generate two colored noise signals and plot their CSD with a confidence interval of 95%. Specify a length 1024 FFT, a 500 point triangular window with no overlap, and a sampling frequency of 10 Hz:

```
h = fir1(30,0.2,boxcar(31));
h1 = ones(1,10)/sqrt(10);
r = randn(16384,1);
x = filter(h1,1,r);
y = filter(h,1,x);
csd(x,y,1024,10000,triang(500),0,[])
```



Algorithm

`csd` implements the Welch method of spectral density estimation (see references [1] and [2]):

1. It applies the window specified by the `window` vector to each successive detrended section.
2. It transforms each section with an `nfft`-point FFT.
3. It forms the periodogram of each section by scaling the product of the transform of the `y` section and the conjugate of the transformed `x` section.

Meta Platforms, Inc. - Exhibit 1009
Page 406 of 720

4 It averages the periodograms of the successive overlapping sections to form Pxy, the cross spectral density of x and y.

The number of sections that csd averages is k, where k is

```
fix((length(x)-noverlap)/(length(window)-noverlap))
```

Diagnostics

An appropriate diagnostic message is displayed when incorrect arguments to csd are used:

```
Requires window's length to be no greater than the FFT length.
Requires NOVERLAP to be strictly less than the window length.
Requires positive integer values for NFFT and NOVERLAP.
Requires vector (either row or column) input.
Requires inputs X and Y to have the same length.
Requires confidence parameter to be a scalar between 0 and 1.
```

See Also

cohere	Estimate magnitude squared coherence function between two signals.
pburg	Power spectrum estimate using the Burg method.
pmtm	Power spectrum estimate using the multitaper method (MTM).
pmusic	Power spectrum estimate using MUSIC eigenvector method.
pwelch	Estimate the power spectral density (PSD) of a signal using Welch's method.
pyulear	Power spectrum estimate using Yule-Walker AR method.
tfe	Transfer function estimate from input and output.

References

[1] Rabiner, L.R., and B. Gold. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975. Pgs. 414-419.

[2] Welch, P.D. "The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified Periodograms." *IEEE Trans. Audio Electroacoust*. Vol. AU-15 (June 1967). Pgs. 70-73.

Meta Platforms, Inc. - Exhibit 1009
Page 407 of 720

[3] Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989. Pg. 737.

Meta Platforms, Inc. - Exhibit 1009
Page 408 of 720

Purpose

Chirp *z*-transform.

Syntax

```
y = czt(x,m,w,a)
y = czt(x)
```

Description

`y = czt(x,m,w,a)` returns the chirp *z*-transform of signal `x`. The chirp *z*-transform is the *z*-transform of `x` along a spiral contour defined by `w` and `a`. `m` is a scalar that specifies the length of the transform, `w` is the ratio between points along the *z*-plane spiral contour of interest, and scalar `a` is the complex starting point on that contour. The contour, a spiral or "chirp" in the *z*-plane, is given by

```
z = a*(w.^-(0:m-1))
```

`y = czt(x)` uses the following default values:

- `m = length(x)`
- `w = exp(j*2*pi/m)`
- `a = 1`

With these defaults, `czt` returns the *z*-transform of `x` at `m` equally spaced points around the unit circle. This is equivalent to the discrete Fourier transform of `x`, or `fft(x)`. The empty matrix `[]` specifies the default value for a parameter.

If `x` is a matrix, `czt(x,m,w,a)` transforms the columns of `x`.

Examples

Create a random vector `x` of length 1013 and compute its DFT using `czt`. This is faster than the `fft` function on the same sequence.

```
x = randn(1013,1);
y = czt(x);
```

Use `czt` to zoom in on a narrow-band section (100 to 150 Hz) of a filter's frequency response. First design the filter:

```
h = fir1(30,125/500,boxcar(31)); % filter
```


Meta Platforms, Inc. - Exhibit 1009
Page 409 of 720

Establish frequency and CZT parameters:

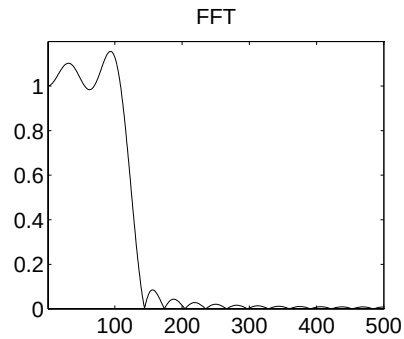
```
Fs = 1000; f1 = 100; f2 = 150;  % in Hertz
m = 1024;
w = exp(-j*2*pi*(f2-f1)/(m*Fs));
a = exp(j*2*pi*f1/Fs);
```

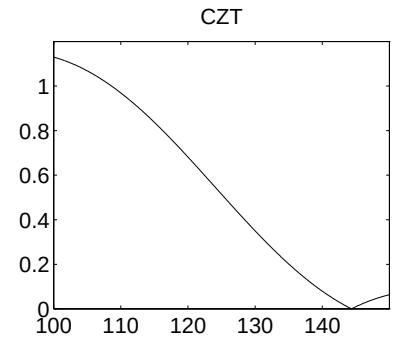
Compute both the DFT and CZT of the filter:

```
y = fft(h,1000);
z = czt(h,m,w,a);
```

Create frequency vectors and compare the results:

```
fy = (0:length(y)-1)'*1000/length(y);
fz = ((0:length(z)-1)'*(f2-f1)/length(z)) + f1;
plot(fy(1:500),abs(y(1:500))); axis([1 500 0 1.2])
title('FFT')
figure
plot(fz,abs(z)); axis([f1 f2 0 1.2])
title('CZT')
```





Algorithm

`czt` uses the next power-of-2 length FFT to perform a fast convolution when computing the *z*-transform on a specified chirp contour [1]. `czt` can be significantly faster than `fft` for large, prime-length sequences.

Diagnostics

If `m`, `w`, or `a` is not a scalar, `czt` gives the following error message:

```
Inputs M, W, and A must be scalars.
```


Meta Platforms, Inc. - Exhibit 1009
Page 410 of 720

See Also

`fft`	One-dimensional fast Fourier transform.
`freqz`	Frequency response of digital filters.

References

[1] Rabiner, L.R., and B. Gold. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975. Pgs. 393-399.

Meta Platforms, Inc. - Exhibit 1009
Page 411 of 720

Purpose Discrete cosine transform (DCT).

Syntax

```
y = dct(x)
y = dct(x,n)
```

Description `y = dct(x)` returns the unitary discrete cosine transform of `x`

$$y(k) = w(k) \sum_{n=1}^{N} x(n) \cos \frac{\pi(2n-1)(k-1)}{2N}, \qquad k = 1, \ldots, N$$

where

$$w(k) = \begin{cases} \frac{1}{\sqrt{N}}, & k = 1 \\ \sqrt{\frac{2}{N}}, & 2 \le k \le N \end{cases}$$

N is the length of `x`, and `x` and `y` are the same size. If `x` is a matrix, `dct` transforms its columns. The series is indexed from $n = 1$ and $k = 1$ instead of the usual $n = 0$ and $k = 0$ because MATLAB vectors run from 1 to N instead of from 0 to N- 1.

`y = dct(x,n)` pads or truncates `x` to length `n` before transforming.

The DCT is closely related to the discrete Fourier transform. You can often reconstruct a sequence very accurately from only a few DCT coefficients, a useful property for applications requiring data reduction.

Meta Platforms, Inc. - Exhibit 1009
Page 412 of 720

Example

Find how many DCT coefficients represent 99% of the energy in a sequence:

```
x = (1:100) + 50*cos((1:100)*2*pi/40);
X = dct(x);
[XX,ind] = sort(abs(X)); ind = fliplr(ind);
i = 1;
while (norm([X(ind(1:i)) zeros(1,100-i)])/norm(X)<.99)
   i = i + 1;
end
```

```
i =
      3
```

See Also

`fft`	One-dimensional fast Fourier transform.
`idct`	Inverse discrete cosine transform.
`dct2`	Two-dimensional DCT (see *Image Processing Toolbox User's Guide*).
`idct2`	Two-dimensional inverse DCT (see *Image Processing Toolbox User's Guide*).

References

[1] Jain, A.K. *Fundamentals of Digital Image Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[2] Pennebaker, W.B., and J.L. Mitchell. *JPEG Still Image Data Compression Standard*. New York, NY: Van Nostrand Reinhold, 1993. Chapter 4.

Meta Platforms, Inc. - Exhibit 1009
Page 413 of 720

Purpose

Decrease the sampling rate for a sequence (decimation).

Syntax

```
y = decimate(x,r)
y = decimate(x,r,n)
y = decimate(x,r,'fir')
y = decimate(x,r,n,'fir')
```

Description

Decimation reduces the original sampling rate for a sequence to a lower rate. It is the opposite of interpolation. The decimation process filters the input data with a lowpass filter and then resamples the resulting smoothed signal at a lower rate.

`y = decimate(x,r)` reduces the sample rate of `x` by a factor `r`. The decimated vector `y` is `r` times shorter in length than the input vector `x`. By default, `decimate` employs an eighth-order lowpass Chebyshev type I filter. It filters the input sequence in both the forward and reverse directions to remove all phase distortion, effectively doubling the filter order.

`y = decimate(x,r,n)` uses an order `n` Chebyshev filter. Orders above 13 are not recommended because of numerical instability. MATLAB displays a warning in this case.

`y = decimate(x,r,'fir')` uses a 30-point FIR filter, instead of the Chebyshev IIR filter. Here `decimate` filters the input sequence in only one direction. This technique conserves memory and is useful for working with long sequences.

`y = decimate(x,r,n,'fir')` uses a length `n` FIR filter.

Example

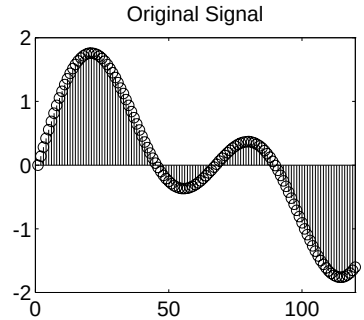
Decimate a signal by a factor of four:

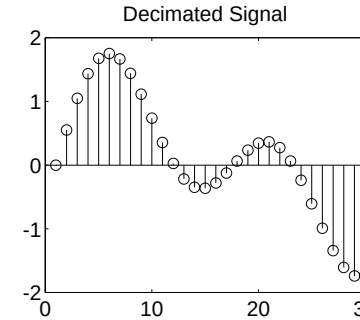
```
t = 0:.00025:1;                        % time vector
x = sin(2*pi*30*t) + sin(2*pi*60*t);
y = decimate(x,4);
```


Meta Platforms, Inc. - Exhibit 1009
Page 414 of 720

View the original and decimated signals:

```
stem(x(1:120)), axis([0 120 -2 2])   % original signal
title('Original Signal')
figure
stem(y(1:30))                        % decimated signal
title('Decimated Signal')
```





Algorithm

`decimate` uses decimation algorithms 8.2 and 8.3 from [1]:

1. It designs a lowpass filter. By default, `decimate` uses a Chebyshev type I filter with normalized cutoff frequency 0.8/`r` and 0.05 dB of passband ripple. For the `fir` option, `decimate` designs a lowpass FIR filter with cutoff frequency `1/r` using `fir1`.
2. For the FIR filter, `decimate` applies the filter to the input vector in one direction. In the IIR case, `decimate` applies the filter in forward and reverse directions with `filtfilt`.
3. `decimate` resamples the filtered data by selecting every `r`-th point.

Diagnostics

If `r` is not an integer, `decimate` gives the following error message:

```
Resampling rate R must be an integer.
```

If `n` specifies an IIR filter with order greater than 13, `decimate` gives the following warning:

```
Warning: IIR filters above order 13 may be unreliable.
```


Meta Platforms, Inc. - Exhibit 1009
Page 415 of 720

See Also

`interp`	Increase sampling rate by an integer factor (interpolation).
`resample`	Change sampling rate by any rational factor.
`spline`	Cubic spline interpolation (see the online *MATLAB Function Reference*).
`upfirdn`	Upsample, apply an FIR filter, and downsample.

References

[1] IEEE. *Programs for Digital Signal Processing*. IEEE Press. New York: John Wiley & Sons, 1979. Chapter 8.

Meta Platforms, Inc. - Exhibit 1009
Page 416 of 720

Purpose

Deconvolution and polynomial division.

Syntax

```
[q,r] = deconv(b,a)
```

Description

`[q,r] = deconv(b,a)` deconvolves vector `a` out of vector `b`, using long division. The result (quotient) is returned in vector `q` and the remainder in vector `r` such that `b = conv(q,a) + r`.

If `a` and `b` are vectors of polynomial coefficients, convolving them is equivalent to polynomial multiplication, and deconvolution is equivalent to polynomial division. The result of dividing `b` by `a` is quotient `q` and remainder `r`.

The `deconv` function is part of the standard MATLAB language.

Example

The convolution of `a = [1 2 3]` and `b = [4 5 6]` is

```
c = conv(a,b)

c =
    4    13    28    27    18
```

Use `deconv` to divide `b` back out:

```
[q,r] = deconv(c,a)

q =
    4    5    6

r =
    0    0    0    0    0
```

Algorithm

This function calls `filter` to compute the deconvolution as the impulse response of an IIR filter.

See Also

`conv`	Convolution and polynomial multiplication.
`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
`residuez`	*z*-transform partial fraction expansion.

Meta Platforms, Inc. - Exhibit 1009
Page 417 of 720

Purpose

Demodulation for communications simulation.

Syntax

```
x = demod(y,Fc,Fs,'method')
x = demod(y,Fc,Fs,'method',opt)
x = demod(y,Fc,Fs,'pwm','centered')
[x1,x2] = demod(y,Fc,Fs,'qam')
```

Description

`demod` performs demodulation, that is, it obtains the original signal from a modulated version of the signal. `demod` undoes the operation performed by `modulate`.

`x = demod(y,Fc,Fs,'method')` and

`x = demod(y,Fc,Fs,'method',opt)` demodulate the real carrier signal `y` with a carrier frequency `Fc` and sampling frequency `Fs`, using one of the options listed below for *`method`*. (Note that some methods accept an option, `opt`.)

`amdsb–sc` or `am`

Amplitude demodulation, double sideband, suppressed carrier. Multiplies `y` by a sinusoid of frequency `Fc` and applies a fifth-order Butterworth lowpass filter using `filtfilt`:

```
x = y.*cos(2*pi*Fc*t);
[b,a] = butter(5,Fc*2/Fs);
x = filtfilt(b,a,x);
```

`amdsb–tc`

Amplitude demodulation, double sideband, transmitted carrier. Multiplies `y` by a sinusoid of frequency `Fc`, and applies a fifth-order Butterworth lowpass filter using `filtfilt`:

```
x = y.*cos(2*pi*Fc*t);
[b,a] = butter(5,Fc*2/Fs);
x = filtfilt(b,a,x);
```

If you specify `opt`, `demod` subtracts scalar `opt` from `x`. The default value for `opt` is 0.

`amssb`

Amplitude demodulation, single sideband. Multiplies `y` by a sinusoid of frequency `Fc` and applies a fifth-order Butterworth lowpass filter using `filtfilt`:

```
x = y.*cos(2*pi*Fc*t);
[b,a] = butter(5,Fc*2/Fs);
x = filtfilt(b,a,x);
```


Meta Platforms, Inc. - Exhibit 1009
Page 418 of 720

`fm` **Frequency demodulation.** Demodulates the FM waveform by modulating the Hilbert transform of `y` by a complex exponential of frequency –`Fc` Hz and obtains the instantaneous frequency of the result.

`pm` **Phase demodulation.** Demodulates the PM waveform by modulating the Hilbert transform of `y` by a complex exponential of frequency –`Fc` Hz and obtains the instantaneous phase of the result.

`ptm` **Pulse-time demodulation.** Finds the pulse times of a pulse-time modulated signal `y`. For correct demodulation, the pulses cannot overlap. `x` is length `length(t)*Fc/Fs`.

`pwm` **Pulse-width demodulation.** Finds the pulse widths of a pulse-width modulated signal `y`. `demod` returns in `x` a vector whose elements specify the width of each pulse in fractions of a period. The pulses in `y` should start at the beginning of each carrier period, that is, they should be left justified.

`qam` **Quadrature amplitude demodulation.** `[x1,x2] = demod(y,Fc,Fs,'qam')` multiplies `y` by a cosine and a sine of frequency `Fc` and applies a fifth-order Butterworth lowpass filter using `filtfilt`:

```
x1 = y.*cos(2*pi*Fc*t);
x2 = y.*sin(2*pi*Fc*t);
[b,a] = butter(5,Fc*2/Fs);
x1 = filtfilt(b,a,x1);
x2 = filtfilt(b,a,x2);
```

The default method is `'am'`. Except for the `'ptm'` and `'pwm'` cases, `x` is the same size as `y`.

If `y` is a matrix, `demod` demodulates its columns.

`x = demod(y,Fc,Fs,'pwm','centered')` finds the pulse widths assuming they are centered at the beginning of each period. `x` is length `length(y)*Fc/Fs`.

Meta Platforms, Inc. - Exhibit 1009
Page 419 of 720

See Also

`modulate`	Modulation for communications simulation.
`vco`	Voltage controlled oscillator.

Meta Platforms, Inc. - Exhibit 1009
Page 420 of 720

Purpose

Discrete Fourier transform matrix.

Syntax

```
A = dftmtx(n)
```

Description

A *discrete Fourier transform matrix* is a complex matrix of values around the unit circle, whose matrix product with a vector computes the discrete Fourier transform of the vector.

`A = dftmtx(n)` returns the `n`-by-`n` complex matrix `A` that, when multiplied into a length `n` column vector `x`:

```
y = A*x
```

computes the discrete Fourier transform of `x`.

The inverse discrete Fourier transform matrix is

```
Ai = conj(dftmtx(n))/n
```

Example

In practice, the discrete Fourier transform is computed more efficiently and uses less memory with an FFT algorithm

```
x = 1:256;
y1 = fft(x);
```

than by using the Fourier transform matrix:

```
n = length(x);
y2 = x*dftmtx(n);
norm(y1-y2)

ans =
    1.8297e-009
```

Algorithm

`dftmtx` uses an outer product to generate the transform matrix.

See Also

`convmtx`	Convolution matrix.
`fft`	One-dimensional fast Fourier transform.

Meta Platforms, Inc. - Exhibit 1009
Page 421 of 720

Purpose

Dirichlet or periodic sinc function.

Syntax

```
y = diric(x,n)
```

Description

`y = diric(x,n)` returns a vector or array `y` the same size as `x`. The elements of `y` are the Dirichlet function of the elements of `x`. `n` must be a positive integer.

The Dirichlet function, or periodic sinc function, is

$$\mathrm{diric}(x, n) = \begin{cases} -1^{\frac{x}{2\pi}(n-1)} & x = 0, \pm 2\pi, \pm 4\pi, \ldots \\ \dfrac{\sin(nx/2)}{n\sin(x/2)} & \text{else} \end{cases}$$

for any nonzero integer `n`. This function has period 2π for `n` odd and period 4π for `n` even. Its peak value is 1, and its minimum value is -1 for `n` even. The magnitude of this function is (1/`n`) times the magnitude of the discrete-time Fourier transform of the `n`-point rectangular window.

Diagnostics

If `n` is not a positive integer, `diric` gives the following error message:

```
Requires n to be a positive integer.
```

See Also

`cos`	Cosine of vector/matrix elements (see the online *MATLAB Function Reference*).
`gauspuls`	Gaussian-modulated sinusoidal pulse generator.
`pulstran`	Pulse train generator.
`rectpuls`	Sampled aperiodic rectangle generator.
`sawtooth`	Sawtooth or triangle wave generator.
`sin`	Sine of vector/matrix elements (see the online *MATLAB Function Reference*).
`sinc`	Sinc or sin(πt)/πt function.
`square`	Square wave generator.
`tripuls`	Sampled aperiodic triangle generator.

Meta Platforms, Inc. - Exhibit 1009
Page 422 of 720

Purpose

Discrete prolate spheroidal sequences (Slepian sequences).

Syntax

```
[e,v] = dpss(n,nw)
[e,v] = dpss(n,nw,k)
[e,v] = dpss(n,nw,[k1 k2])
[e,v] = dpss(n,nw,'spline')
[e,v] = dpss(n,nw,'spline',Ni)
[e,v] = dpss(n,nw,'linear')
[e,v] = dpss(n,nw,'linear',Ni)
[e,v] = dpss(...,'trace')
[e,v] = dpss(...,'int','trace')
```

Description

`[e,v] = dpss(n,nw)` generates the first `2*nw` *discrete prolate spheroidal sequences* (DPSS) of length `n` in the columns of `e`, and their corresponding concentrations in vector `v`. They are also generated in the DPSS MAT-file database `dpss.mat`. `nw` must be less than `n/2`.

`[e,v] = dpss(n,nw,k)` returns the `k` most band-limited sequences of the `2*nw` discrete prolate spheroidal sequences calculated. `k` must be between `0` and `2*nw`.

`[e,v] = dpss(n,nw,[k1 k2])` returns the `k1`-th through the `k2`-th sequences from the `2*nw` discrete prolate spheroidal sequences calculated, where $1 \le k1 \le k2 \le (2*nw)$.

For all of the above forms,

- The Slepian sequences are calculated directly.
- The sequences are generated in the frequency band $|\omega| \le (2\pi W)$, where $W = nw/n$ is the half-bandwidth and ω is in radians.
- `e(:,1)` is the length `n` signal most concentrated in the frequency band $|\omega| \le (2\pi W)$ radians, `e(:,2)` is the signal orthogonal to `e(:,1)` that is most concentrated in this band, `e(:,3)` is the signal orthogonal to both `e(:,1)` and `e(:,2)` that is most concentrated in this band, etc.
- For multitaper spectral analysis, typical choices for `nw` are 2, 5/2, 3, 7/2, or 4.

`[e,v] = dpss(n,nw,'spline')` uses spline interpolation to compute `e` and `v` from the sequences in `dpss.mat` with length closest to `n`.

Meta Platforms, Inc. - Exhibit 1009
Page 423 of 720

`[e,v] = dpss(n,nw,'spline',Ni)` interpolates from existing length `Ni` sequences.

`[e,v] = dpss(n,nw,'linear')` and

`[e,v] = dpss(n,nw,'linear',Ni)` use linear interpolation, which is much faster but less accurate than spline interpolation. `'linear'` requires `Ni` > `n`.

`[e,v] = dpss(...,'trace')` and

`[e,v] = dpss(...,'int','trace')` use a trailing `'trace'` argument to find out which method DPSS uses, where `'int'` is either `'spline'` or `'linear'`.

See Also

`dpssclear`	Remove discrete prolate spheroidal sequences from database.
`dpssdir`	Discrete prolate spheroidal sequences database directory.
`dpssload`	Load discrete prolate spheroidal sequences from database.
`dpsssave`	Save discrete prolate spheroidal sequences in database.
`pmtm`	Power spectrum estimate using the multitaper method (MTM).

References

[1] Percival, D.B., and A.T. Walden. *Spectral Analysis for Physical Applications: Multitaper and Conventional Univariate Techniques*. Cambridge: Cambridge University Press, 1993.

Meta Platforms, Inc. - Exhibit 1009
Page 424 of 720

Purpose

Remove discrete prolate spheroidal sequences from database.

Syntax

```
dpssclear(n,nw)
```

Description

`dpssclear(n,nw)` removes sequences with length `n` and time-bandwidth product `nw` from the DPSS MAT-file database `dpss.mat`.

See Also

`dpss`	Discrete prolate spheroidal sequences (Slepian sequences).
`dpssdir`	Discrete prolate spheroidal sequences database directory.
`dpssload`	Load discrete prolate spheroidal sequences from database.
`dpsssave`	Save discrete prolate spheroidal sequences in database.

Meta Platforms, Inc. - Exhibit 1009
Page 425 of 720

dpssdir

Purpose

Discrete prolate spheroidal sequences database directory.

Syntax

```
dpssdir
dpssdir(n)
dpssdir(nw,'nw')
dpssdir(n,nw)
index = dpssdir
```

Description

`dpssdir` manages the database directory that contains the generated DPSS samples in the DPSS MAT-file database `dpss.mat`.

`dpssdir` lists the directory of saved sequences in `dpss.mat`.

`dpssdir(n)` lists the sequences saved with length `n`.

`dpssdir(nw,'nw')` lists the sequences saved with time-bandwidth product `nw`.

`dpssdir(n,nw)` lists the sequences saved with length `n` and time-bandwidth product `nw`.

`index = dpssdir` is a structure array describing the DPSS database. Pass `n` and `nw` options as for the no output case to get a filtered `index`.

See Also

`dpss`	Discrete prolate spheroidal sequences (Slepian sequences).
`dpssclear`	Remove discrete prolate spheroidal sequences from database.
`dpssload`	Load discrete prolate spheroidal sequences from database.
`dpsssave`	Save discrete prolate spheroidal sequences in database.

Meta Platforms, Inc. - Exhibit 1009
Page 426 of 720

Purpose

Load discrete prolate spheroidal sequences from database.

Syntax

```
[e,v] = dpssload(n,nw)
```

Description

`[e,v] = dpssload(n,nw)` loads all sequences with length `n` and time-bandwidth product `nw` in the columns of `e` and their corresponding concentrations in vector `v` from the DPSS MAT-file database `dpss.mat`.

See Also

`dpss`	Discrete prolate spheroidal sequences (Slepian sequences).
`dpssclear`	Remove discrete prolate spheroidal sequences from database.
`dpssdir`	Discrete prolate spheroidal sequences database directory.
`dpsssave`	Save discrete prolate spheroidal sequences in database.

Meta Platforms, Inc. - Exhibit 1009
Page 427 of 720

dpsssave

Purpose

Save discrete prolate spheroidal sequences in database.

Syntax

```
dpsssave(nw,e,v)
status = dpsssave(nw,e,v)
```

Description

`dpsssave(nw,e,v)` saves the sequences in the columns of `e` and their corresponding concentrations in vector `v` in the DPSS MAT-file database `dpss.mat`.

- It is not necessary to specify sequence length, because the length of the sequence is determined by the number of rows of `e`.
- `nw` is the *time-bandwidth product* that was specified when the sequence was created using `dpss`.

`status = dpsssave(nw,e,v)` returns 0 if the save was successful and 1 if there was an error.

See Also

`dpss`	Discrete prolate spheroidal sequences (Slepian sequences).
`dpssclear`	Remove discrete prolate spheroidal sequences from database.
`dpssdir`	Discrete prolate spheroidal sequences database directory.
`dpssload`	Load discrete prolate spheroidal sequences from database.

Meta Platforms, Inc. - Exhibit 1009
Page 428 of 720

Purpose

Elliptic (Cauer) filter design.

Syntax

```
[b,a] = ellip(n,Rp,Rs,Wn)
[b,a] = ellip(n,Rp,Rs,Wn,'ftype')
[b,a] = ellip(n,Rp,Rs,Wn,'s')
[b,a] = ellip(n,Rp,Rs,Wn,'ftype','s')
[z,p,k] = ellip(...)
[A,B,C,D] = ellip(...)
```

Description

`ellip` designs lowpass, bandpass, highpass, and bandstop digital and analog elliptic filters. Elliptic filters offer steeper rolloff characteristics than Butterworth or Chebyshev filters, but are equiripple in both the pass- and stopbands. In general, elliptic filters meet given performance specifications with the lowest order of any filter type.

Digital Domain

`[b,a] = ellip(n,Rp,Rs,Wn)` designs an order `n` lowpass digital elliptic filter with cutoff frequency `Wn`, `Rp` dB of ripple in the passband, and a stopband `Rs` dB down from the peak value in the passband. It returns the filter coefficients in the length `n + 1` row vectors `b` and `a`, with coefficients in descending powers of z:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + \cdots + b(n+1)z^{-n}}{1 + a(2)z^{-1} + \cdots + a(n+1)z^{-n}}$$

The *cutoff frequency* is the edge of the passband, at which the magnitude response of the filter is –`Rp` dB. For `ellip`, the cutoff frequency `Wn` is a number between 0 and 1, where 1 corresponds to half the sample frequency (Nyquist frequency). Smaller values of passband ripple `Rp` and larger values of stopband attenuation `Rs` both lead to wider transition widths (shallower rolloff characteristics).

If `Wn` is a two-element vector, `Wn = [w1 w2]`, `ellip` returns an order `2*n` bandpass filter with passband `w1 < ω < w2`.

`[b,a] = ellip(n,Rp,Rs,Wn,'ftype')` designs a highpass or bandstop filter, where *`ftype`* is:

Meta Platforms, Inc. - Exhibit 1009
Page 429 of 720

- `high` for a highpass digital filter with cutoff frequency `Wn`
- `stop` for an order `2*n` bandstop digital filter if `Wn` is a two-element vector, `Wn = [w1 w2]`

  The stopband is `w1` < ω < `w2`.

With different numbers of output arguments, `ellip` directly obtains other realizations of the filter. To obtain zero-pole-gain form, use three output arguments:

`[z,p,k] = ellip(n,Rp,Rs,Wn)` or

`[z,p,k] = ellip(n,Rp,Rs,Wn,'ftype')` returns the zeros and poles in length `n` column vectors `z` and `p` and the gain in the scalar `k`.

To obtain state-space form, use four output arguments:

`[A,B,C,D] = ellip(n,Rp,Rs,Wn)` or

`[A,B,C,D] = ellip(n,Rp,Rs,Wn,'ftype')` where `A`, `B`, `C`, and `D` are

$$\begin{aligned} x[n+1] &= Ax[n] + Bu[n] \\ y[n] &= Cx[n] + Du[n] \end{aligned}$$

and u is the input, x is the state vector, and y is the output.

Analog Domain

`[b,a] = ellip(n,Rp,Rs,Wn,'s')` designs an order `n` lowpass analog elliptic filter with cutoff frequency `Wn` and returns the filter coefficients in the length `n + 1` row vectors `b` and `a`, in descending powers of s:

$$H(s) = \frac{B(s)}{A(s)} = \frac{b(1)s^n + b(2)s^{n-1} + \cdots + b(n+1)}{s^n + a(2)s^{n-1} + \cdots + a(n+1)}$$

The *cutoff frequency* is the edge of the passband, at which the magnitude response of the filter is –`Rp` dB. For `ellip`, the cutoff frequency `Wn` must be greater than 0.

If `Wn` is a two-element vector with `w1 < w2`, then `ellip(n,Rp,Rs,Wn,'s')` returns an order `2*n` bandpass analog filter with passband `w1` < ω < `w2`.

Meta Platforms, Inc. - Exhibit 1009
Page 430 of 720

`[b,a] = ellip(n,Rp,Rs,Wn,'ftype','s')` designs a highpass or bandstop filter, where *ftype* is:

- `high` for a highpass analog filter with cutoff frequency `Wn`
- `stop` for an order `2*n` bandstop analog filter. `Wn` is a two-element vector, `[w1 w2]`, specifying the stopband `w1` < ω < `w2`.

With different numbers of output arguments, `ellip` directly obtains other realizations of the analog filter. To obtain zero-pole-gain form, use three output arguments:

`[z,p,k] = ellip(n,Rp,Rs,Wn,'s')` or

`[z,p,k] = ellip(n,Rp,Rs,Wn,'ftype','s')` returns the zeros and poles in length `n` or `2*n` column vectors `z` and `p` and the gain in the scalar `k`.

To obtain state-space form, use four output arguments:

`[A,B,C,D] = ellip(n,Rp,Rs,Wn,'s')` or

`[A,B,C,D] = ellip(n,Rp,Rs,Wn,'ftype','s')` where `A`, `B`, `C`, and `D` are

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

and u is the input, x is the state vector, and y is the output.

Examples

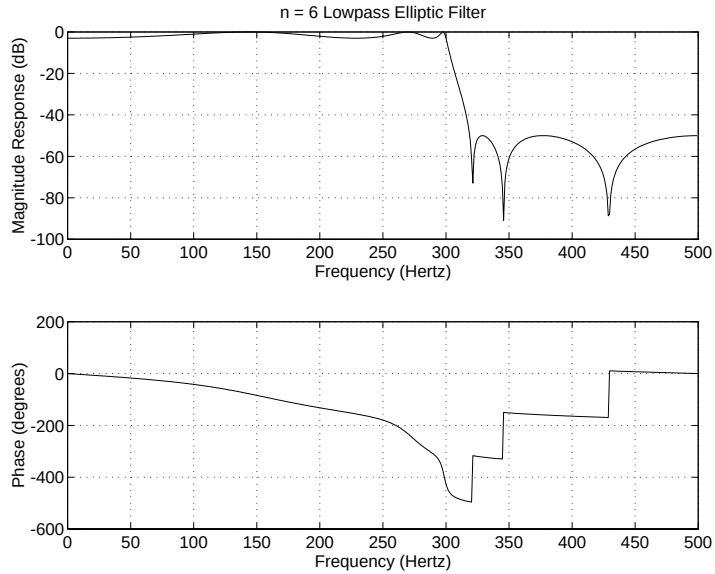
For data sampled at 1000 Hz, design a sixth-order lowpass elliptic filter with a cutoff frequency of 300 Hz, 3 dB of ripple in the passband, and 50 dB of attenuation in the stopband:

```
[b,a] = ellip(6,3,50,300/500);
```


Meta Platforms, Inc. - Exhibit 1009
Page 431 of 720

The filter's frequency response is

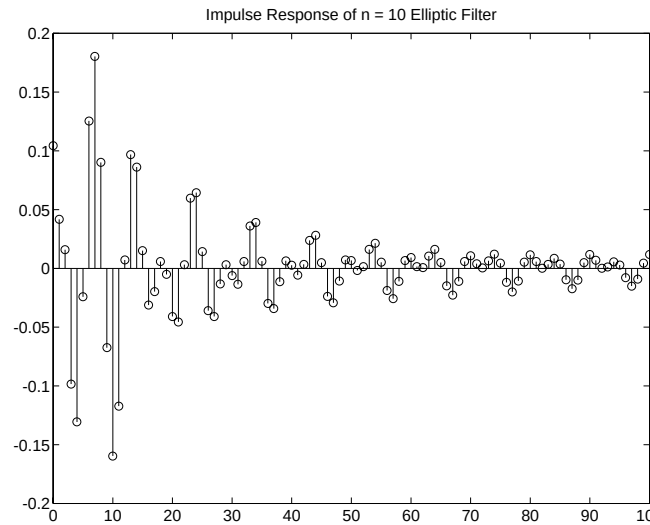
```
freqz(b,a,512,1000)
title('n=6 Lowpass Elliptic Filter')
```



Meta Platforms, Inc. - Exhibit 1009
Page 432 of 720

Design a 20th-order bandpass elliptic filter with a passband from 100 to 200 Hz and plot its impulse response:

```
n = 10; Rp = 0.5; Rs = 20;
Wn = [100 200]/500;
[b,a] = ellip(n,Rp,Rs,Wn);
[y,t] = impz(b,a,101); stem(t,y)
title('Impulse Response of n=10 Elliptic Filter')
```



Limitations

For high order filters, the state-space form is the most numerically accurate, followed by the zero-pole-gain form. The transfer function form is the least accurate; numerical problems can arise for filter orders as low as 15.

Algorithm

The design of elliptic filters is the most difficult and computationally intensive of the Butterworth, Chebyshev type I and II, and elliptic designs. `ellip` uses a five-step algorithm:

1. It finds the lowpass analog prototype poles, zeros, and gain using the `ellipap` function.
2. It converts the poles, zeros, and gain into state-space form.
3. It transforms the lowpass filter to a bandpass, highpass, or bandstop filter with the desired cutoff frequencies using a state-space transformation.

Meta Platforms, Inc. - Exhibit 1009
Page 433 of 720

4. For digital filter design, `ellip` uses `bilinear` to convert the analog filter into a digital filter through a bilinear transformation with frequency prewarping. Careful frequency adjustment guarantees that the analog filters and the digital filters will have the same frequency response magnitude at `Wn` or `w1` and `w2`.
5. It converts the state-space filter back to transfer function or zero-pole-gain form, as required.

See Also

`besself`	Bessel analog filter design.
`butter`	Butterworth analog and digital filter design.
`cheby1`	Chebyshev type I filter design (passband ripple).
`cheby2`	Chebyshev type II filter design (stopband ripple).
`ellipap`	Elliptic analog lowpass filter prototype.
`ellipord`	Elliptic filter order selection.

Meta Platforms, Inc. - Exhibit 1009
Page 434 of 720

Purpose

Elliptic analog lowpass filter prototype.

Syntax

```
[z,p,k] = ellipap(n,Rp,Rs)
```

Description

`[z,p,k] = ellipap(n,Rp,Rs)` returns the zeros, poles, and gain of an order `n` elliptic analog lowpass filter prototype, with `Rp` dB of ripple in the passband, and a stopband `Rs` dB down from the peak value in the passband. The zeros and poles are returned in length `n` column vectors `z` and `p` and the gain in scalar `k`. If `n` is odd, `z` is length `n` – 1. The transfer function is

$$H(s) = \frac{z(s)}{p(s)} = k\frac{(s-z(1))(s-z(2))\cdots(s-z(n))}{(s-p(1))(s-p(2))\cdots(s-p(n))}$$

Elliptic filters are equiripple in both the passband and stopband. They offer steeper rolloff characteristics than Butterworth and Chebyshev filters, but they are equiripple in both the passband and the stopband. Of the four classical filter types, elliptic filters usually meet a given set of filter performance specifications with the lowest filter order.

`ellip` sets the cutoff frequency ω_0 of the elliptic filter to 1 for a normalized result. The *cutoff frequency* is the frequency at which the passband ends and the filter has a magnitude response of $10^{-Rp/20}$.

Algorithm

`ellipap` uses the algorithm outlined in [1]. It employs the M-file `ellipk` to calculate the complete elliptic integral of the first kind and the M-file `ellipj` to calculate Jacobi elliptic functions.

See Also

`besselap`	Bessel analog lowpass filter prototype.
`buttap`	Butterworth analog lowpass filter prototype.
`cheb1ap`	Chebyshev type I analog lowpass filter prototype.
`cheb2ap`	Chebyshev type II analog lowpass filter prototype.
`ellip`	Elliptic (Cauer) filter design.

References

[1] Parks, T.W., and C.S. Burrus. *Digital Filter Design*. New York: John Wiley & Sons, 1987. Chapter 7.

Meta Platforms, Inc. - Exhibit 1009
Page 435 of 720

Purpose Elliptic filter order selection.

Syntax

```
[n,Wn] = ellipord(Wp,Ws,Rp,Rs)
[n,Wn] = ellipord(Wp,Ws,Rp,Rs,'s')
```

Description `ellipord` selects the minimum order digital or analog elliptic filter required to meet a set of lowpass filter design specifications:

`Wp`	Passband corner frequency. `Wp`, the cutoff frequency, has a value between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency).
`Ws`	Stopband corner frequency. `Ws` is in the same units as `Wp`; it has a value between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency).
`Rp`	Passband ripple, in decibels. This value is the maximum permissible passband loss in decibels. The passband is `0 < w < Wp`.
`Rs`	Stopband attenuation, in decibels. This value is the number of decibels the stopband is down from the passband. The stopband is `Ws < w < 1`.

Digital Domain

`[n,Wn] = ellipord(Wp,Ws,Rp,Rs)` returns the order `n` of the lowest order elliptic filter that loses no more than `Rp` dB in the passband and has at least `Rs` dB of attenuation in the stopband. The passband runs from 0 to `Wp` and the stopband extends from `Ws` to 1, the Nyquist frequency. `ellipord` also returns `Wn`, the cutoff frequency that allows `ellip` to achieve the given specifications.

Use `ellipord` for lowpass, highpass, bandpass, and bandstop filters. For highpass filters, `Wp` is greater than `Ws`. For bandpass and bandstop filters, `Wp` and `Ws` are two-element vectors that specify the corner frequencies at both edges of the filter, lower frequency edge first. For the band filters, `ellipord` returns `Wn` as a two-element row vector for input to `ellip`.

If filter specifications call for a bandpass or bandstop filter with unequal ripple in each of the passbands or stopbands, design the filter as separate lowpass and highpass sections and cascade the two filters together.

Meta Platforms, Inc. - Exhibit 1009
Page 436 of 720

Analog Domain

`[n,Wn] = ellipord(Wp,Ws,Rp,Rs,'s')` finds the minimum order `n` and cutoff frequencies `Wn` for an analog filter. In this case the frequencies in `Wp` and `Ws` are in radians per second and may be greater than 1.

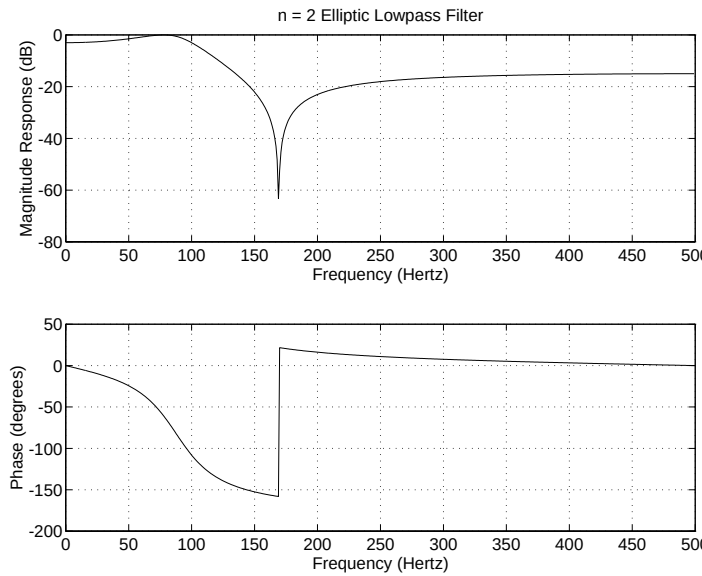
Use `ellipord` for lowpass, highpass, bandpass, and bandstop filters as described under "Digital Domain."

Meta Platforms, Inc. - Exhibit 1009
Page 437 of 720

Examples

For 1000 Hz data, design a lowpass filter with less than 3 dB of attenuation from 0 to 100 Hz and at least 15 dB of attenuation from 150 Hz to the Nyquist frequency:

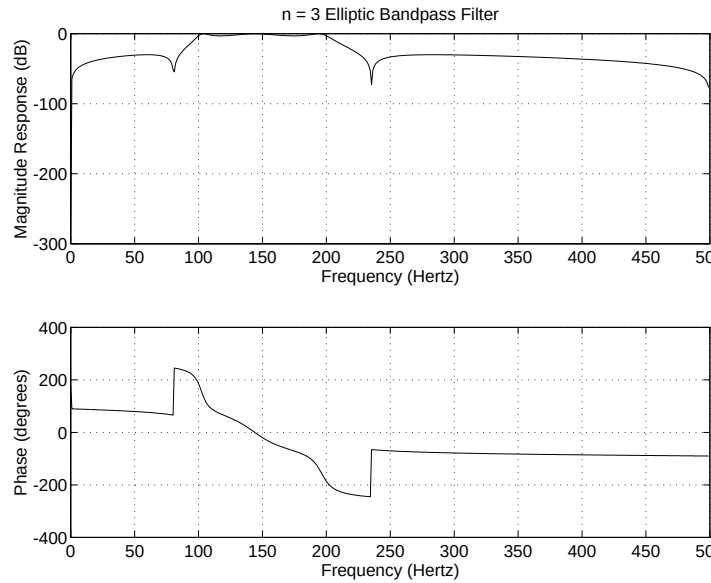
```
Wp = 100/500; Ws = 150/500;
Rp = 3; Rs = 15;
[n,Wn] = ellipord(Wp,Ws,Rp,Rs)

n =
     2

Wn =
    0.2000

[b,a] = ellip(n,Rp,Rs,Wn);
freqz(b,a,512,1000);
title('n=2 Elliptic Lowpass Filter')
```



Meta Platforms, Inc. - Exhibit 1009
Page 438 of 720

Now design a bandpass filter with a passband from 90 Hz to 200 Hz, less than 3 dB of ripple throughout the passband, and 30 dB stopbands 50 Hz out on both sides of the passband:

```
Wp = [90 200]/500; Ws = [50 250]/500;
Rp = 3; Rs = 30;
[n,Wn] = ellipord(Wp,Ws,Rp,Rs)

n =
     3

Wn =
    0.1800    0.4000

[b,a] = ellip(n,Rp,Rs,Wn);
freqz(b,a,512,1000);
title('n=3 Elliptic Bandpass Filter')
```



Algorithm

`ellipord` uses the elliptic lowpass filter order prediction formula described in [1]. The function performs its calculations in the analog domain for both the analog and digital cases. For the digital case, it converts the frequency

Meta Platforms, Inc. - Exhibit 1009
Page 439 of 720

parameters to the *s*-domain before the order and natural frequency estimation process, then converts them back to the *z*-domain.

`ellipord` initially develops a lowpass filter prototype by transforming the passband frequencies of the desired filter to 1 rad/sec (for low- and highpass filters) and to -1 and 1 rad/sec (for bandpass and bandstop filters). It then computes the minimum order required for a lowpass filter to meet the stopband specification.

See Also

`buttord`	Butterworth filter order selection.
`cheb1ord`	Chebyshev type I filter order selection.
`cheb2ord`	Chebyshev type II filter order selection.
`ellip`	Elliptic (Cauer) filter design.

References

[1] Rabiner, L.R., and B. Gold. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975. Pg. 241.

Meta Platforms, Inc. - Exhibit 1009
Page 440 of 720

Purpose

One-dimensional fast Fourier transform.

Syntax

```
y = fft(x)
y = fft(x,n)
```

Description

`fft` computes the discrete Fourier transform of a vector or matrix. This function implements the transform given by

$$X(k+1) = \sum_{n=0}^{N-1} x(n+1) W_n^{kn}$$

where $W_N = e^{-j(2\pi/N)}$ and N = `length(x)`. Note that the series is indexed as $n + 1$ and $k + 1$ instead of the usual n and k because MATLAB vectors run from 1 to N instead of from 0 to N-1.

`y = fft(x)` is the discrete Fourier transform of vector `x`, computed with a fast Fourier transform (FFT) algorithm. If `x` is a matrix, `y` is the FFT of each column of the matrix. The `fft` function employs a radix-2 fast Fourier transform algorithm if the length of the sequence is a power of two, and a slower algorithm if it is not; see the "Algorithm" section for details.

`y = fft(x,n)` is the `n`-point FFT. If the length of `x` is less than `n`, `fft` pads `x` with trailing zeros to length `n`. If the length of `x` is greater than `n`, `fft` truncates the sequence `x`. If `x` is an array, `fft` adjusts the length of the columns in the same manner.

The `fft` function is part of the standard MATLAB language.

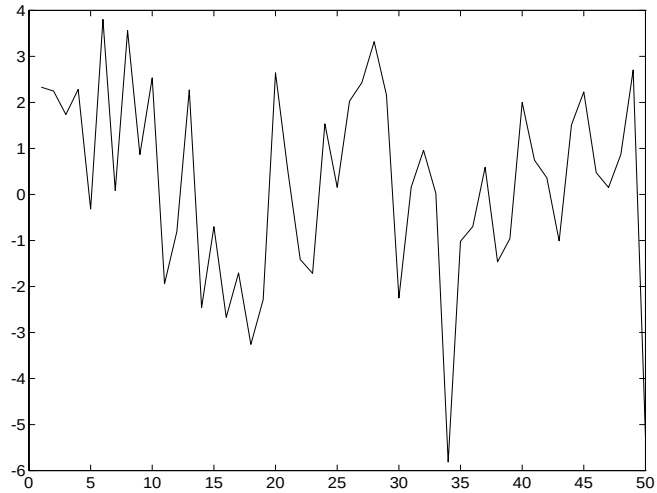
Example

A common use of the Fourier transform is to find the frequency components of a time-domain signal buried in noise. Consider data sampled at 1000 Hz. Form

Meta Platforms, Inc. - Exhibit 1009
Page 441 of 720

a signal consisting of 50 Hz and 120 Hz sinusoids and corrupt the signal with zero-mean random noise:

```
t = 0:0.001:0.6;
x = sin(2*pi*50*t) + sin(2*pi*120*t);
y = x + 2*randn(1,length(t));
plot(y(1:50))
```



It is difficult to identify the frequency components by studying the original signal. Convert to the frequency domain by taking the discrete Fourier transform of the noisy signal y using a 512-point fast Fourier transform (FFT):

```
Y = fft(y,512);
```

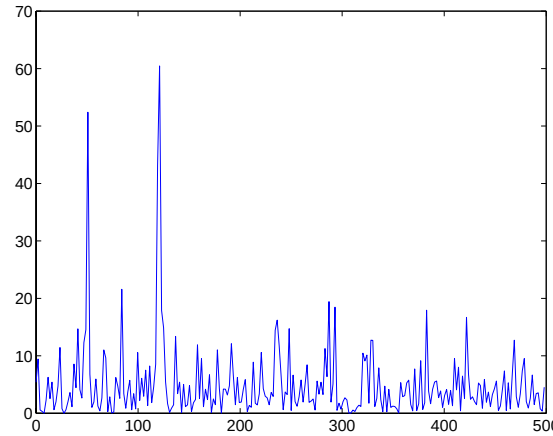
The power spectral density, a measurement of the energy at various frequencies, is

```
Pyy = Y.*conj(Y) / 512;
```


Meta Platforms, Inc. - Exhibit 1009
Page 442 of 720

Graph the first 256 points (the other 256 points are symmetric) on a meaningful frequency axis:

```
f = 1000*(0:255)/512;
plot(f,Pyy(1:256))
```



See the `pwelch` function for details on calculating spectral density.

Sometimes it is useful to normalize the output of `fft` so that a unit sinusoid in the time domain corresponds to unit amplitude in the frequency domain. To produce a normalized discrete-time Fourier transform in this manner, use

```
Pn = abs(fft(x))*2/length(x)
```

Algorithm

`fft` is a built-in MATLAB function. When the sequence length is a power of two, `fft` uses a high-speed radix-2 fast Fourier transform algorithm. The radix-2 FFT routine is optimized to perform a real FFT if the input sequence is purely real; otherwise it computes the complex FFT. This causes a real power-of-two FFT to be about 40% faster than a complex FFT of the same length.

When the sequence length is not an exact power of two, a separate algorithm finds the prime factors of the sequence length and computes the mixed-radix discrete Fourier transforms of the shorter sequences.

Meta Platforms, Inc. - Exhibit 1009
Page 443 of 720

The execution time for `fft` depends on the sequence length. If the length of a sequence has many prime factors, the function computes the FFT quickly; if the length has few prime factors, execution is slower. For sequences whose lengths are prime numbers, `fft` uses the raw (and slow) DFT algorithm. For this reason it is usually better to use power-of-two FFTs, if this is supported by your application. For example, on one machine a 4096-point real FFT takes 2.1 seconds and a complex FFT of the same length takes 3.7 seconds. The FFTs of neighboring sequences of length 4095 and 4097, however, take 7 seconds and 58 seconds, respectively.

Suppose a sequence x of N points is obtained at a sample frequency of f_s. Then, for up to the Nyquist frequency, or point $n = N/2 + 1$, the relationship between the actual frequency and the index k into x (out of N possible indices) is

$$f = (k-1) * f_s \big| N$$

See Also

`dct`	Discrete cosine transform (DCT).
`dftmtx`	Discrete Fourier transform matrix.
`fft2`	Two-dimensional fast Fourier transform.
`fftshift`	Rearrange the outputs of `fft` and `fft2`.
`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
`freqz`	Frequency response of digital filters.
`ifft`	One-dimensional inverse fast Fourier transform.
`pwelch`	Estimate the power spectral density (PSD) of a signal using Welch's method.

Meta Platforms, Inc. - Exhibit 1009
Page 444 of 720

Purpose

Two-dimensional fast Fourier transform.

Syntax

```
Y = fft2(X)
Y = fft2(X,m,n)
```

Description

`Y = fft2(X)` performs a two-dimensional FFT, producing a result `Y` the same size as `X`. If `X` is a vector, `Y` has the same orientation as `X`.

`Y = fft2(X,m,n)` truncates or zero pads `X`, if necessary, to create an `m`-by-`n` array before performing the FFT. The result `Y` is also `m`-by-`n`.

The `fft2` function is part of the standard MATLAB language.

Algorithm

`fft2(x)` is simply

```
fft(fft(x).').'
```

This computes the one-dimensional `fft` of each column of `x`, then of each row of the result. The time required to compute `fft2(x)` depends on the number of prime factors in `[m,n] = size(x)`. `fft2` is fastest when `m` and `n` are powers of 2.

See Also

`fft`	One-dimensional fast Fourier transform.
`fftshift`	Rearrange the outputs of `fft` and `fft2`.
`ifft`	One-dimensional inverse fast Fourier transform.
`ifft2`	Two-dimensional inverse fast Fourier transform.

Meta Platforms, Inc. - Exhibit 1009
Page 445 of 720

Purpose

FFT-based FIR filtering using the overlap-add method.

Syntax

```
y = fftfilt(b,x)
y = fftfilt(b,x,n)
```

Description

`fftfilt` filters data using the efficient FFT-based method of *overlap-add*, a frequency domain filtering technique that works only for FIR filters.

`y = fftfilt(b,x)` filters the data in vector `x` with the filter described by coefficient vector `b`. It returns the data vector `y`. The operation performed by `fftfilt` is described in the *time domain* by the difference equation

$$y(n) = b(1)x(n) + b(2)x(n-1) + \cdots + b(nb+1)x(n-nb)$$

An equivalent representation is the z-transform or *frequency domain* description

$$Y(z) = (b(1) + b(2)z^{-1} + \cdots + b(nb+1)z^{-nb})X(z)$$

By default, `fftfilt` chooses an FFT length and data block length that guarantee efficient execution time.

`y = fftfilt(b,x,n)` uses an FFT length of `nfft = 2^nextpow2(n)` and a data block length of `nfft - length(b) + 1`.

`fftfilt` works for both real and complex inputs.

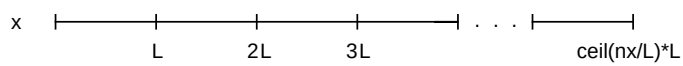
Example

Show that the results from `fftfilt` and `filter` are identical:

```
b = [1 2 3 4];
x = [1 zeros(1,99)]';
norm(fftfilt(b,x) - filter(b,1,x))

ans =
    9.5914e-15
```

Algorithm

`fftfilt` uses `fft` to implement the *overlap-add method* [1], a technique that combines successive frequency domain filtered blocks of an input sequence. `fftfilt` breaks an input sequence `x` into length `L` data blocks:

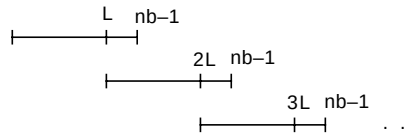


Meta Platforms, Inc. - Exhibit 1009
Page 446 of 720

and convolves each block with the filter b by

```
y = ifft(fft(x(i:i+L-1),nfft).*fft(b,nfft));
```

where nfft is the FFT length. fftfilt overlaps successive output sections by nb-1 points, where nb is the length of the filter, and sums them:



fftfilt chooses the key parameters L and nfft in different ways, depending on whether you supply an FFT length n and on the lengths of the filter and signal. If you do not specify a value for n (which determines FFT length), fftfilt chooses these key parameters automatically:

- If length(x) > length(b), fftfilt chooses values that minimize the number of blocks times the number of flops per FFT.
- If length(b) >= length(x), fftfilt uses a single FFT of length

  ```
  2^nextpow2(length(b) + length(x) - 1)
  ```

  This essentially computes

  ```
  y = ifft(fft(B,nfft).*fft(X,nfft))
  ```

If you supply a value for n, fftfilt chooses an FFT length, nfft, of 2^nextpow2(n) and a data block length of nfft - length(b) + 1. If n is less than length(b), fftfilt sets n to length(b).

See Also

conv	Convolution and polynomial multiplication.
filter	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
filtfilt	Zero-phase digital filtering.

References

[1] Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

Meta Platforms, Inc. - Exhibit 1009
Page 447 of 720

Purpose Rearrange the outputs of the FFT functions.

Syntax

```
y = fftshift(x)
```

Description `y = fftshift(x)` rearranges the outputs of `fft` and `fft2` by moving the zero frequency component to the center of the spectrum, which is sometimes a more convenient form.

For vectors, `fftshift(x)` returns a vector with the left and right halves swapped.

For arrays, `fftshift(x)` swaps quadrants one and three with quadrants two and four.

The `fftshift` function is part of the standard MATLAB language.

Example For any array `X`,

```
Y = fft2(x)
```

has `Y(1,1) = sum(sum(X))`; the DC component of the signal is in the upper-left corner of the two-dimensional FFT. For

```
Z = fftshift(Y)
```

the DC component is near the center of the matrix.

See Also

`fft`	One-dimensional fast Fourier transform.
`fft2`	Two-dimensional fast Fourier transform.

Meta Platforms, Inc. - Exhibit 1009
Page 448 of 720

Purpose

Filter data with a recursive (IIR) or nonrecursive (FIR) filter.

Syntax

```
y = filter(b,a,x)
[y,zf] = filter(b,a,x)
[...] = filter(b,a,x,zi)
[...] = filter(b,a,x,zi,dim)
```

Description

`filter` filters real or complex data using a digital filter. The filter realization is the *transposed direct form II* structure [1], which can handle both FIR and IIR filters.

If $a(1) \neq 1$, `filter` normalizes the filter coefficients by $a(1)$. If $a(1) = 0$, the input is in error.

`y = filter(b,a,x)` filters the data in vector `x` with the filter described by coefficient vectors `a` and `b` to create the filtered data vector `y`. When `x` is a matrix, filter operates on the columns of `x`. When `x` is an *N*-dimensional array, filter operates on the first non-singleton dimension.

`[y,zf] = filter(b,a,x)` returns the final values of the states in the vector `zf`.

`[...] = filter(b,a,x,zi)` specifies initial state conditions in the vector `zi`.

The size of the initial/final condition vector is `max(length(b),length(a))-1`. `zi` or `zf` can also be an array of such vectors, one for each column of `x` if `x` is a matrix. If `x` is a multidimensional array, `filter` works across the first nonsingleton dimension of `x` by default.

`[...] = filter(b,a,x,zi,dim)` works across the dimension `dim` of `x`. Set `zi` to empty to get the default initial conditions.

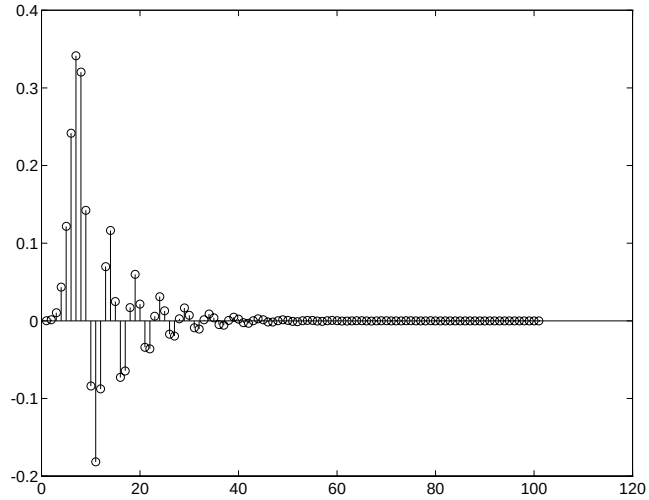
The `filter` function is part of the standard MATLAB language.

Meta Platforms, Inc. - Exhibit 1009
Page 449 of 720

Example

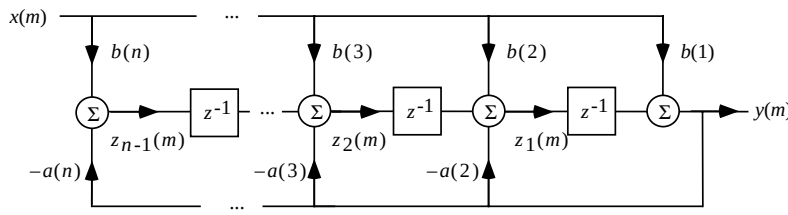
Find and graph the 100-point unit impulse response of a digital filter:

```
x = [1 zeros(1,100)];
[b,a] = butter(12,400/1000);
y = filter(b,a,x);
stem(y)
```



Algorithm

`filter` is implemented as a transposed direct form II structure



where n-1 is the filter order.

Meta Platforms, Inc. - Exhibit 1009
Page 450 of 720

The operation of `filter` at sample m is given by the time domain difference equations for y and the states z_i:

$$
\begin{aligned}
y(m) &= b(1)x(m) + z_1(m-1) - a(1)y(m) \\
z_1(m) &= b(2)x(m) + z_2(m-1) - a(2)y(m) \\
\vdots &= \vdots \\
z_{n-2}(m) &= b(n-1)x(m) + z_{n-1}(m-1) - a(n-1)y(m) \\
z_{n-1}(m) &= b(n)x(m) - a(n)y(m)
\end{aligned}
$$

You can use `filtic` to generate the state vector $z_i(0)$ from past inputs and outputs.

The input-output description of this filtering operation in the z-transform domain is a rational transfer function:

$$
Y(z) = \frac{b(1) + b(2)z^{-1} + \cdots + b(nb+1)z^{-nb}}{a(1) + a(2)z^{-1} + \cdots + a(na+1)z^{-na}} X(z)
$$

Diagnostics

If $a(1) = 0$, `filter` gives the following error message:

```
First denominator coefficient must be nonzero.
```

If the length of the initial condition vector is not the greater of na and nb, `filter` gives the following error message:

```
Initial condition vector has incorrect dimensions.
```

See Also

`fftfilt`	FFT-based FIR filtering using the overlap-add method.
`filter2`	Two-dimensional digital filtering.
`filtfilt`	Zero-phase digital filtering.
`filtic`	Make initial conditions for `filter` function.

References

[1] Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989. Pgs. 311-312.

Meta Platforms, Inc. - Exhibit 1009
Page 451 of 720

Purpose

Two-dimensional digital filtering.

Syntax

```
Y = filter2(B,X)
Y = filter2(B,X,'shape')
```

Description

`Y = filter2(B,X)` filters the two-dimensional data in `X` with the two-dimensional FIR filter in the matrix `B`. The result, `Y`, is computed using two-dimensional convolution and is the same size as `X`.

`Y = filter2(B,X,'shape')` returns `Y` computed with size specified by *shape*:

- `same` returns the central part of the convolution that is the same size as `X` (default).
- `full` returns the full two-dimensional convolution, `size(Y)` > `size(X)`.
- `valid` returns only those parts of the convolution that are computed without the zero-padded edges, `size(Y)` < `size(X)`.

The `filter2` function is part of the standard MATLAB language.

Algorithm

The `filter2` function uses `conv2` to compute the full two-dimensional convolution of the FIR filter with the input matrix. By default, `filter2` extracts and returns the central part of the convolution that is the same size as the input matrix. Use the *shape* parameter to specify an alternate part of the convolution for return.

See Also

`conv2`	Two-dimensional convolution.
`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.

Meta Platforms, Inc. - Exhibit 1009
Page 452 of 720

Purpose

Zero-phase digital filtering.

Syntax

```
y = filtfilt(b,a,x)
```

Description

`y = filtfilt(b,a,x)` performs zero-phase digital filtering by processing the input data in both the forward and reverse directions (see problem 5.39 in [1]). After filtering in the forward direction, it reverses the filtered sequence and runs it back through the filter. The resulting sequence has precisely zero-phase distortion and double the filter order. `filtfilt` minimizes start-up and ending transients by matching initial conditions, and works for both real and complex inputs.

Note that `filtfilt` should not be used with differentiator and Hilbert FIR filters, since the operation of these filters depends heavily on their phase response.

Algorithm

`filtfilt` is an M-file that uses the `filter` function. In addition to the forward-reverse filtering, it attempts to minimize startup transients by adjusting initial conditions to match the DC component of the signal and by prepending several filter lengths of a flipped, reflected copy of the input signal.

See Also

`fftfilt`	FFT-based FIR filtering using the overlap-add method.
`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
`filter2`	Two-dimensional digital filtering.

References

[1] Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989. Pgs. 311-312.

Meta Platforms, Inc. - Exhibit 1009
Page 453 of 720

Purpose Find initial conditions for a transposed direct form II filter implementation.

Syntax

```
z = filtic(b,a,y,x)
z = filtic(b,a,y)
```

Description `z = filtic(b,a,y,x)` finds the initial conditions `z` for the delays in the *transposed direct form II* filter implementation given past outputs `y` and inputs `x`. The vectors `b` and `a` represent the numerator and denominator coefficients, respectively, of the filter's transfer function.

The vectors `x` and `y` contain the most recent input or output first, and oldest input or output last:

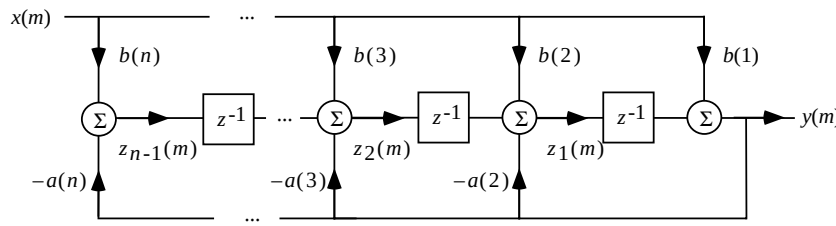
$$x = \{x(-1), x(-2), x(-3), \ldots, x(-nb), \ldots\}$$
$$y = \{y(-1), y(-2), y(-3), \ldots, y(-na), \ldots\}$$

where `nb` is `length(b)-1` (the numerator order) and `na` is `length(a)-1` (the denominator order). If `length(x)` is less than `nb`, `filtic` pads it with zeros to length `nb`; if `length(y)` is less than `na`, `filtic` pads it with zeros to length `na`. Elements of `x` beyond `x(nb-1)` and elements of `y` beyond `y(na-1)` are unnecessary so `filtic` ignores them.

Output `z` is a column vector of length equal to the larger of *nb* and *na*. `z` describes the state of the delays given past inputs `x` and past outputs `y`.

`z = filtic(b,a,y)` assumes that the input `x` is 0 in the past.

The transposed direct form II structure is



where *n*-1 is the filter order.

`filtic` works for both real and complex inputs.

Meta Platforms, Inc. - Exhibit 1009
Page 454 of 720

Algorithm

`filtic` performs a reverse difference equation to obtain the delay states `z`.

Diagnostics

If any of the input arguments `y`, `x`, `b`, or `a` is not a vector (that is, if any argument is a scalar or array), `filtic` gives the following error message:

```
Requires vector inputs.
```

See Also

`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
`filtfilt`	Zero-phase digital filtering.

References

[1] Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989. Pgs. 296, 301-302.

Meta Platforms, Inc. - Exhibit 1009
Page 455 of 720

Purpose

Window-based finite impulse response filter design – standard response.

Syntax

```
b = fir1(n,Wn)
b = fir1(n,Wn,'ftype')
b = fir1(n,Wn,window)
b = fir1(n,Wn,'ftype',window)
b = fir1(...,'noscale')
```

Description

`fir1` implements the classical method of windowed linear-phase FIR digital filter design [1]. It designs filters in standard lowpass, bandpass, highpass, and bandpass configurations. (For windowed filters with arbitrary frequency response, use `fir2`.)

`b = fir1(n,Wn)` returns row vector `b` containing the `n+1` coefficients of an order `n` lowpass FIR filter. This is a Hamming-windowed, linear-phase filter with cutoff frequency `Wn`. The output filter coefficients, `b`, are ordered in descending powers of z:

$$B(z) = b(1) + b(2)z^{-1} + \cdots + b(n+1)z^{-n}$$

`Wn`, the cutoff frequency, is a number between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency).

If `Wn` is a two-element vector, `Wn = [w1 w2]`, `fir1` returns a bandpass filter with passband `w1` < ω < `w2`.

If `Wn` is a multi-element vector, `Wn = [w1 w2 w3 w4 w5 ... wn]`, `fir1` returns an order `n` multiband filter with bands `0` < ω < `w1`, `w1` < ω < `w2`, ..., `wn` < ω < `1`.

By default, the filter is scaled so that the center of the first passband has magnitude exactly 1 after windowing.

`b = fir1(n,Wn,'ftype')` specifies a filter type, where *`ftype`* is

- `high` for a highpass filter with cutoff frequency `Wn`
- `stop` for a bandstop filter, if `Wn = [w1 w2]`

  The stopband is `w1` < ω < `w2`.
- `'DC-1'` to make the first band of a multiband filter a passband
- `'DC-0'` to make the first band of a multiband filter a stopband

Meta Platforms, Inc. - Exhibit 1009
Page 456 of 720

`fir1` always uses an even filter order for the highpass and bandstop configurations. This is because for odd orders, the frequency response at the Nyquist frequency is 0, which is inappropriate for highpass and bandstop filters. If you specify an odd-valued `n`, `fir1` increments it by 1.

`b = fir1(n,Wn,window)` uses the window specified in column vector `window` for the design. The vector `window` must be `n+1` elements long. If no window is specified, `fir1` employs a Hamming window.

`b = fir1(n,Wn,'ftype',window)` accepts both *`ftype`* and *`window`* parameters.

`b = fir1(...,'noscale')` turns off the default scaling.

The group delay of the FIR filter designed by `fir1` is `n/2`.

Algorithm

`fir1` uses the window method of FIR filter design [1]. If $w(n)$ denotes a window, where $1 \le n \le N$, and the impulse response of the ideal filter is $h(n)$, where $h(n)$ is the inverse Fourier transform of the ideal frequency response, then the windowed digital filter coefficients are given by

$$b(n) = w(n)h(n), \qquad 1 \le n \le N$$

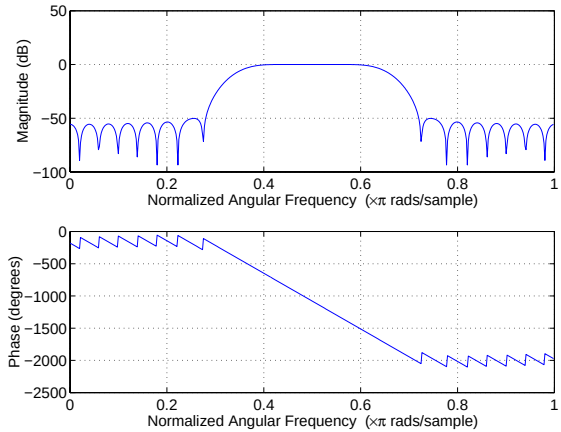
Meta Platforms, Inc. - Exhibit 1009
Page 457 of 720

Examples

Example 1

Design a 48th-order FIR bandpass filter with passband $0.35 \le w \le 0.65$:

```
b = fir1(48,[0.35 0.65]);
freqz(b,1,512)
```



Example 2

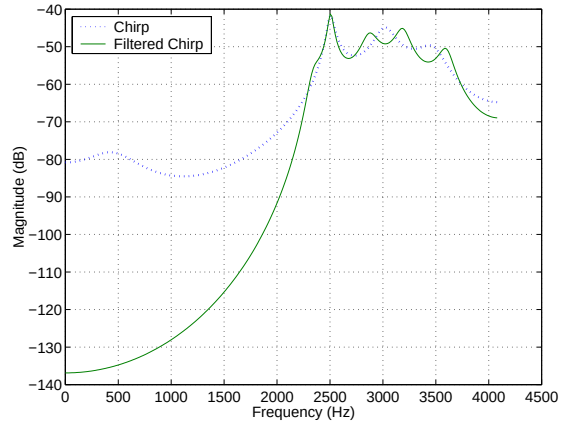
The `chirp.mat` file contains a signal, `y`, that has most of its power above `Fs/4`, or half the Nyquist frequency. Design a 34th-order FIR highpass filter to attenuate the components of the signal below `Fs/4`. Use a cutoff frequency of 0.48 and a Chebyshev window with 30 dB of ripple:

```
load chirp     % loads y and Fs
b = fir1(34,0.48,'high',chebwin(35,30));
yfilt = filter(b,1,y);
```


Meta Platforms, Inc. - Exhibit 1009
Page 458 of 720

Check the spectra of the original and filtered signals.

```
[Py,fy] = pburg(y,10,512,Fs);
[Pyfilt,fyfilt] = pburg(yfilt,10,512,Fs);
plot(fy,10*log10(Py),':',fyfilt,10*log10(Pyfilt)); grid
ylabel('Magnitude (dB)'); xlabel('Frequency (Hz)');
legend('Chirp','Filtered Chirp')
```



Play the two signals using `sound`.

```
sound(y,Fs);
sound(yfilt,Fs);
```

Diagnostics

If `n` is odd and you specify a bandstop or highpass filter, `fir1` gives the following warning message:

```
For highpass and bandstop filters, N must be even.
Order is being increased by 1.
```


Meta Platforms, Inc. - Exhibit 1009
Page 459 of 720

See Also

`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
`fir2`	Window-based finite impulse response filter design—arbitrary response.
`fircls`	Constrained least square FIR filter design for multiband filters.
`fircls1`	Constrained least square filter design for lowpass and highpass linear phase FIR filters.
`firls`	Least square linear-phase FIR filter design.
`freqz`	Frequency response of digital filters.
`kaiserord`	Estimate parameters for `fir1` with Kaiser window.
`remez`	Parks-McClellan optimal FIR filter design.

References

[1] IEEE. *Programs for Digital Signal Processing*. IEEE Press. New York: John Wiley & Sons, 1979. Algorithm 5.2.

Meta Platforms, Inc. - Exhibit 1009
Page 460 of 720

Purpose

Window-based finite impulse response filter design – arbitrary response.

Syntax

```
b = fir2(n,f,m)
b = fir2(n,f,m,window)
b = fir2(n,f,m,npt)
b = fir2(n,f,m,npt,window)
b = fir2(n,f,m,npt,lap)
b = fir2(n,f,m,npt,lap,window)
```

Description

`fir2` designs windowed digital FIR filters with arbitrarily shaped frequency response. (For standard lowpass, bandpass, highpass, and bandstop configurations, use `fir1`.)

`b = fir2(n,f,m)` returns row vector `b` containing the `n+1` coefficients of an order `n` FIR filter. The frequency-magnitude characteristics of this filter match those given by vectors `f` and `m`:

- `f` is a vector of frequency points in the range from 0 to 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency). The first point of `f` must be 0 and the last point 1. The frequency points must be in increasing order.
- `m` is a vector containing the desired magnitude response at the points specified in `f`.
- `f` and `m` must be the same length.
- Duplicate frequency points are allowed, corresponding to steps in the frequency response.

Use `plot(f,m)` to view the filter shape.

The output filter coefficients, `b`, are ordered in descending powers of z:

$$b(z) = b(1) + b(2)z^{-1} + \cdots + b(n+1)z^{-n}$$

`b = fir2(n,f,m,window)` uses the window specified in column vector `window` for the filter design. The vector `window` must be `n+1` elements long. If no window is specified, `fir2` employs a Hamming window.

Meta Platforms, Inc. - Exhibit 1009
Page 461 of 720

`b = fir2(n,f,m,npt)` and

`b = fir2(n,f,m,npt, window)` specify the number of points `npt` for the grid onto which `fir2` interpolates the frequency response, with or without a `window` specification.

`b = fir2(n,f,m,npt,lap)` and

`b = fir2(n,f,m,npt,lap,window)` specify the size of the region, `lap`, that `fir2` inserts around duplicate frequency points, with or without a `window` specification.

See the "Algorithm" section for more on `npt` and `lap`.

Algorithm

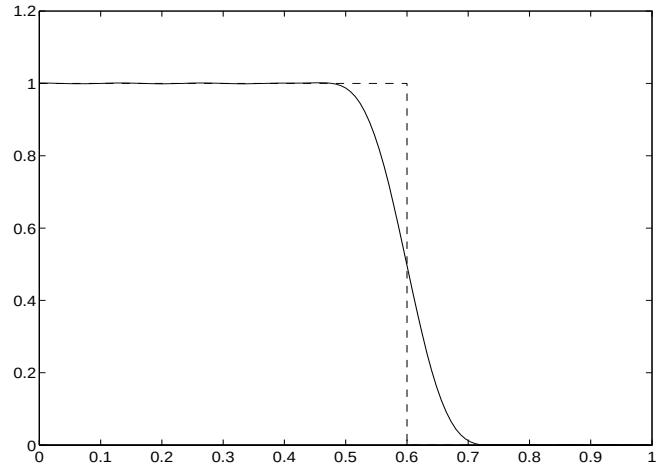
The desired frequency response is interpolated onto a dense, evenly spaced grid of length `npt`. `npt` is 512 by default. If two successive values of `f` are the same, a region of `lap` points is set up around this frequency to provide a smooth but steep transition in the requested frequency response. By default, `lap` is 25. The filter coefficients are obtained by applying an inverse fast Fourier transform to the grid and multiplying by a window; by default, this is a Hamming window.

Meta Platforms, Inc. - Exhibit 1009
Page 462 of 720

Example

Design a 30th-order lowpass filter and overplot the desired frequency response with the actual frequency response:

```
f = [0 0.6 0.6 1]; m = [1 1 0 0];
b = fir2(30,f,m);
[h,w] = freqz(b,1,128);
plot(f,m,w/pi,abs(h))
```



See Also

`butter`	Butterworth analog and digital filter design.
`cheby1`	Chebyshev type I filter design (passband ripple).
`cheby2`	Chebyshev type II filter design (stopband ripple).
`ellip`	Elliptic (Cauer) filter design.
`fir1`	Window-based finite impulse response filter design – standard response.
`maxflat`	Generalized digital Butterworth filter design.
`remez`	Parks-McClellan optimal FIR filter design.
`yulewalk`	Recursive digital filter design.

Meta Platforms, Inc. - Exhibit 1009
Page 463 of 720

Purpose

Constrained least square FIR filter design for multiband filters.

Syntax

```
b = fircls(n,f,amp,up,lo)
fircls(n,f,amp,up,lo,'design_flag')
```

Description

`b = fircls(n,f,amp,up,lo)` generates a length `n+1` linear phase FIR filter `b`. The frequency-magnitude characteristics of this filter match those given by vectors `f` and `amp`:

- `f` is a vector of transition frequencies in the range from 0 to 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency). The first point of `f` must be 0 and the last point 1. The frequency points must be in increasing order.
- `amp` is a vector describing the piecewise constant desired amplitude of the frequency response. The length of `amp` is equal to the number of bands in the response and should be equal to `length(f)-1`.
- `up` and `lo` are vectors with the same length as `amp`. They define the upper and lower bounds for the frequency response in each band.

`fircls(n,f,amp,up,lo,'design_flag')` enables you to monitor the filter design, where *`design_flag`* can be:

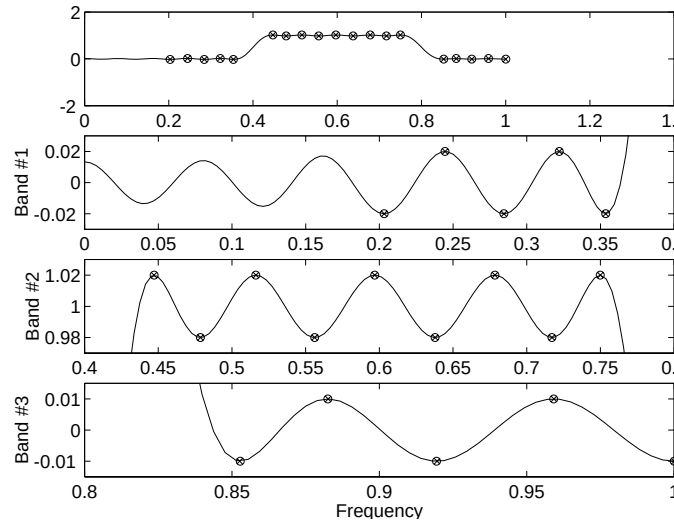
- `trace`, for a textual display of the design error at each iteration step.
- `plots`, for a collection of plots showing the filter's full-band magnitude response and a zoomed view of the magnitude response in each sub-band. All plots are updated at each iteration step.
- `both`, for both the textual display and plots.

Meta Platforms, Inc. - Exhibit 1009
Page 464 of 720

Example

Design an order 50 bandpass filter:

```
n = 50;
f = [0 0.4 0.8 1];
amp = [0 1 0];
up = [0.02 1.02 0.01];
lo = [-0.02 0.98 -0.01];
b = fircls(n,f,amp,up,lo,'plots');  % plots magnitude response
```



NOTE Normally, the lower value in the stopband will be specified as negative. By setting `lo` equal to 0 in the stopbands, a nonnegative frequency response amplitude can be obtained. Such filters can be spectrally factored to obtain minimum phase filters.

Algorithm

The algorithm is a multiple exchange algorithm that uses Lagrange multipliers and Kuhn-Tucker conditions on each iteration.

Meta Platforms, Inc. - Exhibit 1009
Page 465 of 720

See Also

`fircls1`	Constrained least square filter design for lowpass and highpass linear phase FIR filters.
`firls`	Least square linear-phase FIR filter design.
`remez`	Parks-McClellan optimal FIR filter design.

References

[1] Selesnick, I.W., M. Lang, and C.S. Burrus. “Constrained Least Square Design of FIR Filters without Specified Transition Bands.” *Proceedings of the IEEE Int. Conf. Acoust., Speech, Signal Processing*. Vol. 2 (May 1995). Pgs. 1260-1263.

[2] Selesnick, I.W., M. Lang, and C.S. Burrus. “Constrained Least Square Design of FIR Filters without Specified Transition Bands.” *IEEE Transactions on Signal Processing*, Vol. 44, No. 8 (August 1996).

Meta Platforms, Inc. - Exhibit 1009
Page 466 of 720

Purpose

Constrained least square filter design for lowpass and highpass linear phase FIR filters.

Syntax

```
b = fircls1(n,wo,dp,ds)
b = fircls1(n,wo,dp,ds,'high')
b = fircls1(n,wo,dp,ds,wt)
b = fircls1(n,wo,dp,ds,wt,'high')
b = fircls1(n,wo,dp,ds,wp,ws,k)
b = fircls1(n,wo,dp,ds,wp,ws,k,'high')
b = fircls1(n,wo,dp,ds,…,'design_flag')
```

Description

`b = fircls1(n,wo,dp,ds)` generates a lowpass FIR filter `b`. `n+1` is the filter length, `wo` is the normalized cutoff frequency in the range between 0 and 1 (where 1 corresponds to half the sampling frequency, that is, the Nyquist frequency), `dp` is the maximum passband deviation from 1 (passband ripple), and `ds` is the maximum stopband deviation from 0 (stopband ripple).

`b = fircls1(n,wo,dp,ds,'high')` generates a highpass FIR filter `b`.

`b = fircls1(n,wo,dp,ds,wt)` and

`b = fircls1(n,wo,dp,ds,wt,'high')` specify a frequency `wt` above which (for `wt>wo`) or below which (for `wt<wo`) the filter is guaranteed to meet the given band criterion. This will help you design a filter that meets a passband or stopband edge requirement. There are four cases:

- Lowpass:
  - `0<wt<wo<1`: the amplitude of the filter is within `dp` of 1 over the frequency range $0 < \omega <$ `wt`.
  - `0<wo<wt<1`: the amplitude of the filter is within `ds` of 0 over the frequency range `wt` $< \omega < 1$.
- Highpass:
  - `0<wt<wo<1`: the amplitude of the filter is within `ds` of 0 over the frequency range $0 < \omega <$ `wt`.
  - `0<wo<wt<1`: the amplitude of the filter is within `dp` of 1 over the frequency range `wt` $< \omega < 1$.

Meta Platforms, Inc. - Exhibit 1009
Page 467 of 720

`b = fircls1(n,wo,dp,ds,wp,ws,k)` generates a lowpass FIR filter `b` with a weighted function. `n+1` is the filter length, `wo` is the normalized cutoff frequency, `dp` is the maximum passband deviation from 1 (passband ripple), and `ds` is the maximum stopband deviation from 0 (stopband ripple). `wp` is the passband edge of the L2 weight function and `ws` is the stopband edge of the L2 weight function, where `wp` < `wo` < `ws`. `k` is the ratio (passband L2 error)/(stopband L2 error):

$$\frac{\int_0^{w_p} |A(\omega) - D(\omega)|^2 \, d\omega}{\int_{w_s}^{\pi} |A(\omega) - D(\omega)|^2 \, d\omega} = k$$

`b = fircls1(n,wo,dp,ds,wp,ws,k,'high')` generates a highpass FIR filter `b` with a weighted function, where `ws` < `wo` < `wp`.

`b = fircls1(n,wo,dp,ds,…,'`*`design_flag`*`')` enables you to monitor the filter design, where *`design_flag`* can be

- `trace`, for a textual display of the design table used in the design
- `plots`, for plots of the filter's magnitude, group delay, and zeros and poles
- `both`, for both the textual display and plots

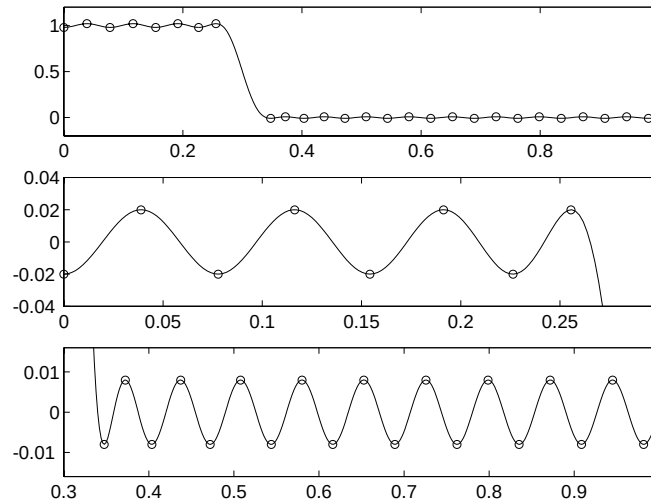
NOTE In the design of very narrow band filters with small `dp` and `ds`, there may not exist a filter of the given length that meets the specifications.

Meta Platforms, Inc. - Exhibit 1009
Page 468 of 720

Example

Design an order 55 lowpass filter with a cutoff frequency at 0.3:

```
n = 55; wo = 0.3;
dp = 0.02; ds = 0.008;
b = fircls1(n,wo,dp,ds,'plots'); % plot magnitude response
```



Algorithm

The algorithm is a multiple exchange algorithm that uses Lagrange multipliers and Kuhn-Tucker conditions on each iteration.

See Also

`fircls`	Constrained least square FIR filter design for multiband filters.
`firls`	Least square linear-phase FIR filter design.
`remez`	Parks-McClellan optimal FIR filter design.

References

[1] Selesnick, I.W., M. Lang, and C.S. Burrus. “Constrained Least Square Design of FIR Filters without Specified Transition Bands.” *Proceedings of the IEEE Int. Conf. Acoust., Speech, Signal Processing.* Vol. 2 (May 1995). Pgs. 1260-1263.

[2] Selesnick, I.W., M. Lang, and C.S. Burrus. “Constrained Least Square Design of FIR Filters without Specified Transition Bands.” *IEEE Transactions on Signal Processing*, Vol. 44, No. 8 (August 1996).

Meta Platforms, Inc. - Exhibit 1009
Page 469 of 720

Purpose

Least square linear-phase FIR filter design.

Syntax

```
b = firls(n,f,a)
b = firls(n,f,a,w)
b = firls(n,f,a,'ftype')
b = firls(n,f,a,w,'ftype')
```

Description

`firls` designs a linear-phase FIR filter that minimizes the weighted, integrated squared error between an ideal piecewise linear function and the magnitude response of the filter over a set of desired frequency bands.

`b = firls(n,f,a)` returns row vector `b` containing the `n+1` coefficients of the order `n` FIR filter whose frequency-amplitude characteristics approximately match those given by vectors `f` and `a`. The output filter coefficients, or "taps," in `b` obey the symmetry relation

$$b(k) = b(n + 2 - k), \qquad k = 1, \ldots, n + 1$$

These are type I (`n` odd) and type II (`n` even) linear-phase filters. Vectors `f` and `a` specify the frequency-amplitude characteristics of the filter:

- `f` is a vector of pairs of frequency points, specified in the range between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency). The frequencies must be in increasing order. Duplicate frequency points are allowed and, in fact, can be used to design a filter exactly the same as those returned by the `fir1` and `fir2` functions with a rectangular or `boxcar` window.
- `a` is a vector containing the desired amplitude at the points specified in `f`.

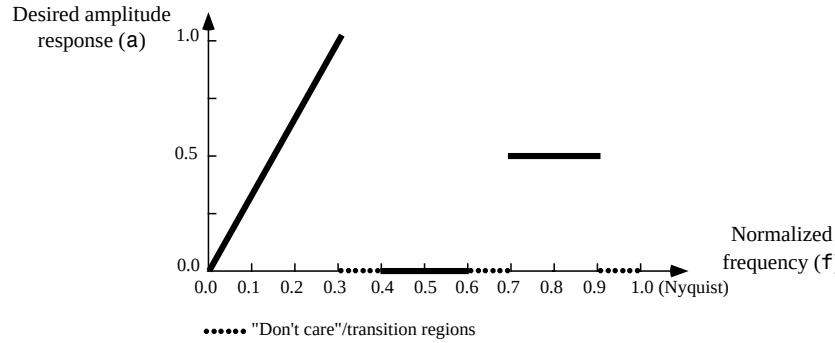
  The desired amplitude function at frequencies between pairs of points ($f(k)$, $f(k+1)$) for k odd is the line segment connecting the points ($f(k)$, $a(k)$) and ($f(k+1)$, $a(k+1)$).

  The desired amplitude function at frequencies between pairs of points ($f(k)$, $f(k+1)$) for k even is unspecified. These are transition or "don't care" regions.
- `f` and `a` are the same length. This length must be an even number.

Meta Platforms, Inc. - Exhibit 1009
Page 470 of 720

The relationship between the `f` and `a` vectors in defining a desired amplitude response is

```
f = [0 .3 .4 .6 .7 .9]
a = [0  1  0  0 .5 .5]
```



`b = firls(n,f,a,w)` uses the weights in vector `w` to weight the fit in each frequency band. The length of `w` is half the length of `f` and `a`, so there is exactly one weight per band.

`b = firls(n,f,a,'`*`ftype`*`')` and

`b = firls(n,f,a,w,'`*`ftype`*`')` specify a filter type, where *`ftype`* is

- `hilbert` for linear-phase filters with odd symmetry (type III and type IV)

  The output coefficients in `b` obey the relation $b(k) = -b(n + 2 - k)$, $k = 1, ..., n + 1$. This class of filters includes the Hilbert transformer, which has a desired amplitude of 1 across the entire band.

- `differentiator` for type III and type IV filters, using a special weighting technique

  For nonzero amplitude bands, the integrated squared error has a weight of $(1/f)^2$ so that the error at low frequencies is much smaller than at high frequencies. For FIR differentiators, which have an amplitude characteristic proportional to frequency, the filters minimize the relative integrated squared error (the integral of the square of the ratio of the error to the desired amplitude).

Meta Platforms, Inc. - Exhibit 1009
Page 471 of 720

Examples

Design an order 255 lowpass filter with transition band:

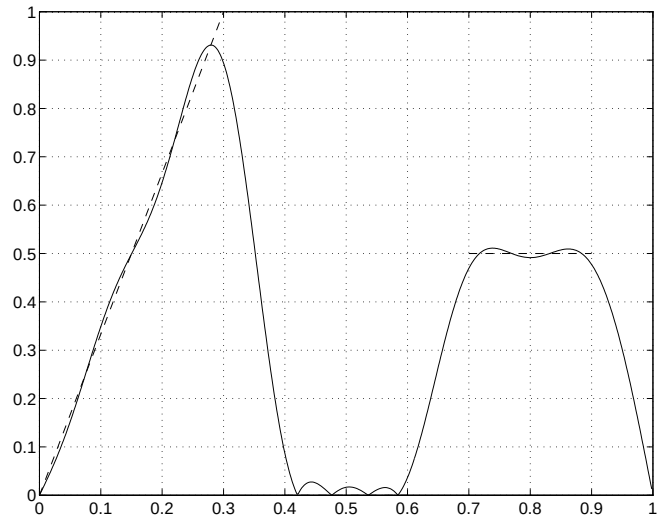
```
b = firls(255,[0 0.25 0.3 1],[1 1 0 0]);
```

Design a 31 coefficient differentiator:

```
b = firls(30,[0 0.9],[0 0.9],'differentiator');
```

Design a 24th-order anti-symmetric filter with piecewise linear passbands and plot the desired and actual frequency response:

```
F = [0 0.3  0.4 0.6  0.7 0.9];
A = [0  1   0  0  0.5 0.5];
b = firls(24,F,A,'hilbert');
for i=1:2:6,
   plot([F(i) F(i+1)],[A(i) A(i+1)],'- -'), hold on
end
[H,f] = freqz(b,1,512,2);
plot(f,abs(H)), grid on, hold off
```



Algorithm

Reference [1] describes the theoretical approach that `firls` takes. The function solves a system of linear equations involving an inner product matrix of size roughly `n/2` using MATLAB's \ operator.

Meta Platforms, Inc. - Exhibit 1009
Page 472 of 720

This function designs type I, II, III, and IV linear-phase filters. Type I and II are the defaults for n even and odd respectively, while the `'hilbert'` and `'differentiator'` flags produce type III (n even) and IV (n odd) filters. The various filter types have different symmetries and constraints on their frequency responses (see [2] for details).

Linear Phase Filter Type	Filter Order n	Symmetry of Coefficients	Response H(f), f = 0	Response H(f), f = 1 (Nyquist)
Type I	Even	even: $b(k) = b(n+2-k), \quad k = 1, \ldots, n+1$	No restriction	No restriction
Type II	Odd		No restriction	$H(1) = 0$
Type III	Even	odd: $b(k) = -b(n+2-k), \quad k = 1, \ldots, n+1$	$H(0) = 0$	$H(1) = 0$
Type IV	Odd		$H(0) = 0$	No restriction

Diagnostics

An appropriate diagnostic message is displayed when incorrect arguments are used:

```
F must be even length.
F and A must be equal lengths.
Requires symmetry to be 'hilbert' or 'differentiator'.
Requires one weight per band.
Frequencies in F must be nondecreasing.
Frequencies in F must be in range [0,1].
```

A more serious warning message is

```
Warning: Matrix is close to singular or badly scaled.
```

This tends to happen when the filter length times the transition width grows large. In this case, the filter coefficients b might not represent the desired filter. You can check the filter by looking at its frequency response.

Meta Platforms, Inc. - Exhibit 1009
Page 473 of 720

See Also

`fir1`	Window-based finite impulse response filter design – standard response.
`fir2`	Window-based finite impulse response filter design – arbitrary response.
`firrcos`	Raised cosine FIR filter design.
`remez`	Parks-McClellan optimal FIR filter design.

References

[1] Parks, T.W., and C.S. Burrus. *Digital Filter Design*. New York: John Wiley & Sons, 1987. Pgs. 54-83.

[2] Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989. Pgs. 256-266.

Meta Platforms, Inc. - Exhibit 1009
Page 474 of 720

Purpose

Raised cosine FIR filter design.

```
b = firrcos(n,F0,df,Fs)
b = firrcos(n,F0,df)
b = firrcos(n,F0,r,Fs,'rolloff')
b = firrcos(...,'type')
b = firrcos(...,'type',delay)
b = firrcos(...,'type',delay,window)
[b,a] = firrcos(...)
```

Description

`firrcos(n,F0,df,Fs)` returns an order `n` lowpass linear-phase FIR filter with a raised cosine transition band. The filter has cutoff frequency `F0`, transition bandwidth `df`, and sampling frequency `Fs`, all in Hertz. `df` must be small enough so that `F0` ± `df/2` is between `0` and `Fs/2`. The coefficients in `b` are normalized so that the nominal passband gain is always equal to one.

`firrcos(n,F0,df)` uses a default sampling frequency of `Fs = 2`.

`b = firrcos(n,F0,r,Fs,'rolloff')` interprets the third argument, `r`, as the rolloff factor instead of the transition bandwidth, `df`. `r` must be in the range `[0,1]`.

`b = firrcos(...,'type')` designs either a normal raised cosine filter or a square root raised cosine filter depending on the *`type`* specification, which can be

- `normal`, for a regular raised cosine filter. This is the default, and is also in effect when the '*`type`*' argument is left empty, `[]`.
- `sqrt`, for a square root raised cosine filter.

`b = firrcos(...,'type',delay)` specifies an integer delay in the range `[0,n+1]`. The default is `n/2` for even `n` and `(n+1)/2` for odd `n`.

`b = firrcos(...,'type',delay,window)` applies a length `n+1` window to the designed filter to reduce the ripple in the frequency response. `window` must be a `n+1` long column vector. If no window is specified, a boxcar (rectangular) window is used. Care must be exercised when using a window with a delay other than the default.

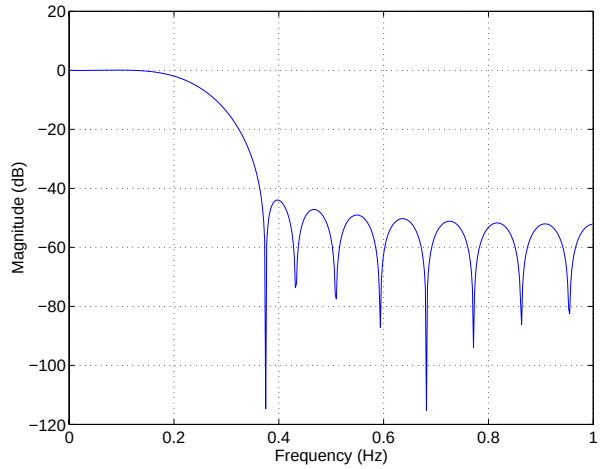
`[b,a] = firrcos(...)` always returns `a = 1`.

Meta Platforms, Inc. - Exhibit 1009
Page 475 of 720

Example

Design an order 20 raised cosine FIR filter with cutoff frequency 0.25 of the Nyquist frequency and a transition bandwidth of 0.25:

```
h = firrcos(20,0.25,0.25);
freqz(h,1,'Fs',2,'phase','no')
```



See Also

`fir1`	Window-based finite impulse response filter design – standard response.
`fir2`	Window-based finite impulse response filter design – arbitrary response.
`firls`	Least square linear-phase FIR filter design.
`remez`	Parks-McClellan optimal FIR filter design.

Meta Platforms, Inc. - Exhibit 1009
Page 476 of 720

Purpose

Frequency response of analog filters.

Syntax

```
h = freqs(b,a,w)
[h,w] = freqs(b,a)
[h,w] = freqs(b,a,n)
freqs(b,a)
```

Description

`freqs` returns the complex frequency response *H(jw)* (Laplace transform) of an analog filter:

$$H(s) = \frac{B(s)}{A(s)} = \frac{b(1)s^{nb} + b(2)s^{nb-1} + \cdots + b(nb+1)}{a(1)s^{na} + a(2)s^{na-1} + \cdots + a(na+1)}$$

given the numerator and denominator coefficients in vectors `b` and `a`.

`h = freqs(b,a,w)` returns the complex frequency response of the analog filter specified by coefficient vectors `b` and `a`. `freqs` evaluates the frequency response along the imaginary axis in the complex plane at the frequencies specified in real vector `w`.

`[h,w] = freqs(b,a)` automatically picks a set of 200 frequency points `w` on which to compute the frequency response `h`.

`[h,w] = freqs(b,a,n)` picks `n` frequencies on which to compute the frequency response `h`.

`freqs` with no output arguments plots the magnitude and phase response versus frequency in the current figure window.

`freqs` works only for real input systems and positive frequencies.

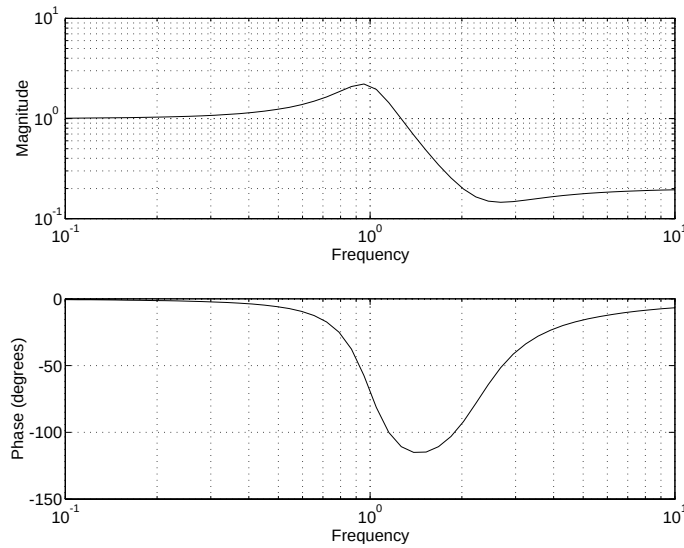
Meta Platforms, Inc. - Exhibit 1009
Page 477 of 720

Example

Find and graph the frequency response of the transfer function given by

$$H(s) = \frac{0.2s^2 + 0.3s + 1}{s^2 + 0.4s + 1}$$

```
a = [1 0.4 1];
b = [0.2 0.3 1];
w = logspace(-1,1);
freqs(b,a,w)
```



You can also create the plot with

```
h = freqs(b,a,w);
mag = abs(h);
phase = angle(h);
subplot(2,1,1), loglog(w,mag)
subplot(2,1,2), semilogx(w,phase)
```

To convert to Hertz, degrees, and decibels, use

```
f = w/(2*pi);
mag = 20*log10(mag);
phase = phase*180/pi;
```


Meta Platforms, Inc. - Exhibit 1009
Page 478 of 720

Algorithm

`freqs` evaluates the polynomials at each frequency point, then divides the numerator response by the denominator response:

```
s = i*w;
h = polyval(b,s)./polyval(a,s);
```

See Also

`abs`	Absolute value (magnitude).
`angle`	Phase angle.
`freqz`	Frequency response of digital filters.
`invfreqs`	Continuous-time (analog) filter identification from frequency data.
`logspace`	Generate logarithmically spaced vectors (see the online *MATLAB Function Reference*).
`polyval`	Polynomial evaluation (see the online *MATLAB Function Reference*).

Meta Platforms, Inc. - Exhibit 1009
Page 479 of 720

freqspace

Purpose

Frequency spacing for frequency response.

Syntax

```
f = freqspace(n)
f = freqspace(n,'whole')
[f1,f2] = freqspace(n)
[f1,f2] = freqspace([m n])
[x1,y1] = freqspace(n,'meshgrid')
[x1,y1] = freqspace([m n],'meshgrid')
```

Description

`freqspace` returns the implied frequency range for equally spaced frequency responses. This is useful when creating frequency vectors for use with `freqz`.

`f = freqspace(n)` returns the frequency vector `f` assuming `n` evenly spaced points around the unit circle. For `n` even or odd, `f` is `(0:2/n:1)`. For `n` even, `freqspace` returns `(n + 2)/2` points. For `N` odd, it returns `(n + 1)/2` points.

`f = freqspace(n,'whole')` returns `n` evenly spaced points around the whole unit circle. In this case, `f` is `0:2/n:2*(n–1)/n`.

`[f1,f2] = freqspace(n)` returns the two-dimensional frequency vectors `f1` and `f2` for an `n`-by-`n` matrix. For `n` odd, both `f1` and `f2` are `[–1 + 1/n:2/n:1–1/n]`. For `n` even, both `f1` and `f2` are `[–1:2/n:1–2/n]`.

`[f1,f2] = freqspace([m n])` returns the two-dimensional frequency vectors `f1` and `f2` for an `m`-by-`n` matrix.

`[x1,y1] = freqspace(n,'meshgrid')` and

`[x1,y1] = freqspace([m n],'meshgrid')` are equivalent to

```
[f1,f2] = freqspace(...);
[x1,y1] = meshgrid(f1,f2);
```

See the online *MATLAB Function Reference* for details on the `meshgrid` function.

See Also

`freqz`	Frequency response of digital filters.
`invfreqz`	Discrete-time filter identification from frequency data.

Meta Platforms, Inc. - Exhibit 1009
Page 480 of 720

Purpose

Frequency response of digital filters.

Syntax

```
[h,w] = freqz(b,a,n)
[h,f] = freqz(b,a,n,Fs)
[h,w] = freqz(b,a,n,'whole')
[h,f] = freqz(b,a,n,'whole',Fs)
h = freqz(b,a,w)
h = freqz(b,a,f,Fs)
freqz(...)
```

Description

`freqz` returns the complex frequency response $H(e^{j\omega})$ of a digital filter, given the (real or complex) numerator and denominator coefficients in vectors `b` and `a`.

`[h,w] = freqz(b,a,n)` returns the `n`-point complex frequency response of the digital filter

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + \cdots + b(nb+1)z^{-nb}}{a(1) + a(2)z^{-1} + \cdots + a(na+1)z^{-na}}$$

evaluated at $z=e^{j\omega}$ given the coefficient vectors `b` and `a`. `freqz` returns both `h`, the complex frequency response, and `w`, a vector containing the `n` frequency points in units of rads/sample. `freqz` evaluates the frequency response at `n` points equally spaced around the upper half of the unit circle, so `w` contains `n` points between 0 and π.

It is best, although not necessary, to choose a value for `n` that is an exact power of two, because this allows fast computation using an FFT algorithm. If you do not specify a value for `n`, it defaults to 512.

`[h,f] = freqz(b,a,n,Fs)` specifies a positive sampling frequency `Fs`, in Hertz. The default for `Fs` is 1. It returns a vector `f` containing the actual frequency points between 0 and `Fs/2` (the Nyquist frequency) at which it calculated the frequency response. `f` is of length `n`.

`[h,w] = freqz(b,a,n,'whole')` uses `n` points around the whole unit circle, so `w` has range $[0,2\pi)$.

`[h,f] = freqz(b,a,n,'whole',Fs)` uses `n` points around the whole unit circle, so `f` has range `[0,Fs)`.

Meta Platforms, Inc. - Exhibit 1009
Page 481 of 720

`h = freqz(b,a,w)` returns the frequency response at the frequencies in vector `w` (specified in rads/sample).

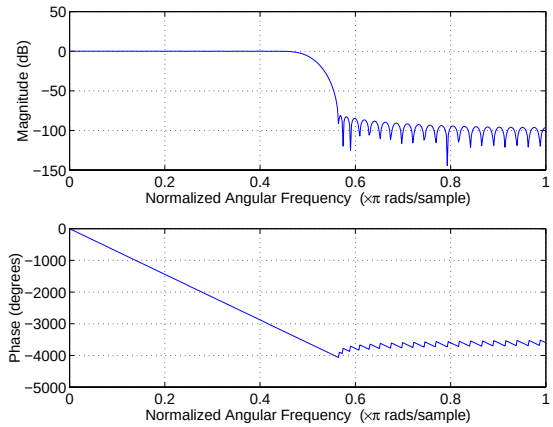
`h = freqz(b,a,f,Fs)` returns the frequency response at the frequencies in vector `f` (specified in Hz).

`freqz(...)` with no output arguments plots the magnitude and phase response versus frequency in the current figure window.

Examples

Plot the magnitude and phase response of an FIR filter.

```
b = fir1(80,0.5,kaiser(81,8));
freqz(b,1);
```



Algorithm

`freqz` uses an FFT algorithm when argument `n` is present. It computes the frequency response as the ratio of the transformed numerator and denominator coefficients, padded with zeros to the desired length:

```
h = fft(b,n)./fft(a,n)
```

If `n` is not a power of two, the FFT algorithm is not as efficient and may cause long computation times.

When a frequency vector `w` or `f` is present, or if `n` is less than `max(length(b),length(a))`, `freqz` evaluates the polynomials at each frequency point using Horner's method of polynomial evaluation and then divides the numerator response by the denominator response.

Meta Platforms, Inc. - Exhibit 1009
Page 482 of 720

See Also

`abs`	Absolute value (magnitude).
`angle`	Phase angle.
`fft`	One-dimensional fast Fourier transform.
`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
`freqs`	Frequency response of analog filters.
`impz`	Impulse response of digital filters.
`invfreqz`	Discrete-time filter identification from frequency data.
`logspace`	Generate logarithmically spaced vectors (see the online *MATLAB Function Reference*).

Meta Platforms, Inc. - Exhibit 1009
Page 483 of 720

gauspuls

Purpose

Gaussian-modulated sinusoidal pulse generator.

Syntax

```
yi = gauspuls(t,fc,bw)
yi = gauspuls(t,fc,bw,bwr)
[yi,yq] = gauspuls(…)
[yi,yq,ye] = gauspuls(…)
tc = gauspuls('cutoff',fc,bw,bwr,tpe)
```

Description

`gauspuls` generates Gaussian-modulated sinusoidal pulses.

`yi = gauspuls(t,fc,bw)` returns a unity-amplitude Gaussian RF pulse at the times indicated in array `t`, with a center frequency `fc` in Hertz and a fractional bandwidth `bw`, which must be greater than 0. The default value for `fc` is 1000 Hz and for `bw` is 0.5.

`yi = gauspuls(t,fc,bw,bwr)` returns a unity-amplitude Gaussian RF pulse with a fractional bandwidth of `bw` as measured at a level of `bwr` dB with respect to the normalized signal peak. The fractional bandwidth reference level `bwr` must be less than 0, because it indicates a reference level less than the peak (unity) envelope amplitude. The default value for `bwr` is -6 dB.

`[yi,yq] = gauspuls(…)` returns both the in-phase and quadrature pulses.

`[yi,yq,ye] = gauspuls(…)` returns the RF signal envelope.

`tc = gauspuls('cutoff',fc,bw,bwr,tpe)` returns the cutoff time `tc` (greater than or equal to 0) at which the trailing pulse envelope falls below `tpe` dB with respect to the peak envelope amplitude. The trailing pulse envelope level `tpe` must be less than 0, because it indicates a reference level less than the peak (unity) envelope amplitude. The default value for `tpe` is -60 dB.

Remarks

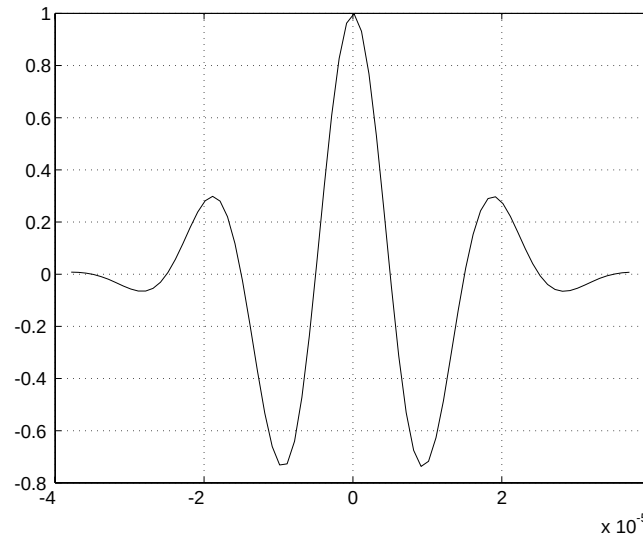
Default values are substituted for empty or omitted trailing input arguments.

Meta Platforms, Inc. - Exhibit 1009
Page 484 of 720

Example

Plot a 50 kHz Gaussian RF pulse with 60% bandwidth, sampled at a rate of 1 MHz. Truncate the pulse where the envelope falls 40 dB below the peak:

```
tc = gauspuls('cutoff',50e3,0.6,[],-40);
t = -tc : 1e-6 : tc;
yi = gauspuls(t,50e3,0.6);
plot(t,yi)
```



See Also

`chirp`	Swept-frequency cosine generator.
`cos`	Cosine of vector/matrix elements (see the online *MATLAB Function Reference*).
`diric`	Dirichlet or periodic sinc function.
`pulstran`	Pulse train generator.
`rectpuls`	Sampled aperiodic rectangle generator.
`sawtooth`	Sawtooth or triangle wave generator.
`sin`	Sine of vector/matrix elements (see the online *MATLAB Function Reference*).
`sinc`	Sinc or sin(πt)/πt function.
`square`	Square wave generator.
`tripuls`	Sampled aperiodic triangle generator.

Meta Platforms, Inc. - Exhibit 1009
Page 485 of 720

Purpose

Average filter delay (group delay).

Syntax

```
[gd,w] = grpdelay(b,a,n)
[gd,f] = grpdelay(b,a,n,Fs)
[gd,w] = grpdelay(b,a,n,'whole')
[gd,f] = grpdelay(b,a,n,'whole',Fs)
gd = grpdelay(b,a,w)
gd = grpdelay(b,a,f,Fs)
grpdelay(b,a)
```

Description

The *group delay* of a filter is a measure of the average delay of the filter as a function of frequency. It is the negative first derivative of the phase response of the filter. If the complex frequency response of a filter is $H(e^{j\omega})$, then the group delay is

$$\tau_g(\omega) = -\frac{d\theta(\omega)}{d\omega}$$

where ω is frequency and θ is the phase angle of $H(e^{j\omega})$.

`[gd,w] = grpdelay(b,a,n)` returns the `n`-point group delay, $\tau_g(\omega)$, of the digital filter

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + \cdots + b(nb+1)z^{-nb}}{a(1) + a(2)z^{-1} + \cdots + a(na+1)z^{-na}}$$

given the numerator and denominator coefficients in vectors `b` and `a`. `grpdelay` returns both `gd`, the group delay, and `w`, a vector containing the `n` frequency points in radians. `grpdelay` evaluates the group delay at `n` points equally spaced around the upper half of the unit circle, so `w` contains `n` points between 0 and π. A value for `n` that is an exact power of two allows fast computation using an FFT algorithm.

`[gd,f] = grpdelay(b,a,n,Fs)` specifies a positive sampling frequency `Fs` in Hertz. It returns a length `n` vector `f` containing the actual frequency points at which the group delay is calculated, also in Hertz. `f` contains `n` points between 0 and `Fs/2`.

Meta Platforms, Inc. - Exhibit 1009
Page 486 of 720

`[gd,w] = grpdelay(b,a,n,'whole')` and

`[gd,f] = grpdelay(b,a,n,'whole',Fs)` use `n` points around the whole unit circle (from 0 to 2π, or from 0 to `Fs`).

`gd = grpdelay(b,a,w)` and

`gd = grpdelay(b,a,f,Fs)` return the group delay evaluated at the points in `w` (in radians) or `f` (in Hertz), respectively, where `Fs` is the sampling frequency in Hertz.

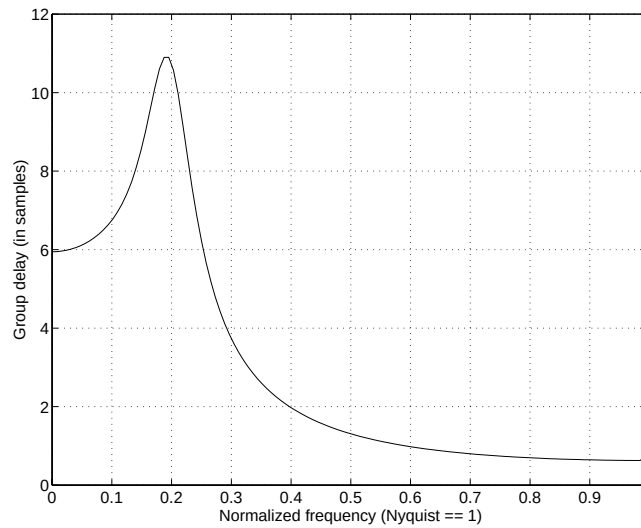
`grpdelay` with no output arguments plots the group delay versus frequency in the current figure window.

`grpdelay` works for both real and complex input systems.

Examples

Plot the group delay of Butterworth filter $b(z)/a(z)$:

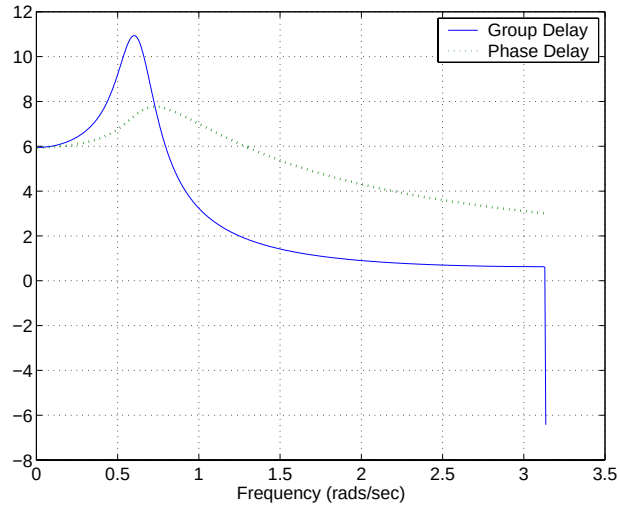
```
[b,a] = butter(6,0.2);
grpdelay(b,a,128)
```



Meta Platforms, Inc. - Exhibit 1009
Page 487 of 720

Plot both the group and phase delays of a system on the same graph:

```
gd = grpdelay(b,a,512);
gd(1) = []; % avoid NaNs
[h,w] = freqz(b,a,512); h(1) = []; w(1) = [];
pd = -unwrap(angle(h))./w;
plot(w,gd,w,pd,':')
xlabel('Frequency (rads/sec)'); grid;
legend('Group Delay','Phase Delay');
```



Algorithm

`grpdelay` multiplies the filter coefficients by a unit ramp. After Fourier transformation, this process corresponds to differentiation.

See Also

`cceps`	Complex cepstral analysis.
`fft`	One-dimensional fast Fourier transform.
`freqz`	Frequency response of digital filters.
`hilbert`	Hilbert transform.
`icceps`	Inverse complex cepstrum.
`rceps`	Real cepstrum and minimum phase reconstruction.

Meta Platforms, Inc. - Exhibit 1009
Page 488 of 720

Purpose

Hamming window.

Syntax

```
w = hamming(n)
w = hamming(n,sflag)
```

Description

`w = hamming(n)` returns an `n`-point symmetrically sampled Hamming window in the column vector `w`. `n` should be a nonnegative integer. The coefficients of a Hamming window are

$$w[k+1] = 0.54 - 0.46\cos\left(2\pi\frac{k}{n-1}\right), \qquad k = 0, \ldots, n-1$$

`w = hamming(n,sflag)` returns an `n`-point Hamming window using the window sampling specified by `sflag`, which can be either `'periodic'` or `'symmetric'` (the default). When `'periodic'` is specified, `hamming` computes a length `n+1` window and returns the first `n` points.

Diagnostics

An error message is displayed when incorrect arguments are used:

```
Order cannot be less than zero.
Sampling must be either 'symmetric' or 'periodic'.
```

A warning message is displayed for noninteger `n`:

```
Warning: Rounding order to nearest integer.
```

See Also

`bartlett`	Bartlett window.
`blackman`	Blackman window.
`boxcar`	Rectangular window.
`chebwin`	Chebyshev window.
`hanning`	Hanning window.
`kaiser`	Kaiser window.
`triang`	Triangular window.

References

[1] Oppenheim, A.V., and R.W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

Meta Platforms, Inc. - Exhibit 1009
Page 489 of 720

Purpose

Hanning window.

Syntax

```
w = hanning(n)
w = hanning(n,sflag)
```

Description

`w = hanning(n)` returns an `n`-point symmetrically sampled Hanning window in the column vector `w`. `n` should be a nonnegative integer. The coefficients of a Hanning window are

$$w[k] = 0.5\left(1 - \cos\left(2\pi\frac{k}{n+1}\right)\right), \qquad k = 1, \ldots, n$$

`w = hanning(n,sflag)` returns an `n`-point Hanning window using the window sampling specified by `sflag`, which can be either `'periodic'` or `'symmetric'` (the default). When `'periodic'` is specified, `hanning` computes a length `n+1` window and returns the first `n` points.

Diagnostics

An error message is displayed when incorrect arguments are used:

```
Order cannot be less than zero.
Sampling must be either 'symmetric' or 'periodic'.
```

A warning message is displayed for noninteger `n`:

```
Warning: Rounding order to nearest integer.
```

See Also

`bartlett`	Bartlett window.
`blackman`	Blackman window.
`boxcar`	Rectangular window.
`chebwin`	Chebyshev window.
`hamming`	Hamming window.
`kaiser`	Kaiser window.
`triang`	Triangular window.

References

[1] Oppenheim, A.V., and R.W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

Meta Platforms, Inc. - Exhibit 1009
Page 490 of 720

Purpose

Hilbert transform.

Syntax

```
y = hilbert(x)
```

Description

`y = hilbert(x)` returns a complex helical sequence, sometimes called the *analytic signal*, from a real data sequence. The analytic signal has a real part, which is the original data, and an imaginary part, which contains the Hilbert transform. The imaginary part is a version of the original real sequence with a 90° phase shift. Sines are therefore transformed to cosines and vice versa. The Hilbert transformed series has the same amplitude and frequency content as the original real data and includes phase information that depends on the phase of the original data.

If `x` is a matrix, `y = hilbert(x)` operates columnwise on the matrix, finding the Hilbert transform of each column.

The Hilbert transform is useful in calculating instantaneous attributes of a time series, especially the amplitude and frequency. The instantaneous amplitude is the amplitude of the complex Hilbert transform; the instantaneous frequency is the time rate of change of the instantaneous phase angle. For a pure sinusoid, the instantaneous amplitude and frequency are constant. The instantaneous phase, however, is a sawtooth, reflecting the way in which the local phase angle varies linearly over a single cycle. For mixtures of sinusoids, the attributes are short term, or local, averages spanning no more than two or three points.

Reference [1] describes the Kolmogorov method for minimum phase reconstruction, which involves taking the Hilbert transform of the logarithm of the spectrum of a time series. The toolbox function `rceps` performs this reconstruction.

Algorithm

The analytic signal for a sequence `x` has a *one-sided Fourier transform*, that is, negative frequencies are 0. To approximate the analytic signal, `hilbert` calculates the FFT of the input sequence, replaces those FFT coefficients that correspond to negative frequencies with zeros, and calculates the inverse FFT of the result.

In detail, `hilbert` uses a four-step algorithm:

Meta Platforms, Inc. - Exhibit 1009
Page 491 of 720

1. It calculates the FFT of the input sequence, storing the result in a vector `y`. Before transforming, it zero pads the input sequence so its length `n` is the closest power of two, if necessary. This ensures the most efficient FFT computation.
2. It creates a vector `h` whose elements `h(i)` have the values
   - 1 for `i` = 1, `(n/2) + 1`
   - 2 for `i` = 2, 3, ..., `(n/2)`
   - 0 for `i` = `(n/2) + 2`, ... , `n`
3. It calculates the element-wise product of `y` and `h`.
4. It calculates the inverse FFT of the sequence obtained in step 3 and returns the first `n` elements of the result.

If the input data `x` is a matrix, `hilbert` operates in a similar manner, extending each step above to handle the matrix case.

See Also

`fft`	One-dimensional fast Fourier transform.
`ifft`	One-dimensional inverse fast Fourier transform.
`rceps`	Real cepstrum and minimum phase reconstruction.

References

[1] Claerbout, J.F. *Fundamentals of Geophysical Data Processing*. New York: McGraw-Hill, 1976. Pgs. 59-62.

Meta Platforms, Inc. - Exhibit 1009
Page 492 of 720

Purpose Inverse complex cepstrum.

Syntax

```
x = icceps(xhat,nd)
```

Description `x = icceps(xhat,nd)` returns the inverse complex cepstrum of the (assumed real) sequence `xhat`, removing `nd` samples of delay. If `xhat` was obtained with `cceps(x)`, then the amount of delay that was added to `x` was the element of `round(unwrap(angle(fft(x)))/pi)` corresponding to π radians.

See Also

`cceps`	Complex cepstral analysis.
`hilbert`	Hilbert transform.
`rceps`	Real cepstrum and minimum phase reconstruction.
`unwrap`	Unwrap phase angles.

References

[1] Oppenheim, A.V., and R.W. Schafer. *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1975.

Meta Platforms, Inc. - Exhibit 1009
Page 493 of 720

Purpose Inverse discrete cosine transform.

Syntax

```
x = idct(y)
x = idct(y,n)
```

Description The inverse discrete cosine transform reconstructs a sequence from its discrete cosine transform (DCT) coefficients. The `idct` function is the inverse of the `dct` function.

`x = idct(y)` returns the inverse discrete cosine transform of `y`

$$x(n) = \sum_{k=1}^{N} w(k)y(k)\cos\frac{\pi(2n-1)(k-1)}{2N}, \qquad n = 1, \ldots, N$$

where

$$w(k) = \begin{cases} \frac{1}{\sqrt{N}}, & k = 1 \\ \sqrt{\frac{2}{N}}, & 2 \le k \le N \end{cases}$$

and N = `length(x)`, which is the same as `length(y)`. The series is indexed from $n = 1$ and $k = 1$ instead of the usual $n = 0$ and $k = 0$ because MATLAB vectors run from 1 to N instead of from 0 to N-1.

`x = idct(y,n)` appends zeros or truncates the vector `y` to length `n` before transforming.

If `y` is a matrix, `idct` transforms its columns.

See Also

`dct`	Discrete cosine transform (DCT).
`dct2`	Two-dimensional DCT (see *Image Processing Toolbox User's Guide*).
`idct2`	Two-dimensional inverse DCT (see *Image Processing Toolbox User's Guide*).
`ifft`	One-dimensional inverse fast Fourier transform.

Meta Platforms, Inc. - Exhibit 1009
Page 494 of 720

References

[1] Jain, A.K. *Fundamentals of Digital Image Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[2] Pennebaker, W.B., and J.L. Mitchell. *JPEG Still Image Data Compression Standard*. New York, NY: Van Nostrand Reinhold, 1993. Chapter 4.

Meta Platforms, Inc. - Exhibit 1009
Page 495 of 720

Purpose

One-dimensional inverse fast Fourier transform.

Syntax

```
y = ifft(x)
y = ifft(x,n)
```

Description

`ifft` computes the inverse Fourier transform of a vector or array. This function implements the inverse transform given by

$$x(n+1) = \frac{1}{N} \sum_{k=0}^{N-1} X(k+1) W_n^{-kn}$$

where $W_N = e^{-j(2p/N)}$ and N = `length(x)`. Note that the series is indexed as $n + 1$ and $k + 1$ instead of the usual n and k because MATLAB vectors run from 1 to N instead of from 0 to N-1.

`y = ifft(x)` is the inverse Fourier transform of vector `x`. If `x` is an array, `y` is the inverse FFT of each column of the matrix.

`y = ifft(x,n)` is the `n`-point inverse FFT. If the length of `x` is less than `n`, `ifft` pads `x` with trailing zeros to length `n`. If the length of `x` is greater than `n`, `ifft` truncates the sequence `x`. When `x` is an array, `ifft` adjusts the length of the columns in the same manner.

The `ifft` function is part of the standard MATLAB language.

Algorithm

The `ifft` function is an M-file. The algorithm for `ifft` is the same as that for `fft`, except for a sign change and a scale factor of `n = length(x)`. The execution time is fastest when `n` is a power of two and slowest when `n` is a large prime.

See Also

`fft`	One-dimensional fast Fourier transform.
`fft2`	Two-dimensional fast Fourier transform.
`fftshift`	Rearrange the outputs of `fft` and `fft2`.
`ifft2`	Two-dimensional inverse fast Fourier transform.

Meta Platforms, Inc. - Exhibit 1009
Page 496 of 720

Purpose

Two-dimensional inverse fast Fourier transform.

Syntax

```
Y = ifft2(X)
Y = ifft2(X,m,n)
```

Description

`Y = ifft2(X)` returns the two-dimensional inverse fast Fourier transform (FFT) of the array `X`. If `X` is a vector, `Y` has the same orientation as `X`.

`Y = ifft2(X,m,n)` truncates or zero pads `X`, if necessary, to create an `m`-by-`n` array before performing the inverse FFT. The result `Y` is also `m`-by-`n`.

For any `X`, `ifft2(fft2(X))` equals `X` to within roundoff error. If `X` is real, `ifft2(fft2(X))` may have small imaginary parts.

The `ifft2` function is part of the standard MATLAB language.

Algorithm

The algorithm for `ifft2` is the same as that for `fft2`, except for a sign change and scale factors of `[m n]= size(X)`. The execution time is fastest when `m` and `n` are powers of two and slowest when they are large primes.

See Also

`fft`	One-dimensional fast Fourier transform.
`fft2`	Two-dimensional fast Fourier transform.
`fftn`	*N*-dimensional fast Fourier transform (see the online *MATLAB Function Reference*).
`fftshift`	Rearrange the outputs of `fft` and `fft2`.
`ifft`	One-dimensional inverse fast Fourier transform.
`ifftn`	*N*-dimensional inverse fast Fourier transform (see the online *MATLAB Function Reference*).

Meta Platforms, Inc. - Exhibit 1009
Page 497 of 720

Purpose

Impulse invariance method of analog-to-digital filter conversion.

Syntax

```
[bz,az] = impinvar(b,a,Fs)
[bz,az] = impinvar(b,a)
[bz,az] = impinvar(b,a,Fs,tol)
```

Description

`[bz,az] = impinvar(b,a,Fs)` creates a digital filter with numerator and denominator coefficients `bz` and `az`, respectively, whose impulse response is equal to the impulse response of the analog filter with coefficients `b` and `a`, scaled by `1/Fs`.

`[bz,az] = impinvar(b,a)` uses the default value of 1 Hz for `Fs`.

`[bz,az] = impinvar(b,a,Fs,tol)` uses the tolerance specified by `tol` to determine whether poles are repeated. A larger tolerance increases the likelihood that `impinvar` will consider nearby poles to be repeated. The default is 0.001, or 0.1% of a pole's magnitude. Note that the accuracy of the pole values is still limited to the accuracy obtainable by the `roots` function.

Example

Convert an analog lowpass filter to a digital filter using `impinvar` with a sampling frequency of 10 Hz:

```
[b,a] = butter(4,0.3,'s');
[bz,az] = impinvar(b,a,10)

bz =

  1.0e-006 *

   -0.0000    0.1324    0.5192    0.1273         0

az =

    1.0000   -3.9216    5.7679   -3.7709    0.9246
```

Algorithm

`impinvar` performs the impulse-invariant method of analog-to-digital transfer function conversion discussed in reference [1]:

1. It finds the partial fraction expansion of the system represented by `b` and `a`.
2. It replaces the poles `p` by the poles `exp(p/Fs)`.

Meta Platforms, Inc. - Exhibit 1009
Page 498 of 720

3 It finds the transfer function coefficients of the system from the residues from step 1 and the poles from step 2.

See Also

`bilinear`	Bilinear transformation method of analog-to-digital filter conversion.
`lp2bp`	Lowpass to bandpass analog filter transformation.
`lp2bs`	Lowpass to bandstop analog filter transformation.
`lp2hp`	Lowpass to highpass analog filter transformation.
`lp2lp`	Lowpass to lowpass analog filter transformation.

References

[1] Parks, T.W., and C.S. Burrus. *Digital Filter Design*. New York: John Wiley & Sons, 1987. Pgs. 206-209.

Meta Platforms, Inc. - Exhibit 1009
Page 499 of 720

Purpose

Impulse response of digital filters.

Syntax

```
[h,t] = impz(b,a)
[h,t] = impz(b,a,n)
[h,t] = impz(b,a,n,Fs)
impz(b,a)
impz(...)
```

Description

`[h,t] = impz(b,a)` computes the impulse response of the filter with numerator coefficients `b` and denominator coefficients `a`. `impz` chooses the number of samples and returns the response in column vector `h` and times (or sample intervals) in column vector `t` (where `t = (0:n-1)'` and `n` is the computed impulse response length).

`[h,t] = impz(b,a,n)` computes `n` samples of the impulse response. If `n` is a vector of integers, `impz` computes the impulse response at those integer locations where 0 is the starting point of the filter.

`[h,t] = impz(b,a,n,Fs)` computes `n` samples and scales `t` so that samples are spaced `1/Fs` units apart. `Fs` is 1 by default. Leave `n` empty, `[]`, to let `impz` select the number of samples.

`impz` with no output arguments plots the impulse response in the current figure window using `stem(t,h)`.

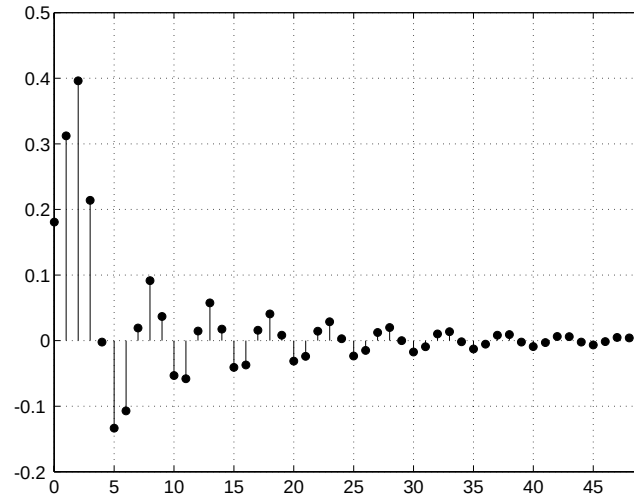
`impz` works for both real and complex input systems.

Meta Platforms, Inc. - Exhibit 1009
Page 500 of 720

Example

Plot the first 50 samples of the impulse response of a fourth-order lowpass elliptic filter with cutoff frequency of 0.4 times the Nyquist frequency:

```
[b,a] = ellip(4,0.5,20,0.4);
impz(b,a,50)
```



Algorithm

`impz` filters a length `n` impulse sequence using

```
filter(b,a,[1 zeros(1,n-1)])
```

To compute `n` in the auto-length case, `impz` either uses `n = length(b)` for the FIR case or first finds the poles using `p = roots(a)`, if `length(a)` is greater than `1`.

If the filter is unstable, `n` is chosen to be the point at which the term from the largest pole reaches `10^6` times its original value.

If the filter is stable, `n` is chosen to be the point at which the term due to the largest amplitude pole is `5*10^-5` of its original amplitude.

If the filter is oscillatory (poles on the unit circle only), `impz` computes five periods of the slowest oscillation.

If the filter has both oscillatory and damped terms, `n` is chosen to equal five periods of the slowest oscillation or the point at which the term due to the

Meta Platforms, Inc. - Exhibit 1009
Page 501 of 720

largest (nonunity) amplitude pole is `5*10^–5` of its original amplitude, whichever is greater.

`impz` also allows for delay in the numerator polynomial, which it adds to the resulting `n`.

See Also

`impulse`	Unit impulse response (see *Control System Toolbox User's Guide*).
`stem`	Plot discrete sequence data (see the online *MATLAB Function Reference*).

Meta Platforms, Inc. - Exhibit 1009
Page 502 of 720

Purpose Increase sampling rate by an integer factor (interpolation).

Syntax

```
y = interp(x,r)
y = interp(x,r,l,alpha)
[y,b] = interp(x,r,l,alpha)
```

Description Interpolation increases the original sampling rate for a sequence to a higher rate. `interp` performs lowpass interpolation by inserting zeros into the original sequence and then applying a special lowpass filter.

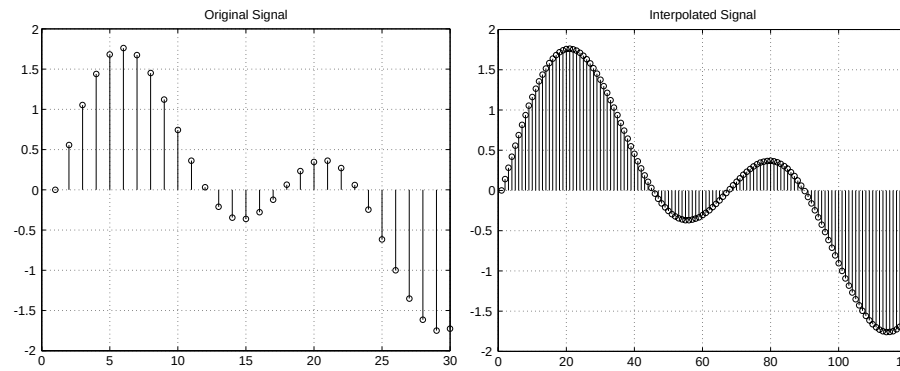
`y = interp(x,r)` increases the sampling rate of `x` by a factor of `r`. The interpolated vector `y` is `r` times longer than the original input `x`.

`y = interp(x,r,l,alpha)` specifies `l` (filter length) and `alpha` (cut-off frequency). The default value for `l` is 4 and the default value for `alpha` is 0.5.

`[y,b] = interp(x,r,l,alpha)` returns vector `b` containing the filter coefficients used for the interpolation.

Example Interpolate a signal by a factor of four:

```
t = 0:0.001:1; % time vector
x = sin(2*pi*30*t) + sin(2*pi*60*t);
y = interp(x,4);
stem(x(1:30)); title('Original Signal');
figure
stem(y(1:120)); title('Interpolated Signal');
```



Meta Platforms, Inc. - Exhibit 1009
Page 503 of 720

Algorithm

`interp` uses the lowpass interpolation Algorithm 8.1 described in [1]:

1. It expands the input vector to the correct length by inserting zeros between the original data values.
2. It designs a special symmetric FIR filter that allows the original data to pass through unchanged and interpolates between so that the mean-square errors between the interpolated points and their ideal values are minimized.
3. It applies the filter to the input vector to produce the interpolated output vector.

The length of the FIR lowpass interpolating filter is `2*l*r+1`. The number of original sample values used for interpolation is `2*l`. Ordinarily, `l` should be less than or equal to 10. The original signal is assumed to be band limited with normalized cutoff frequency $0 \leq$ `alpha` ≤ 1, where 1 is half the original sampling frequency (the Nyquist frequency). The default value for `l` is 4 and the default value for `alpha` is 0.5.

Diagnostics

If `r` is not an integer, `interp` gives the following error message:

```
Resampling rate R must be an integer.
```

See Also

`decimate`	Decrease the sampling rate for a sequence (decimation).
`interp1`	One-dimensional data interpolation (table lookup) (see the online *MATLAB Function Reference*).
`resample`	Change sampling rate by any rational factor.
`spline`	Cubic spline interpolation (see the online *MATLAB Function Reference*).
`upfirdn`	Upsample, apply an FIR filter, and downsample.

References

[1] IEEE. *Programs for Digital Signal Processing*. IEEE Press. New York: John Wiley & Sons, 1979. Algorithm 8.1.

Meta Platforms, Inc. - Exhibit 1009
Page 504 of 720

Purpose

Interpolation FIR filter design.

Syntax

```
b = intfilt(r,l,alpha)
b = intfilt(r,n,'Lagrange')
```

Description

`b = intfilt(r,l,alpha)` designs a linear phase FIR filter that performs ideal bandlimited interpolation using the nearest `2*l` nonzero samples, when used on a sequence interleaved with `r–1` consecutive zeros every `r` samples. It assumes an original bandlimitedness of `alpha` times the Nyquist frequency. The returned filter is identical to that used by `interp`.

`b = intfilt(r,n,'Lagrange')` or `b = intfilt(r,n,'l')` designs an FIR filter that performs `n`th-order Lagrange polynomial interpolation on a sequence interleaved with `r–1` consecutive zeros every `r` samples. `b` has length `(n + 1)*r` for `n` even, and length `(n + 1)*r–1` for `n` odd.

Both types of filters are basically lowpass and are intended for interpolation and decimation.

Examples

Design a digital interpolation filter to upsample a signal by four, using the bandlimited method:

```
alpha = 0.5; % "bandlimitedness" factor
h1 = intfilt(4,2,alpha); % bandlimited interpolation
```

The filter `h1` works best when the original signal is bandlimited to `alpha` times the Nyquist frequency. Create a bandlimited noise signal:

```
randn('seed',0)
x = filter(fir1(40,0.5),1,randn(200,1)); % bandlimit
```

Now zero pad the signal with three zeros between every sample. The resulting sequence is four times the length of `x`:

```
xr = reshape([x zeros(length(x),3)]',4*length(x),1);
```

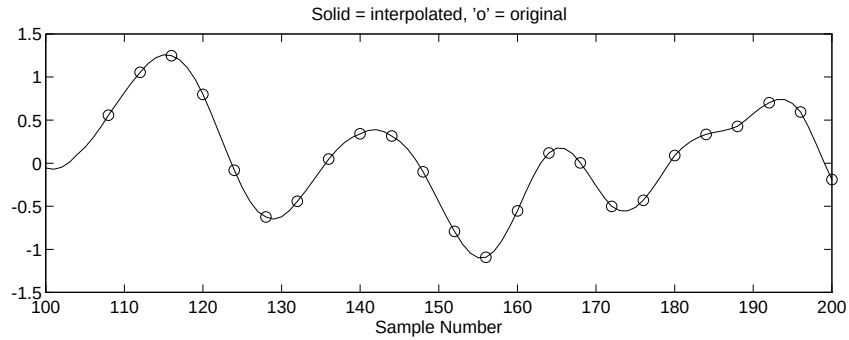
Interpolate using the `filter` command:

```
y = filter(h1,1,xr);
```


Meta Platforms, Inc. - Exhibit 1009
Page 505 of 720

y is an interpolated version of x, delayed by seven samples (the group-delay of the filter). Zoom in on a section to see this:

```
plot(100:200,y(100:200),7+(101:4:196),x(26:49),'o')
```



intfilt's other type of filter performs Lagrange polynomial interpolation of the original signal. For example, first-order polynomial interpolation is just linear interpolation, which is accomplished with a triangular filter:

```
h2 = intfilt(4,1,'l')   % Lagrange interpolation

h2 =
   0.2500   0.5000   0.7500   1.0000   0.7500   0.5000  0.2500
```

Algorithm

The bandlimited method uses firls to design an interpolation FIR equivalent to that presented in [1]. The polynomial method uses Lagrange's polynomial interpolation formula on equally spaced samples to construct the appropriate filter.

See Also

decimate	Decrease the sampling rate for a sequence (decimation).
interp	Increase sampling rate by an integer factor (interpolation).
resample	Change sampling rate by any rational factor.

References

[1] Oetken, Parks, and Schüßler. "New Results in the Design of Digital Interpolators." IEEE *Trans. Acoust., Speech, Signal Processing*. Vol. ASSP-23 (June 1975). Pgs. 301-309.

Meta Platforms, Inc. - Exhibit 1009
Page 506 of 720

Purpose

Continuous-time (analog) filter identification from frequency data.

Syntax

```
[b,a] = invfreqs(h,w,nb,na)
[b,a] = invfreqs(h,w,nb,na,wt)
[b,a] = invfreqs(h,w,nb,na,wt,iter)
[b,a] = invfreqs(h,w,nb,na,wt,iter,tol)
[b,a] = invfreqs(h,w,nb,na,wt,iter,tol,'trace')
[b,a] = invfreqs(h,w,'complex',nb,na,...)
```

Description

`invfreqs` is the inverse operation of `freqs`; it finds a continuous-time transfer function that corresponds to a given complex frequency response. From a laboratory analysis standpoint, `invfreqs` is useful in converting magnitude and phase data into transfer functions.

`[b,a] = invfreqs(h,w,nb,na)` returns the real numerator and denominator coefficient vectors `b` and `a` of the transfer function

$$H(s) = \frac{B(s)}{A(s)} = \frac{b(1)s^{nb} + b(2)s^{nb-1} + \cdots + b(nb+1)}{a(1)s^{na} + a(2)s^{na-1} + \cdots + a(na+1)}$$

whose complex frequency response is given in vector `h` at the frequency points specified in vector `w`. Scalars `nb` and `na` specify the desired orders of the numerator and denominator polynomials.

Frequency is specified in radians between 0 and π, and the length of `h` must be the same as the length of `w`. `invfreqs` uses `conj(h)` at `–w` to ensure the proper frequency domain symmetry for a real filter.

`[b,a] = invfreqs(h,w,nb,na,wt)` weights the fit-errors versus frequency. `wt` is a vector of weighting factors the same length as `w`.

`invfreqs(h,w,nb,na,wt,iter)` and

`invfreqs(h,w,nb,na,wt,iter,tol)` provide a superior algorithm that guarantees stability of the resulting linear system and searches for the best fit using a numerical, iterative scheme. The `iter` parameter tells `invfreqs` to end the iteration when the solution has converged, or after `iter` iterations, whichever comes first. `invfreqs` defines convergence as occurring when the

Meta Platforms, Inc. - Exhibit 1009
Page 507 of 720

norm of the (modified) gradient vector is less than `tol`. `tol` is an optional parameter that defaults to 0.01. To obtain a weight vector of all ones, use

```
invfreqs(h,w,nb,na,[],iter,tol)
```

`invfreqs(h,w,nb,na,wt,iter,tol,'trace')` displays a textual progress report of the iteration.

`invfreqs(h,w,'complex',nb,na,...)` creates a complex filter. In this case no symmetry is enforced, and the frequency is specified in radians between -π and π.

Remarks

When building higher order models using high frequencies, it is important to scale the frequencies, dividing by a factor such as half the highest frequency present in `w`, so as to obtain well conditioned values of `a` and `b`. This corresponds to a rescaling of time.

Examples

Convert a simple transfer function to frequency response data and then back to the original filter coefficients:

```
a = [1 2 3 2 1 4]; b = [1 2 3 2 3];
[h,w] = freqs(b,a,64);
[bb,aa] = invfreqs(h,w,4,5)

bb =

    1.0000    2.0000    3.0000    2.0000    3.0000

aa =

    1.0000    2.0000    3.0000    2.0000    1.0000    4.0000
```


Meta Platforms, Inc. - Exhibit 1009
Page 508 of 720

Notice that bb and aa are equivalent to b and a, respectively. However, aa has poles in the left half-plane and thus the system is unstable. Use invfreqs's iterative algorithm to find a stable approximation to the system:

```
[bbb,aaa] = invfreqs(h,w,4,5,[],30)

bbb =

    0.6816    2.1015    2.6694    0.9113   -0.1218

aaa =

    1.0000    3.4676    7.4060    6.2102    2.5413    0.0001
```

Suppose you have two vectors, mag and phase, that contain magnitude and phase data gathered in a laboratory, and a third vector w of frequencies. You can convert the data into a continuous-time transfer function using invfreqs:

```
[b,a] = invfreqs(mag.*exp(j*phase),w,2,3);
```

Algorithm

By default, invfreqs uses an equation error method to identify the best model from the data. This finds b and a in

$$\min_{b,\,a} \sum_{k=1}^{n} wt(k)\left|h(k)A(w(k)) - B(w(k))\right|^2$$

by creating a system of linear equations and solving them with MATLAB's \ operator. Here $A(w(k))$ and $B(w(k))$ are the Fourier transforms of the polynomials a and b, respectively, at the frequency $w(k)$, and n is the number of frequency points (the length of h and w). This algorithm is based on Levi [1]. Several variants have been suggested in the literature, where the weighting function wt gives less attention to high frequencies.

The superior ("output-error") algorithm uses the damped Gauss-Newton method for iterative search [2], with the output of the first algorithm as the initial estimate. This solves the direct problem of minimizing the weighted sum of the squared error between the actual and the desired frequency response points:

$$\min_{b,\,a} \sum_{k=1}^{n} wt(k)\left|h(k) - \frac{B(w(k))}{A(w(k))}\right|^2$$

Meta Platforms, Inc. - Exhibit 1009
Page 509 of 720

See Also

`freqs`	Frequency response of analog filters.
`freqz`	Frequency response of digital filters.
`invfreqz`	Discrete-time filter identification from frequency data.
`prony`	Prony's method for time domain IIR filter design.

References

[1] Levi, E.C. "Complex-Curve Fitting." *IRE Trans. on Automatic Control*. Vol. AC-4 (1959). Pgs. 37-44.

[2] Dennis, J.E., Jr., and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice Hall, 1983.

Meta Platforms, Inc. - Exhibit 1009
Page 510 of 720

Purpose

Discrete-time filter identification from frequency data.

Syntax

```
[b,a] = invfreqz(h,w,nb,na)
[b,a] = invfreqz(h,w,nb,na,wt)
[b,a] = invfreqz(h,w,nb,na,wt,iter)
[b,a] = invfreqz(h,w,nb,na,wt,iter,tol)
[b,a] = invfreqz(h,w,nb,na,wt,iter,tol,'trace')
[b,a] = invfreqz(h,w,'complex',nb,na,...)
```

Description

`invfreqz` is the inverse operation of `freqz`; it finds a discrete-time transfer function that corresponds to a given complex frequency response. From a laboratory analysis standpoint, `invfreqz` can be used to convert magnitude and phase data into transfer functions.

`[b,a] = invfreqz(h,w,nb,na)` returns the real numerator and denominator coefficients in vectors `b` and `a` of the transfer function

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + \cdots + b(nb+1)z^{-nb}}{a(1) + a(2)z^{-1} + \cdots + a(na+1)z^{-na}}$$

whose complex frequency response is given in vector `h` at the frequency points specified in vector `w`. Scalars `nb` and `na` specify the desired orders of the numerator and denominator polynomials.

Frequency is specified in radians between 0 and π, and the length of `h` must be the same as the length of `w`. `invfreqz` uses `conj(h)` at –`w` to ensure the proper frequency domain symmetry for a real filter.

`[b,a] = invfreqz(h,w,nb,na,wt)` weights the fit-errors versus frequency. `wt` is a vector of weighting factors the same length as `w`.

`invfreqz(h,w,nb,na,wt,iter)` and

`invfreqz(h,w,nb,na,wt,iter,tol)` provide a superior algorithm that guarantees stability of the resulting linear system and searches for the best fit using a numerical, iterative scheme. The `iter` parameter tells `invfreqz` to end the iteration when the solution has converged, or after `iter` iterations, whichever comes first. `invfreqz` defines convergence as occurring when the

Meta Platforms, Inc. - Exhibit 1009
Page 511 of 720

norm of the (modified) gradient vector is less than `tol`. `tol` is an optional parameter that defaults to 0.01. To obtain a weight vector of all ones, use

```
invfreqz(h,w,nb,na,[],iter,tol)
```

`invfreqz(h,w,nb,na,wt,iter,tol,'trace')` displays a textual progress report of the iteration.

`invfreqz(h,w,'complex',nb,na,...)` creates a complex filter. In this case no symmetry is enforced, and the frequency is specified in radians between -π and π.

Example

Convert a simple transfer function to frequency response data and then back to the original filter coefficients:

```
a = [1 2 3 2 1 4]; b = [1 2 3 2 3];
[h,w] = freqz(b,a,64);
[bb,aa] = invfreqz(h,w,4,5)

bb =

    1.0000    2.0000    3.0000    2.0000    3.0000

aa =

    1.0000    2.0000    3.0000    2.0000    1.0000    4.0000
```

Notice that `bb` and `aa` are equivalent to `b` and `a`, respectively. However, `aa` has poles outside the unit circle and thus the system is unstable. Use `invfreqz`'s iterative algorithm to find a stable approximation to the system:

```
[bbb,aaa] = invfreqz(h,w,4,5,[],30)

bbb =

    0.2427    0.2788    0.0069    0.0971    0.1980

aaa =

    1.0000   -0.8944    0.6954    0.9997   -0.8933    0.6949
```

Algorithm

By default, `invfreqz` uses an equation error method to identify the best model from the data. This finds `b` and `a` in

Meta Platforms, Inc. - Exhibit 1009
Page 512 of 720

$$\min_{b,a} \sum_{k=1}^{n} wt(k)\left|h(k)A(w(k)) - B(w(k))\right|^2$$

by creating a system of linear equations and solving them with MATLAB's \ operator. Here $A(w(k))$ and $B(w(k))$ are the Fourier transforms of the polynomials a and b, respectively, at the frequency $w(k)$, and n is the number of frequency points (the length of h and w). This algorithm is a based on Levi [1].

The superior ("output-error") algorithm uses the damped Gauss-Newton method for iterative search [2], with the output of the first algorithm as the initial estimate. This solves the direct problem of minimizing the weighted sum of the squared error between the actual and the desired frequency response points:

$$\min_{b,a} \sum_{k=1}^{n} wt(k)\left|h(k) - \frac{B(w(k))}{A(w(k))}\right|^2$$

See Also

`freqs`	Frequency response of analog filters.
`freqz`	Frequency response of digital filters.
`invfreqs`	Continuous-time (analog) filter identification from frequency data.
`prony`	Prony's method for time domain IIR filter design.

References

[1] Levi, E.C. "Complex-Curve Fitting." *IRE Trans. on Automatic Control*. Vol. AC-4 (1959). Pgs. 37-44.

[2] Dennis, J.E., Jr., and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice Hall, 1983.

Meta Platforms, Inc. - Exhibit 1009
Page 513 of 720

Purpose

Kaiser window.

Syntax

```
w = kaiser(n,beta)
```

Description

`w = kaiser(n,beta)` returns an `n`-point Kaiser (I_0 - sinh) window in the column vector `w`. `beta` is the Kaiser window β parameter that affects the sidelobe attenuation of the Fourier transform of the window.

To obtain a Kaiser window that designs an FIR filter with sidelobe height $-\alpha$ dB, use the following β:

$$\beta = \begin{cases} 0.1102(\alpha - 8.7), & \alpha > 50 \\ 0.5842(\alpha - 21)^{0.4} + 0.07886(\alpha - 21), & 50 \geq \alpha \geq 21 \\ 0, & \alpha < 21 \end{cases}$$

Increasing `beta` widens the mainlobe and decreases the amplitude of the sidelobes (increases the attenuation).

See Also

`bartlett`	Bartlett window.
`blackman`	Blackman window.
`boxcar`	Rectangular window.
`chebwin`	Chebyshev window.
`hamming`	Hamming window.
`hanning`	Hanning window.
`kaiserord`	Estimate parameters for `fir1` with Kaiser window.
`triang`	Triangular window.

References

[1] Kaiser, J.F. “Nonrecursive Digital Filter Design Using the I_0 - sinh Window Function.” *Proc. 1974 IEEE Symp. Circuits and Syst.* (April 1974). Pgs. 20-23.

[2] IEEE. *Digital Signal Processing II*. IEEE Press. New York: John Wiley & Sons, 1975.

Meta Platforms, Inc. - Exhibit 1009
Page 514 of 720

Purpose

Estimate parameters for an FIR filter design with Kaiser window.

Syntax

```
[n,Wn,beta,ftype] = kaiserord(f,a,dev)
[n,Wn,beta,ftype] = kaiserord(f,a,dev,Fs)
c = kaiserord(f,a,dev,Fs,'cell')
```

Description

`kaiserord` returns a filter order `n` and `beta` parameter to specify a Kaiser window for use with the `fir1` function. Given a set of specifications in the frequency domain, `kaiserord` estimates the minimum FIR filter order that will approximately meet the specifications. `kaiserord` converts the given filter specifications into passband and stopband ripples and converts cutoff frequencies into the form needed for windowed FIR filter design.

NOTE If the band ripples are specified as unequal, the smallest one is used, since the Kaiser window method is constrained to give filters with equal ripple heights in all the passbands and stopbands.

`[n,Wn,beta,ftype] = kaiserord(f,a,dev)` finds the approximate order `n`, normalized frequency band edges `Wn`, and weights that meet input specifications `f`, `a`, and `dev`. `f` is a vector of band edges and `a` is a vector specifying the desired amplitude on the bands defined by `f`. The length of `f` is twice the length of `a`, minus 2. Together, `f` and `a` define a desired piecewise constant response function. `dev` is a vector the same size as `a` that specifies the maximum allowable error or deviation between the frequency response of the output filter and its desired amplitude, for each band.

`fir1` can use the resulting order `n`, frequency vector `Wn`, multiband magnitude type `ftype`, and the Kaiser window parameter `beta`. The `ftype` string is intended for use with `fir1`; it is equal to `'high'` for a highpass filter and `'stop'` for a bandstop filter. For multiband filters, it can be equal to `'dc–0'` when the first band is a stopband (starting at $f = 0$) or `'dc–1'` when the first band is a passband.

To design a filter `b` that approximately meets the specifications given by `kaiser` parameters `f`, `a`, and `dev`:

```
b = fir1(n,Wn,kaiser(n+1,beta),ftype,'noscale')
```


Meta Platforms, Inc. - Exhibit 1009
Page 515 of 720

`[n,Wn,beta,ftype] = kaiserord(f,a,dev,Fs)` specifies a sampling frequency `Fs`. If not present, `Fs` defaults to 2 Hz, implying a Nyquist frequency of 1 Hz. You can therefore specify band edges scaled to a particular application's sampling frequency.

`c = kaiserord(f,a,dev,Fs,'cell')` is a cell-array whose elements are the parameters to `fir1`.

NOTE In some cases, `kaiserord` underestimates or overestimates the order `n`. If the filter does not meet the specifications, try a higher order such as `n+1`, `n+2`, and so on, or a lower order.

Results are inaccurate if the cutoff frequencies are near 0 or the Nyquist frequency, or if `dev` is large (greater than 10%).

Algorithm

`kaiserord` uses empirically derived formulas for estimating the orders of lowpass filters, as well as differentiators and Hilbert transformers. Estimates for multiband filters (such as bandpass filters) are derived from the lowpass design formulas.

The design formulas that underlie the Kaiser window and its application to FIR filter design are

$$\beta = \begin{cases} 0.1102(\alpha - 8.7), & \alpha > 50 \\ 0.5842(\alpha - 21)^{0.4} + 0.07886(\alpha - 21), & 50 \geq \alpha \geq 21 \\ 0, & \alpha < 21 \end{cases}$$

where $\alpha = -20\log_{10}\delta$ is the stopband attenuation expressed in decibels (recall that $\delta_p = \delta_s$ is required). The design formula is:

$$n = \frac{\alpha - 7.95}{2.285(\Delta\omega)}$$

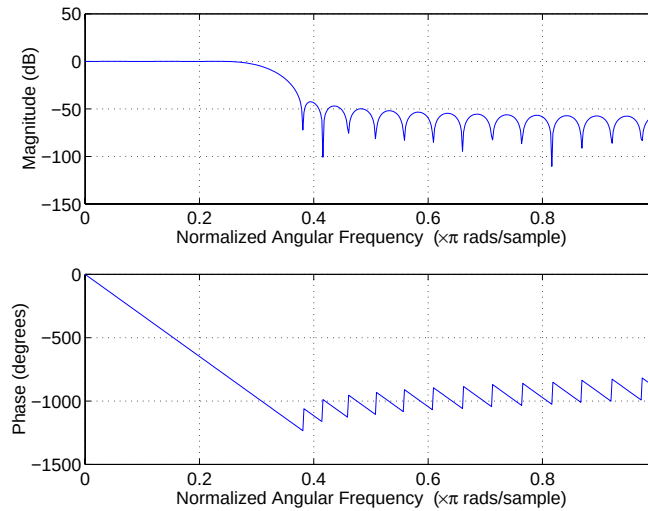
where n is the filter order and $\Delta\omega$ is the width of the smallest transition region.

Meta Platforms, Inc. - Exhibit 1009
Page 516 of 720

Examples

Design a lowpass filter with passband from 0 to 1 kHz and stopband from 1500 Hz to 4 kHz. Specify passband ripple of 5% and stopband attenuation of 40 dB:

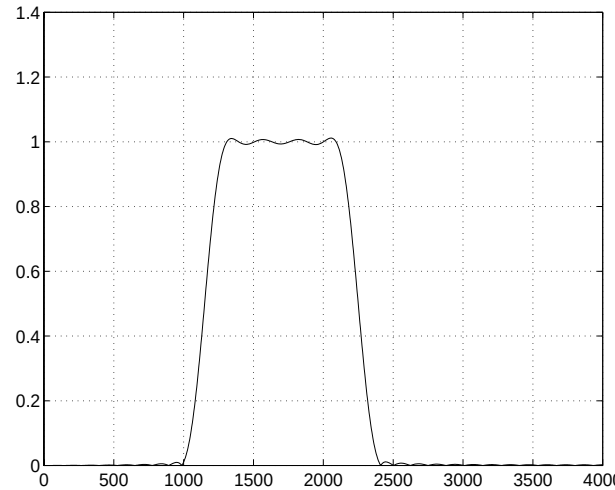
```
fsamp = 8000;
fcuts = [1000 1500];
mags = [1 0];
devs = [0.05 0.01];
[n,Wn,beta,ftype] = kaiserord(fcuts,mags,devs,fsamp);
hh = fir1(n,Wn,ftype,kaiser(n+1,beta),'noscale');
freqz(hh)
```



Meta Platforms, Inc. - Exhibit 1009
Page 517 of 720

Design an odd-length bandpass filter (note that odd length means even order, so the input to `fir1` must be an even integer):

```
fsamp = 8000;
fcuts = [1000 1300 2210 2410];
mags = [0 1 0];
devs = [0.01 0.05 0.01];
[n,Wn,beta,ftype] = kaiserord(fcuts,mags,devs,fsamp);
n = n + rem(n,2);
hh = fir1(n,Wn,ftype,kaiser(n+1,beta),'noscale');
[H,f] = freqz(hh,1,1024,fsamp);
plot(f,abs(H)), grid on
```



Design a lowpass filter with a passband cutoff of 1500 Hz, a stopband cutoff of 2000 Hz, passband ripple of 0.01, stopband ripple of 0.1, and a sampling frequency of 8000 Hz:

```
[n,Wn,beta,ftype] = kaiserord([1500 2000],[1 0],[0.01 0.1],8000);
b = fir1(n,Wn,ftype,kaiser(n+1,beta),'noscale');
```

This is equivalent to

```
c = kaiserord([1500 2000],[1 0],[0.01 0.1],8000,'cell');
b = fir1(c{:});
```


Meta Platforms, Inc. - Exhibit 1009
Page 518 of 720

Remarks

Be careful to distinguish between the meanings of filter length and filter order. The filter *length* is the number of impulse response samples in the FIR filter. Generally, the impulse response is indexed from $n = 0$ to $n = L-1$, where L is the filter length. The filter *order* is the highest power in a z-transform representation of the filter. For an FIR transfer function, this representation is a polynomial in z, where the highest power is z^{L-1} and the lowest power is z^0. The filter order is 1 less than the length (L-1) and is also equal to the number of zeros of the z polynomial.

Diagnostics

The lengths of the frequency, magnitude, and deviation vectors must match; otherwise, `kaiserord` gives the following error message:

```
Requires M and DEV to be the same length.
Length of F must be 2*length(M)–2.
```

When magnitudes are not specified correctly, `kaiserord` gives the following error messages:

```
Stopbands must be zero.
All passbands must have the same height.
```

If the band edges are not strictly increasing, `kaiserord` gives the following error message:

```
Bandedges must be strictly increasing.
```

See Also

`fir1`	Window-based finite impulse response filter design – standard response.
`kaiser`	Kaiser window.
`remezord`	Parks-McClellan optimal FIR filter order estimation.

References

[1] Kaiser, J.F. “Nonrecursive Digital Filter Design Using the I_0 - sinh Window Function.” P*roc. 1974 IEEE Symp. Circuits and Syst.* (April 1974). Pgs. 20-23.

[2] IEEE. *Digital Signal Processing II*. IEEE Press. New York: John Wiley & Sons, 1975. Pgs. 123-126.

Meta Platforms, Inc. - Exhibit 1009
Page 519 of 720

Purpose

Lattice filter to transfer function conversion.

Syntax

```
[num,den] = latc2tf(k,v)
[num,den] = latc2tf(k,'iir')
num = latc2tf(k,'fir')
num = latc2tf(k)
```

Description

`[num,den] = latc2tf(k,v)` finds the transfer function numerator `num` and denominator `den` from the IIR lattice coefficients `k` and ladder coefficients `v`.

`[num,den] = latc2tf(k,'iir')` assumes that `k` is associated with an all-pole IIR lattice filter.

`num = latc2tf(k,'fir')` and

`num = latc2tf(k)` find the transfer function numerators from the FIR lattice coefficients specified by `k`.

See Also

`latcfilt`	Lattice and lattice-ladder filter implementation.
`tf2latc`	Conversion of transfer function to lattice filter.

Meta Platforms, Inc. - Exhibit 1009
Page 520 of 720

Purpose

Lattice and lattice-ladder filter implementation.

Syntax

```
[f,g] = latcfilt(k,x)
[f,g] = latcfilt(k,v,x)
[f,g] = latcfilt(k,1,x)
[f,g,zf] = latcfilt(...,'ic',zi)
```

Description

`[f,g] = latcfilt(k,x)` filters `x` with the FIR lattice coefficients in vector `k`. `f` is the forward lattice filter result and `g` is the backward filter result.

If `k` and `x` are vectors, the result is a (signal) vector.

Matrix arguments are permitted under the following rules:

- If `x` is a matrix and `k` is a vector, each column of `x` is processed through the lattice filter specified by `k`.
- If `x` is a vector and `k` is a matrix, each column of `k` is used to filter `x`, and a signal matrix is returned.
- If `x` and `k` are both matrices with the same number of columns, then the *i*-th column of `k` is used to filter the *i*-th column of `x`. A signal matrix is returned.

`[f,g] = latcfilt(k,v,x)` filters `x` with the IIR lattice coefficients `k` and ladder coefficients `v`. `k` and `v` must be vectors, while `x` may be a signal matrix.

`[f,g] = latcfilt(k,1,x)` filters `x` with the IIR all-pole lattice specified by `k`. `k` and `x` may be vectors or matrices according to the rules given for the FIR lattice.

`[f,g,zf] = latcfilt(...,'ic',zi)` accepts a length-`k` vector `zi` specifying the initial condition of the lattice states. Output `zf` is a length-`k` vector specifying the final condition of the lattice states.

See Also

`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
`latc2tf`	Lattice filter to transfer function conversion.
`tf2latc`	Conversion of transfer function to lattice filter.

Meta Platforms, Inc. - Exhibit 1009
Page 521 of 720

Purpose

Levinson-Durbin recursion.

Syntax

```
a = levinson(r,p)
```

Description

The Levinson-Durbin recursion is an algorithm for finding an all-pole IIR filter with a prescribed deterministic autocorrelation sequence. It has applications in filter design, coding, and spectral estimation. The filter that `levinson` produces is minimum phase.

`a = levinson(r,p)` finds the coefficients of an `p`th-order autoregressive linear process which has `r` as its autocorrelation sequence. `r` is a real or complex deterministic autocorrelation sequence (a vector), and `p` is the order of denominator polynomial $A(z)$; that is, `a = [1 a(2) ... a(p+1)]`. The filter coefficients are ordered in descending powers of z:

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

Algorithm

`levinson` solves the symmetric Toeplitz system of linear equations.

$$\begin{bmatrix} r(1) & r(2)^* & \cdots & r(p)^* \\ r(2) & r(1) & \cdots & r(p-1)^* \\ \vdots & \ddots & \ddots & \vdots \\ r(p) & \cdots & r(2) & r(1) \end{bmatrix} \begin{bmatrix} a(2) \\ a(3) \\ \vdots \\ a(p+1) \end{bmatrix} = \begin{bmatrix} -r(2) \\ -r(3) \\ \vdots \\ -r(p+1) \end{bmatrix}$$

where `r = [`$r(1)$ ... $r(p+1)$`]` is the input autocorrelation vector, and $r(i)^*$ denotes the complex conjugate of $r(i)$. The algorithm requires $O(p^2)$ flops and is thus much more efficient than the MATLAB \ command for large `p`. However, the `levinson` function uses \ for low orders to provide the fastest possible execution.

See Also

`lpc`	Linear prediction coefficients.
`prony`	Prony's method for time domain IIR filter design.
`rlevinson`	Reverse Levinson-Durbin recursion.
`stmcb`	Linear model using Steiglitz-McBride iteration.

Meta Platforms, Inc. - Exhibit 1009
Page 522 of 720

References

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

Meta Platforms, Inc. - Exhibit 1009
Page 523 of 720

Purpose Lowpass to bandpass analog filter transformation.

Syntax

```
[bt,at] = lp2bp(b,a,Wo,Bw)
[At,Bt,Ct,Dt] = lp2bp(A,B,C,D,Wo,Bw)
```

Description `lp2bp` transforms analog lowpass filter prototypes with a cutoff frequency of 1 rad/sec into bandpass filters with desired bandwidth and center frequency. The transformation is one step in the digital filter design process for the `butter`, `cheby1`, `cheby2`, and `ellip` functions.

`lp2bp` can perform the transformation on two different linear system representations: transfer function form and state-space form. In both cases, the input system must be an analog filter prototype.

Transfer Function Form (Polynomial)

`[bt,at] = lp2bp(b,a,Wo,Bw)` transforms an analog lowpass filter prototype given by polynomial coefficients into a bandpass filter with center frequency `Wo` and bandwidth `Bw`. Row vectors `b` and `a` specify the coefficients of the numerator and denominator of the prototype in descending powers of s:

$$\frac{b(s)}{a(s)} = \frac{b(1)s^{nb} + \cdots + b(nb)s + b(nb+1)}{a(1)s^{na} + \cdots + a(na)s + a(na+1)}$$

Scalars `Wo` and `Bw` specify the center frequency and bandwidth in units of radians/second. For a filter with lower band edge `w1` and upper band edge `w2`, use `Wo = sqrt(w1*w2)` and `Bw = w2–w1`.

`lp2bp` returns the frequency transformed filter in row vectors `bt` and `at`.

State-Space Form

`[At,Bt,Ct,Dt] = lp2bp(A,B,C,D,Wo,Bw)` converts the continuous-time state-space lowpass filter prototype in matrices `A`, `B`, `C`, `D`:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

Meta Platforms, Inc. - Exhibit 1009
Page 524 of 720

into a bandpass filter with center frequency `Wo` and bandwidth `Bw`. For a filter with lower band edge `w1` and upper band edge `w2`, use `Wo = sqrt(w1*w2)` and `Bw = w2-w1`.

The bandpass filter is returned in matrices `At`, `Bt`, `Ct`, `Dt`.

Algorithm

`lp2bp` is a highly accurate state-space formulation of the classic analog filter frequency transformation. Consider the state-space system:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

where u is the input, x is the state vector, and y is the output. The Laplace transform of the first equation (assuming zero initial conditions) is

$$sX(s) = AX(s) + BU(s)$$

Now if a bandpass filter is to have center frequency ω_0 and bandwidth B_w, the standard s-domain transformation is

$$s = Q(p^2 + 1)/p$$

where $Q = \omega_0/B_w$ and $p = s/\omega_0$. Substituting this for s in the Laplace transformed state-space equation, and considering the operator p as d/dt:

$$Q\ddot{x} + Qx = \dot{A}x + B\dot{u}$$

or

$$Q\ddot{x} - \dot{A}x - B\dot{u} = -Qx$$

Now define

$$Q\dot{\omega} = -Qx$$

which, when substituted, leads to

$$Q\dot{x} = Ax + Q\omega + Bu$$

Meta Platforms, Inc. - Exhibit 1009
Page 525 of 720

The last two equations give equations of state. Write them in standard form and multiply the differential equations by ω_0 to recover the time/frequency scaling represented by p and find state matrices for the bandpass filter:

```
Q = Wo/Bw; [ma,na] = size(A);
At = Wo*[A/Q eye(ma,na);-eye(ma,na) zeros(ma,na)];
Bt = Wo*[B/Q; zeros(ma,nb)];
Ct = [C zeros(mc,ma)];
Dt = d;
```

If the input to `lp2bp` is in transfer function form, the function transforms it into state-space form before applying this algorithm.

See Also

`bilinear`	Bilinear transformation method of analog-to-digital filter conversion.
`impinvar`	Impulse invariance method of analog-to-digital filter conversion.
`lp2bs`	Lowpass to bandstop analog filter transformation.
`lp2hp`	Lowpass to highpass analog filter transformation.
`lp2lp`	Lowpass to lowpass analog filter transformation.

Meta Platforms, Inc. - Exhibit 1009
Page 526 of 720

Purpose

Lowpass to bandstop analog filter transformation.

Syntax

```
[bt,at] = lp2bs(b,a,Wo,Bw)
[At,Bt,Ct,Dt] = lp2bs(A,B,C,D,Wo,Bw)
```

Description

`lp2bs` transforms analog lowpass filter prototypes with a cutoff frequency of 1 rad/sec into bandstop filters with desired bandwidth and center frequency. The transformation is one step in the digital filter design process for the `butter`, `cheby1`, `cheby2`, and `ellip` functions.

`lp2bs` can perform the transformation on two different linear system representations: transfer function form and state-space form. In both cases, the input system must be an analog filter prototype.

Transfer Function Form (Polynomial)

`[bt,at] = lp2bs(b,a,Wo,Bw)` transforms an analog lowpass filter prototype given by polynomial coefficients into a bandstop filter with center frequency `Wo` and bandwidth `Bw`. Row vectors `b` and `a` specify the coefficients of the numerator and denominator of the prototype in descending powers of *s*:

$$\frac{b(s)}{a(s)} = \frac{b(1)s^{nb} + \cdots + b(nb)s + b(nb+1)}{a(1)s^{na} + \cdots + a(na)s + a(na+1)}$$

Scalars `Wo` and `Bw` specify the center frequency and bandwidth in units of radians/second. For a filter with lower band edge `w1` and upper band edge `w2`, use `Wo = sqrt(w1*w2)` and `Bw = w2-w1`.

`lp2bs` returns the frequency transformed filter in row vectors `bt` and `at`.

State-Space Form

`[At,Bt,Ct,Dt] = lp2bs(A,B,C,D,Wo,Bw)` converts the continuous-time state-space lowpass filter prototype in matrices `A`, `B`, `C`, `D`:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

Meta Platforms, Inc. - Exhibit 1009
Page 527 of 720

into a bandstop filter with center frequency `Wo` and bandwidth `Bw`. For a filter with lower band edge `w1` and upper band edge `w2`, use `Wo = sqrt(w1*w2)` and `Bw = w2-w1`.

The bandstop filter is returned in matrices `At`, `Bt`, `Ct`, `Dt`.

Algorithm

`lp2bs` is a highly accurate state-space formulation of the classic analog filter frequency transformation. If a bandstop filter is to have center frequency ω_0 and bandwidth B_w, the standard s-domain transformation is

$$s = \frac{p}{Q(p^2 + 1)}$$

where $Q = \omega_0/B_w$ and $p = s/\omega_0$. The state-space version of this transformation is

```
Q = Wo/Bw;
At = [Wo/Q*inv(A) Wo*eye(ma);-Wo*eye(ma) zeros(ma)];
Bt = -[Wo/Q*(A B); zeros(ma,nb)];
Ct = [C/A zeros(mc,ma)];
Dt = D - C/A*B;
```

See `lp2bp` for a derivation of the bandpass version of this transformation.

See Also

`bilinear`	Bilinear transformation method of analog-to-digital filter conversion.
`impinvar`	Impulse invariance method of analog-to-digital filter conversion.
`lp2bp`	Lowpass to bandpass analog filter transformation.
`lp2hp`	Lowpass to highpass analog filter transformation.
`lp2lp`	Lowpass to lowpass analog filter transformation.

Meta Platforms, Inc. - Exhibit 1009
Page 528 of 720

Purpose

Lowpass to highpass analog filter transformation.

Syntax

```
[bt,at] = lp2hp(b,a,Wo)
[At,Bt,Ct,Dt] = lp2hp(A,B,C,D,Wo)
```

Description

`lp2hp` transforms analog lowpass filter prototypes with a cutoff frequency of 1 rad/sec into highpass filters with desired cutoff frequency. The transformation is one step in the digital filter design process for the `butter`, `cheby1`, `cheby2`, and `ellip` functions.

The `lp2hp` function can perform the transformation on two different linear system representations: transfer function form and state-space form. In both cases, the input system must be an analog filter prototype.

Transfer Function Form (Polynomial)

`[bt,at] = lp2hp(b,a,Wo)` transforms an analog lowpass filter prototype given by polynomial coefficients into a highpass filter with cutoff frequency `Wo`. Row vectors `b` and `a` specify the coefficients of the numerator and denominator of the prototype in descending powers of *s*:

$$\frac{b(s)}{a(s)} = \frac{b(1)s^{nb} + \cdots + b(nb)s + b(nb+1)}{a(1)s^{na} + \cdots + a(na)s + a(na+1)}$$

Scalar `Wo` specifies the cutoff frequency in units of radians/second. The frequency transformed filter is returned in row vectors `bt` and `at`.

State-Space Form

`[At,Bt,Ct,Dt] = lp2hp(A,B,C,D,Wo)` converts the continuous-time state-space lowpass filter prototype in matrices `A`, `B`, `C`, `D`:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

into a highpass filter with cutoff frequency `Wo`. The highpass filter is returned in matrices `At`, `Bt`, `Ct`, `Dt`.

Meta Platforms, Inc. - Exhibit 1009
Page 529 of 720

Algorithm

`lp2hp` is a highly accurate state-space formulation of the classic analog filter frequency transformation. If a highpass filter is to have cutoff frequency ω_0, the standard s-domain transformation is

$$s = \frac{\omega_0}{p}$$

The state-space version of this transformation is

```
At = Wo*inv(A);
Bt = -Wo*(A\B);
Ct = C/A;
Dt = D - C/A*B;
```

See `lp2bp` for a derivation of the bandpass version of this transformation.

See Also

`bilinear`	Bilinear transformation method of analog-to-digital filter conversion.
`impinvar`	Impulse invariance method of analog-to-digital filter conversion.
`lp2bp`	Lowpass to bandpass analog filter transformation.
`lp2bs`	Lowpass to bandstop analog filter transformation.
`lp2lp`	Lowpass to lowpass analog filter transformation.

Meta Platforms, Inc. - Exhibit 1009
Page 530 of 720

Purpose

Lowpass to lowpass analog filter transformation.

Syntax

```
[bt,at] = lp2lp(b,a,Wo)
[At,Bt,Ct,Dt] = lp2lp(A,B,C,D,Wo)
```

Description

`lp2lp` transforms an analog lowpass filter prototype with a cutoff frequency of 1 rad/sec into a lowpass filter with any specified cutoff frequency. The transformation is one step in the digital filter design process for the `butter`, `cheby1`, `cheby2`, and `ellip` functions.

The `lp2lp` function can perform the transformation on two different linear system representations: transfer function form and state-space form. In both cases, the input system must be an analog filter prototype.

Transfer Function Form (Polynomial)

`[bt,at] = lp2lp(b,a,Wo)` transforms an analog lowpass filter prototype given by polynomial coefficients into a lowpass filter with cutoff frequency `Wo`. Row vectors `b` and `a` specify the coefficients of the numerator and denominator of the prototype in descending powers of *s*:

$$\frac{b(s)}{a(s)} = \frac{b(1)s^{nb} + \cdots + b(nb)s + b(nb+1)}{a(1)s^{na} + \cdots + a(na)s + a(na+1)}$$

Scalar `Wo` specifies the cutoff frequency in units of radians/second. `lp2lp` returns the frequency transformed filter in row vectors `bt` and `at`.

State-Space Form

`[At,Bt,Ct,Dt] = lp2lp(A,B,C,D,Wo)` converts the continuous-time state-space lowpass filter prototype in matrices `A`, `B`, `C`, `D`:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

into a lowpass filter with cutoff frequency `Wo`. `lp2lp` returns the lowpass filter in matrices `At`, `Bt`, `Ct`, `Dt`.

Meta Platforms, Inc. - Exhibit 1009
Page 531 of 720

Algorithm

`lp2lp` is a highly accurate state-space formulation of the classic analog filter frequency transformation. If a lowpass filter is to have cutoff frequency ω_0, the standard s-domain transformation is

$$s = p/\omega_0$$

The state-space version of this transformation is

```
At = Wo*A;
Bt = Wo*B;
Ct = C;
Dt = D;
```

See `lp2bp` for a derivation of the bandpass version of this transformation.

See Also

`bilinear`	Bilinear transformation method of analog-to-digital filter conversion.
`impinvar`	Impulse invariance method of analog-to-digital filter conversion.
`lp2bp`	Lowpass to bandpass analog filter transformation.
`lp2bs`	Lowpass to bandstop analog filter transformation.
`lp2hp`	Lowpass to highpass analog filter transformation.

Meta Platforms, Inc. - Exhibit 1009
Page 532 of 720

Purpose

Linear prediction coefficients.

Syntax

```
a = lpc(x,p)
```

Description

`lpc` determines the coefficients of a forward linear predictor by minimizing the prediction error in the least-squares sense. It has applications in filter design and speech coding.

`a = lpc(x,p)` finds the coefficients of a `p`th-order linear predictor (FIR filter) that predicts the current value of the real time series `x` based on past samples:

$$\hat{x}(n) = -a(2)x(n-1) - a(3)x(n-2) - \cdots - a(p+1)x(n-p)$$

`p` is the order of the prediction polynomial, `a = [1 a(2) ... a(p+1)]`.

If `p` is unspecified, `lpc` uses as a default `p = length(x)-1`. If `x` is a matrix containing a separate signal in each column, `lpc` returns a model estimate for each column in the rows of matrix `a`.

Example

Estimate a data series using a 3rd-order forward predictor, and compare to the original signal.

First, create the signal data as the output of an autoregressive process driven by white noise. Use the last 4096 samples of the AR process output to avoid start-up transients.

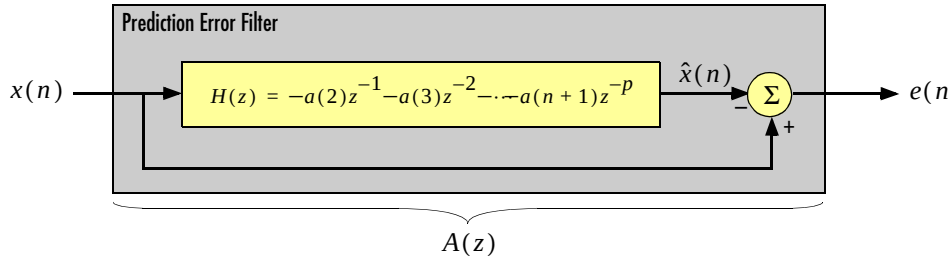
```
noise = randn(50000,1);    % Normalized white Gaussian noise
x = filter(1,[1 1/2 1/3 1/4],noise);
x = x(45904:50000);
```

Compute the predictor coefficients, estimated signal, prediction error, and autocorrelation sequence of the prediction error.

```
a = lpc(x,3);

est_x = filter([0 -a(2:end)],1,x);    % Estimated signal
e = x - est_x;                        % Prediction error
[acs,lags] = xcorr(e,'coeff');        % ACS of prediction error
```

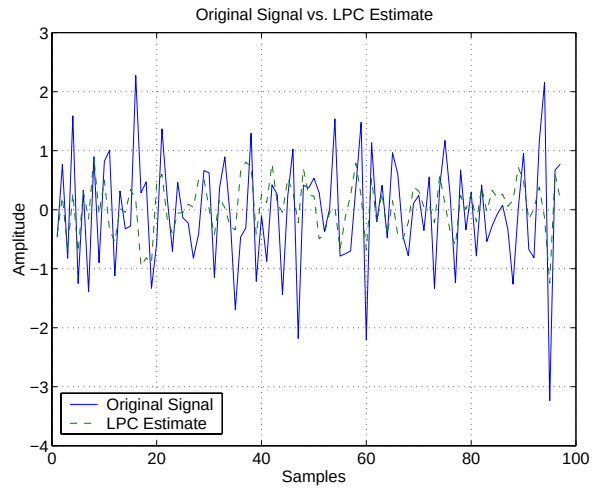

Meta Platforms, Inc. - Exhibit 1009
Page 533 of 720

The prediction error, $e(n)$, can be viewed as the output of the prediction error filter $A(z)$ shown below, where $H(z)$ is the optimal linear predictor, $x(n)$ is the input signal, and $\hat{x}(n)$ is the predicted signal.



Compare the predicted signal to the original signal.

```
plot(1:97,x(4001:4097),1:97,est_x(4001:4097),'--');
title('Original Signal vs. LPC Estimate');
xlabel('Samples'); ylabel('Amplitude'); grid;
legend('Original Signal','LPC Estimate')
```

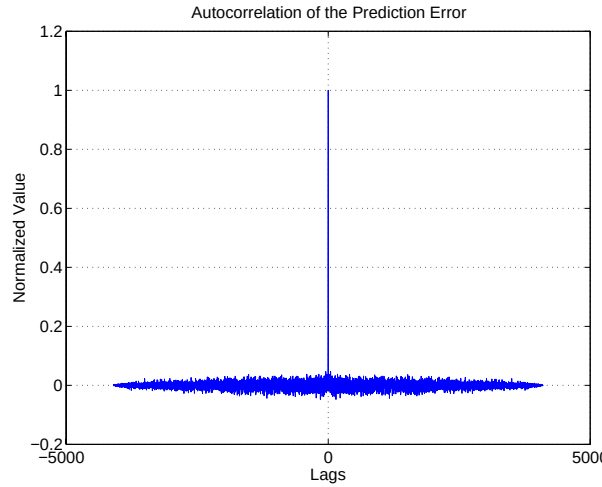


Look at the autocorrelation of the prediction error.

```
plot(lags,acs);
title('Autocorrelation of the Prediction Error');
xlabel('Lags'); ylabel('Normalized Value'); grid;
```


Meta Platforms, Inc. - Exhibit 1009
Page 534 of 720

The prediction error is approximately white Gaussian noise, as expected for a 3rd-order AR input process.



Algorithm

`lpc` uses the autocorrelation method of autoregressive (AR) modeling to find the filter coefficients. The generated filter might not model the process exactly even if the data sequence is truly an AR process of the correct order. This is because the autocorrelation method implicitly windows the data, that is, it assumes that signal samples beyond the length of `x` are 0.

`lpc` computes the least-squares solution to

$$Xa \approx b$$

where

$$X = \begin{bmatrix} x(1) & 0 & \cdots & 0 \\ x(2) & x(1) & \ddots & \vdots \\ \vdots & x(2) & \ddots & 0 \\ x(m) & \vdots & \ddots & x(1) \\ 0 & x(m) & \ddots & x(2) \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & x(m) \end{bmatrix}, \quad a = \begin{bmatrix} 1 \\ a(2) \\ \vdots \\ a(p+1) \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Meta Platforms, Inc. - Exhibit 1009
Page 535 of 720

and `m` is the length of `x`. Solving the least-squares problem via the normal equations

$$X^H X a = X^H b$$

leads to the Yule-Walker equations

$$\begin{bmatrix} r(1) & r(2)^* & \cdots & r(p)^* \\ r(2) & r(1) & \ddots & \vdots \\ \vdots & \ddots & \ddots & r(2)^* \\ r(p) & \cdots & r(2) & r(1) \end{bmatrix} \begin{bmatrix} a(2) \\ a(3) \\ \vdots \\ a(p+1) \end{bmatrix} = \begin{bmatrix} -r(2) \\ -r(3) \\ \vdots \\ -r(p+1) \end{bmatrix}$$

where $r = [r(1)\ r(2) \ldots r(p+1)]$ is an autocorrelation estimate for `x` computed using `xcorr`. The Yule-Walker equations are solved in $O(p^2)$ flops by the Levinson-Durbin algorithm (see `levinson`).

See Also

`aryule`	Compute an estimate of AR model parameters using the Yule-Walker method.
`levinson`	Levinson-Durbin recursion.
`prony`	Prony's method for time domain IIR filter design.
`pyulear`	Power spectrum estimate using Yule-Walker AR method.
`stmcb`	Linear model using Steiglitz-McBride iteration.

References

[1] Jackson, L.B. *Digital Filters and Signal Processing*. Second Ed. Boston: Kluwer Academic Publishers, 1989. Pgs. 255-257.

Meta Platforms, Inc. - Exhibit 1009
Page 536 of 720

Purpose

Generalized digital Butterworth filter design.

Syntax

```
[b,a,] = maxflat(nb,na,Wn)
b = maxflat(nb,'sym',Wn)
[b,a,b1,b2] = maxflat(nb,na,Wn)
[...] = maxflat(nb,na,Wn,'design_flag')
```

Description

`[b,a,] = maxflat(nb,na,Wn)` is a lowpass Butterworth filter with numerator and denominator coefficients `b` and `a` of orders `nb` and `na` respectively. `Wn` is the cutoff frequency at which the magnitude response of the filter is equal to $1/\sqrt{2}$ (approx. -3 dB). `Wn` must be between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency).

`b = maxflat(nb,'sym',Wn)` is a symmetric FIR Butterworth filter. `nb` must be even, and `Wn` is restricted to a subinterval of [0,1]. The function raises an error if `Wn` is specified outside of this subinterval.

`[b,a,b1,b2] = maxflat(nb,na,Wn)` returns two polynomials `b1` and `b2` whose product is equal to the numerator polynomial `b` (that is, `b = conv(b1,b2)`). `b1` contains all the zeros at `z = –1`, and `b2` contains all the other zeros.

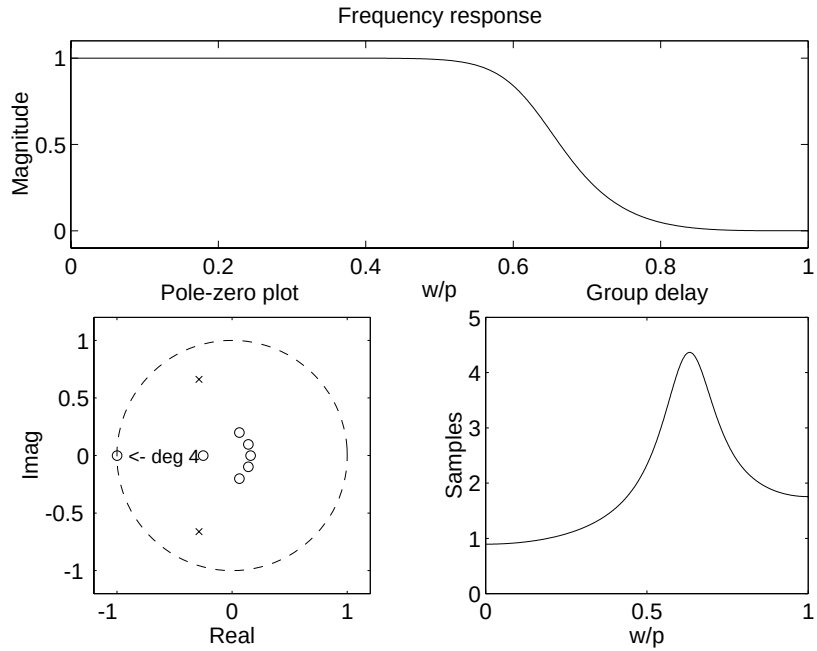
`[...] = maxflat(nb,na,Wn,'design_flag')` enables you to monitor the filter design, where *`design_flag`* is

- `trace`, for a textual display of the design table used in the design
- `plots`, for plots of the filter's magnitude, group delay, and zeros and poles
- `both`, for both the textual display and plots

Meta Platforms, Inc. - Exhibit 1009
Page 537 of 720

Examples

```
nb = 10; na = 2; Wn = 0.2*pi;
[b,a,b1,b2] = maxflat(nb,na,Wn,'plots')
```



Algorithm

The method consists of the use of formulae, polynomial root finding, and a transformation of polynomial roots.

See Also

`butter`	Butterworth analog and digital filter design.
`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
`freqz`	Frequency response of digital filters.

References

[1] Selesnick, I.W., and C.S. Burrus. “Generalized Digital Butterworth Filter Design.” *Proceedings of the IEEE Int. Conf. Acoust., Speech, Signal Processing*. Vol. 3 (May 1996).

Meta Platforms, Inc. - Exhibit 1009
Page 538 of 720

Purpose

One-dimensional median filtering.

Syntax

```
y = medfilt1(x,n)
y = medfilt1(x,n,blksz)
```

Description

`y = medfilt1(x,n)` applies an order `n`, one-dimensional median filter to vector `x`. `y` is the same length as `x`; the function treats the signal as if it is 0 beyond the end points.

For `n` odd, `y(k)` is the median of `x(k–(n–1)/2:k+(n–1)/2)`.

For `n` even, `y(k)` is the median of `x(k–n/2)`, `x(k–(n/2)+1)`, ..., `x(k+(n/2)–1)`. In this case, `medfilt1` sorts the numbers, then takes the average of the `(n–1)/2` and `((n–1)/2)+1` elements.

The default for `n` is 3.

`y = medfilt1(x,n,blksz)` uses a `for`-loop to compute `blksz` (block size) output samples at a time. Use `blksz << length(x)` if you are low on memory, since `medfilt1` uses a working matrix of size `n`-by-`blksz`. By default, `blksz = length(x)`; this is the fastest execution if you have sufficient memory.

If `x` is a matrix, `medfilt1` median filters its columns using

```
y(:,i) = medfilt1(x(:,i),n,blksz)
```

in a loop over the columns of `x`.

See Also

`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
`medfilt2`	Two-dimensional median filtering (see *Image Processing Toolbox User's Guide*).
`median`	Median value (see the online *MATLAB Function Reference*).

References

[1] Pratt, W.K. *Digital Image Processing*. New York: John Wiley & Sons, 1978. Pgs. 330-333.

Meta Platforms, Inc. - Exhibit 1009
Page 539 of 720

Purpose

Modulation for communications simulation.

Syntax

```
y = modulate(x,Fc,Fs,'method')
y = modulate(x,Fc,Fs,'method',opt)
[y,t] = modulate(x,Fc,Fs)
```

Description

`y = modulate(x,Fc,Fs,'method')` and

`y = modulate(x,Fc,Fs,'method',opt)` modulate the real message signal `x` with a carrier frequency `Fc` and sampling frequency `Fs`, using one of the options listed below for *`method`*. Note that some methods accept an option, `opt`.

`amdsb-sc` or `am` **Amplitude modulation, double sideband, suppressed carrier.** Multiplies `x` by a sinusoid of frequency `Fc`:

```
y = x.*cos(2*pi*Fc*t)
```

`amdsb-tc` **Amplitude modulation, double sideband, transmitted carrier.** Subtracts scalar `opt` from `x` and multiplies the result by a sinusoid of frequency `Fc`:

```
y = (x-opt).*cos(2*pi*Fc*t)
```

If the `opt` parameter is not present, `modulate` uses a default of `min(min(x))` so that the message signal `(x-opt)` is entirely non-negative and has a minimum value of 0.

`amssb` **Amplitude modulation, single sideband.** Multiplies `x` by a sinusoid of frequency `Fc` and adds the result to the Hilbert transform of `x` multiplied by a phase shifted sinusoid of frequency `Fc`:

```
y =
x.*cos(2*pi*Fc*t)+imag(hilbert(x)).*sin(2*pi*Fc*t)
```

This effectively removes the upper sideband.

Meta Platforms, Inc. - Exhibit 1009
Page 540 of 720

`fm` **Frequency modulation.** Creates a sinusoid with instantaneous frequency that varies with the message signal `x`:

```
y = cos(2*pi*Fc*t + opt*cumsum(x))
```

`cumsum` is a rectangular approximation to the integral of `x`. `modulate` uses `opt` as the constant of frequency modulation. If `opt` is not present, `modulate` uses a default of

```
opt = (Fc/Fs)*2*pi/(max(max(x)))
```

so the maximum frequency excursion from `Fc` is `Fc` Hz.

`pm` **Phase modulation.** Creates a sinusoid of frequency `Fc` whose phase varies with the message signal `x`:

```
y = cos(2*pi*Fc*t + opt*x)
```

`modulate` uses `opt` as the constant of phase modulation. If `opt` is not present, `modulate` uses a default of

```
opt = pi/(max(max(x)))
```

so the maximum phase excursion is π radians.

`pwm` **Pulse-width modulation.** Creates a pulse-width modulated signal from the pulse widths in `x`. The elements of `x` must be between 0 and 1, specifying the width of each pulse in fractions of a period. The pulses start at the beginning of each period, that is, they are left justified.

```
modulate(x,Fc,Fs,'pwm','centered')
```

yields pulses centered at the beginning of each period. `y` is length `length(x)*Fs/Fc`.

`ptm` **Pulse time modulation.** Creates a pulse time modulated signal from the pulse times in `x`. The elements of `x` must be between 0 and 1, specifying the left edge of each pulse in fractions of a period. `opt` is a scalar between 0 and 1 that specifies the length of each pulse in fractions of a period. The default for `opt` is 0.1. `y` is length `length(x)*Fs/Fc`.

`qam` **Quadrature amplitude modulation.** Creates a quadrature amplitude modulated signal from signals `x` and `opt`:

```
y = x.*cos(2*pi*Fc*t) + opt.*sin(2*pi*Fc*t)
```

`opt` must be the same size as `x`.

Meta Platforms, Inc. - Exhibit 1009
Page 541 of 720

If you do not specify *method*, then `modulate` assumes `am`. Except for the `pwm` and `ptm` cases, `y` is the same size as `x`.

If `x` is an array, `modulate` modulates its columns.

`[y,t] = modulate(x,Fc,Fs)` returns the internal time vector `t` that `modulate` uses in its computations.

See Also

`demod`	Demodulation for communications simulation.
`vco`	Voltage controlled oscillator.

Meta Platforms, Inc. - Exhibit 1009
Page 542 of 720

Purpose

Power spectrum estimate using the Burg method.

Syntax

```
Pxx = pburg(x,p,nfft)
[Pxx,freq] = pburg(x,p,nfft)
[Pxx,freq] = pburg(x,p,nfft,Fs)
[Pxx,freq] = pburg(x,p,nfft,Fs,'range')
pburg(...)
pburg(...,'squared')
```

Description

`pburg` estimates the power spectral density (PSD) of the signal vector $x[n]$ using the Burg method. This method fits an autoregressive (AR) model to the signal by minimizing (least squares) the forward and backward prediction errors while constraining the AR parameters to satisfy the Levinson-Durbin recursion. The spectral estimate returned by `pburg` is the magnitude squared frequency response of this AR model. The correct choice of the model order `p` is important.

`Pxx = pburg(x,p,nfft)` returns `Pxx`, the power spectrum estimate. `x` is the input signal, `p` is the model order for the all-pole filter, and `nfft` is the FFT length (defaults to 256 if not specified). `Pxx` has length `(nfft/2+1)` for `nfft` even, `(nfft+1)/2` for `nfft` odd, and `nfft` if `x` is complex.

`[Pxx,freq] = pburg(x,p,nfft)` returns `Pxx`, the power spectrum estimate, and `freq`, a vector of frequencies at which the PSD was estimated. If the input signal is real-valued, the range of `freq` is $[0,\pi)$. If the input signal is complex, the range of `freq` is $[0,2\pi)$.

`[Pxx,freq] = pburg(x,p,nfft,Fs)` uses the signal's sampling frequency, `Fs`, to scale both the PSD vector (`Pxx`) and the frequency vector (`freq`). `Pxx` is scaled by `1/Fs`. If the input signal is real-valued, the range of `freq` is `[0,Fs/2)`. If the input signal is complex, the range of `freq` is `[0,Fs)`. `Fs` defaults to `1` if left empty, `[]`.

`[Pxx,freq] = pburg(x,p,nfft,Fs,'range')` specifies the range of frequency values to include in `freq`. *range* can be:

- `half`, to compute the PSD over the range `[0,Fs/2)` for real `x`, and `[0,Fs)` for complex `x`. If `Fs` is left blank, `[]`, the range is `[0,1/2)` for real `x`, and `[0,1)`

Meta Platforms, Inc. - Exhibit 1009
Page 543 of 720

for complex x. If Fs is omitted entirely, the range is [0,pi) for real x, and [0,2*pi) for complex x. half is the default range.

- whole, to compute the PSD over the range [0,Fs) for all x. If Fs is left blank, [], the range is [0,1) for all x. If Fs is omitted entirely, the range is [0,2*pi) for all x.

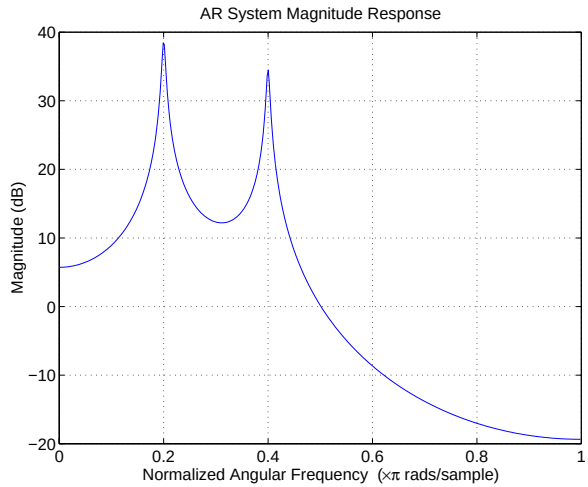
pburg(...) plots the power spectral density in the current figure window. The frequency range on the plot is the same as the range of output freq for a given set of parameters.

pburg(...,'squared') plots the magnitude of Pxx directly, rather than converting the values to dB.

Example

Because the method estimates the spectrum by fitting an AR model to the signal, first define the AR system (all-pole filter) that generates the input. Check the magnitude response of the process with freqz.

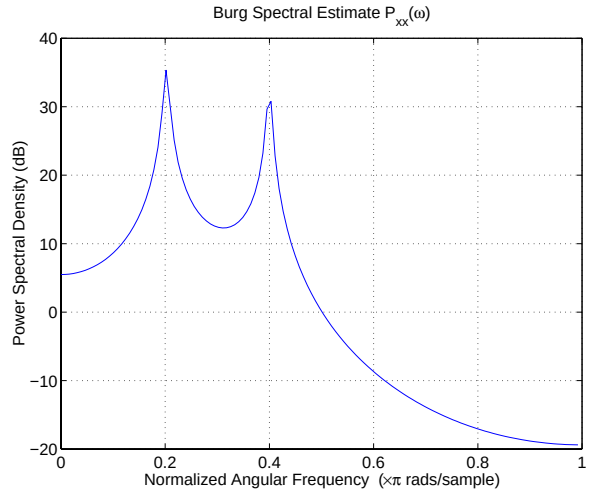
```
a = [1 -2.2137 2.9403 -2.1697 0.9606];  % AR system coefficients
freqz(1,a,'phase','no')  % AR system magnitude response
title('AR System Magnitude Response')
```



Meta Platforms, Inc. - Exhibit 1009
Page 544 of 720

Now generate the input signal x by filtering white noise through the AR system. Estimate the PSD of x based on a fourth-order AR model (since we know that the original AR system, a, has order 4).

```
x = filter(1,a,randn(150,1));     % AR system output
pburg(x,4)                        % 4th order estimate
```



Diagnostics

The first input argument must be a full vector; otherwise pburg generates the following error message:

```
Input signal cannot be sparse.
```

If you specify an empty matrix for the second argument, pburg generates the following error message:

```
Model order must be an integer.
```


Meta Platforms, Inc. - Exhibit 1009
Page 545 of 720

See Also

`arburg`	Compute an estimate of AR model parameters using the Burg method.
`lpc`	Linear prediction coefficients.
`pcov`	Power spectrum estimate using the covariance method.
`pmcov`	Power spectrum estimate using the modified covariance method.
`pmtm`	Power spectrum estimate using the multitaper method (MTM).
`pmusic`	Power spectrum estimate using MUSIC eigenvector method.
`prony`	Prony's method for time domain IIR filter design.
`pwelch`	Estimate the power spectral density (PSD) of a signal using Welch's method.
`pyulear`	Power spectrum estimate using Yule-Walker AR method.

References

[1] Marple, S.L. *Digital Spectral Analysis*. Englewood Cliffs, NJ: Prentice Hall, 1987. Chapter 7.

Meta Platforms, Inc. - Exhibit 1009
Page 546 of 720

Purpose

Power spectrum estimate using the covariance method.

Syntax

```
Pxx = pcov(x,p,nfft)
[Pxx,freq] = pcov(x,p,nfft)
[Pxx,freq] = pcov(x,p,nfft,Fs)
[Pxx,freq] = pcov(x,p,nfft,Fs,'range')
pcov(...)
pcov(...,'squared')
```

Description

`pcov` estimates the power spectral density (PSD) of the signal vector $x[n]$ 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 spectral estimate returned by `pcov` is the magnitude squared frequency response of this AR model. The correct choice of the model order `p` is important.

`Pxx = pcov(x,p,nfft)` returns `Pxx`, the power spectrum estimate. `x` is the input signal, `p` is the model order for the all-pole filter, and `nfft` is the FFT length (defaults to 256 if not specified). `Pxx` has length `(nfft/2+1)` for `nfft` even, `(nfft+1)/2` for `nfft` odd, and `nfft` if `x` is complex.

`[Pxx,freq] = pcov(x,p,nfft)` returns `Pxx`, the power spectrum estimate, and `freq`, a vector of frequencies at which the PSD was estimated. If the input signal is real-valued, the range of `freq` is $[0,\pi)$. If the input signal is complex, the range of `freq` is $[0,2\pi)$.

`[Pxx,freq] = pcov(x,p,nfft,Fs)` uses the signal's sampling frequency, `Fs`, to scale both the PSD vector (`Pxx`) and the frequency vector (`freq`). `Pxx` is scaled by `1/Fs`. If the input signal is real-valued, the range of `freq` is `[0,Fs/2)`. If the input signal is complex, the range of `freq` is `[0,Fs)`. `Fs` defaults to `1` if left empty, `[]`.

`[Pxx,freq] = pcov(x,p,nfft,Fs,'range')` specifies the range of frequency values to include in `freq`. *range* can be:

- `half`, to compute the PSD over the range `[0,Fs/2)` for real `x`, and `[0,Fs)` for complex `x`. If `Fs` is left blank, `[]`, the range is `[0,1/2)` for real `x`, and `[0,1)`

Meta Platforms, Inc. - Exhibit 1009
Page 547 of 720

for complex x. If Fs is omitted entirely, the range is [0,pi) for real x, and [0,2*pi) for complex x. half is the default range.

- whole, to compute the PSD over the range [0,Fs) for all x. If Fs is left blank, [], the range is [0,1) for all x. If Fs is omitted entirely, the range is [0,2*pi) for all x.

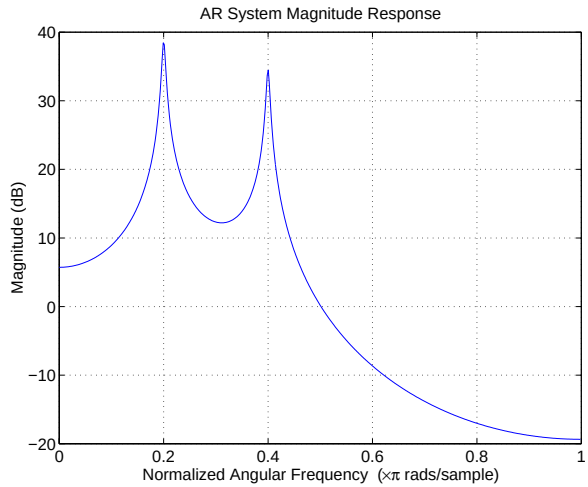
pcov(...) plots the power spectral density in the first available figure window. The frequency range on the plot is the same as the range of output freq for a given set of parameters.

pcov(...,'squared') plots the magnitude of Pxx directly, rather than converting the values to dB.

Example

Because the method estimates the spectrum by fitting an AR model to the signal, first define the AR system (all-pole filter) that generates the input. Check the magnitude response of the process with freqz.

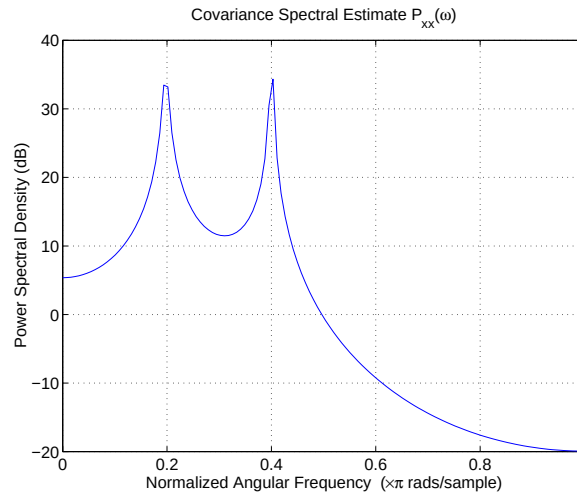
```
a = [1 -2.2137 2.9403 -2.1697 0.9606];  % AR system coefficients
freqz(1,a,'phase','no')  % AR system magnitude response
title('AR System Magnitude Response')
```



Meta Platforms, Inc. - Exhibit 1009
Page 548 of 720

Now generate the input signal x by filtering white noise through the AR system. Estimate the PSD of x based on a fourth-order AR model (since we know that the original AR system, a, has order 4).

```
x = filter(1,a,randn(150,1));     % AR system output
pcov(x,4)                         % 4th order estimate
```



Diagnostics

The first input argument must be a full vector; otherwise pcov generates the following error message:

```
Input signal cannot be sparse.
```

If you specify an empty matrix for the second argument, pcov generates the following error message:

```
Model order must be an integer.
```


Meta Platforms, Inc. - Exhibit 1009
Page 549 of 720

See Also

`arcov`	Compute an estimate of AR model parameters using the covariance method.
`lpc`	Linear prediction coefficients.
`pburg`	Power spectrum estimate using the Burg method.
`pmcov`	Power spectrum estimate using the modified covariance method.
`pmtm`	Power spectrum estimate using the multitaper method (MTM).
`pmusic`	Power spectrum estimate using MUSIC eigenvector method.
`prony`	Prony's method for time domain IIR filter design.
`pwelch`	Estimate the power spectral density (PSD) of a signal using Welch's method.
`pyulear`	Power spectrum estimate using Yule-Walker AR method.

References

[1] Marple, S.L. *Digital Spectral Analysis*. Englewood Cliffs, NJ: Prentice Hall, 1987. Chapter 7.

Meta Platforms, Inc. - Exhibit 1009
Page 550 of 720

Purpose

Power spectrum estimate using the modified covariance method.

Syntax

```
Pxx = pmcov(x,p,nfft)
[Pxx,freq] = pmcov(x,p,nfft)
[Pxx,freq] = pmcov(x,p,nfft,Fs)
[Pxx,freq] = pmcov(x,p,nfft,Fs,'range')
pmcov(...)
pmcov(...,'squared')
```

Description

`pmcov` estimates the power spectral density (PSD) of the signal vector $x[n]$ using the modified covariance method. This method fits an autoregressive (AR) model to the signal by minimizing the forward and backward prediction errors in the least-squares sense. The spectral estimate returned by `pmcov` is the magnitude squared frequency response of this AR model. The correct choice of the model order `p` is important.

`Pxx = pmcov(x,p,nfft)` returns `Pxx`, the power spectrum estimate. `x` is the input signal, `p` is the model order for the all-pole filter, and `nfft` is the FFT length (defaults to 256 if not specified). `Pxx` has length `(nfft/2+1)` for `nfft` even, `(nfft+1)/2` for `nfft` odd, and `nfft` if `x` is complex.

`[Pxx,freq] = pmcov(x,p,nfft)` returns `Pxx`, the power spectrum estimate, and `freq`, a vector of frequencies at which the PSD was estimated. If the input signal is real-valued, the range of `freq` is $[0,\pi)$. If the input signal is complex, the range of `freq` is $[0,2\pi)$.

`[Pxx,freq] = pmcov(x,p,nfft,Fs)` uses the signal's sampling frequency, `Fs`, to scale both the PSD vector (`Pxx`) and the frequency vector (`freq`). `Pxx` is scaled by `1/Fs`. If the input signal is real-valued, the range of `freq` is `[0,Fs/2)`. If the input signal is complex, the range of `freq` is `[0,Fs)`. `Fs` defaults to `1` if left empty, `[]`.

`[Pxx,freq] = pmcov(x,p,nfft,Fs,'range')` specifies the range of frequency values to include in `freq`. *range* can be:

- `half`, to compute the PSD over the range `[0,Fs/2)` for real `x`, and `[0,Fs)` for complex `x`. If `Fs` is left blank, `[]`, the range is `[0,1/2)` for real `x`, and `[0,1)`

Meta Platforms, Inc. - Exhibit 1009
Page 551 of 720

for complex x. If Fs is omitted entirely, the range is [0,pi) for real x, and [0,2*pi) for complex x. half is the default range.

- whole, to compute the PSD over the range [0,Fs) for all x. If Fs is left blank, [], the range is [0,1) for all x. If Fs is omitted entirely, the range is [0,2*pi) for all x.

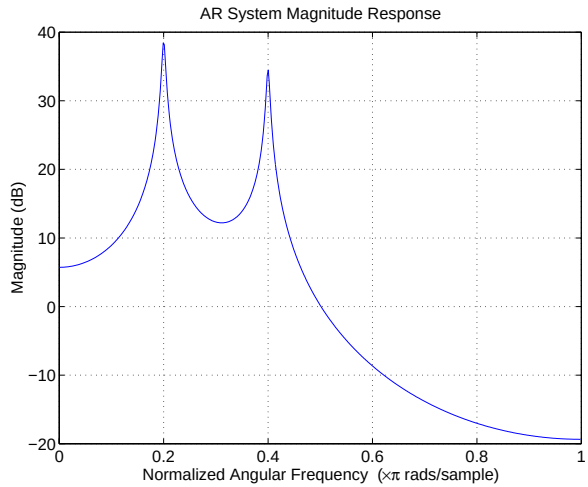
pmcov(...) plots the power spectral density in the first available figure window. The frequency range on the plot is the same as the range of output freq for a given set of parameters.

pmcov(...,'squared') plots the magnitude of Pxx directly, rather than converting the values to dB.

Example

Because the method estimates the spectrum by fitting an AR model to the signal, first create the AR system (all-pole filter) that generates the input. Check the magnitude response of the process with freqz.

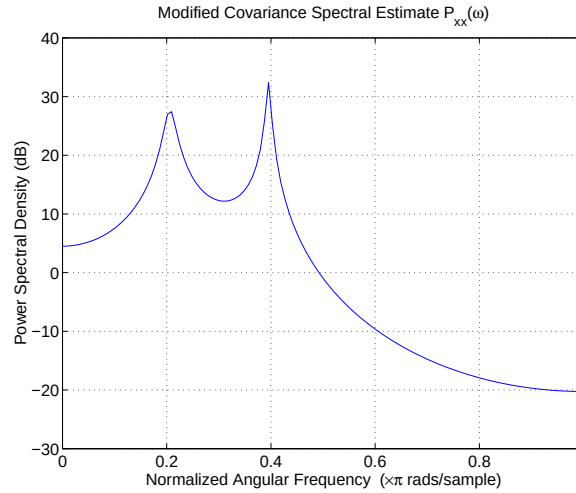
```
a = [1 -2.2137 2.9403 -2.1697 0.9606];  % AR system coefficients
freqz(1,a,'phase','no')  % AR system magnitude response
title('AR System Magnitude Response')
```



Meta Platforms, Inc. - Exhibit 1009
Page 552 of 720

Now generate the input signal x by filtering white noise through the AR system. Estimate the PSD of x based on a fourth-order AR model (since we know that the original AR system, a, has order 4).

```
x = filter(1,a,randn(150,1));     % AR system output
pmcov(x,4)                        % 4th order estimate
```



Diagnostics

The first input argument must be a full vector; otherwise pmcov generates the following error message:

```
Input signal cannot be sparse.
```

If you specify an empty matrix for the second argument, pmcov generates the following error message:

```
Model order must be an integer.
```


Meta Platforms, Inc. - Exhibit 1009
Page 553 of 720

See Also

`armcov`	Compute an estimate of AR model parameters using the modified covariance method.
`lpc`	Linear prediction coefficients.
`pburg`	Power spectrum estimate using the Burg method.
`pcov`	Power spectrum estimate using the covariance method.
`pmtm`	Power spectrum estimate using the multitaper method (MTM).
`pmusic`	Power spectrum estimate using MUSIC eigenvector method.
`prony`	Prony's method for time domain IIR filter design.
`pwelch`	Estimate the power spectral density (PSD) of a signal using Welch's method.
`pyulear`	Power spectrum estimate using Yule-Walker AR method.

References

[1] Marple, S.L. *Digital Spectral Analysis*. Englewood Cliffs, NJ: Prentice Hall, 1987. Chapter 7.

Meta Platforms, Inc. - Exhibit 1009
Page 554 of 720

Purpose

Power spectrum estimate using the multitaper method (MTM).

Syntax

```
Pxx = pmtm(x,nw)
Pxx = pmtm(x,nw,nfft)
[Pxx,f] = pmtm(x,nw,nfft,Fs)
[Pxx,f] = pmtm(x,nw,nfft,Fs,'method')
[Pxx,Pxxc,f] = pmtm(x,nw,nfft,Fs,'method')
[Pxx,Pxxc,f] = pmtm(x,nw,nfft,Fs,'method',p)
[Pxx,Pxxc,f] = pmtm(x,e,v,nfft,Fs,'method',p)
[Pxx,Pxxc,f] = pmtm(x,dpss_params,nfft,Fs,'method',p)
```

Description

`pmtm` estimates the power spectral density (PSD) of the real time series *x* using the multitaper method (MTM), described in [1].

`Pxx = pmtm(x,nw)` estimates the PSD using `nw` as the time-bandwidth product for the discrete prolate spheroidal sequences (Slepian sequences) that are used as data windows. The default for `nw` is 4; other typical choices are 2, 5/2, 3, 7/2. The number of sequences used to form `Pxx` is `2*nw–1`.

`Pxx = pmtm(x,nw,nfft)` defines the frequency grid as length `nfft`. When `x` is real, `Pxx` is length `(nfft/2+1)` for `nfft` even and `(nfft+1)/2` for `nfft` odd; when `x` is complex, `Pxx` is length `nfft`. The default for `nfft` is 256 or the next power of 2 greater than the length of `x`, whichever is larger.

`[Pxx,f] = pmtm(x,nw,nfft,Fs)` returns `f`, the vector of frequencies at which the PSD is estimated, for the sampling frequency `Fs`. The default for `Fs` is 1 Hz.

`[Pxx,f] = pmtm(x,nw,nfft,Fs,'method')` specifies the algorithm used for combining the individual spectral estimates, where method is:

- `adapt`, to specify Thomson's adaptive nonlinear combination (default)
- `unity`, to specify a linear combination with unity weights
- `eigen`, to specify a linear combination with eigenvalue weights

`[Pxx,Pxxc,f] = pmtm(x,nw,nfft,Fs,'method')` returns `Pxxc`, the 95% confidence interval for `Pxx`, and

`[Pxx,Pxxc,f] = pmtm(x,nw,nfft,Fs,'method',p)` returns `Pxxc`, the `p*100%` confidence interval for `Pxx`, where `p` is a scalar between 0 and 1. Confidence

Meta Platforms, Inc. - Exhibit 1009
Page 555 of 720

intervals are computed using a chi-squared approach, where `Pxxc(:,1)` is the lower bound and `Pxxc(:,2)` is the upper bound of the confidence interval.

`[Pxx,Pxxc,f] = pmtm(x,e,v,nfft,Fs,'method',p)` returns the PSD estimate `Pxx`, the confidence interval `Pxxc`, and the frequency vector `f` from the data tapers in `e` and their concentrations `v`.

`[Pxx,Pxxc,f] = pmtm(x,dpss_params,nfft,Fs,'method',p)` returns the PSD estimate `Pxx`, the confidence interval `Pxxc`, and the frequency vector `f` from the data tapers computed using `dpss` with parameters from the cell array `dpss_params`, whose first element is the *second* input to `dpss`. The first `dpss` parameter (`n`) is determined by the length of `x`. For example, `pmtm(x,{3.5,'trace'},512,Fs)` calculates the Slepian sequences for `nw = 3.5`, and displays the method that `dpss` uses. See `dpss` for other options.

Remarks

`pmtm` with no output arguments plots the PSD in the current or next available figure, with confidence intervals.

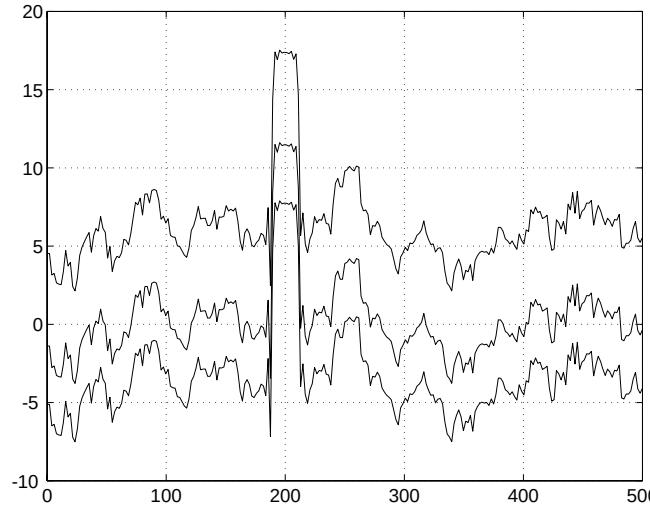
To use default parameters for any argument in an expression, insert an empty matrix `[]`. For example, `pmtm(x,[],[],1000)` uses defaults for the second and third elements, in this case, `nw` and `nfft`.

Meta Platforms, Inc. - Exhibit 1009
Page 556 of 720

Example

This example analyzes a sinusoid in white noise:

```
Fs = 1000; t = 0:1/Fs:0.3;
x = cos(2*pi*t*200) + randn(size(t));
[Pxx,Pxxc,f] = pmtm(x,3.5,512,Fs,[],0.99);
plot(f,10*log10([Pxx Pxxc]))
```



See Also

`dpss`	Discrete prolate spheroidal sequences (Slepian sequences).
`pburg`	Power spectrum estimate using the Burg method.
`pcov`	Power spectrum estimate using the covariance method.
`pmcov`	Power spectrum estimate using the modified covariance method.
`pmusic`	Power spectrum estimate using MUSIC eigenvector method.
`pwelch`	Estimate the power spectral density (PSD) of a signal using Welch's method.
`pyulear`	Power spectrum estimate using Yule-Walker AR method.

Meta Platforms, Inc. - Exhibit 1009
Page 557 of 720

References

[1] Percival, D.B., and A.T. Walden. *Spectral Analysis for Physical Applications: Multitaper and Conventional Univariate Techniques*. Cambridge: Cambridge University Press, 1993.

[2] Thomson, D.J. “Spectrum estimation and harmonic analysis.” In *Proceedings of the IEEE*. Vol. 70 (1982). Pgs. 1055-1096.

Meta Platforms, Inc. - Exhibit 1009
Page 558 of 720

Purpose

Power spectrum estimate using MUSIC eigenvector method.

Syntax

```
[Pxx,f] = pmusic(x,p)
[Pxx,f] = pmusic(x,[p thresh])
[Pxx,f] = pmusic(x,[p thresh],nfft,Fs,window,noverlap)
[Pxx,f] = pmusic(x,...,'corr')
[Pxx,f] = pmusic(x,...,'ev')
[Pxx,f,evects,svals] = pmusic(x,...)
```

Description

`pmusic` estimates the power spectral density (PSD) of a signal or correlation matrix using Schmidt's eigen-analysis method [1]. The name MUSIC is an acronym for MUltiple SIgnal Classification. The *eigenvector method*, which uses eigenvalue weighting, is also supported [2]. The calling syntax is similar to that of `pwelch`, which also performs spectrum estimation. `pwelch` uses the classical FFT-based approach while `pmusic` performs eigen-analysis of the signal's correlation matrix.

`[Pxx,f] = pmusic(x,p)` and

`[Pxx,f] = pmusic(x,[p thresh])` return `Pxx`, the power spectrum estimate, and `f`, a vector of frequencies at which the PSD is estimated. `x` is the input signal, where:

- A row or column vector represents one observation of the process output (for example, one "signal")
- A rectangular (possibly square) array assumes that each column of `x` is a separate observation of the process output (for example, each column is one output of an array of sensors, as in array processing)
- A square matrix, given the trailing argument `'corr'`, represents a correlation matrix

The second argument is a one- or two-element vector, either `p` or `[p thresh]`. If only `p` is specified, the signal subspace dimension is `p`. If `[p thresh]` is specified, `thresh` is multiplied by λ_{min}, the smallest eigenvalue; eigenvalues below the threshold λ_{min}`*thresh` are assigned to the noise subspace. In this case, `p` is the maximum dimension of the signal subspace.

Meta Platforms, Inc. - Exhibit 1009
Page 559 of 720

NOTE `pmusic` must assign eigenvectors to the noise and signal subspaces, but this is very difficult to do in practice. The two parameters `p` and `thresh` are provided for flexibility and control.

`[Pxx,f] = pmusic(x,[p thresh],nfft,Fs,window,noverlap)` specifies the FFT length `nfft` (default is 256) and the sampling frequency for the signal `Fs` (default is 1). If `Fs` is specified, the output frequency vector `f` is scaled by this value. If the input signal is real-valued, the frequency range is 0 to `Fs/2`; for the complex case, it is 0 to `Fs`. `window` is a scalar specifying the rectangular window length, or a vector giving the actual window coefficients. `noverlap`, used in conjunction with `window`, is a scalar that gives the number of points by which to overlap successive windows.

`[Pxx,f] = pmusic(x,...,'corr')` forces `x` to be taken as a correlation matrix. In this case, the arguments `window` and `noverlap` are ignored.

`[Pxx,f] = pmusic(x,...,'ev')` selects the eigenvector variant of the MUSIC estimator. See the “Algorithm” section below for an explanation of how this is different from the MUSIC method.

`[Pxx,f,evects,svals] = pmusic(x,...)` returns two additional arguments. `evects` is a matrix of eigenvectors spanning the noise subspace (one per column). `svals` is either a vector of singular values (squared) from `svd` or a vector of eigenvalues of the correlation matrix when the `'corr'` option is present.

Remarks

The input `x` can be a vector or a matrix. `x` can be interpreted as signal data or as a correlation matrix, in one of three ways:

- `x` is a vector of signal values (row or column). In this case, the dimension of the eigenvectors must be given. This is done either by taking the default value of `2*p` or by specifying a window length using `window`.
- `x` is a rectangular (`m`-by-`n`, possibly square) matrix. In this case, each column of `x` is a separate observation signal that enters into the SVD analysis, `n` is

Meta Platforms, Inc. - Exhibit 1009
Page 560 of 720

the number of observations, and the dimension of the eigenvectors is equal to m, the length of a column.

- x is a square matrix and the trailing 'corr' is present. x is treated as a correlation matrix. In this case, the matrix must have only real, nonnegative eigenvalues.

The inputs p and thresh can determine the number of noise eigenvectors in one of four ways:

- If thresh < 1, or if it is unspecified, the number of eigenvectors spanning the signal subspace will be equal to p. p must be an integer satisfying $0 \le p < n$, where n is the dimension of the eigenvectors. This dimension n is the column length in the data matrix case, the matrix size in the correlation matrix case, or the window length for signal vectors. The value of thresh is unused.
- If $p \ge n$, thresh must be at least 1. thresh is used as the multiplier to determine an absolute threshold for splitting the eigenvalues between the signal and noise subspaces:

$$\lambda_k \le (\texttt{thresh})\min(\lambda_k) \qquad \Rightarrow \{\lambda_k, v_k\} \quad \text{belong to noise subspace}$$

- If thresh < 1, there will be no noise eigenvectors. This case is not allowed and gives the following error message:

```
Noise subspace dimension cannot be zero.
```

- When p < n and thresh ≥ 1, p specifies the maximum number of signal eigenvectors. However, the threshold test specified by thresh can also take eigenvectors from the signal subspace and assign them to the noise subspace.

Examples

This example analyzes a signal vector xx, assuming that two real signals are present in the signal subspace. In this case, the dimension of the signal subspace is 4 because each real sinusoid is the sum of two complex exponentials:

```
nn = 0:199;
xx = cos(0.257*pi*nn) + sin(0.2*pi*nn) + 0.01*randn(size(nn));
[PP,ff] = pmusic(xx,4);
```

This example analyzes the same signal vector xx with an eigenvalue cutoff of 10% above the minimum. Setting p = Inf forces the signal/noise subspace

Meta Platforms, Inc. - Exhibit 1009
Page 561 of 720

decision to be based on `thresh`. Use eigenvectors of dimension 7 and a sampling frequency `Fs` of 8 kHz:

```
[PP,ff] = pmusic(xx,[Inf,1.1],[],8000,7); % window length = 7
```

With the third and fourth outputs, by plotting the zeros of the noise-eigenvector polynomials, it is possible to create a "Root-MUSIC" algorithm, as the following `zplane` plot illustrates:

```
[PP,ff,v_noise] = pmusic(xx,4);
for kk = 1:size(v_noise,2)
   rr(:,kk) = roots(v_noise(:,kk));
end
zplane(rr)
```

Assume that `RR` is a square correlation matrix (for example, 7-by-7):

```
RR = toeplitz(cos(0.1*pi*[0:6])) + 0.1*eye(7);
[PP,ff] = pmusic(RR,4,'corr');
```

Make an observation matrix `xx` that is rectangular (100-by-7):

```
xx = reshape(cos(0.257*pi*(0:699)),7,100) + 0.1*randn(7,100);
[PP,ff] = pmusic(xx,4);
```

Use the same signal, but let `pmusic` form the 100-by-7 data matrix using its window and overlap inputs. In addition, use a longer FFT:

```
yy = xx(:);
[PP,ff] = pmusic(yy,4,512,[],7,0);
```

If we set `p = 0`, all the eigenvectors are assigned to the noise subspace. `'ev'` specifies the eigenvector weighting. This turns out to be equivalent to MVDL (Capon's MLM):

```
[PP,ff] = pmusic(RR,0,'ev','corr');
```

Algorithm

The MUSIC estimate is given by the formula

$$P_{music}(f) = \frac{1}{\mathbf{e}^H(f)\left(\sum_{k=p+1}^{N} \mathbf{v}_k \mathbf{v}_k^H\right)\mathbf{e}(f)} = \frac{1}{\sum_{k=p+1}^{N} \left|\mathbf{v}_k^H \mathbf{e}(f)\right|^2}$$

Meta Platforms, Inc. - Exhibit 1009
Page 562 of 720

where N is the dimension of the eigenvectors and $\mathbf{v}_k$ is the k-th eigenvector of the correlation matrix of the input signal. The integer p is the dimension of the signal subspace, so the eigenvectors $\mathbf{v}_k$ used in the sum correspond to the smallest eigenvalues and also span the noise subspace. The vector $\mathbf{e}(f)$ consists of complex exponentials, so the inner product

$$\mathbf{v}_k^H \mathbf{e}(f)$$

amounts to a Fourier transform. The second form is preferred for computation because the FFT is computed for each $\mathbf{v}_k$ and then the squared magnitudes are summed.

In the eigenvector method, the summation is weighted by the eigenvalues λ_k of the correlation matrix:

$$P_{ev}(f) = \frac{1}{\left(\sum_{k=p+1}^{N} \left| \mathbf{v}_k^H \mathbf{e}(f) \right|^2 \right) / \lambda_k}$$

The function relies on the `svd` matrix decomposition in the signal case, and it uses the `eig` function for analyzing the correlation matrix. If SVD is used, the correlation matrix is never explicitly computed, but the singular values are the λ_k.

Diagnostics

There must be at least one output argument and at least two inputs; otherwise, `pmusic` stops and gives one of the following error messages:

```
Must have at least 1 output argument.
Must have at least 2 input arguments.
```

The first argument must be a full matrix, otherwise `pmusic` gives the following error message:

```
Input signal or correlation cannot be sparse.
```


Meta Platforms, Inc. - Exhibit 1009
Page 563 of 720

If the second argument was entered as an empty matrix, or if it has more than two elements, or if it has negative or non-integer elements, pmusic gives one of the following error messages:

```
P cannot be empty.
Second input must have only 1 or 2 elements.
P must be an integer.
Second input must contain non-negative entries.
```

If the value of p is too large with respect to the eigenvector dimension, and thresh is less than 1, no eigenvectors can be assigned to the noise subspace and the algorithm fails. In this case, pmusic gives the following error message:

```
Noise subspace dimension cannot be zero.
```

If the 'corr' parameter is used, then the first input must be a square correlation matrix. If it is not, pmusic gives the following error message:

```
Correlation matrix (R) is not square.
```

The correlation matrix is then checked for validity; if it fails, pmusic gives the following error message:

```
Correlation matrix (R) has negative or complex eigenvalue.
```

See Also

lpc	Linear prediction coefficients.
pburg	Power spectrum estimate using the Burg method.
pcov	Power spectrum estimate using the covariance method.
pmcov	Power spectrum estimate using the modified covariance method.
pmtm	Power spectrum estimate using the multitaper method (MTM).
prony	Prony's method for time domain IIR filter design.
pwelch	Estimate the power spectral density (PSD) of a signal using Welch's method.
pyulear	Power spectrum estimate using Yule-Walker AR method.

Meta Platforms, Inc. - Exhibit 1009
Page 564 of 720

References

[1] Schmidt, R.O. “Multiple Emitter Location and Signal Parameter Estimation.” *IEEE Trans. Antennas Propagation*. Vol. AP-34 (March 1986). Pgs. 276-280.

[2] Marple, S.L. *Digital Spectral Analysis*. Englewood Cliffs, NJ: Prentice Hall, 1987. Pgs. 373-378.

Meta Platforms, Inc. - Exhibit 1009
Page 565 of 720

Purpose Conversion of prediction polynomial to autocorrelation sequence.

Syntax

```
r = poly2ac(a,efinal)
```

Description `r = poly2ac(a,efinal)` finds the autocorrelation sequence corresponding to prediction polynomial `a` and final prediction error `efinal`. If `a(1)` is not equal to `1`, `poly2ac` normalizes the prediction polynomial by `a(1)`. `a(1)` cannot be 0.

See Also

`ac2poly`	Conversion of autocorrelation sequence to prediction polynomial.
`poly2rc`	Conversion of prediction polynomial to reflection coefficients.
`rc2ac`	Conversion of reflection coefficients to autocorrelation sequence.

References

[1] Kay, S.M. *Modern Spectral Estimation*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

Meta Platforms, Inc. - Exhibit 1009
Page 566 of 720

Purpose

Conversion of prediction polynomial to reflection coefficients.

Syntax

```
k = poly2rc(a)
[k,r0] = poly2rc(a,efinal)
```

Description

`k = poly2rc(a)` finds the reflection coefficients of the AR lattice structure corresponding to prediction polynomial `a`. `a` must be real, and `a(1)` cannot be 0. If a(1) is not equal to `1`, `poly2rc` normalizes the prediction polynomial by `a(1)`. `k` is a row vector of size `length(a)-1`.

`[k,r0] = poly2rc(a,efinal)` returns the zero-lag autocorrelation, `r0`, based on the final prediction error, `efinal`.

A simple, fast way to check if `a` has all of its roots inside the unit circle is to check if each of the elements of `k` has magnitude less than 1:

```
stable = all(abs(poly2rc(a))<1)
```

Example

Consider an IIR filter given by

```
a = [1.0000   0.6149   0.9899   0.0000   0.0031  -0.0082];
```

Its reflection coefficient representation is

```
k = poly2rc(a)

k =
    0.3090
    0.9801
    0.0031
    0.0081
   -0.0082
```

Limitations

If `abs(k(i)) == 1` for any `i`, finding the reflection coefficients is an ill-conditioned problem. `poly2rc` will return some `NaN`s and provide a warning message in this case.

Algorithm

`poly2rc` implements the recursive relationship:

Meta Platforms, Inc. - Exhibit 1009
Page 567 of 720

$$k(n) = a_n(n)$$

$$a_{n-1}(m) = \frac{a_n(m) - k(n)a_n(n-m)}{1 - k(n)^2}, \qquad m = 1, 2, \ldots, n-1$$

This relationship is based on Levinson's recursion [1]. To implement it, `poly2rc` loops through `a` in reverse order after discarding its first element. For each loop iteration `i`, the function

1. sets `k(i)` equal to `a(i)`
2. applies the second relationship above to elements 1 through `i` of the vector `a`:

```
a = (a-k(i)*fliplr(a))/(1-k(i)^2);
```

See Also

`ac2rc`	Conversion of autocorrelation sequence to reflection coefficients.
`latc2tf`	Lattice filter to transfer function conversion.
`latcfilt`	Lattice and lattice-ladder filter implementation.
`poly2ac`	Conversion of prediction polynomial to autocorrelation sequence.
`rc2poly`	Conversion of reflection coefficients to prediction polynomial.
`tf2latc`	Conversion of transfer function to lattice filter.

References

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

Meta Platforms, Inc. - Exhibit 1009
Page 568 of 720

Purpose

Stabilize polynomial.

Syntax

```
b = polystab(a)
```

Description

`polystab` stabilizes a polynomial with respect to the unit circle; it reflects roots with magnitudes greater than 1 inside the unit circle.

`b = polystab(a)` where `a` is a vector of polynomial coefficients normally in the z-domain:

$$a(z) = a(1) + a(2)z^{-1} + \cdots + a(na + 1)z^{-na}$$

returns a row vector containing the stabilized polynomial.

Example

`polystab` can convert a linear-phase filter into a minimum-phase filter with the same magnitude response:

```
h = fir1(25,0.4);
hmin = polystab(h)*norm(h)/norm(polystab(h));
```

Algorithm

`polystab` finds the roots of the polynomial and maps those roots found outside the unit circle to the inside of the unit circle:

```
v = roots(a);
vs = 0.5*(sign(abs(v)-1)+1);
v = (1-vs).*v + vs./conj(v);
b = a(1)*poly(v);
```

See Also

`roots`	Polynomial roots (see the online *MATLAB Function Reference*).

Meta Platforms, Inc. - Exhibit 1009
Page 569 of 720

Purpose

Prony's method for time domain IIR filter design.

Syntax

```
[b,a] = prony(h,nb,na)
```

Description

Prony's method is an algorithm for finding an IIR filter with a prescribed time domain impulse response. It has applications in filter design, exponential signal modeling, and system identification (parametric modeling).

`[b,a] = prony(h,nb,na)` finds a filter with numerator order `nb`, denominator order `na`, and the time domain impulse response in `h`. `prony` returns the filter coefficients in row vectors `b` and `a`, of length `nb + 1` and `na + 1`, respectively. The filter coefficients are in descending powers of z:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + \cdots + b(nb+1)z^{-nb}}{a(1) + a(2)z^{-1} + \cdots + a(na+1)z^{-na}}$$

Example

Recover the coefficients of a Butterworth filter from its impulse response:

```
[b,a] = butter(4,0.2)

b =
    0.0048    0.0193    0.0289    0.0193    0.0048

a =
    1.0000   -2.3695    2.3140   -1.0547    0.1874

h = filter(b,a,[1 zeros(1,25)]);
[bb,aa] = prony(h,4,4)

bb =
    0.0048    0.0193    0.0289    0.0193    0.0048

ab =
    1.0000   -2.3695    2.3140   -1.0547    0.1874
```

Algorithm

`prony` implements the method described in reference [1]. This method uses a variation of the covariance method of AR modeling to find the denominator coefficients `a` and then finds the numerator coefficients `b` for which the impulse response of the output filter matches exactly the first `nb + 1` samples of `x`. The filter is not necessarily stable, but potentially can recover the coefficients

Meta Platforms, Inc. - Exhibit 1009
Page 570 of 720

exactly if the data sequence is truly an autoregressive moving average (ARMA) process of the correct order.

See Also

`butter`	Butterworth analog and digital filter design.
`cheby1`	Chebyshev type I filter design (passband ripple).
`cheby2`	Chebyshev type II filter design (stopband ripple).
`ellip`	Elliptic (Cauer) filter design.
`invfreqz`	Discrete-time filter identification from frequency data.
`levinson`	Levinson-Durbin recursion.
`lpc`	Linear prediction coefficients.
`stmcb`	Linear model using Steiglitz-McBride iteration.

References

[1] Parks, T.W., and C.S. Burrus. *Digital Filter Design*. New York: John Wiley & Sons, 1987. Pgs. 226-228.

Meta Platforms, Inc. - Exhibit 1009
Page 571 of 720

Superseded

The psd function for spectral estimation by Welch's method has been superseded by pwelch.

The pwelch function improves upon psd in three ways:

- The magnitude of the spectrum is scaled by 1/Fs
- The default value of Fs is 1
- No detrending is performed

The psd function will continue to be included in the Signal Processing Toolbox for backwards compatibility.

Meta Platforms, Inc. - Exhibit 1009
Page 572 of 720

Purpose

Pulse train generator.

Syntax

```
y = pulstran(t,d,'func')
y = pulstran(t,d,'func',p1,p2,...)
y = pulstran(t,d,p,Fs)
y = pulstran(t,d,p)
```

Description

`pulstran` generates pulse trains from continuous functions or sampled prototype pulses.

`y = pulstran(t,d,'func')` generates a pulse train based on samples of a continuous function, `'func'`, where *func* is:

- `gauspuls`, for Gaussian-modulated sinusoidal pulse generator
- `rectpuls`, for sampled aperiodic rectangle generator
- `tripuls`, for sampled aperiodic triangle generator

`pulstran` is evaluated `length(d)` times and returns the sum of the evaluations `y = func(t-d(1)) + func(t-d(2)) + ...`

The function is evaluated over the range of argument values specified in array `t`, after removing a scalar argument offset taken from the vector `d`. Note that *func* must be a vectorized function that can take an array `t` as an argument.

An optional gain factor may be applied to each delayed evaluation by specifying `d` as a two-column matrix, with the offset defined in column 1 and associated gain in column 2 of `d`. Note that a row vector will be interpreted as specifying delays only.

`pulstran(t,d,'func',p1,p2,...)` allows additional parameters to be passed to '*func*' as necessary. For example,

```
func(t-d(1),p1,p2,...) + func(t-d(2),p1,p2,...) + ...
```

`pulstran(t,d,p,Fs)` generates a pulse train that is the sum of multiple delayed interpolations of the prototype pulse in vector `p`, sampled at the rate `Fs`, where `p` spans the time interval `[0,(length(p)-1)/Fs]`, and its samples are identically 0 outside this interval. By default, linear interpolation is used for generating delays.

Meta Platforms, Inc. - Exhibit 1009
Page 573 of 720

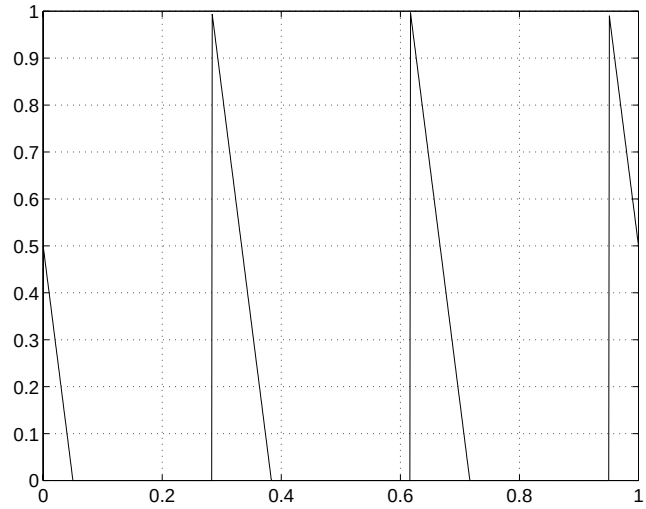
`pulstran(t,d,p)` assumes that the sampling rate `Fs` is equal to 1 Hz.

`pulstran(...,'func')` specifies alternative interpolation methods. See `interp1` for a list of available methods.

Examples

This example generates an asymmetric sawtooth waveform with a repetition frequency of 3 Hz and a sawtooth width of 0.1 sec. It has a signal length of 1 sec and a 1 kHz sample rate:

```
t = 0 : 1/1e3 : 1;          % 1 kHz sample freq for 1 sec
d = 0 : 1/3 : 1;            % 3 Hz repetition freq
y = pulstran(t,d,'tripuls',0.1,-1);
plot(t,y)
```

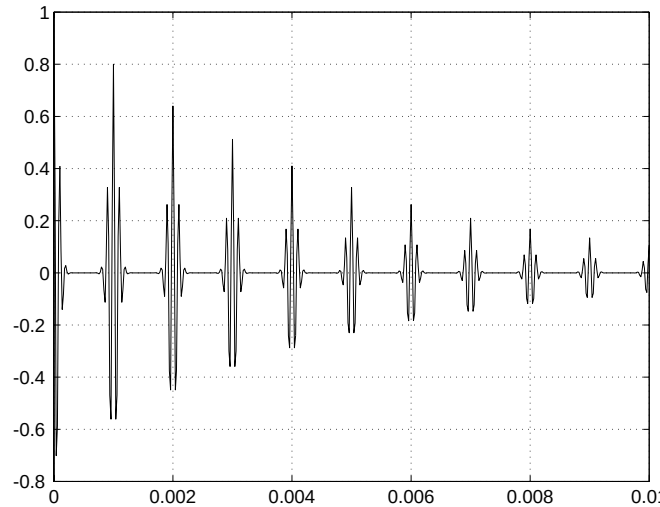


This example generates a periodic Gaussian pulse signal at 10 kHz, with 50% bandwidth. The pulse repetition frequency is 1 kHz, sample rate is 50 kHz, and

Meta Platforms, Inc. - Exhibit 1009
Page 574 of 720

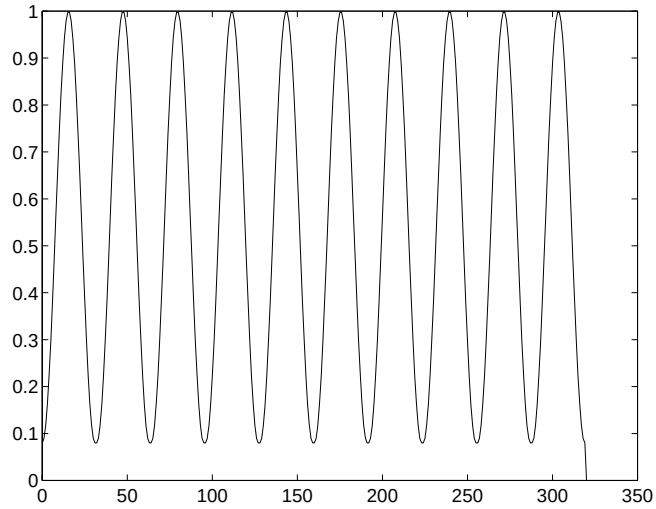
pulse train length is 10 msec. The repetition amplitude should attenuate by 0.8 each time:

```
t = 0 : 1/50E3 : 10e-3;
d = [0 : 1/1E3 : 10e-3 ; 0.8.^(0:10)]';
y = pulstran(t,d,'gauspuls',10e3,0.5);
plot(t,y)
```



This example generates a train of 10 Hamming windows:

```
p = hamming(32);
t = 0:320; d = (0:9)'*32;
y = pulstran(t,d,p);
plot(t,y)
```



See Also

`chirp`	Swept-frequency cosine generator.
`cos`	Cosine of vector/matrix elements (see the online *MATLAB Function Reference*).
`diric`	Dirichlet or periodic sinc function.
`gauspuls`	Gaussian-modulated sinusoidal pulse generator.
`rectpuls`	Sampled aperiodic rectangle generator.
`sawtooth`	Sawtooth or triangle wave generator.
`sin`	Sine of vector/matrix elements (see the online *MATLAB Function Reference*).
`sinc`	Sinc or $\sin(\pi t)/\pi t$ function.
`square`	Square wave generator.
`tripuls`	Sampled aperiodic triangle generator.

Meta Platforms, Inc. - Exhibit 1009
Page 576 of 720

Purpose

Estimate the power spectral density (PSD) of a signal using Welch's method.

Syntax

```
Pxx = pwelch(x)
Pxx = pwelch(x,nfft)
[Pxx,w] = pwelch(x,nfft)
[Pxx,f] = pwelch(x,nfft,Fs)
[Pxx,f] = pwelch(x,nfft,Fs,window)
[Pxx,f] = pwelch(x,nfft,Fs,window,noverlap)
[Pxx,Pxxc,f] = pwelch(x,nfft,Fs,window,noverlap,p)
[Pxx,Pxxc,f] = pwelch(...,'range')
pwelch(...)
pwelch(...,'magunits')
```

Description

`Pxx = pwelch(x)` estimates the power spectrum of the sequence `x` using the Welch method of spectral estimation. If `x` is real, `pwelch` estimates the spectrum at positive frequencies only; in this case, output `Pxx` is a column vector of length `nfft/2+1` for `nfft` even and `(nfft+1)/2` for `nfft` odd. If `x` is complex, `pwelch` estimates the spectrum at both positive and negative frequencies and `Pxx` has length `nfft`.

`Pxx = pwelch(x,nfft)` uses the specified FFT length `nfft` in estimating the power spectrum for `x`. This value determines the number of different frequencies at which the power spectrum is estimated. Specify `nfft` as a power of 2 for fastest execution. Specify an empty matrix for `nfft`, `[]`, to use the default value of `min(256,length(x))`.

`[Pxx,w] = pwelch(x,nfft)` returns a vector `w` of normalized angular frequencies (in rads/sample) at which the function evaluates the PSD. The range of `w` is $[0,\pi)$ for real `x` and $[0,2\pi)$ for complex `x`. Since the frequency vector `w` is the same size as `Pxx`, `plot(w,Pxx)` plots the power spectrum versus the normalized angular frequency.

`[Pxx,f] = pwelch(x,nfft,Fs)` returns a vector `f` of linear frequencies (in Hz) at which the function evaluates the PSD. `Fs` is a scalar that specifies the sampling frequency of `x`, and `Pxx` is scaled by `1/Fs`. The range of `f` is `[0,Fs/2)` for real `x` and `[0,Fs)` for complex `x`. Since the frequency vector `f` is the same size as `Pxx`, `plot(f,Pxx)` plots the power spectrum versus the linear frequency. Specify an empty matrix for `Fs`, `[]`, to use the default value of 1 Hz.

Meta Platforms, Inc. - Exhibit 1009
Page 577 of 720

`[Pxx,f] = pwelch(x,nfft,Fs,window)` specifies a windowing function and the number of samples per windowed section of the `x` vector (e.g., `kaiser(64)`). The length of the window must be less than or equal to `nfft`; `pwelch` zero pads the sections if the length of the window is less than `nfft`. Specify a scalar for `window` to use a Hanning window of that length; specify an empty matrix, `[]`, to use the default value of `hanning(nfft)`.

`[Pxx,f] = pwelch(x,nfft,Fs,window,noverlap)` overlaps the windowed sections of `x` by `noverlap` samples. Specify an empty matrix for `noverlap`, `[]`, to use the default value of 0.

`[Pxx,Pxxc,f] = pwelch(x,nfft,Fs,window,noverlap,p)` where `p` is a positive scalar between 0 and 1 returns a vector `Pxxc` that contains an estimate of the `p*100` percent confidence interval for `Pxx`. `Pxxc` is a two-column matrix that is the same length as `Pxx`. The interval `[Pxxc(:,1),Pxxc(:,2)]` covers the true PSD with probability `p`. `plot(f,[Pxx Pxxc])` plots the power spectrum inside the `p*100` percent confidence interval. Specify an empty matrix for `p`, `[]`, to use the default value of 0.95.

`[Pxx,Pxxc,f] = pwelch(...,'range')` specifies the range of frequency values to include in `f`. *`range`* can be:

- `half`, to compute the PSD over the range $[0,\pi)$ for real or complex `x`. If `Fs` is specified, the range is `[0,Fs/2)` for real or complex `x`. If `Fs` is specified by an empty matrix, `[]`, the default value of 1 is used, and the range is [0,1/2) for real or complex `x`.
- `whole`, to compute the PSD over the range $[0,2\pi)$ for real or complex `x`. If `Fs` is specified, the range is `[0,Fs)` for real or complex `x`. If `Fs` is specified by an empty matrix, `[]`, the default value of 1 is used, and the range is [0,1) for real or complex `x`.

`pwelch(x,...)` with no output arguments plots the PSD against frequency in the current figure window. If `p` is specified, the plot includes the confidence interval.

`pwelch(...,'magunits')` specifies the units in which to express to the magnitude axis of the PSD plot. The options are:

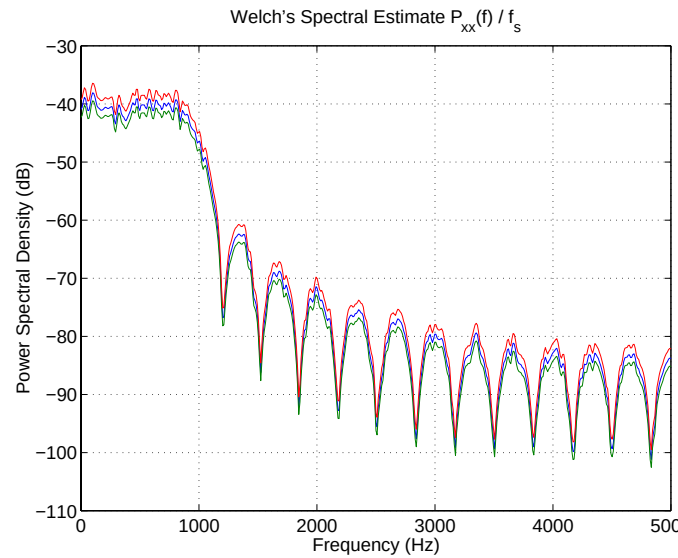
Meta Platforms, Inc. - Exhibit 1009
Page 578 of 720

- 'db', to express the PSD data in decibels (dB). This is the default.
- 'squared', to express the PSD in linear units (magnitude squared).

Example

Generate a colored noise signal and plot its PSD with a confidence interval of 95%. Specify a length 1024 FFT, a 512-point Kaiser window with no overlap, and a sampling frequency of 10 kHz:

```
h = fir1(30,0.2,boxcar(31));      % design a lowpass filter
r = randn(16384,1);               % white noise
x = filter(h,1,r);                % color the noise
pwelch(x,1024,10000,kaiser(512,5),0,0.95)
```



Algorithm

pwelch calculates the power spectral density using Welch's method (see references [1] and [2]):

1. It applies the window specified by the window vector to each successive section of input x.
2. It transforms each section with an nfft-point FFT.
3. It forms the periodogram of each section by scaling the magnitude squared of each transformed section.

Meta Platforms, Inc. - Exhibit 1009
Page 579 of 720

4. It averages the periodograms of the overlapping sections to form `Pxx(f)`.
5. It scales `Pxx(f)` by `1/Fs` to form `Pxx(f)/Fs`, the power spectrum of `x`.

The number of sections that `pwelch` averages is

```
k = fix((length(x)-noverlap)/(length(window)-noverlap))
```

Diagnostics

An appropriate diagnostic message is displayed when incorrect arguments to `pwelch` are used:

```
Requires window's length to be no greater than FFT length.
Requires NOVERLAP to be strictly less than the window length.
Requires positive integer values for NFFT and NOVERLAP.
Requires confidence parameter to be a scalar between 0 and 1.
Requires vector input.
```

See Also

`cohere`	Estimate magnitude squared coherence function between two signals.
`csd`	Estimate the cross spectral density (CSD) of two signals.
`pburg`	Power spectrum estimate using the Burg method.
`pcov`	Power spectrum estimate using the covariance method.
`pmcov`	Power spectrum estimate using the modified covariance method.
`pmtm`	Power spectrum estimate using the multitaper method (MTM).
`pmusic`	Power spectrum estimate using MUSIC eigenvector method.
`pyulear`	Power spectrum estimate using Yule-Walker AR method.
`specgram`	Time-dependent frequency analysis (spectrogram).
`tfe`	Transfer function estimate from input and output.

Meta Platforms, Inc. - Exhibit 1009
Page 580 of 720

References

[1] Rabiner, L.R., and B. Gold. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1975. Pgs. 399-419.

[2] Welch, P.D. "The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified Periodograms." *IEEE Trans. Audio Electroacoust*. Vol. AU-15 (June 1967). Pgs. 70-73.

[3] Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989. Pgs. 311-312.

Meta Platforms, Inc. - Exhibit 1009
Page 581 of 720

Purpose

Power spectrum estimate using Yule-Walker AR method.

Syntax

```
Pxx = pyulear(x,p,nfft)
[Pxx,freq] = pyulear(x,p,nfft)
[Pxx,freq] = pyulear(x,p,nfft,Fs)
[Pxx,freq] = pyulear(x,p,nfft,Fs,'range')
pyulear(...)
pyulear(...,'squared')
```

Description

`pyulear` estimates the power spectral density (PSD) of the signal vector $x[n]$ using the Yule-Walker AR method. This method, also called the autocorrelation method, fits an autoregressive (AR) model to the signal 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. The spectral estimate returned by `pyulear` is the magnitude squared frequency response of this AR model. The correct choice of the model order `p` is important.

`Pxx = pyulear(x,p,nfft)` returns `Pxx`, the power spectrum estimate. `x` is the input signal, `p` is the model order for the all-pole filter, and `nfft` is the FFT length (defaults to 256 if not specified). `Pxx` has length `(nfft/2+1)` for `nfft` even, `(nfft+1)/2` for `nfft` odd, and `nfft` if `x` is complex.

`[Pxx,freq] = pyulear(x,p,nfft)` returns `Pxx`, the power spectrum estimate, and `freq`, a vector of frequencies at which the PSD was estimated. If the input signal is real-valued, the range of `freq` is $[0,\pi)$. If the input signal is complex, the range of `freq` is $[0,2\pi)$.

`[Pxx,freq] = pyulear(x,p,nfft,Fs)` uses the signal's sampling frequency, `Fs`, to scale both the PSD vector (`Pxx`) and the frequency vector (`freq`). `Pxx` is scaled by `1/Fs`. If the input signal is real-valued, the range of `freq` is `[0,Fs/2)`. If the input signal is complex, the range of `freq` is `[0,Fs)`. `Fs` defaults to `1` if left empty, `[]`.

`[Pxx,freq] = pyulear(x,p,nfft,Fs,'range')` specifies the range of frequency values to include in `freq`. *`range`* can be:

- `half`, to compute the PSD over the range `[0,Fs/2)` for real `x`, and `[0,Fs)` for complex `x`. If `Fs` is left blank, `[]`, the range is `[0,1/2)` for real `x`, and `[0,1)`

Meta Platforms, Inc. - Exhibit 1009
Page 582 of 720

for complex `x`. If `Fs` is omitted entirely, the range is `[0,pi)` for real `x`, and `[0,2*pi)` for complex `x`. `half` is the default range.

- `whole`, to compute the PSD over the range `[0,Fs)` for all `x`. If `Fs` is left blank, `[]`, the range is `[0,1)` for all `x`. If `Fs` is omitted entirely, the range is `[0,2*pi)` for all `x`.

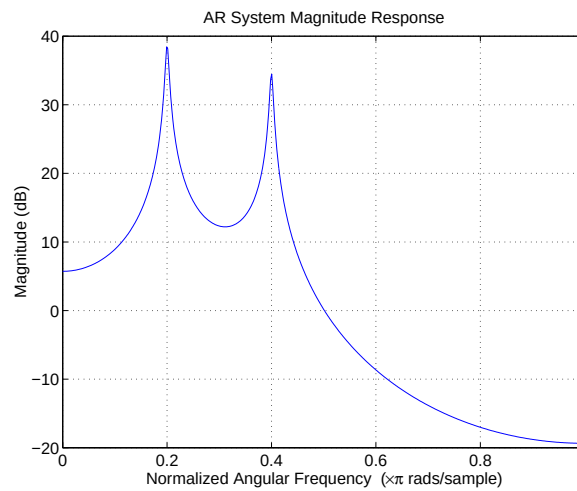
`pyulear(...)` plots the power spectral density in the first available figure window. The frequency range on the plot is the same as the range of output `freq` for a given set of parameters.

`pyulear(...,'squared')` plots the magnitude of `Pxx` directly, rather than comnverting the values to dB.

Example

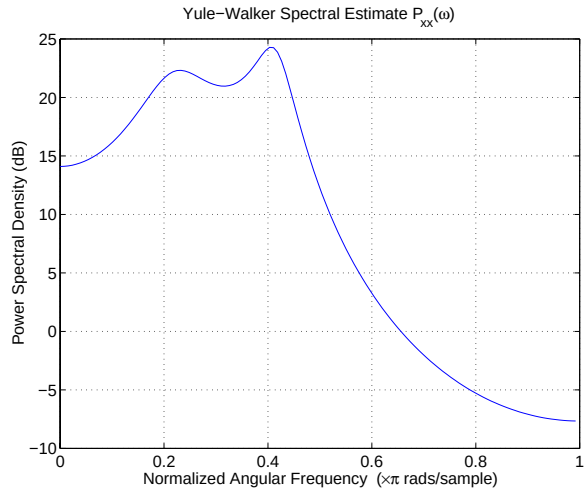
Since the method estimates the spectrum by fitting an AR model to the signal, first define the AR system (all-pole filter) that generates the input. Check the magnitude response of the process with `freqz`.

```
a = [1 -2.2137 2.9403 -2.1697 0.9606];  % AR system coefficients
freqz(1,a,'phase','no')  % AR system magnitude response
title('AR System Magnitude Response')
```



Now generate the input signal x by filtering white noise through the AR system. Estimate the PSD of x based on a fourth-order AR model (since we know that the original AR system, a, has order 4).

```
x = filter(1,a,randn(150,1));     % AR system output
pyulear(x,4)                      % 4th order estimate
```



Diagnostics

The first input argument must be a full vector; otherwise pyulear generates the following error message:

```
Input signal cannot be sparse.
```

If you specify an empty matrix for the second argument, pyulear generates the following error message:

```
Model order must be an integer.
```


Meta Platforms, Inc. - Exhibit 1009
Page 584 of 720

See Also

`aryule`	Compute an estimate of AR model parameters using the Yule-Walker method.
`lpc`	Linear prediction coefficients.
`pburg`	Power spectrum estimate using the Burg method.
`pcov`	Power spectrum estimate using the covariance method.
`pmcov`	Power spectrum estimate using the modified covariance method.
`pmtm`	Power spectrum estimate using the multitaper method (MTM).
`pmusic`	Power spectrum estimate using MUSIC eigenvector method.
`prony`	Prony's method for time domain IIR filter design.
`pwelch`	Estimate the power spectral density (PSD) of a signal using Welch's method.

References

[1] Marple, S.L. *Digital Spectral Analysis*. Englewood Cliffs, NJ: Prentice Hall, 1987. Chapter 7.

Meta Platforms, Inc. - Exhibit 1009
Page 585 of 720

Purpose Conversion of reflection coefficients to autocorrelation sequence.

Syntax

```
r = rc2ac(k,r0)
```

Description `r = rc2ac(k,r0)` finds the autocorrelation coefficients, `r`, of the discrete filter with reflection coefficients `k` and zero-lag autocorrelation `r0`.

See Also

`ac2rc`	Conversion of autocorrelation sequence to reflection coefficients.
`poly2ac`	Conversion of prediction polynomial to autocorrelation sequence.
`rc2poly`	Conversion of reflection coefficients to prediction polynomial.

References

[1] Kay, S.M. *Modern Spectral Estimation*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

Meta Platforms, Inc. - Exhibit 1009
Page 586 of 720

Purpose

Conversion of reflection coefficients to prediction polynomial.

Syntax

```
a = rc2poly(k)
[a,efinal] = rc2poly(k,r0)
```

Description

`a = rc2poly(k)` finds the prediction polynomial `a`, with `a(1) = 1`, from the reflection coefficients `k` of the corresponding lattice structure. Output `a` is row vector of length `k + 1`.

`[a,efinal] = rc2poly(k,r0)` returns the final prediction error, `efinal`, based on the zero-lag autocorrelation, `r0`.

Example

Consider an IIR filter given by reflection coefficients `k`:

```
k = [0.3090    0.9800    0.0031    0.0082   -0.0082];
```

Its polynomial representation is

```
a = rc2poly(k)

a =

1.0000    0.6148    0.9899    0.0000    0.0032   -0.0082
```

Algorithm

`rc2poly` computes output `a` using Levinson's recursion [1]. The function:

1. Sets the output vector `a` to the first element of `k`
2. Loops through the remaining elements of `k`

   For each loop iteration `i`, `a = [a + a(i-1:-1:1)*k(i) k(i)]`.
3. Implements `a = [1 a]`

Meta Platforms, Inc. - Exhibit 1009
Page 587 of 720

See Also

`ac2poly`	Conversion of autocorrelation sequence to prediction polynomial.
`latc2tf`	Lattice filter to transfer function conversion.
`latcfilt`	Lattice and lattice-ladder filter implementation.
`poly2rc`	Conversion of prediction polynomial to reflection coefficients.
`rc2ac`	Conversion of reflection coefficients to autocorrelation sequence.
`tf2latc`	Conversion of transfer function to lattice filter.

References

[1] Kay, S.M. *Modern Spectral Estimation*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

Meta Platforms, Inc. - Exhibit 1009
Page 588 of 720

Purpose

Real cepstrum and minimum phase reconstruction.

Syntax

```
y = rceps(x)
[y,ym] = rceps(x)
```

Description

The *real cepstrum* is the inverse Fourier transform of the real logarithm of the magnitude of the Fourier transform of a sequence.

`rceps(x)` returns the real cepstrum of the real sequence `x`. The real cepstrum is a real-valued function.

`[y,ym] = rceps(x)` returns both the real cepstrum `y` and a minimum phase reconstructed version `ym` of the input sequence.

Algorithm

`rceps` is an M-file implementation of algorithm 7.2 in [2], that is:

```
y = real(ifft(log(abs(fft(x)))));
```

Appropriate windowing in the cepstral domain forms the reconstructed minimum phase signal:

```
w = [1; 2*ones(n/2-1,1); ones(1 - rem(n,2),1); zeros(n/2-1,1)];
ym = real(ifft(exp(fft(w.*y))));
```

See Also

`cceps`	Complex cepstral analysis.
`fft`	One-dimensional fast Fourier transform.
`hilbert`	Hilbert transform.
`icceps`	Inverse complex cepstrum.
`unwrap`	Unwrap phase angles.

References

[1] Oppenheim, A.V., and R.W. Schafer. *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1975.

[2] IEEE. *Programs for Digital Signal Processing*. IEEE Press. New York: John Wiley & Sons, 1979.

Meta Platforms, Inc. - Exhibit 1009
Page 589 of 720

Purpose

Sampled aperiodic rectangle generator.

Syntax

```
y = rectpuls(t)
y = rectpuls(t,w)
```

Description

`y = rectpuls(t)` returns a continuous, aperiodic, unity-height rectangular pulse at the sample times indicated in array `t`, centered about `t = 0` and with a default width of 1. Note that the interval of non-zero amplitude is defined to be open on the right, that is, `rectpuls(–0.5) = 1` while `rectpuls(0.5) = 0`.

`y = rectpuls(t,w)` generates a rectangle of width `w`.

`rectpuls` is typically used in conjunction with the pulse train generating function, `pulstran`.

See Also

`chirp`	Swept-frequency cosine generator.
`cos`	Cosine of vector/matrix elements (see the online *MATLAB Function Reference*).
`diric`	Dirichlet or periodic sinc function.
`gauspuls`	Gaussian-modulated sinusoidal pulse generator.
`pulstran`	Pulse train generator.
`sawtooth`	Sawtooth or triangle wave generator.
`sin`	Sine of vector/matrix elements (see the online *MATLAB Function Reference*).
`sinc`	Sinc or $\sin(\pi t)/\pi t$ function.
`square`	Square wave generator.
`tripuls`	Sampled aperiodic triangle generator.

Meta Platforms, Inc. - Exhibit 1009
Page 590 of 720

Purpose

Parks-McClellan optimal FIR filter design.

Syntax

```
b = remez(n,f,a)
b = remez(n,f,a,w)
b = remez(n,f,a,'ftype')
b = remez(n,f,a,w,'ftype')
b = remez(...,{lgrid})
b = remez(n,f,'fresp',w)
b = remez(n,f,'fresp',w,'ftype')
b = remez(n,f,{'fresp',p1,p2,...},w)
b = remez(n,f,{'fresp',p1,p2,...},w,'ftype')
[b,delta] = remez(...)
[b,delta,opt] = remez(...)
```

Description

`remez` designs a linear-phase FIR filter using the Parks-McClellan algorithm [1]. The Parks-McClellan algorithm uses the Remez exchange algorithm and Chebyshev approximation theory to design filters with an optimal fit between the desired and actual frequency responses. The filters are optimal in the sense that the maximum error between the desired frequency response and the actual frequency response is minimized. Filters designed this way exhibit an equiripple behavior in their frequency responses and hence are sometimes called *equiripple* filters.

`b = remez(n,f,a)` returns row vector `b` containing the `n+1` coefficients of the order `n` FIR filter whose frequency-amplitude characteristics match those given by vectors `f` and `a`.

The output filter coefficients (taps) in `b` obey the symmetry relation

$$b(k) = b(n + 2 - k), \qquad k = 1, \ldots, n + 1$$

Vectors `f` and `a` specify the frequency-magnitude characteristics of the filter:

Meta Platforms, Inc. - Exhibit 1009
Page 591 of 720

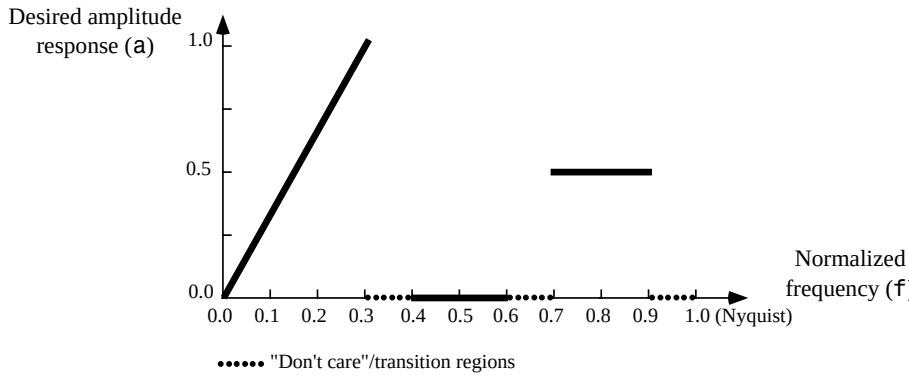
- `f` is a vector of pairs of frequency points, specified in the range between 0 and 1, where 1 corresponds to half the sampling frequency (the Nyquist frequency). The frequencies must be in increasing order.
- `a` is a vector containing the desired amplitudes at the points specified in `f`.

  The desired amplitude at frequencies between pairs of points ($f(k)$, $f(k+1)$) for k odd is the line segment connecting the points ($f(k)$, $a(k)$) and ($f(k+1)$, $a(k+1)$).

  The desired amplitude at frequencies between pairs of points ($f(k)$, $f(k+1)$) for k even is unspecified. The areas between such points are transition or "don't care" regions.
- `f` and `a` must be the same length. The length must be an even number.

The relationship between the `f` and `a` vectors in defining a desired frequency response is shown below:

```
f = [0 .3 .4 .6 .7 .9]
a = [0  1  0  0 .5 .5]
```



`remez(n,f,a,w)` uses the weights in vector `w` to weight the fit in each frequency band. The length of `w` is half the length of `f` and `a`, so there is exactly one weight per band.

`b = remez(n,f,a,'`*`ftype`*`')` and

`b = remez(n,f,a,w,'`*`ftype`*`')` specify a filter type, where *`ftype`* is

Meta Platforms, Inc. - Exhibit 1009
Page 592 of 720

- `hilbert`, for linear-phase filters with odd symmetry (type III and type IV)

  The output coefficients in `b` obey the relation $b(k) = -b(n + 2 - k)$, $k = 1,...,n + 1$. This class of filters includes the Hilbert transformer, which has a desired amplitude of 1 across the entire band.

  For example,

  ```
  h = remez(30,[0.1 0.9],[1 1],'hilbert');
  ```

  designs an approximate FIR Hilbert transformer of length 31.

- `differentiator`, for type III and IV filters, using a special weighting technique

  For nonzero amplitude bands, it weights the error by a factor of $1/f$ so that the error at low frequencies is much smaller than at high frequencies. For FIR differentiators, which have an amplitude characteristic proportional to frequency, these filters minimize the maximum relative error (the maximum of the ratio of the error to the desired amplitude).

`b = remez(...,{lgrid})` uses the integer `lgrid` to control the density of the frequency grid, which has roughly `(lgrid*n)/(2*bw)` frequency points, where `bw` is the fraction of the total frequency band interval [0,1] covered by `f`. Increasing `lgrid` often results in filters that are more exactly equiripple, but which take longer to compute. The default value of `16` is the minimum value that should be specified for `lgrid`. Note that the `{lgrid}` argument must be a 1-by-1 cell array.

`b = remez(n,f,'fresp',w)` returns row vector `b` containing the `n+1` coefficients of the order `n` FIR filter whose frequency-amplitude characteristics best approximate the response specified by function *`fresp`*. The function is called from within `remez` with the following syntax:

```
[dh,dw] = fresp(n,f,gf,w)
```

The arguments are similar to those for `remez`:

- `n` is the filter order.
- `f` is the vector of frequency band edges that appear monotonically between 0 and 1, where 1 is the Nyquist frequency.
- `gf` is a vector of grid points that have been linearly interpolated over each specified frequency band by `remez`. `gf` determines the frequency grid at

Meta Platforms, Inc. - Exhibit 1009
Page 593 of 720

which the response function must be evaluated, and contains the same data returned by `cremez` in the `fgrid` field of the `opt` structure.

- `w` is a vector of real, positive weights, one per band, used during optimization. `w` is optional in the call to `remez`; if not specified, it is set to unity weighting before being passed to `'fresp'`.
- `dh` and `dw` are the desired complex frequency response and band weight vectors, respectively, evaluated at each frequency in grid `gf`.

The predefined frequency response function (*fresp*) that `remez` calls is `remezfrf` in the `signal/private` directory.

`b = remez(n,f,{'fresp',p1,p2,...},w)` allows you to specify additional parameters (`p1`, `p2`, etc.) to pass to *fresp*. Note that `b = remez(n,f,a,w)` is a synonym for `b = remez(n,f,{'remezfrf',a},w)`, where `a` is a vector containing the desired amplitudes at the points specified in `f`.

`b = remez(n,f,'fresp',w,'ftype')` and

`b = remez(n,f,{'fresp',p1,p2,...},w,'ftype')` design antisymmetric (odd) rather than symmetric (even) filters, where `'ftype'` is either `'d'` for a differentiator or `'h'` for a Hilbert transformer.

In the absence of a specification for *ftype*, a preliminary call is made to *fresp* to determine the default symmetry property `sym`. This call is made using the syntax:

```
sym = fresp('defaults',{n,f,[],w,p1,p2,...})
```

The arguments `n`, `f`, `w`, etc., may be used as necessary in determining an appropriate value for `sym`, which `remez` expects to be either `'even'` or `'odd'`. If the *fresp* function does not support this calling syntax, `remez` defaults to even symmetry.

`[b,delta] = remez(...)` returns the maximum ripple height in `delta`.

Meta Platforms, Inc. - Exhibit 1009
Page 594 of 720

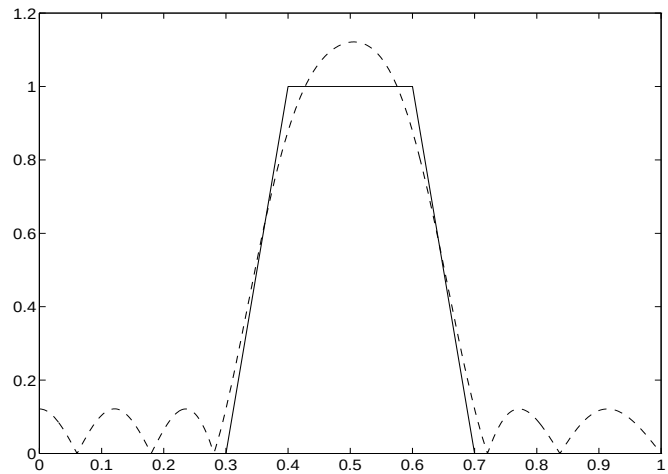
`[b,delta,opt] = remez(...)` returns a structure, `opt`, of optional results with the following fields.

`opt.fgrid`	Frequency grid vector used for the filter design optimization
`opt.des`	Desired frequency response for each point in `opt.fgrid`
`opt.wt`	Weighting for each point in `opt.fgrid`
`opt.H`	Actual frequency response for each point in `opt.fgrid`
`opt.error`	Error at each point in `opt.fgrid` (`opt.des–opt.H`)
`opt.iextr`	Vector of indices into `opt.fgrid` for extremal frequencies
`opt.fextr`	Vector of extremal frequencies

Example

Graph the desired and actual frequency responses of a 17th-order Parks-McClellan bandpass filter:

```
f = [0 0.3 0.4 0.6 0.7 1]; a = [0 0 1 1 0 0];
b = remez(17,f,a);
[h,w] = freqz(b,1,512);
plot(f,a,w/pi,abs(h))
```



Meta Platforms, Inc. - Exhibit 1009
Page 595 of 720

Algorithm

`remez` is a MEX-file version of the original Fortran code from [1], altered to design arbitrarily long filters with arbitrarily many linear bands.

`remez` designs type I, II, III, and IV linear-phase filters. Type I and Type II are the defaults for `n` even and `n` odd, respectively, while Type III (`n` even) and Type IV (`n` odd) are obtained with the `'hilbert'` and `'differentiator'` flags. The different types of filters have different symmetries and certain constraints on their frequency responses (see [5] for more details).

Linear Phase Filter Type	Filter Order n	Symmetry of Coefficients	Response H(f), f = 0	Response H(f), f = 1 (Nyquist)
Type I	Even	even: $b(k) = b(n + 2 - k), \quad k = 1, \ldots, n + 1$	No restriction	No restriction
Type II	Odd		No restriction	$H(1) = 0$
Type III	Even	odd: $b(k) = -b(n + 2 - k), \quad k = 1, \ldots, n + 1$	$H(0) = 0$	$H(1) = 0$
Type IV	Odd		$H(0) = 0$	No restriction

Diagnostics

An appropriate diagnostic message is displayed if incorrect arguments are used:

```
Filter order must be 3 or more.
There should be one weight per band.
Frequency and amplitude vectors must be the same length.
The number of frequency points must be even.
Frequencies must lie between 0 and 1.
Frequencies must be specified in bands.
Frequencies must be nondecreasing.
Adjacent bands not allowed.
```

A more serious warning message is

```
— Failure to Converge —
Probable cause is machine rounding error.
```

In the rare event that you see this message, it is possible that the filter design may still be correct. Verify the design by checking its frequency response.

Meta Platforms, Inc. - Exhibit 1009
Page 596 of 720

See Also

`butter`	Butterworth analog and digital filter design.
`cheby1`	Chebyshev type I filter design (passband ripple).
`cheby2`	Chebyshev type II filter design (stopband ripple).
`cremez`	Complex and nonlinear-phase equiripple FIR filter design.
`ellip`	Elliptic (Cauer) filter design.
`fir1`	Window-based finite impulse response filter design – standard response.
`fir2`	Window-based finite impulse response filter design – arbitrary response.
`fircls`	Constrained least square FIR filter design for multiband filters.
`fircls1`	Constrained least square filter design for lowpass and highpass linear phase FIR filters.
`firls`	Least square linear-phase FIR filter design.
`firrcos`	Raised cosine FIR filter design.
`remezord`	Parks-McClellan optimal FIR filter order estimation.
`yulewalk`	Recursive digital filter design.

References

[1] IEEE. *Programs for Digital Signal Processing*. IEEE Press. New York: John Wiley & Sons, 1979. Algorithm 5.1.

[2] IEEE. *Selected Papers in Digital Signal Processing, II*. IEEE Press. New York: John Wiley & Sons, 1979.

[3] Parks, T.W., and C.S. Burrus. *Digital Filter Design*. New York: John Wiley & Sons, 1987. Pg. 83.

[4] Rabiner, L.R., J.H. McClellan, and T.W. Parks. “FIR Digital Filter Design Techniques Using Weighted Chebyshev Approximations.” *Proc. IEEE* 63 (1975).

[5] Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989. Pgs. 256-266.

Meta Platforms, Inc. - Exhibit 1009
Page 597 of 720

Purpose

Parks-McClellan optimal FIR filter order estimation.

Syntax

```
[n,fo,ao,w] = remezord(f,a,dev)
[n,fo,ao,w] = remezord(f,a,dev,Fs)
c = remezord(f,a,dev,Fs,'cell')
```

Description

`[n,fo,ao,w] = remezord(f,a,dev)` finds the approximate order, normalized frequency band edges, frequency band amplitudes, and weights that meet input specifications `f`, `a`, and `dev`, to use with the `remez` command.

- `f` is a vector of frequency band edges (between 0 and Fs/2, where Fs is the sampling frequency), and `a` is a vector specifying the desired amplitude on the bands defined by `f`. The length of `f` is twice the length of `a`, minus 2. The desired function is piecewise constant.
- `dev` is a vector the same size as `a` that specifies the maximum allowable deviation or ripples between the frequency response and the desired amplitude of the output filter, for each band.

Use `remez` with the resulting order `n`, frequency vector `fo`, amplitude response vector `ao`, and weights `w` to design the filter `b` which approximately meets the specifications given by `remezord` input parameters `f`, `a`, and `dev`:

```
b = remez(n,fo,ao,w)
```

`[n,fo,ao,w] = remezord(f,a,dev,Fs)` specifies a sampling frequency `Fs`. `Fs` defaults to 2 Hz, implying a Nyquist frequency of 1 Hz. You can therefore specify band edges scaled to a particular application's sampling frequency.

In some cases `remezord` underestimates the order `n`. If the filter does not meet the specifications, try a higher order such as `n+1` or `n+2`.

`c = remezord(f,a,dev,Fs,'cell')` specifies a cell-array whose elements are the parameters to `remez`.

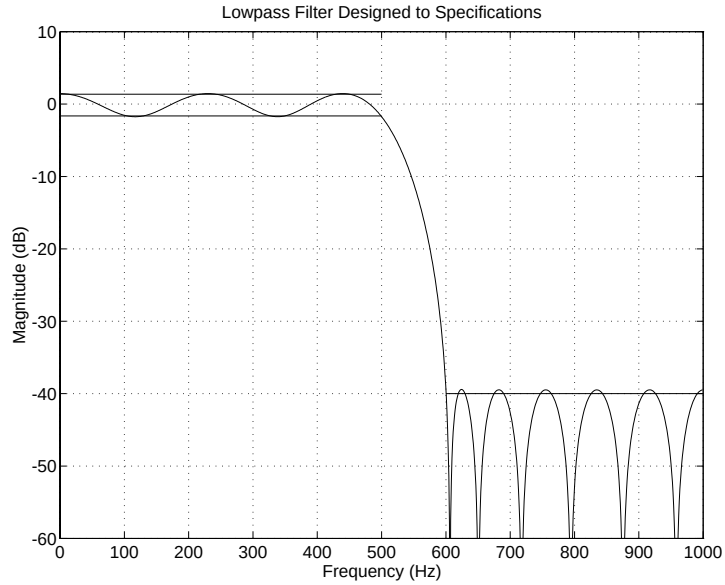
Examples

Design a minimum-order lowpass filter with a 500 Hz passband cutoff frequency and 600 Hz stopband cutoff frequency, with a sampling frequency of

Meta Platforms, Inc. - Exhibit 1009
Page 598 of 720

2000 Hz), at least 40 dB attenuation in the stopband, and less than 3 dB of ripple in the passband:

```
rp = 3;          % passband ripple
rs = 40;         % stopband ripple
Fs = 2000;       % sampling frequency
f = [500 600];   % cutoff frequencies
a = [1 0];       % desired amplitudes

% compute deviations
dev = [(10^(rp/20)-1)/(10^(rp/20)+1)  10^(-rs/20)];

[n,fo,ao,w] = remezord(f,a,dev,Fs);
b = remez(n,fo,ao,w);
[h,f] = freqz(b,1,1024,Fs);
plot(f,20*log10(abs(h)));

title('Lowpass Filter Designed to Specifications');
xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)'); grid;
```



Note that the filter falls slightly short of meeting the stopband attenuation and passband ripple specifications. Using `n+1` in the call to `remez` instead of `n` achieves the desired amplitude characteristics.

Meta Platforms, Inc. - Exhibit 1009
Page 599 of 720

Design a lowpass filter with a 1500 Hz passband cutoff frequency and 2000 Hz stopband cutoff frequency, with a sampling frequency of 8000 Hz, a maximum stopband amplitude of 0.1, and a maximum passband error (ripple) of 0.01:

```
[n,fo,ao,w] = remezord( [1500 2000], [1 0], [0.01 0.1], 8000 );
b = remez(n,fo,ao,w);
```

This is equivalent to

```
c = remezord( [1500 2000], [1 0], [0.01 0.1], 8000, 'cell');
b = remez(c{:});
```

NOTE In some cases, `remezord` underestimates or overestimates the order `n`. If the filter does not meet the specifications, try a higher order such as `n+1` or `n+2`.

Results are inaccurate if the cutoff frequencies are near 0 or the Nyquist frequency.

Algorithm

`remezord` uses the algorithm suggested in [1]. This method is inaccurate for band edges close to either 0 or the Nyquist frequency (`Fs/2`).

Diagnostics

If the input parameter lengths are not consistent, `remezord` gives the following error messages:

```
Requires M and DEV to be the same length.
Length of F must be length(M)-2.
```

See Also

`buttord`	Butterworth filter order selection.
`cheb1ord`	Chebyshev type I filter order selection.
`cheb2ord`	Chebyshev type II filter order selection.
`ellipord`	Elliptic filter order selection.
`kaiserord`	Estimate parameters for an FIR filter design with Kaiser window.
`remez`	Parks-McClellan optimal FIR filter design.

Meta Platforms, Inc. - Exhibit 1009
Page 600 of 720

References

[1] Rabiner, L.R., and O. Herrmann. "The Predictability of Certain Optimum Finite Impulse Response Digital Filters." *IEEE Trans. on Circuit Theory*. Vol. CT-20, No. 4 (July 1973). Pgs. 401-408.

[2] Rabiner, L.R., and B. Gold. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1975. Pgs. 156-157.

Meta Platforms, Inc. - Exhibit 1009
Page 601 of 720

Purpose

Change sampling rate by any rational factor.

Syntax

```
y = resample(x,p,q)
y = resample(x,p,q,n)
y = resample(x,p,q,n,beta)
y = resample(x,p,q,b)
[y,b] = resample(x,p,q)
```

Description

`y = resample(x,p,q)` resamples the sequence in vector `x` at `p/q` times the original sampling rate, using a polyphase filter implementation. The length of `y` is equal to `ceil(length(x)*p/q)`. `p` and `q` must be positive integers. If `x` is a matrix, `resample` works down the columns of `x`.

`resample` applies an anti-aliasing (lowpass) FIR filter to `x` during the resampling process. It designs the filter using `firls` with a Kaiser window.

`y = resample(x,p,q,n)` uses `n` terms on either side of the current sample, `x(k)`, to perform the resampling. The length of the FIR filter `resample` uses is proportional to `n`; larger values of `n` provide better accuracy at the expense of more computation time. The default for `n` is 10. If you let `n = 0`, `resample` performs a nearest-neighbor interpolation:

```
y(k) = x(round((k-1)*q/p)+1)
```

where `y(k)` = 0 if the index to `x` is greater than `length(x)`.

`y = resample(x,p,q,n,beta)` uses `beta` as the design parameter for the Kaiser window that `resample` employs in designing the lowpass filter. The default for `beta` is 5.

`y = resample(x,p,q,b)` filters `x` with `b`, a vector of filter coefficients.

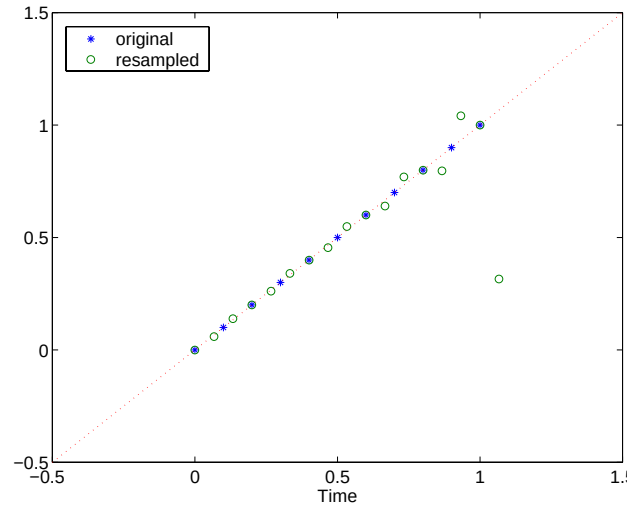
`[y,b] = resample(x,p,q)` returns the vector `b`, which contains the coefficients of the filter applied to `x` during the resampling process.

Meta Platforms, Inc. - Exhibit 1009
Page 602 of 720

Examples

Resample a simple linear sequence at 3/2 the original rate:

```
Fs1 = 10;            % original sampling frequency in Hz
t1 = 0:1/Fs1:1;      % time vector
x = t1;              % define a linear sequence
y = resample(x,3,2); % now resample it

t2 = (0:(length(y)-1))*2/(3*Fs1); % new time vector
plot(t1,x,'*',t2,y,'o',-0.5:0.01:1.5,-0.5:0.01:1.5,':')
legend('original','resampled'); xlabel('Time')
```

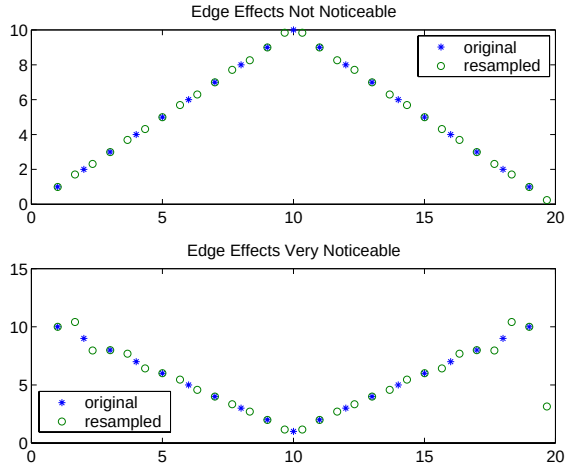


Notice that the last few points of the output `y` are inaccurate. In its filtering process, `resample` assumes the samples at times before and after the given samples in `x` are equal to zero. Thus large deviations from zero at the end

Meta Platforms, Inc. - Exhibit 1009
Page 603 of 720

points of the sequence x can cause inaccuracies in y at its end points. The following two plots illustrate this side effect of resample:

```
x = [1:10 9:-1:1]; y = resample(x,3,2);
subplot(2,1,1);
plot(1:19,x,'*',(0:28)*2/3 + 1,y,'o');
title('Edge Effects Not Noticeable');
legend('original','resampled');

x = [10:-1:1 2:10]; y = resample(x,3,2);
subplot(2,1,2);
plot(1:19,x,'*',(0:28)*2/3 + 1,y,'o')
title('Edge Effects Very Noticeable');
legend('original','resampled');
```



Diagnostics

If p or q are not positive integers, resample gives the appropriate error message:

```
P must be a positive integer.
Q must be a positive integer.
```

If x is not a vector, resample gives the following error message:

```
Input X must be a vector.
```


Meta Platforms, Inc. - Exhibit 1009
Page 604 of 720

See Also

`decimate`	Decrease the sampling rate for a sequence (decimation).
`firls`	Least square linear-phase FIR filter design.
`interp`	Increase sampling rate by an integer factor (interpolation).
`interp1`	One-dimensional data interpolation (table lookup) (see the online *MATLAB Function Reference*).
`intfilt`	Interpolation FIR filter design.
`kaiser`	Kaiser window.
`spline`	Cubic spline interpolation (see the online *MATLAB Function Reference*).
`upfirdn`	Upsample, apply an FIR filter, and downsample.

Meta Platforms, Inc. - Exhibit 1009
Page 605 of 720

Purpose z-transform partial-fraction expansion.

Syntax

```
[r,p,k] = residuez(b,a)
[b,a] = residuez(r,p,k)
```

Description `residuez` converts a discrete time system, expressed as the ratio of two polynomials, to partial fraction expansion, or residue, form. It also converts the partial fraction expansion back to the original polynomial coefficients.

`[r,p,k] = residuez(b,a)` finds the residues, poles, and direct terms of a partial fraction expansion of the ratio of two polynomials, $b(z)$ and $a(z)$. Vectors `b` and `a` specify the coefficients of the polynomials of the discrete-time system $b(z)/a(z)$ in descending powers of z:

$$b(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_m z^{-m}$$

$$a(z) = a_0 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}$$

If there are no multiple roots and `a` > `n`–`1`,

$$\frac{b(z)}{a(z)} = \frac{r(1)}{1 - p(1)z^{-1}} + \cdots + \frac{r(n)}{1 - p(n)z^{-1}} + k(1) + k(2)z^{-1} + \cdots + k(m - n + 1)z^{-(m-n)}$$

The returned column vector `r` contains the residues, column vector `p` contains the pole locations, and row vector `k` contains the direct terms. The number of poles is

```
n = length(a)–1 = length(r) = length(p)
```

The direct term coefficient vector `k` is empty if `length(b) < length(a)`; otherwise

```
length(k) = length(b) – length(a) + 1
```

If `p(j) = ... = p(j+s–1)` is a pole of multiplicity `s`, then the expansion includes terms of the form

$$\frac{r(j)}{1 - p(j)z^{-1}} + \frac{r(j+1)}{(1 - p(j)z^{-1})^2} + \cdots + \frac{r(j + s_r - 1)}{(1 - p(j)z^{-1})^{s_r}}$$

Meta Platforms, Inc. - Exhibit 1009
Page 606 of 720

`[b,a] = residuez(r,p,k)` with three input arguments and two output arguments, converts the partial fraction expansion back to polynomials with coefficients in row vectors `b` and `a`.

The `residue` function in the standard MATLAB language is very similar to `residuez`. It computes the partial fraction expansion of continuous-time systems in the Laplace domain (see reference [1]), rather than discrete-time systems in the z-domain as does `residuez`.

Algorithm

`residuez` applies standard MATLAB functions and partial fraction techniques to find `r`, `p`, and `k` from `b` and `a`. It finds:

1. The direct terms `a` using `deconv` (polynomial long division) when `length(b)>length(a)-1`.
2. The poles using `p = roots(a)`. `mpoles` finds repeated poles and reorders the poles according to their multiplicities.
3. The residue for each nonrepeating pole p_i by multiplying $b(z)/a(z)$ by $1/(1-p_i z^{-1})$ and evaluating the resulting rational function at $z = p_i$.
4. The residues for the repeated poles by solving

   ```
   S2*r2 = h - S1*r1
   ```

   for `r2` using `\`. `h` is the impulse response of the reduced $b(z)/a(z)$, `S1` is a matrix whose columns are impulse responses of the first-order systems made up of the nonrepeating roots, and `r1` is a column containing the residues for the nonrepeating roots. Each column of matrix `S2` is an impulse response. For each root p_j of multiplicity s_j, `S2` contains s_j columns representing the impulse responses of each of the following systems:

   $$\frac{1}{1-pjz^{-1}}, \frac{1}{(1-pjz^{-1})^2}, \ldots, \frac{1}{(1-pjz^{-1})^{s_j}}$$

The vector `h` and matrices `S1` and `S2` have `n + xtra` rows, where `n` is the total number of roots and the internal parameter `xtra`, set to 1 by default, determines the degree of overdetermination of the system of equations.

Diagnostics

If `a(1) == 0` while computing the partial fraction decomposition using `[r,p,k] = residuez(b,a)`, `residuez` gives the following error message:

```
First coefficient in A vector must be nonzero.
```


Meta Platforms, Inc. - Exhibit 1009
Page 607 of 720

If the number of residues `r` and poles `p` is not the same, `residuez` gives the following error message:

```
R and P vectors must be the same size.
```

See Also

`convmtx`	Convolution matrix.
`deconv`	Deconvolution and polynomial division (see the online *MATLAB Function Reference*).
`poly`	Polynomial with specified roots (see the online *MATLAB Function Reference*).
`prony`	Prony's method for time domain IIR filter design.
`residue`	Partial fraction expansion (see the online *MATLAB Function Reference*).
`roots`	Polynomial roots (see the online *MATLAB Function Reference*).
`ss2tf`	Conversion of state-space to zero-pole-gain.
`tf2ss`	Conversion of transfer function to state-space.
`tf2zp`	Conversion of transfer function to zero-pole-gain.
`zp2ss`	Conversion of zero-pole-gain to state-space.

References

[1] Oppenheim, A.V., and R.W. Schafer. *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1975. Pgs. 166-170.

Meta Platforms, Inc. - Exhibit 1009
Page 608 of 720

Purpose

Reverse Levinson-Durbin recursion.

Syntax

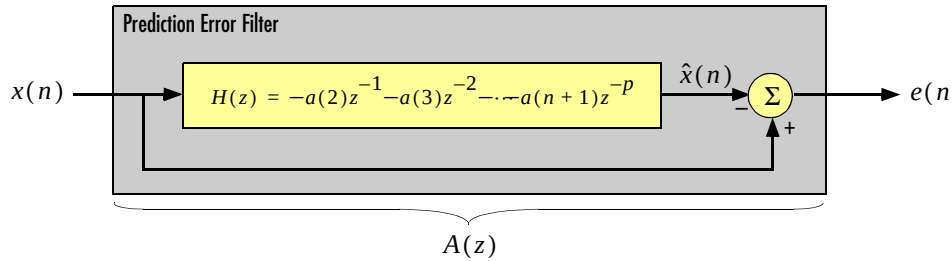
```
r = rlevinson(a,efinal)
[r,u] = rlevinson(a,efinal)
[r,u,k] = rlevinson(a,efinal)
[r,u,k,e] = rlevinson(a,efinal)
```

Description

The reverse Levinson-Durbin recursion implements the step-down algorithm for solving the following symmetric Toeplitz system of linear equations for r, where $r = [r(1) \ \ldots \ r(p+1)]$ and $r(i)^*$ denotes the complex conjugate of $r(i)$.

$$\begin{bmatrix} r(1) & r(2)^* & \cdots & r(p)^* \\ r(2) & r(1) & \cdots & r(p-1)^* \\ \vdots & \ddots & \ddots & \vdots \\ r(p) & \cdots & r(2) & r(1) \end{bmatrix} \begin{bmatrix} a(2) \\ a(3) \\ \vdots \\ a(p+1) \end{bmatrix} = \begin{bmatrix} -r(2) \\ -r(3) \\ \vdots \\ -r(p+1) \end{bmatrix}$$

`r = rlevinson(a,efinal)` solves the above system of equations for r given vector `a`, where `a` = [1 $a(2)$ ... $a(p+1)$]. In linear prediction applications, `r` represents the autocorrelation sequence of the input to the prediction error filter, where `r(1)` is the zero-lag element. The figure below shows the typical filter of this type, where $H(z)$ is the optimal linear predictor, $x(n)$ is the input signal, $\hat{x}(n)$ is the predicted signal, and $e(n)$ is the prediction error.



Input vector `a` represents the polynomial coefficients of this prediction error filter in descending powers of z

$$A(z) = 1 + a(2)z^{-1} + \cdots + a(n+1)z^{-p}$$

Meta Platforms, Inc. - Exhibit 1009
Page 609 of 720

The filter must be minimum phase to generate a valid autocorrelation sequence. `efinal` is the scalar prediction error power, which is equal to the variance of the prediction error signal, $\sigma^2(e)$.

`[r,u] = rlevinson(a,efinal)` returns upper triangular matrix U from the UDU^* decomposition

$$R^{-1} = UE^{-1}U^*$$

where

$$R = \begin{bmatrix} r(1) & r(2)^* & \cdots & r(p)^* \\ r(2) & r(1) & \cdots & r(p-1)^* \\ \vdots & \ddots & \ddots & \vdots \\ r(p) & \cdots & r(2) & r(1) \end{bmatrix}$$

and E is a diagonal matrix with elements returned in output `e` (see below). This decomposition permits the efficient evaluation of the inverse of the autocorrelation matrix, R^{-1}.

Output matrix `u` contains the prediction polynomial, `a`, from each iteration of the reverse Levinson-Durbin recursion

$$U = \begin{bmatrix} a_1(1)^* & a_2(2)^* & \cdots a_{p+1}(p+1)^* \\ 0 & a_2(1)^* & \ddots \quad a_{p+1}(p)^* \\ 0 & 0 & \ddots a_{p+1}(p-1)^* \\ \vdots & \ddots & \ddots \quad \vdots \\ 0 & \cdots & 0 \quad a_{p+1}(1)^* \end{bmatrix}$$

where $a_i(j)$ is the jth coefficient of the ith order prediction polynomial (i.e., step i in the recursion). For example, the 5th order prediction polynomial is

```
a5 = u(5:-1:1,5)'
```

Note that `u(p+1:-1:1,p+1)'` is the input polynomial coefficient vector `a`.

Meta Platforms, Inc. - Exhibit 1009
Page 610 of 720

`[r,u,k] = rlevinson(a,efinal)` returns a vector `k` of length (p+1) containing the reflection coefficients. The reflection coefficients are the conjugates of the values in the first row of `u`

```
k = conj(u(1,2:end))
```

`[r,u,k,e] = rlevinson(a,efinal)` returns a vector of length (p+1) containing the prediction errors from each iteration of the reverse Levinson-Durbin recursion: `e(1)` is the prediction error from the first-order model, `e(2)` is the prediction error from the second-order model, and so on.

These prediction error values form the diagonal of the matrix E in the UDU* decomposition of R^{-1},

$$R^{-1} = UE^{-1}U^{*}.$$

See Also

`levinson`	Levinson-Durbin recursion.
`lpc`	Linear prediction coefficients.
`prony`	Prony's method for time domain IIR filter design.
`stmcb`	Linear model using Steiglitz-McBride iteration.

References

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

Meta Platforms, Inc. - Exhibit 1009
Page 611 of 720

Purpose

Sawtooth or triangle wave generator.

Syntax

```
x = sawtooth(t)
x = sawtooth(t,width)
```

Description

`sawtooth(t)` generates a sawtooth wave with period 2π for the elements of time vector `t`. `sawtooth(t)` is similar to `sin(t)`, but it creates a sawtooth wave with peaks of -1 and 1 instead of a sine wave. The sawtooth wave is defined to be -1 at multiples of 2π and to increase linearly with time with a slope of 1/π at all other times.

`sawtooth(t,width)` generates a modified triangle wave where `width`, a scalar parameter between 0 and 1, determines the fraction between 0 and 2π at which the maximum occurs. The function increases from -1 to 1 on the interval 0 to 2π*`width`, then decreases linearly from 1 to -1 on the interval 2π*`width` to 2π. Thus a parameter of 0.5 specifies a standard triangle wave, symmetric about time instant π with peak-to-peak amplitude of 1. `sawtooth(t,1)` is equivalent to `sawtooth(t)`.

Diagnostics

If the `width` parameter is not a scalar, `sawtooth` gives the following error message:

```
Requires WIDTH parameter to be a scalar.
```

See Also

`chirp`	Swept-frequency cosine generator.
`cos`	Cosine of vector/matrix elements (see the online *MATLAB Function Reference*).
`diric`	Dirichlet or periodic sinc function.
`gauspuls`	Gaussian-modulated sinusoidal pulse generator.
`pulstran`	Pulse train generator.
`rectpuls`	Sampled aperiodic rectangle generator.
`sin`	Sine of vector/matrix elements (see the online *MATLAB Function Reference*).
`sinc`	Sinc or sin(πt)/πt function.
`square`	Square wave generator.
`tripuls`	Sampled aperiodic triangle generator.

Meta Platforms, Inc. - Exhibit 1009
Page 612 of 720

Purpose

Savitzky-Golay filter design.

Syntax

```
b = sgolay(k,f)
b = sgolay(k,f,w)
```

Description

`b = sgolay(k,f)` designs a Savitzky-Golay FIR smoothing filter `b`. The polynomial order `k` must be less than the frame size, `f`, which must be odd. If `k = f–1`, the designed filter produces no smoothing. The output, `b`, is an `f`-by-`f` matrix whose rows represent the time-varying FIR filter coefficients. In a smoothing filter implementation (for example, `sgolayfilt`), the last `(f–1)/2` rows (each an FIR filter) are applied to the signal during the startup transient, and the first `(f–1)/2` rows are applied to the signal during the terminal transient. The center row is applied to the signal in the steady state.

`b = sgolay(k,f,w)` specifies a weighting vector `w` with length `f`, which contains the real, positive-valued weights to be used during the least-squares minimization.

Remarks

Savitzky-Golay smoothing filters (also called digital smoothing polynomial filters or least squares smoothing filters) are typically used to “smooth out” a noisy signal whose frequency span (without noise) is large. In this type of application, Savitzky-Golay smoothing filters perform much better than standard averaging FIR filters, which tend to filter out a significant portion of the signal's high frequency content along with the noise. Although Savitzky-Golay filters are more effective at preserving the pertinent high frequency components of the signal, they are less successful than standard averaging FIR filters at rejecting noise.

Savitzky-Golay filters are optimal in the sense that they minimize the least-squares error in fitting a polynomial to each frame of noisy data.

See Also

`fir1`	Window-based finite impulse response filter design – standard response.
`firls`	Least square linear-phase FIR filter design.
`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
`sgolayfilt`	Savitzky-Golay filtering.

Meta Platforms, Inc. - Exhibit 1009
Page 613 of 720

References

[1] Orfanidis, S.J. *Introduction to Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1996.

Meta Platforms, Inc. - Exhibit 1009
Page 614 of 720

Purpose

Savitzky-Golay filtering.

Syntax

```
y = sgolayfilt(x,k,f)
y = sgolayfilt(x,k,f,w)
```

Description

`y = sgolayfilt(x,k,f)` applies a Savitzky-Golay FIR smoothing filter to the data in vector `x`. If `x` is a matrix, `sgolayfilt` operates on each column. The polynomial order `k` must be less than the frame size, `f`, which must be odd. If `k = f–1`, the filter produces no smoothing.

`y = sgolayfilt(x,k,f,w)` specifies a weighting vector `w` with length `f`, which contains the real, positive-valued weights to be used during the least-squares minimization.

Remarks

Savitzky-Golay smoothing filters (also called digital smoothing polynomial filters or least-squares smoothing filters) are typically used to “smooth out” a noisy signal whose frequency span (without noise) is large. In this type of application, Savitzky-Golay smoothing filters perform much better than standard averaging FIR filters, which tend to filter out a significant portion of the signal’s high frequency content along with the noise. Although Savitzky-Golay filters are more effective at preserving the pertinent high frequency components of the signal, they are less successful than standard averaging FIR filters at rejecting noise.

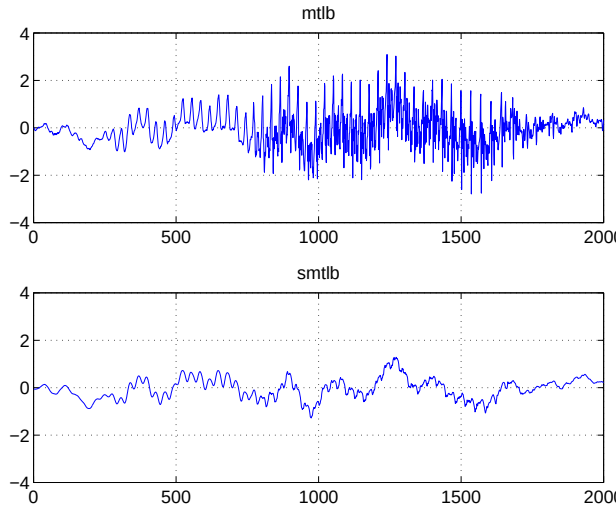
Savitzky-Golay filters are optimal in the sense that they minimize the least-squares error in fitting a polynomial to frames of noisy data.

Meta Platforms, Inc. - Exhibit 1009
Page 615 of 720

Example

Smooth the `mtlb` signal by applying a cubic Savitzky-Golay filter to data frames of length 41.

```
load mtlb                        % load the data
smtlb = sgolayfilt(mtlb,3,41); % apply the 3rd-order filter

subplot(2,1,1)
plot([1:2000],mtlb(1:2000)); axis([0 2000 –4 4]);
title('mtlb'); grid;

subplot(2,1,2)
plot([1:2000],smtlb(1:2000)); axis([0 2000 –4 4]);
title('smtlb'); grid;
```



See Also

`medfilt1`	One-dimensional median filtering.
`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
`sgolay`	Savitzky-Golay filter design.
`sosfilt`	Second-order (biquadratic) IIR filtering.

References

[1] Orfanidis, S.J. *Introduction to Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1996.

Meta Platforms, Inc. - Exhibit 1009
Page 616 of 720

Purpose

Sinc function.

Syntax

```
y = sinc(x)
```

Description

`sinc` computes the sinc function of an input vector or array, where the sinc function is

$$\operatorname{sinc}(t) = \begin{cases} 1, & t = 0 \\ \dfrac{\sin(\pi t)}{\pi t}, & t \neq 0 \end{cases}$$

This function is the continuous inverse Fourier transform of the rectangular pulse of width 2π and height 1:

$$\operatorname{sinc}(t) = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{j\omega t} d\omega$$

`y = sinc(x)` returns an array `y` the same size as `x`, whose elements are the `sinc` function of the elements of `x`.

The space of functions bandlimited in the frequency band $\omega \in [-\pi,\pi]$ is spanned by the infinite (yet countable) set of sinc functions shifted by integers. Thus any such bandlimited function $g(t)$ can be reconstructed from its samples at integer spacings:

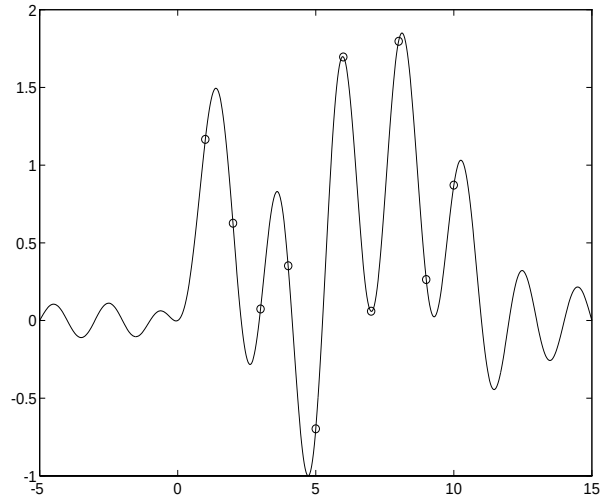
$$g(t) = \sum_{n=-\infty}^{\infty} g(n)\operatorname{sinc}(t-n)$$

Meta Platforms, Inc. - Exhibit 1009
Page 617 of 720

Example

Perform ideal bandlimited interpolation by assuming that the signal to be interpolated is 0 outside of the given time interval and that it has been sampled at exactly the Nyquist frequency:

```
t = (1:10)';            % a column vector of time samples
randn('seed',0);
x = randn(size(t));     % a column vector of data
% ts is times at which to interpolate data
ts = linspace(-5,15,600)';
y = sinc(ts(:,ones(size(t))) - t(:,ones(size(ts)))')*x;
plot(t,x,'o',ts,y)
```



Meta Platforms, Inc. - Exhibit 1009
Page 618 of 720

See Also

`chirp`	Swept-frequency cosine generator.
`cos`	Cosine of vector/matrix elements (see the online *MATLAB Function Reference*).
`diric`	Dirichlet or periodic sinc function.
`gauspuls`	Gaussian-modulated sinusoidal pulse generator.
`pulstran`	Pulse train generator.
`rectpuls`	Sampled aperiodic rectangle generator.
`sawtooth`	Sawtooth or triangle wave generator.
`sin`	Sine of vector/matrix elements (see the online *MATLAB Function Reference*).
`square`	Square wave generator.
`tripuls`	Sampled aperiodic triangle generator.

Meta Platforms, Inc. - Exhibit 1009
Page 619 of 720

Purpose Conversion of second-order sections to state-space.

Syntax

```
[A,B,C,D] = sos2ss(sos)
[A,B,C,D] = sos2ss(sos,g)
```

Description `sos2ss` converts a second-order section representation of a given system to an equivalent state-space representation.

`[A,B,C,D] = sos2ss(sos)` converts the system `sos`, in second-order section form, to a single-input, single-output state-space representation:

$$\begin{aligned} x[n+1] &= Ax[n] + Bu[n] \\ y[n] &= Cx[n] + Du[n] \end{aligned}$$

The discrete transfer function in second-order section form is given by

$$H(z) = \prod_{k=1}^{L} H_k(z) = \prod_{k=1}^{L} \frac{b_{0k} + b_{1k}z^{-1} + b_{2k}z^{-2}}{1 + a_{1k}z^{-1} + a_{2k}z^{-2}}$$

`sos` is a L-by-6 matrix organized as

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

The entries of `sos` must be real for proper conversion to state space. The returned matrix `A` is size N-by-N, where $N = 2L$-1, `B` is a length N-1 column vector, `C` is a length N-1 row vector, and `D` is a scalar.

`[A,B,C,D] = sos2ss(sos,g)` converts the system `sos` in second-order section form with gain `g`:

$$H(z) = g \prod_{k=1}^{L} H_k(z)$$

Meta Platforms, Inc. - Exhibit 1009
Page 620 of 720

Example

Compute the state-space representation of a simple second-order section system with a gain of 2:

```
sos = [1  1  1  1  0 -1; -2  3  1  1 10  1];
[A,B,C,D] = sos2ss(sos)

A =
   -10     0    10     1
     1     0     0     0
     0     1     0     0
     0     0     1     0

B =
     1
     0
     0
     0

C =
    21     2   -16    -1

D =
    -2
```

Algorithm

`sos2ss` first converts from second-order sections to transfer function using `sos2tf`, and then from transfer function to state-space using `tf2ss`.

See Also

`sos2tf`	Conversion of second-order sections to transfer function.
`sos2zp`	Conversion of second-order sections to zero-pole-gain.
`ss2sos`	Conversion of state-space to second-order sections.
`tf2ss`	Conversion of transfer function to state-space.
`zp2ss`	Conversion of zero-pole-gain to state-space.

Meta Platforms, Inc. - Exhibit 1009
Page 621 of 720

Purpose Conversion of second-order sections to transfer function.

Syntax

```
[b,a] = sos2tf(sos)
[b,a] = sos2tf(sos,g)
```

Description `sos2tf` converts a second-order section representation of a given system to an equivalent transfer function representation.

`[b,a] = sos2tf(sos)` returns the numerator coefficients `b` and denominator coefficients `a` of the transfer function that describes a discrete-time system given by `sos` in second-order section form. The second-order section format of $H(z)$ is given by

$$H(z) = \prod_{k=1}^{L} H_k(z) = \prod_{k=1}^{L} \frac{b_{0k} + b_{1k}z^{-1} + b_{2k}z^{-2}}{1 + a_{1k}z^{-1} + a_{2k}z^{-2}}$$

`sos` is an L-by-6 matrix that contains the coefficients of each second-order section stored in its rows:

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

Row vectors `b` and `a` contain the numerator and denominator coefficients of $H(z)$ stored in descending powers of z:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2z^{-1} + \cdots + b_{n+1}z^{-n}}{a_1 + a_2z^{-1} + \cdots + a_{m+1}z^{-m}}$$

`[b,a] = sos2tf(sos,g)` returns the transfer function that describes a discrete-time system given by `sos` in second-order section form with gain `g`:

$$H(z) = g \prod_{k=1}^{L} H_k(z)$$

Meta Platforms, Inc. - Exhibit 1009
Page 622 of 720

Algorithm

`sos2tf` uses the `conv` function to multiply all of the numerator and denominator second-order polynomials together.

Example

Compute the transfer function representation of a simple second-order section system:

```
sos = [1  1  1  1  0 –1; –2  3  1  1 10  1];
[b,a] = sos2tf(sos)

b =
   –2     1     2     4     1

a =
    1    10     0   –10    –1
```

See Also

`latc2tf`	Lattice filter to transfer function conversion.
`sos2ss`	Conversion of second-order sections to state-space.
`sos2zp`	Conversion of second-order sections to zero-pole-gain.
`ss2tf`	Conversion of state-space to transfer function.
`tf2sos`	Conversion of transfer function to second-order sections.
`zp2tf`	Conversion of zero-pole-gain to transfer function.

Meta Platforms, Inc. - Exhibit 1009
Page 623 of 720

Purpose Conversion of second-order sections to zero-pole-gain.

Syntax

```
[z,p,k] = sos2zp(sos)
[z,p,k] = sos2zp(sos,g)
```

Description `sos2zp` converts a second-order section representation of a given system to an equivalent zero-pole-gain representation.

`[z,p,k] = sos2zp(sos)` returns the zeros `z`, poles `p`, and gain `k` of the system given by `sos` in second-order section form. The second-order section format of $H(z)$ is given by

$$H(z) = \prod_{k=1}^{L} H_k(z) = \prod_{k=1}^{L} \frac{b_{0k} + b_{1k}z^{-1} + b_{2k}z^{-2}}{1 + a_{1k}z^{-1} + a_{2k}z^{-2}}$$

`sos` is an L-by-6 matrix that contains the coefficients of each second-order section stored in its rows:

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

Column vectors `z` and `p` contain the zeros and poles of the transfer function $H(z)$:

$$H(z) = k\frac{(z - z_1)(z - z_2)\cdots(z - z_n)}{(p - p_1)(p - p_2)\cdots(p - p_m)}$$

where the orders n and m are determined by the matrix `sos`.

`[z,p,k] = sos2zp(sos,g)` returns the zeros `z`, poles `p`, and gain `k` of the system given by `sos` in second-order section form with gain `g`:

$$H(z) = g\prod_{k=1}^{L} H_k(z)$$

Meta Platforms, Inc. - Exhibit 1009
Page 624 of 720

Example

Compute the poles, zeros, and gain of a simple system in second-order section form:

```
sos = [1  1  1  1  0 -1; -2  3  1  1 10  1];
[z,p,k] = sos2zp(sos)

z =
  -0.5000 + 0.8660i
  -0.5000 - 0.8660i
   1.7808
  -0.2808

p =
   -1.0000
    1.0000
   -9.8990
   -0.1010

k
     -2
```

Algorithm

`sos2zp` finds the poles and zeros of each second-order section by repeatedly calling `tf2zp`.

See Also

`sos2ss`	Conversion of second-order sections to state-space.
`sos2tf`	Conversion of second-order sections to transfer function.
`ss2zp`	Conversion of state-space to zero-pole-gain.
`tf2zp`	Conversion of transfer function to zero-pole-gain.
`zp2sos`	Conversion of zero-pole-gain to second-order sections.

Meta Platforms, Inc. - Exhibit 1009
Page 625 of 720

Purpose

Second-order (biquadratic) IIR filtering.

Syntax

```
y = sosfilt(sos,x)
```

Description

`y = sosfilt(sos,x)` applies the second-order section filter `sos` to vector `x`. The output, `y`, is the same length as `x`.

`sos` represents the second-order section filter $H(z)$

$$H(z) = \prod_{k=1}^{L} H_k(z) = \prod_{k=1}^{L} \frac{b_{0k} + b_{1k}z^{-1} + b_{2k}z^{-2}}{1 + a_{1k}z^{-1} + a_{2k}z^{-2}}$$

by an L-by-6 matrix containing the coefficients of each second-order section in its rows

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

If `x` is a matrix, `sosfilt` applies the filter to each column of `x` independently. Output `y` is a matrix of the same size, containing the filtered data corresponding to each column of `x`.

See Also

`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
`medfilt1`	One-dimensional median filtering.
`sgolayfilt`	Savitzky-Golay filtering.

References

[1] Orfanidis, S.J. *Introduction to Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1996.

Meta Platforms, Inc. - Exhibit 1009
Page 626 of 720

Purpose

Time-dependent frequency analysis (spectrogram).

Syntax

```
B = specgram(a)
B = specgram(a,nfft)
[B,f] = specgram(a,nfft,Fs)
[B,f,t] = specgram(a,nfft,Fs)
B = specgram(a,nfft,Fs,window)
B = specgram(a,nfft,Fs,window,noverlap)
specgram(a)
B = specgram(a,f,Fs,window,noverlap)
```

Description

`specgram` computes the windowed discrete-time Fourier transform of a signal using a sliding window. The spectrogram is the magnitude of this function.

`B = specgram(a)` calculates the spectrogram for the signal in vector `a`. This syntax uses the default values:

- `nfft = min(256,length(a))`
- `Fs = 2`
- `window = hanning(nfft)`
- `noverlap = length(window)/2`

`nfft` specifies the FFT length that `specgram` uses. This value determines the frequencies at which the discrete-time Fourier transform is computed. `Fs` is a scalar that specifies the sampling frequency. `window` specifies a windowing function and the number of samples `specgram` uses in its sectioning of vector `a`. `noverlap` is the number of samples by which the sections overlap. Any arguments that you omit from the end of the input parameter list use the default values shown above.

If `a` is real, `specgram` computes the discrete-time Fourier transform at positive frequencies only. If `n` is even, `specgram` returns `nfft/2+1` rows (including the zero and Nyquist frequency terms). If `n` is odd, `specgram` returns `nfft/2` rows. The number of columns in `B` is

```
k = fix((n–noverlap)/(length(window)–noverlap))
```

If `a` is complex, `specgram` computes the discrete-time Fourier transform at both positive and negative frequencies. In this case, `B` is a complex matrix with `nfft`

Meta Platforms, Inc. - Exhibit 1009
Page 627 of 720

rows. Time increases linearly across the columns of `B`, starting with sample 1 in column 1. Frequency increases linearly down the rows, starting at 0.

`B = specgram(a,nfft)` uses the specified FFT length `nfft` in its calculations. Specify `nfft` as a power of 2 for fastest execution.

`[B,f] = specgram(a,nfft,Fs)` returns a vector `f` of frequencies at which the function computes the discrete-time Fourier transform. `Fs` has no effect on the output `B`; it is a frequency scaling multiplier.

`[B,f,t] = specgram(a,nfft,Fs)` returns frequency and time vectors `f` and `t` respectively. `t` is a column vector of scaled times, with length equal to the number of columns of `B`. `t(j)` is the earliest time at which the *j*-th window intersects `a`. `t(1)` is always equal to 0.

`B = specgram(a,nfft,Fs,window)` specifies a windowing function and the number of samples per section of the `x` vector. If you supply a scalar for `window`, `specgram` uses a Hanning window of that length. The length of the window must be less than or equal to `nfft`; `specgram` zero pads the sections if the length of the window exceeds `nfft`.

`B = specgram(a,nfft,Fs,window,noverlap)` overlaps the sections of `x` by `noverlap` samples.

You can use the empty matrix `[]` to specify the default value for any input argument. For example,

```
B = specgram(x,[],10000)
```

is equivalent to

```
B = specgram(x)
```

but with a sampling frequency of 10,000 Hz instead of the default 2 Hz.

`specgram` with no output arguments displays the scaled logarithm of the spectrogram in the current figure window using

```
imagesc(t,f,20*log10(abs(b))),axis xy,colormap(jet)
```


Meta Platforms, Inc. - Exhibit 1009
Page 628 of 720

The `axis xy` mode displays the low-frequency content of the first portion of the signal in the lower-left corner of the axes. `specgram` uses `Fs` to label the axes according to true time and frequency.

`B = specgram(a,f,Fs,window,noverlap)` computes the spectrogram at the frequencies specified in `f`, using either the chirp *z*-transform (for more than 20 evenly spaced frequencies) or a polyphase decimation filter bank. `f` is a vector of frequencies in Hertz; it must have at least two elements.

Algorithm

`specgram` calculates the spectrogram for a given signal as follows:

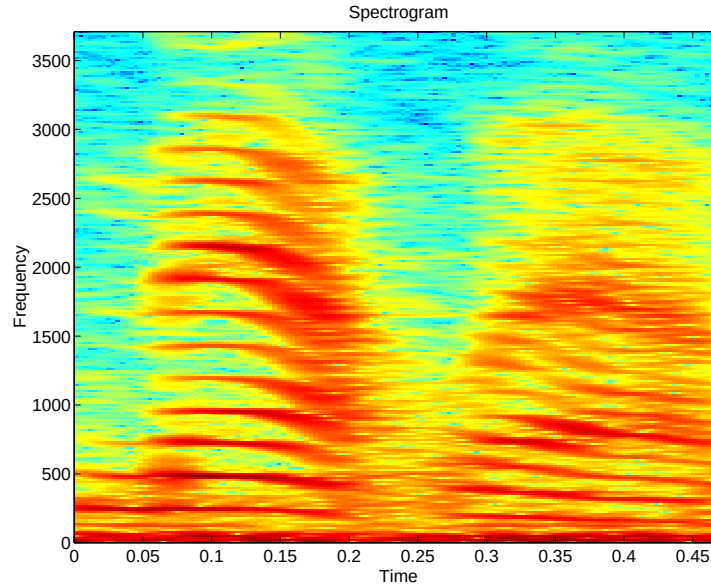
1. It splits the signal into overlapping sections and applies the window specified by the `window` parameter to each section.
2. It computes the discrete-time Fourier transform of each section with a length `nfft` FFT to produce an estimate of the short-term frequency content of the signal; these transforms make up the columns of `B`. `specgram` zero pads the windowed sections if `nfft > length(window)`, so the quantity `(length(window) – noverlap)` specifies by how many samples `specgram` shifts the window.
3. For real input, `specgram` truncates the spectrogram to the first `nfft/2 + 1` points for `nfft` even and `(nfft + 1)/2` for `nfft` odd.

Meta Platforms, Inc. - Exhibit 1009
Page 629 of 720

Example

Plot the spectrogram of a digitized speech signal:

```
load mtlb
specgram(mtlb,512,Fs,kaiser(500,5),475)
title('Spectrogram')
```



Diagnostics

An appropriate diagnostic message is displayed when incorrect arguments are used:

```
Requires window's length to be no greater than the FFT length.
Requires NOVERLAP to be strictly less than the window length.
Requires positive integer values for NFFT and NOVERLAP.
Requires vector input.
```

See Also

`cohere`	Estimate magnitude squared coherence function between two signals.
`csd`	Estimate the cross spectral density (CSD) of two signals.
`pwelch`	Estimate the power spectral density (PSD) of a signal using Welch's method.
`tfe`	Transfer function estimate from input and output.

Meta Platforms, Inc. - Exhibit 1009
Page 630 of 720

References

[1] Oppenheim, A.V., and R.W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989. Pgs. 713-718.

[2] Rabiner, L.R., and R.W. Schafer. *Digital Processing of Speech Signals*. Englewood Cliffs, NJ: Prentice Hall, 1978.

Meta Platforms, Inc. - Exhibit 1009
Page 631 of 720

Purpose Interactive digital signal processing tool (SPTool).

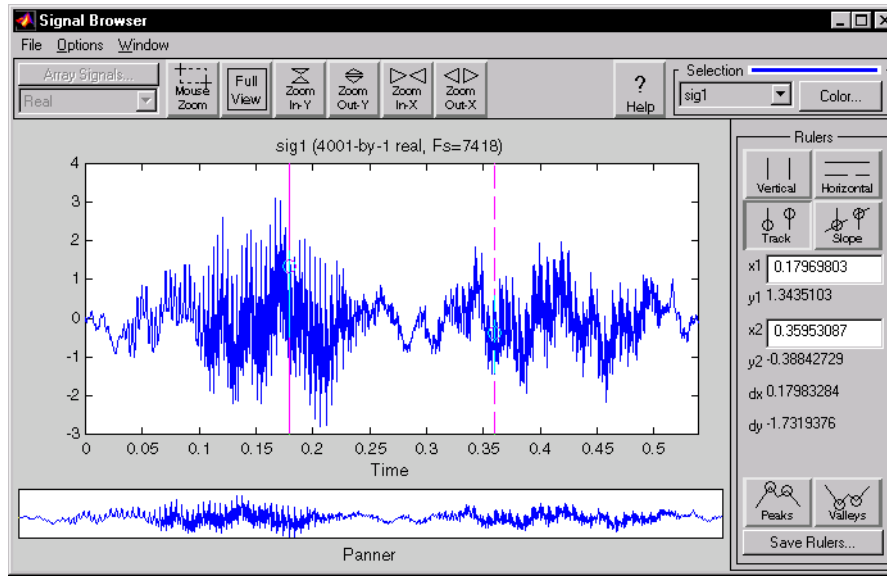
Syntax

```
sptool
```

Description The `sptool` command invokes a suite of graphical user interface (GUI) tools that provides access to many of the signal, filter, and spectral analysis functions in the toolbox in a powerful, easy-to-use interactive signal display and exploration environment.

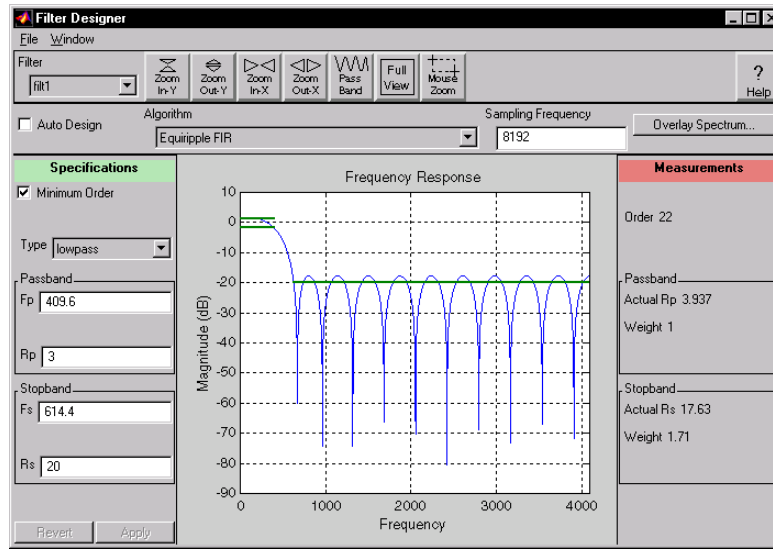
Using SPTool, you can import, export, and manage signals, filters, and spectra. From SPTool, you can activate its four integrated signal processing tools:

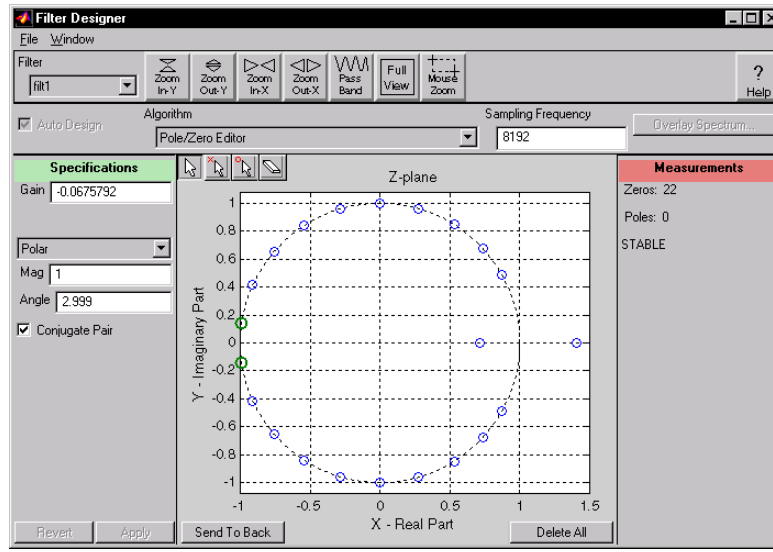
- The *Signal Browser*, for viewing, measuring, and analyzing the time-domain information of one or more signals.



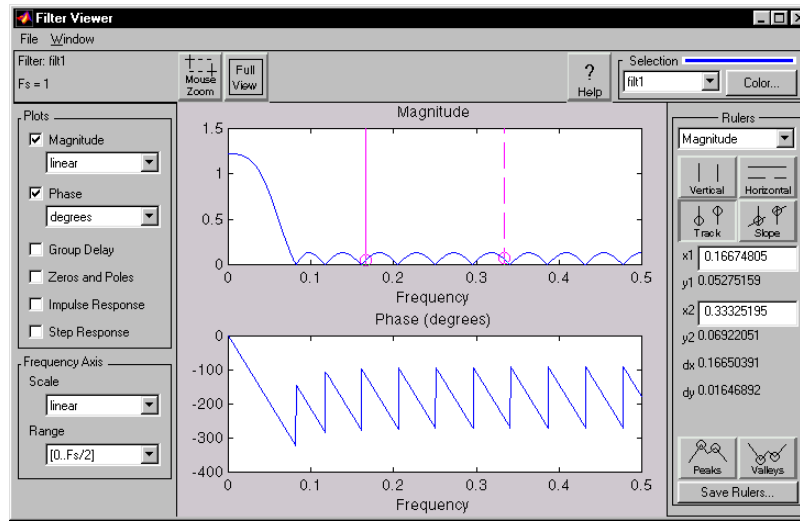
Meta Platforms, Inc. - Exhibit 1009
Page 632 of 720

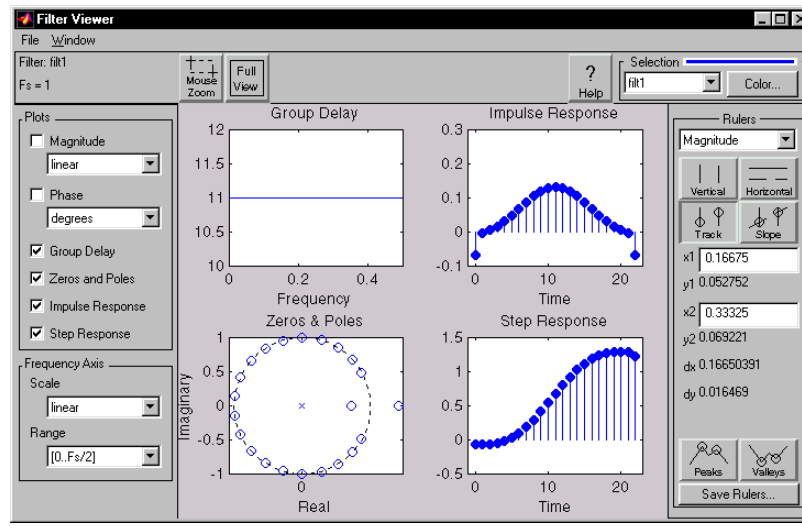
- The *Filter Designer*, for designing and editing FIR and IIR filters of various lengths and types, with standard (lowpass, highpass, bandpass, bandstop, and multiband) configurations.





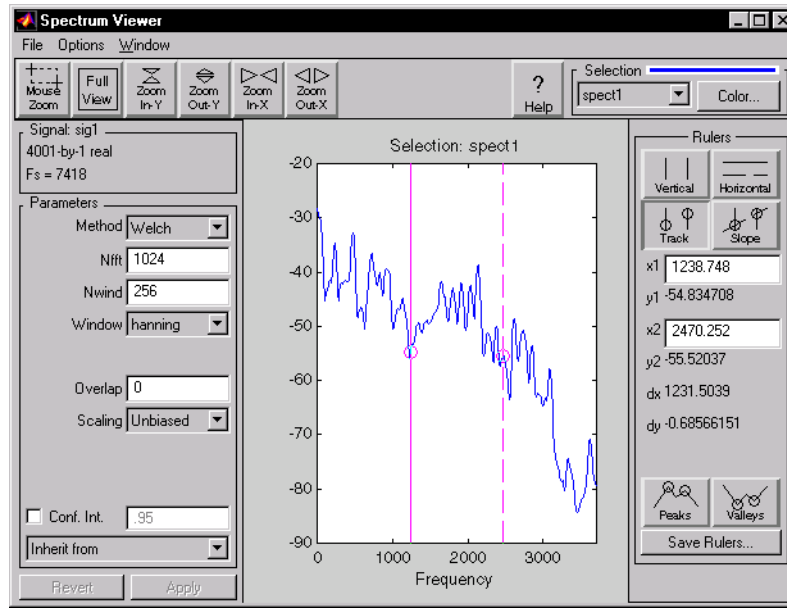
- The *Filter Viewer*, for viewing the characteristics of a designed or imported filter, including its magnitude response, phase response, group delay, pole-zero plot, impulse response, and step response.





Meta Platforms, Inc. - Exhibit 1009
Page 634 of 720

- The *Spectrum Viewer*, for graphical analysis of frequency-domain data using a variety of methods of spectral density estimation, including the Burg method, the FFT method, the multitaper method (MTM), the MUSIC eigenvector method, Welch's method, and the Yule-Walker AR method.



See Chapter 5, "Interactive Tools" for a full discussion of how to use SPTool.

Meta Platforms, Inc. - Exhibit 1009
Page 635 of 720

Purpose

Square wave generator.

Syntax

```
x = square(t)
x = square(t,duty)
```

Description

`x = square(t)` generates a square wave with period 2π for the elements of time vector `t`. `square(t)` is similar to `sin(t)`, but it creates a square wave with peaks of ±1 instead of a sine wave.

`x = square(t,duty)` generates a square wave with specified duty cycle, `duty`. The *duty cycle* is the percent of the period in which the signal is positive.

See Also

`chirp`	Swept-frequency cosine generator.
`cos`	Cosine of vector/matrix elements (see the online *MATLAB Function Reference*).
`diric`	Dirichlet or periodic sinc function.
`gauspuls`	Gaussian-modulated sinusoidal pulse generator.
`pulstran`	Pulse train generator.
`rectpuls`	Sampled aperiodic rectangle generator.
`sawtooth`	Sawtooth or triangle wave generator.
`sin`	Sine of vector/matrix elements (see the online *MATLAB Function Reference*).
`sinc`	Sinc or $\sin(\pi t)/\pi t$ function.
`tripuls`	Sampled aperiodic triangle generator.

Meta Platforms, Inc. - Exhibit 1009
Page 636 of 720

Purpose

Conversion of state-space to second-order sections.

Syntax

```
[sos,g] = ss2sos(A,B,C,D)
[sos,g] = ss2sos(A,B,C,D,iu)
[sos,g] = ss2sos(A,B,C,D,'order')
[sos,g] = ss2sos(A,B,C,D,iu,'order')
[sos,g] = ss2sos(A,B,C,D,iu,'order','scale')
sos = ss2sos(...)
```

Description

`ss2sos` converts a state-space representation of a given system to an equivalent second-order section representation.

`[sos,g] = ss2sos(A,B,C,D)` finds a matrix `sos` in second-order section form with gain `g` that is equivalent to the state-space system represented by input arguments `A`, `B`, `C` and `D`. The input system must be single output and real. `sos` is an L-by-6 matrix

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

whose rows contain the numerator and denominator coefficients b_{ik} and a_{ik} of the second-order sections of $H(z)$:

$$H(z) = g \prod_{k=1}^{L} H_k(z) = g \prod_{k=1}^{L} \frac{b_{0k} + b_{1k}z^{-1} + b_{2k}z^{-2}}{1 + a_{1k}z^{-1} + a_{2k}z^{-2}}$$

`[sos,g] = ss2sos(A,B,C,D,iu)` specifies a scalar `iu` that determines which input of the state-space system `A`, `B`, `C`, `D` is used in the conversion. The default for `iu` is 1.

`[sos,g] = ss2sos(A,B,C,D,'order')` and

`[sos,g] = ss2sos(A,B,C,D,iu,'order')` specify the order of the rows in `sos`, where *`order`* is:

Meta Platforms, Inc. - Exhibit 1009
Page 637 of 720

- down, to order the sections so the first row of sos contains the poles closest to the unit circle
- up, to order the sections so the first row of sos contains the poles farthest from the unit circle (default)

The zeros are always paired with the poles closest to them.

`[sos,g] = ss2sos(A,B,C,D,iu,'order','scale')` specifies the desired scaling of the gain and the numerator coefficients of all second-order sections, where *scale* is:

- none, to apply no scaling (default)
- inf, to apply infinity-norm scaling
- two, to apply 2-norm scaling

Using infinity-norm scaling in conjunction with up-ordering minimizes the probability of overflow in the realization. Using 2-norm scaling in conjunction with down-ordering minimizes the peak round-off noise.

`sos = ss2sos(...)` embeds the overall system gain, g, in the first section, $H_1(z)$, so that:

$$H(z) = \prod_{k=1}^{L} H_k(z)$$

Example

Find a second-order section form of a Butterworth lowpass filter:

```
[A,B,C,D] = butter(5,0.2);
sos = ss2sos(A,B,C,D)

sos =
    0.0013    0.0013         0    1.0000   -0.5095         0
    1.0000    2.0008    1.0008    1.0000   -1.0966    0.3554
    1.0000    1.9979    0.9979    1.0000   -1.3693    0.6926
```

Algorithm

ss2sos uses a four-step algorithm to determine the second-order section representation for an input state-space system:

Meta Platforms, Inc. - Exhibit 1009
Page 638 of 720

1. It finds the poles and zeros of the system given by `A`, `B`, `C` and `D`.
2. It uses the function `zp2sos`, which first groups the zeros and poles into complex conjugate pairs using the `cplxpair` function. `zp2sos` then forms the second-order sections by matching the pole and zero pairs according to the following rules:
   a. Match the poles closest to the unit circle with the zeros closest to those poles.
   b. Match the poles next closest to the unit circle with the zeros closest to those poles.
   c. Continue until all of the poles and zeros are matched.

   `ss2sos` groups real poles into sections with the real poles closest to them in absolute value. The same rule holds for real zeros.
3. It orders the sections according to the proximity of the pole pairs to the unit circle. `ss2sos` normally orders the sections with poles closest to the unit circle last in the cascade. You can tell `ss2sos` to order the sections in the reverse order by specifying the `'down'` flag.
4. `ss2sos` scales the sections by the norm specified in the `'scale'` argument. For arbitrary $H(\omega)$, the scaling is defined by:

$$\|H\|_p = \left[\frac{1}{2\pi} \int_0^{2\pi} |H(\omega)|^p \, d\omega \right]^{\frac{1}{p}}$$

   where p can be either ∞ or 2. See the references for details on the scaling.

   This scaling is an attempt to minimize overflow or peak round-off noise in fixed point filter implementations.

Diagnostics

If there is more than one input to the system, `ss2sos` gives the following error message:

```
State-space system must have only one input.
```


Meta Platforms, Inc. - Exhibit 1009
Page 639 of 720

See Also

`cplxpair`	Group complex numbers into complex conjugate pairs.
`sos2ss`	Conversion of second-order sections to state-space.
`ss2tf`	Conversion of state-space to transfer function.
`ss2zp`	Conversion of state-space to zero-pole-gain.
`tf2sos`	Conversion of transfer function to second-order sections.
`zp2sos`	Conversion of zero-pole-gain to second-order sections.

References

[1] Jackson, L.B. *Digital Filters and Signal Processing*. 3rd ed. Boston: Kluwer Academic Publishers, 1996. Chapter 11.

[2] Mitra, S.K. *Digital Signal Processing: A Computer-Based Approach*. New York: McGraw-Hill, 1998. Chapter 9.

[3] Vaidyanathan, P.P. “Robust Digital Filter Structures.” *Handbook for Digital Signal Processing*. S.K. Mitra and J.F. Kaiser, ed. Chapter 7. New York: John Wiley & Sons, 1993.

Meta Platforms, Inc. - Exhibit 1009
Page 640 of 720

Purpose

Conversion of state-space to transfer function.

Syntax

```
[b,a] = ss2tf(A,B,C,D,iu)
```

Description

`ss2tf` converts a state-space representation of a given system to an equivalent transfer function representation.

`[b,a] = ss2tf(A,B,C,D,iu)` returns the transfer function

$$H(s) = \frac{B(s)}{A(s)} = C(sI - A)^{-1}B + D$$

of the system

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

from the `iu`-th input. Vector `a` contains the coefficients of the denominator in descending powers of s. The numerator coefficients are returned in array `b` with as many rows as there are outputs y. `ss2tf` also works with systems in discrete time, in which case it returns the z-transform representation.

The `ss2tf` function is part of the standard MATLAB language.

Algorithm

The `ss2tf` function uses `poly` to find the characteristic polynomial det(sI-A) and the equality

$$H(s) = C(sI - A)^{-1}B = \frac{\det(sI - A + BC) - \det(sI - A)}{\det(sI - A)}$$

See Also

`latc2tf`	Lattice filter to transfer function conversion.
`sos2tf`	Conversion of second-order sections to transfer function.
`ss2sos`	Conversion of state-space to second-order sections.
`ss2zp`	Conversion of state-space to zero-pole-gain.
`tf2ss`	Conversion of transfer function to state-space.
`zp2tf`	Conversion of zero-pole-gain to transfer function.

Meta Platforms, Inc. - Exhibit 1009
Page 641 of 720

Purpose Conversion of state-space to zero-pole-gain.

Syntax

```
[z,p,k] = ss2zp(A,B,C,D,iu)
```

Description `ss2zp` converts a state-space representation of a given system to an equivalent zero-pole-gain representation. The zeros, poles, and gains of state-space systems represent the transfer function in factored form.

`[z,p,k] = ss2zp(A,B,C,D,iu)` calculates the transfer function in factored form

$$H(s) = \frac{Z(s)}{P(s)} = k\frac{(s-z_1)(s-z_2)\cdots(s-z_n)}{(s-p_1)(s-p_2)\cdots(s-p_n)}$$

of the system

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

from the `iu`-th input. Returned column vector `p` contains the pole locations of the denominator coefficients of the transfer function. Matrix `z` contains the numerator zeros in its columns, with as many columns as there are outputs y. Column vector `k` contains the gains for each numerator transfer function.

`ss2zp` also works with systems in discrete time, in which case it returns the z-transform representation. The input state-space system must be real.

The `ss2zp` function is part of the standard MATLAB language.

Meta Platforms, Inc. - Exhibit 1009
Page 642 of 720

Example

Here are two ways of finding the zeros, poles, and gains of a system:

```
b = [2 3];
a = [1 0.4 1];
[z,p,k] = tf2zp(b,a)

z =
    -1.5000

p =
    -0.2000 + 0.9798i
    -0.2000 - 0.9798i

k =
    2

[A,B,C,D] = tf2ss(b,a);
[z,p,k] = ss2zp(A,B,C,D,1)

z =
    -1.5000

p =
    -0.2000 + 0.9798i
    -0.2000 - 0.9798i

k =
    2
```

Algorithm

ss2zp finds the poles from the eigenvalues of the A array. The zeros are the finite solutions to a generalized eigenvalue problem:

```
z = eig([A B;C D], diag([ones(1,n) 0]);
```

In many situations this algorithm produces spurious large, but finite, zeros. ss2zp interprets these large zeros as infinite.

ss2zp finds the gains by solving for the first nonzero Markov parameters.

Meta Platforms, Inc. - Exhibit 1009
Page 643 of 720

See Also

`pzmap`	Pole-zero map of LTI system (*Control System Toolbox User's Guide*).
`sos2zp`	Conversion of second-order sections to zero-pole-gain.
`ss2sos`	Conversion of state-space to second-order sections.
`ss2tf`	Conversion of state-space to transfer function.
`tf2zp`	Conversion of transfer function to zero-pole-gain.
`zp2ss`	Conversion of zero-pole-gain to state-space.

References

[1] Laub, A.J., and B.C. Moore. "Calculation of Transmission Zeros Using QZ Techniques." *Automatica* 14 (1978). Pg. 557.

Meta Platforms, Inc. - Exhibit 1009
Page 644 of 720

Purpose

Linear model using Steiglitz-McBride iteration.

Syntax

```
[b,a] = stmcb(x,nb,na)
[b,a] = stmcb(x,u,nb,na)
[b,a] = stmcb(x,nb,na,niter)
[b,a] = stmcb(x,u,nb,na,niter)
[b,a] = stmcb(x,nb,na,niter,ai)
[b,a] = stmcb(x,u,nb,na,niter,ai)
```

Description

Steiglitz-McBride iteration is an algorithm for finding an IIR filter with a prescribed time domain impulse response. It has applications in both filter design and system identification (parametric modeling).

`[b,a] = stmcb(x,nb,na)` finds the coefficients `b` and `a` of the system $b(z)/a(z)$ with approximate impulse response `x`, exactly `nb` zeros, and exactly `na` poles.

`[b,a] = stmcb(x,u,nb,na)` finds the system coefficients `b` and `a` of the system that, given `u` as input, has `x` as output. `x` and `u` must be the same length.

`[b,a] = stmcb(x,nb,na,niter)` and

`[b,a] = stmcb(x,u,nb,na,niter)` use `niter` iterations. The default for `niter` is 5.

`[b,a] = stmcb(x,nb,na,niter,ai)` and

`[b,a] = stmcb(x,u,nb,na,niter,ai)` use the vector `ai` as the initial estimate of the denominator coefficients. If `ai` is not specified, `stmcb` uses the output argument from `[b,ai] = prony(x,0,na)` as the vector `ai`.

`stmcb` returns the IIR filter coefficients in length `nb+1` and `na+1` row vectors `b` and `a`. The filter coefficients are ordered in descending powers of z:

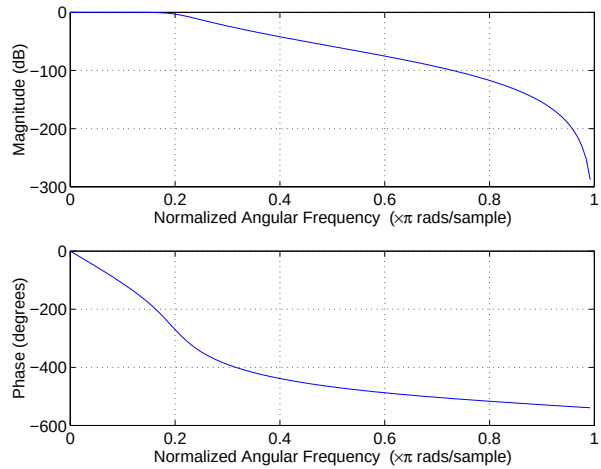
$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + \cdots + b(nb+1)z^{-nb}}{a(1) + a(2)z^{-1} + \cdots + a(na+1)z^{-na}}$$

Meta Platforms, Inc. - Exhibit 1009
Page 645 of 720

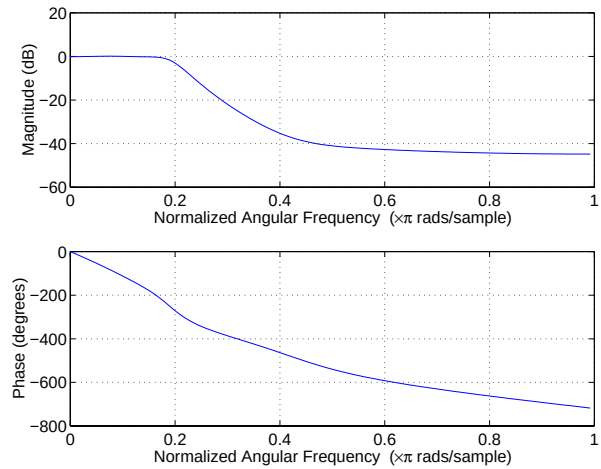
Example

Approximate the impulse response of a Butterworth filter with a system of lower order:

```
[b,a] = butter(6,0.2);
h = filter(b,a,[1 zeros(1,100)]);
freqz(b,a,128)
```




```
[bb,aa] = stmcb(h,4,4);
freqz(bb,aa,128)
```



Meta Platforms, Inc. - Exhibit 1009
Page 646 of 720

Algorithm

stmcb attempts to minimize the squared error between the impulse response x' of $b(z)/a(z)$ and the input signal x:

$$\min_{a,\,b} \sum_{i=0}^{\infty} |x(i) - x'(i)|^2$$

stmcb iterates using two steps:

1. It prefilters x and u using $1/a(z)$.
2. It solves a system of linear equations for b and a using \.

stmcb repeats this process niter times. No checking is done to see if the b and a coefficients have converged in fewer than niter iterations.

Diagnostics

If x and u have different lengths, stmcb gives the following error message:

```
X and U must have same length.
```

See Also

levinson	Levinson-Durbin recursion.
lpc	Linear prediction coefficients.
aryule	Compute an estimate of AR model parameters using the Yule-Walker method.
prony	Prony's method for time domain IIR filter design.

References

[1] Steiglitz, K., and L.E. McBride. "A Technique for the Identification of Linear Systems." *IEEE Trans. Automatic Control*. Vol. AC-10 (1965). Pgs. 461-464.

[2] Ljung, L. *System Identification: Theory for the User*. Englewood Cliffs, NJ: Prentice Hall, 1987. Pg. 297.

Meta Platforms, Inc. - Exhibit 1009
Page 647 of 720

Purpose

Strip plot.

Syntax

```
strips(x)
strips(x,n)
strips(x,sd,Fs)
strips(x,sd,Fs,scale)
```

Description

`strips(x)` plots vector `x` in horizontal strips of length 250. If `x` is a matrix, `strips(x)` plots each column of `x`. The left-most column (column 1) is the top horizontal strip.

`strips(x,n)` plots vector `x` in strips that are each `n` samples long.

`strips(x,sd,Fs)` plots vector `x` in strips of duration `sd` seconds, given a sampling frequency of `Fs` samples per second.

`strips(x,sd,Fs,scale)` scales the vertical axes.

If `x` is a matrix, `strips(x,n)`, `strips(x,sd,Fs)`, and `strips(x,sd,Fs,scale)` plot the different columns of `x` on the same strip plot.

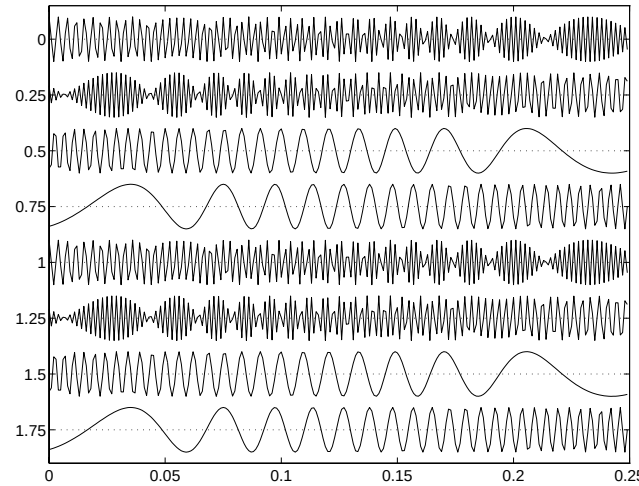
`strips` ignores the imaginary part of `x` if it is complex.

Meta Platforms, Inc. - Exhibit 1009
Page 648 of 720

Example

Plot two seconds of a frequency modulated sinusoid in 0.25 second strips:

```
Fs = 1000;                          % sampling frequency
t = 0:1/Fs:2;                       % time vector
x = vco(sin(2*pi*t),[10 490],Fs);   % FM waveform
strips(x,0.25,Fs)
```



See Also

`plot`	Linear two-dimensional plot (see the online *MATLAB Function Reference*).
`stem`	Plot discrete sequence data (see the online *MATLAB Function Reference*).

Meta Platforms, Inc. - Exhibit 1009
Page 649 of 720

Purpose

Conversion of transfer function to lattice filter.

Syntax

```
[k,v] = tf2latc(b,a)
k = tf2latc(1,a)
[k,v] = tf2latc(1,a)
k = tf2latc(b)
```

Description

`[k,v] = tf2latc(b,a)` finds the lattice parameters `k` and the ladder parameters `v` for an IIR (ARMA) lattice-ladder filter, normalized by `a(1)`. Note that an error is generated if one or more of the lattice parameters are exactly equal to 1.

`k = tf2latc(1,a)` finds the lattice parameters `k` for an IIR all-pole (AR) lattice filter.

`[k,v] = tf2latc(1,a)` returns the scalar ladder coefficient at the correct position in vector `v`. All other elements of `v` are zero.

`k = tf2latc(b)` finds the lattice parameters `k` for an FIR (MA) lattice filter, normalized by `b(1)`.

See Also

`latc2tf`	Lattice filter to transfer function conversion.
`latcfilt`	Lattice and lattice-ladder filter implementation.
`tf2sos`	Conversion of transfer function to second-order sections.
`tf2ss`	Conversion of transfer function to state-space.
`tf2zp`	Conversion of transfer function to zero-pole-gain.

Meta Platforms, Inc. - Exhibit 1009
Page 650 of 720

Purpose

Conversion of transfer function to second-order sections.

Syntax

```
[sos,g] = tf2sos(b,a)
[sos,g] = tf2sos(b,a,'order')
[sos,g] = tf2sos(b,a,'order','scale')
sos = tf2sos(...)
```

Description

`tf2sos` converts a transfer function representation of a given system to an equivalent second-order section representation.

`[sos,g] = tf2sos(b,a)` finds a matrix `sos` in second-order section form with gain `g` that is equivalent to the system represented by transfer function coefficient vectors `a` and `b`.

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2 z^{-1} + \cdots + b_{nb+1} z^{-nb}}{a_1 + a_2 z^{-1} + \cdots + a_{na+1} z^{-na}}$$

`sos` is an L-by-6 matrix

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

whose rows contain the numerator and denominator coefficients b_{ik} and a_{ik} of the second-order sections of $H(z)$:

$$H(z) = g \prod_{k=1}^{L} H_k(z) = g \prod_{k=1}^{L} \frac{b_{0k} + b_{1k} z^{-1} + b_{2k} z^{-2}}{1 + a_{1k} z^{-1} + a_{2k} z^{-2}}$$

`[sos,g] = tf2sos(b,a,'order')` specifies the order of the rows in `sos`, where *`order`* is:

Meta Platforms, Inc. - Exhibit 1009
Page 651 of 720

- `down`, to order the sections so the first row of `sos` contains the poles closest to the unit circle
- `up`, to order the sections so the first row of `sos` contains the poles farthest from the unit circle (default)

`[sos,g] = tf2sos(b,a,'order','scale')` specifies the desired scaling of the gain and numerator coefficients of all second-order sections, where *`scale`* is:

- `none`, to apply no scaling (default)
- `inf`, to apply infinity-norm scaling
- `two`, to apply 2-norm scaling

Using infinity-norm scaling in conjunction with `up`-ordering minimizes the probability of overflow in the realization. Using 2-norm scaling in conjunction with `down`-ordering minimizes the peak round-off noise.

`sos = tf2sos(...)` embeds the overall system gain, `g`, in the first section, $H_1(z)$, so that

$$H(z) = \prod_{k=1}^{L} H_k(z)$$

See Also

`cplxpair`	Group complex numbers into complex conjugate pairs.
`sos2tf`	Conversion of second-order sections to transfer function.
`ss2sos`	Conversion of state-space to second-order sections.
`tf2ss`	Conversion of transfer function to state-space.
`tf2zp`	Conversion of transfer function to zero-pole-gain.
`zp2sos`	Conversion of zero-pole-gain to second-order sections.

Meta Platforms, Inc. - Exhibit 1009
Page 652 of 720

References

[1] Jackson, L.B. *Digital Filters and Signal Processing*. 3rd ed. Boston: Kluwer Academic Publishers, 1996. Chapter 11.

[2] Mitra, S.K. *Digital Signal Processing: A Computer-Based Approach*. New York: McGraw-Hill, 1998. Chapter 9.

[3] Vaidyanathan, P.P. "Robust Digital Filter Structures." *Handbook for Digital Signal Processing*. S.K. Mitra and J.F. Kaiser, ed. Chapter 7. New York: John Wiley & Sons, 1993.

Meta Platforms, Inc. - Exhibit 1009
Page 653 of 720

Purpose Conversion of transfer function to state-space.

Syntax

```
[A,B,C,D] = tf2ss(b,a)
```

Description `tf2ss` converts a transfer function representation of a given system to an equivalent state-space representation.

`[A,B,C,D] = tf2ss(b,a)` finds a state-space representation:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

given a system in transfer function form:

$$H(s) = \frac{B(s)}{A(s)} = C(sI - A)^{-1}B + D$$

from a single input. Input vector `a` contains the denominator coefficients in descending powers of s. Matrix `b` contains the numerator coefficients with as many rows as there are outputs y. `tf2ss` returns the `A`, `B`, `C`, and `D` matrices in controller canonical form.

`tf2ss` also works for discrete systems, but you must pad the numerator with trailing zeros to make it the same length as the denominator.

The `tf2ss` function is part of the standard MATLAB language.

Example Consider the system

$$H(s) = \frac{\begin{bmatrix} 2s + 3 \\ s^2 + 2s + 1 \end{bmatrix}}{s^2 + 0.4s + 1}$$

Meta Platforms, Inc. - Exhibit 1009
Page 654 of 720

To convert this system to state-space:

```
b = [0 2 3; 1 2 1];
a = [1 0.4 1];
[A,B,C,D] = tf2ss(b,a)
```

```
A =
   -0.4000   -1.0000
    1.0000         0

B =
     1
     0

C =
    2.0000    3.0000
    1.6000         0

D =
     0
     1
```

There is disagreement in the literature on naming conventions for the canonical forms. It is easy, however, to generate similarity transformations that convert to other forms. For example:

```
T = fliplr(eye(n));
A = T\A*T;
```

Algorithm

`tf2ss` writes the output in controller canonical form by inspection.

See Also

`sos2ss`	Conversion of second-order sections to state-space.
`ss2tf`	Conversion of state-space to transfer function.
`tf2sos`	Conversion of transfer function to second-order sections.
`tf2zp`	Conversion of transfer function to zero-pole-gain.
`zp2ss`	Conversion of zero-pole-gain to state-space.

Meta Platforms, Inc. - Exhibit 1009
Page 655 of 720

Purpose Conversion of transfer function to zero-pole-gain.

Syntax

```
[z,p,k] = tf2zp(b,a)
```

Description `tf2zp` finds the zeros, poles, and gains of a system in polynomial transfer function form.

`[z,p,k] = tf2zp(b,a)` finds the single-input, multi-output (SIMO) factored transfer function form:

$$H(s) = \frac{Z(s)}{P(s)} = k\frac{(s-Z_1)(s-Z_2)\cdots(s-Z_m)}{(s-p_1)(s-p_2)\cdots(s-p_n)}$$

given a SIMO system in polynomial transfer function form:

$$\frac{B(s)}{A(s)} = \frac{b_1 s^{(nb-1)} + \cdots + b_{(nb-1)}s + b_{(nb)}}{a_1 s^{(na-1)} + \cdots + a_{(na-1)}s + a_{(na)}}$$

Vector `a` specifies the coefficients of the denominator in descending powers of *s*. Matrix `b` indicates the numerator coefficients with as many rows as there are outputs. The zero locations are returned in the columns of matrix `z`, with as many columns as there are rows in `b`. The pole locations are returned in column vector `p` and the gains for each numerator transfer function in vector `k`.

The `tf2zp` function also works for discrete systems, and is part of the standard MATLAB language.

Meta Platforms, Inc. - Exhibit 1009
Page 656 of 720

Example

Find the zeros, poles, and gains of the system

$$H(s) = \frac{2s + 3}{s^2 + 0.4s + 1}$$

```
b = [2 3];
a = [1 0.4 1];
[z,p,k] = tf2zp(b,a)

z =

   -1.5000

p =

   -0.2000 + 0.9798i
   -0.2000 - 0.9798i

k =

   2
```

Algorithm

The system is converted to state-space using `tf2ss` and then to zeros, poles, and gains using `ss2zp`.

See Also

`sos2zp`	Conversion of second-order sections to zero-pole-gain.
`ss2zp`	Conversion of state-space to zero-pole-gain.
`tf2sos`	Conversion of transfer function to second-order sections.
`tf2ss`	Conversion of transfer function to state-space.
`zp2tf`	Conversion of zero-pole-gain to transfer function.

Meta Platforms, Inc. - Exhibit 1009
Page 657 of 720

Purpose

Transfer function estimate from input and output.

Syntax

```
Txy = tfe(x,y)
Txy = tfe(x,y,nfft)
[Txy,f] = tfe(x,y,nfft,Fs)
Txy = tfe(x,y,nfft,Fs,window)
Txy = tfe(x,y,nfft,Fs,window,noverlap)
Txy = tfe(x,y,...,'dflag')
tfe(x,y)
```

Description

`Txy = tfe(x,y)` finds a transfer function estimate `Txy` given input signal vector `x` and output signal vector `y`. The *transfer function* is the quotient of the cross spectrum of `x` and `y` and the power spectrum of `x`:

$$T_{xy}(f) = \frac{P_{xy}(f)}{P_{xx}(f)}$$

The relationship between the input `x` and output `y` is modeled by the linear, time-invariant transfer function `Txy`.

Vectors `x` and `y` must be the same length. `Txy = tfe(x,y)` uses the following default values:

- `nfft = min(256,(length(x))`
- `Fs = 2`
- `window = hanning(nfft)`
- `noverlap = 0`

`nfft` specifies the FFT length that `tfe` uses. This value determines the frequencies at which the power spectrum is estimated. `Fs` is a scalar that specifies the sampling frequency. `window` specifies a windowing function and the number of samples `tfe` uses in its sectioning of the `x` and `y` vectors. `noverlap` is the number of samples by which the sections overlap. Any arguments that are omitted from the end of the parameter list use the default values shown above.

If `x` is real, `tfe` estimates the transfer function at positive frequencies only; in this case, the output `Txy` is a column vector of length `nfft/2+1` for `nfft` even and `(nfft+1)/2` for `nfft` odd. If `x` or `y` is complex, `tfe` estimates the transfer function for both positive and negative frequencies and `Txy` has length `nfft`.

Meta Platforms, Inc. - Exhibit 1009
Page 658 of 720

`Txy = tfe(x,y,nfft)` uses the specified FFT length `nfft` in estimating the transfer function. Specify `nfft` as a power of 2 for fastest execution.

`[Txy,f] = tfe(x,y,nfft,Fs)` returns a vector `f` of frequencies at which `tfe` estimates the transfer function. `Fs` is the sampling frequency. `f` is the same size as `Txy`, so `plot(f,Txy)` plots the transfer function estimate versus properly scaled frequency. `Fs` has no effect on the output `Txy`; it is a frequency scaling multiplier.

`Txy = tfe(x,y,nfft,Fs,window)` specifies a windowing function and the number of samples per section of the `x` vector. If you supply a scalar for `window`, `Txy` uses a Hanning window of that length. The length of the window must be less than or equal to `nfft`; `tfe` zero pads the sections if the length of the window exceeds `nfft`.

`Txy = tfe(x,y,nfft,Fs,window,noverlap)` overlaps the sections of `x` by `noverlap` samples.

You can use the empty matrix `[]` to specify the default value for any input argument except `x` or `y`. For example,

```
Txy = tfe(x,y,[],[],kaiser(128,5))
```

uses 256 as the value for `nfft` and 2 as the value for `Fs`.

`Txy = tfe(x,y,...,'dflag')` specifies a detrend option, where *`dflag`* is:

- `linear`, to remove the best straight-line fit from the prewindowed sections of `x` and `y`
- `mean`, to remove the mean from the prewindowed sections of `x` and `y`
- `none`, for no detrending (default)

The *`dflag`* parameter must appear last in the list of input arguments. `tfe` recognizes a *`dflag`* string no matter how many intermediate arguments are omitted.

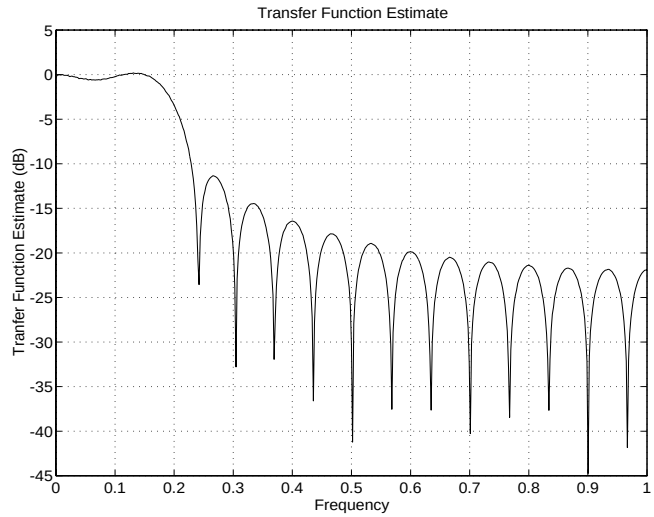
`tfe(...)` with no output arguments plots the magnitude of the transfer function estimate in decibels versus frequency in the current figure window.

Meta Platforms, Inc. - Exhibit 1009
Page 659 of 720

Example

Compute and plot the transfer function estimate between two colored noise sequences x and y:

```
h = fir1(30,0.2,boxcar(31));
x = randn(16384,1);
y = filter(h,1,x);
tfe(x,y,1024,[],[],512)
title('Transfer Function Estimate')
```



Algorithm

tfe uses a four-step algorithm:

1. It multiplies the detrended sections by window.
2. It takes the length nfft FFT of each section.
3. It averages the squares of the spectra of the x sections to form Pxx and averages the products of the spectra of the x and y sections to form Pxy.
4. It calculates Txy:

   ```
   Txy = Pxy./Pxx
   ```


Meta Platforms, Inc. - Exhibit 1009
Page 660 of 720

Diagnostics

An appropriate diagnostic message is displayed when incorrect arguments are used:

```
Requires window's length to be no greater than the FFT length.
Requires NOVERLAP to be strictly less than the window length.
Requires positive integer values for NFFT and NOVERLAP.
Requires vector (either row or column) input.
Requires inputs X and Y to have the same length.
```

See Also

`etfe`	Compute empirical transfer function estimate and periodogram (see *System Identification Toolbox User's Guide*).
`cohere`	Estimate magnitude squared coherence function between two signals.
`csd`	Estimate the cross spectral density (CSD) of two signals.
`pwelch`	Estimate the power spectral density (PSD) of a signal using Welch's method.
`spa`	Perform spectral analysis for input-output data (see *System Identification Toolbox User's Guide*).

Meta Platforms, Inc. - Exhibit 1009
Page 661 of 720

Purpose Triangular window.

Syntax

```
w = triang(n)
```

Description `triang(n)` returns an `n`-point triangular window in the column vector `w`. The coefficients of a triangular window are

For `n` odd:

$$w[k] = \begin{cases} \dfrac{2k}{n+1}, & 1 \le k \le \dfrac{n+1}{2} \\ \dfrac{2(n-k+1)}{n+1}, & \dfrac{n+1}{2} \le k \le n \end{cases}$$

For `n` even:

$$w[k] = \begin{cases} \dfrac{2k-1}{n}, & 1 \le k \le \dfrac{n}{2} \\ \dfrac{2(n-k+1)}{n}, & \dfrac{n}{2} + 1 \le k \le n \end{cases}$$

The triangular window is very similar to a Bartlett window. The Bartlett window always ends with zeros at samples 1 and `n`, while the triangular window is nonzero at those points. For `n` odd, the center `n–2` points of `triang(n–2)` are equivalent to `bartlett(n)`.

See Also

`bartlett`	Bartlett window.
`blackman`	Blackman window.
`boxcar`	Rectangular window.
`chebwin`	Chebyshev window.
`hamming`	Hamming window.
`hanning`	Hanning window.
`kaiser`	Kaiser window.

References

[1] Oppenheim, A.V., and R.W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

Meta Platforms, Inc. - Exhibit 1009
Page 662 of 720

Purpose

Sampled aperiodic triangle generator.

Syntax

```
y = tripuls(T)
y = tripuls(T,w)
y = tripuls(T,w,s)
```

Description

`y = tripuls(T)` returns a continuous, aperiodic, symmetric, unity-height triangular pulse at the times indicated in array `T`, centered about `T=0` and with a default width of 1.

`y = tripuls(T,w)` generates a triangular pulse of width `w`.

`y = tripuls(T,w,s)` generates a triangular pulse with skew `s`, where $-1<s<1$. When `s` is 0, a symmetric triangular pulse is generated.

See Also

`chirp`	Swept-frequency cosine generator.
`cos`	Cosine of vector/matrix elements (see the online *MATLAB Function Reference*).
`diric`	Dirichlet or periodic sinc function.
`gauspuls`	Gaussian-modulated sinusoidal pulse generator.
`pulstran`	Pulse train generator.
`rectpuls`	Sampled aperiodic rectangle generator.
`sawtooth`	Sawtooth or triangle wave generator.
`sin`	Sine of vector/matrix elements (see the online *MATLAB Function Reference*).
`sinc`	Sinc or $\sin(\pi t)/\pi t$ function.
`square`	Square wave generator.

Meta Platforms, Inc. - Exhibit 1009
Page 663 of 720

Purpose Unwrap phase angles.

Syntax

```
p = unwrap(p)
```

Description `p = unwrap(p)` corrects the phase angles in vector `p` by adding multiples of $\pm 2\pi$, where needed, to smooth the transitions across branch cuts. When `p` is a matrix, `unwrap` corrects the phase angles down each column. The phase must be in radians.

The `unwrap` function is part of the standard MATLAB language.

Limitations `unwrap` tries to detect branch cut crossings, but it can be fooled by sparse, rapidly changing phase values.

See Also

`abs`	Absolute value (magnitude).
`angle`	Phase angle.

Meta Platforms, Inc. - Exhibit 1009
Page 664 of 720

Purpose

Upsample, apply an FIR filter, and downsample.

Syntax

```
yout = upfirdn(xin,h)
yout = upfirdn(xin,h,p)
yout = upfirdn(xin,h,p,q)
```

Description

`upfirdn` performs a cascade of three operations:

1. Upsampling by `p` (zero inserting)
2. FIR filtering with the impulse response given in `h`
3. Downsampling by `q` (throwing away samples)

`upfirdn` has been implemented as a MEX-file for maximum speed, so only the outputs actually needed are computed. The FIR filter is usually a lowpass filter, which you must design using another function such as `remez` or `fir1`.

NOTE The function `resample` performs an FIR design using `firls`, followed by rate changing implemented with `upfirdn`.

`yout = upfirdn(xin,h)` returns the output signal `yout`. If `yout` is a row or column vector, then it represents one signal; if `yout` is an array, then each column is a separate output. `xin` is the input signal. If `xin` is a row or column vector, then it represents one signal; if `xin` is an array, then each column is filtered. `h` is the impulse response of the FIR filter. If `h` is a row or column vector, then it represents one filter; if `h` is an array, then each column is a separate impulse response.

`yout = upfirdn(xin,h,p)` specifies the upsampling factor `p`. `p` is an integer with a default of 1.

`yout = upfirdn(xin,h,p,q)` specifies the downsampling factor `q`. `q` is an integer with a default of 1.

Meta Platforms, Inc. - Exhibit 1009
Page 665 of 720

NOTE Since `upfirdn` performs convolution and rate changing, the `yout` signals have a different length than `xin`. The length of $y[n]$ is approximately `p/q` times the length of $x[n]$.

Remarks

Usually the inputs `xin` and the filter `h` are vectors, in which case only one output signal is produced. However, when these arguments are arrays, each column is treated as a separate signal or filter. Valid combinations are

1. `xin` is a vector and `h` is a vector.

   There is one filter and one signal, so the function convolves `xin` with `h`. The output signal `yout` is a row vector if `xin` is a row; otherwise, it is a column vector.

2. `xin` is an array and `h` is a vector.

   There is one filter and many signals, so the function convolves `h` with each column of `xin`. The resulting `yout` will be an array with the same number of columns as `xin`.

3. `xin` is a vector and `h` is an array.

   There are many filters and one signal, so the function convolves each column of `h` with `xin`. The resulting `yout` will be an array with the same number of columns as `h`.

4. `xin` is an array and `h` is an array, both with the same number of columns.

   There are many filters and many signals, so the function convolves corresponding columns of `xin` and `h`. The resulting `yout` is an array with the same number of columns as `xin` and `h`.

Examples

If both `p` and `q` are equal to 1 (that is, there is no rate changing), the result is ordinary convolution of two signals (equivalent to `conv`):

```
yy = upfirdn(xx,hh);
```

This example implements a seven-channel filter bank by convolving seven different filters with one input signal, then downsamples by five:

```
% Assume that hh is an L-by-7 array of filters.
yy = upfirdn(xx,hh,1,5);
```


Meta Platforms, Inc. - Exhibit 1009
Page 666 of 720

Implement a rate change from 44.1 kHz (CD sampling rate) to 48 kHz (DAT rate), a ratio of 160/147. This requires a lowpass filter with cutoff frequency at $\omega_c = 2\pi/160$:

```
% Design lowpass filter with cutoff at 1/160th of Fs.
hh = fir1(300,2/160); % need a very long lowpass filter
yy = upfirdn(xx,hh,160,147);
```

In this example, the filter design and resampling are separate steps. Note that `resample` would do both steps as one.

Algorithm

`upfirdn` uses a polyphase interpolation structure. The number of multiply-add operations in the polyphase structure is approximately $(L_h L_x - pL_x)/q$ where L_h and L_x are the lengths of $h[n]$ and $x[n]$, respectively.

A more accurate flops count is computed in the program, but the actual count is still approximate. For long signals $x[n]$, the formula is quite often exact.

Diagnostics

There must be one output argument and at least two input arguments. If either of these conditions are violated, `upfirdn` gives the appropriate error message:

```
UPFIRDN needs at least two input arguments.
UPFIRDN should have exactly one output argument.
```

If the arrays are sparse, `upfirdn` gives the error message

```
H must be full numeric matrix.
```

When the input signals are in the columns of a matrix and there are multiple filters also in the columns of a matrix, the number of signals and filters must be the same. If they are not, `upfirdn` gives the error message

```
X and H must have the same number of columns, if more than one.
```

The arguments `p` and `q` must be integers. If they are not, `upfirdn` gives the error message

```
P and/or Q must be greater than zero
```

If the arguments `p` and `q` are not relatively prime, `upfirdn` gives the warning message

```
WARNING (upfirdn) p & q have common factor
```


Meta Platforms, Inc. - Exhibit 1009
Page 667 of 720

See Also

`conv`	Convolution and polynomial multiplication.
`decimate`	Decrease the sampling rate for a sequence (decimation).
`filter`	Filter data with a recursive (IIR) or nonrecursive (FIR) filter.
`interp`	Increase sampling rate by an integer factor (interpolation).
`intfilt`	Interpolation FIR filter design.
`resample`	Change sampling rate by any rational factor.

References

[1] Crochiere, R.E., and L.R. Rabiner. *Multi-Rate Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1983. Pgs. 88-91.

[2] Crochiere, R.E. “A General Program to Perform Sampling Rate Conversion of Data by Rational Ratios.” In *Programs for Digital Signal Processing*. IEEE Press. New York: John Wiley & Sons, 1979. Pgs. 8.2-1 to 8.2-7.

Meta Platforms, Inc. - Exhibit 1009
Page 668 of 720

Purpose

Voltage controlled oscillator.

Syntax

```
y = vco(x,Fc,Fs)
y = vco(x,[Fmin Fmax],Fs)
```

Description

`y = vco(x,Fc,Fs)` creates a signal that oscillates at a frequency determined by the real input vector or array `x` with sampling frequency `Fs`. `Fc` is the carrier or reference frequency; when `x` is 0, `y` is an `Fc` Hz cosine with amplitude 1 sampled at `Fs` Hz. `x` ranges from -1 to 1, where -1 corresponds to a 0 frequency output, 0 to `Fc`, and 1 to `2*Fc`. `y` is the same size as `x`.

`y = vco(x,[Fmin Fmax],Fs)` scales the frequency modulation range so that -1 and 1 values of `x` yield oscillations of `Fmin` Hz and `Fmax` Hz respectively. For best results, `Fmin` and `Fmax` should be in the range 0 to `Fs/2`.

By default, `Fs` is 1 and `Fc` is `Fs/4`.

If `x` is a matrix, `vco` produces a matrix whose columns oscillate according to the columns of `x`.

Example

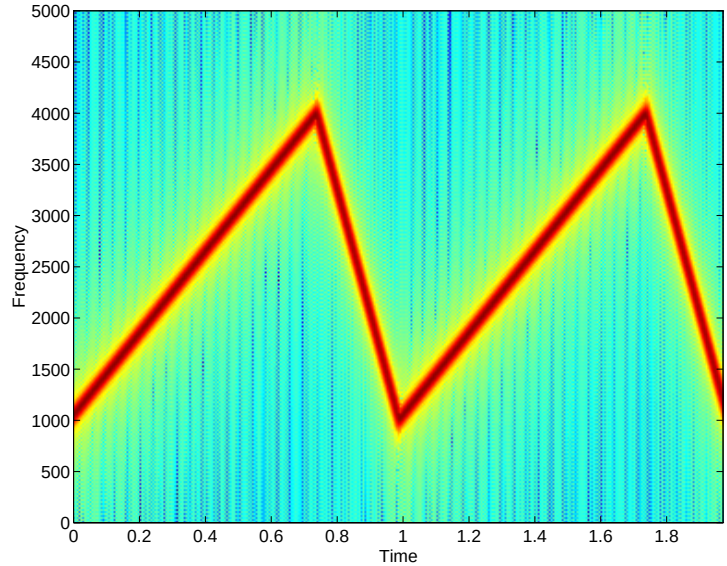
Generate two seconds of a signal sampled at 10,000 samples/second whose instantaneous frequency is a triangle function of time:

```
Fs = 10000;
t = 0:1/Fs:2;
x = vco(sawtooth(2*pi*t,0.75),[.1 0.4]*Fs,Fs);
```


Meta Platforms, Inc. - Exhibit 1009
Page 669 of 720

Plot the spectrogram of the generated signal.

```
specgram(x,512,Fs,kaiser(256,5),220)
```



Algorithm

vco performs FM modulation using the modulate function.

Diagnostics

If any values of x lie outside [-1,1], vco gives the following error message:

```
X outside of range [-1,1].
```

See Also

demod	Demodulation for communications simulation.
modulate	Modulation for communications simulation.

Meta Platforms, Inc. - Exhibit 1009
Page 670 of 720

Purpose

Cross-correlation function estimate.

Syntax

```
c = xcorr(x,y)
c = xcorr(x)
c = xcorr(x,y,'option')
c = xcorr(x,'option')
c = xcorr(x,y,maxlags)
c = xcorr(x,maxlags)
c = xcorr(x,y,maxlags,'option')
c = xcorr(x,maxlags,'option')
[c,lags] = xcorr(...)
```

Description

`xcorr` estimates the cross-correlation sequence of a random process. Autocorrelation is handled as a special case.

The true cross-correlation sequence is

$$\gamma_{xy}(m) = E\{x_n y^*_{n+m}\}$$

where x_n and y_n are stationary random processes, $-\infty < n < \infty$, and $E\{\}$ is the expected value operator. `xcorr` must estimate the sequence because, in practice, only a finite segment of the infinite-length random process is available.

`c = xcorr(x,y)` returns the cross-correlation sequence in a length $2N$-1 vector, where `x` and `y` are length N vectors (N>1). If `x` and `y` are not the same length, the shorter vector is zero-padded to the length of the longer vector.

By default, `xcorr` computes raw correlations with no normalization:

$$c_{xy}(m) = \begin{cases} \sum_{n=0}^{N-|m|-1} x_n y^*_{n+m} & m \geq 0 \\ c^*_{yx}(-m) & m < 0 \end{cases}$$

The output vector `c` has elements given by $c(m) = c_{xy}(m\text{-}N)$, m=1, ..., $2N$-1.

In general, the correlation function requires normalization to produce an accurate estimate (see below).

Meta Platforms, Inc. - Exhibit 1009
Page 671 of 720

`c = xcorr(x)` is the autocorrelation sequence for the vector `x`. If `x` is an N-by-P matrix, `c` is a matrix with $2N$-1 rows whose P^2 columns contain the cross-correlation sequences for all combinations of the columns of `x`.

`c = xcorr(x,y,'option')` specifies a normalization option for the cross-correlation, where `'option'` is:

- `biased`, for biased estimates of the cross-correlation function

$$c_{xy,\,biased}(m) = \frac{1}{N} c_{xy}(m)$$

- `unbiased`, for unbiased estimates of the cross-correlation function

$$c_{xy,\,unbiased}(m) = \frac{1}{N - |m|} c_{xy}(m)$$

- `coeff`, to normalize the sequence so the autocorrelations at zero lag are identically 1.0
- `none`, to use the raw, unscaled cross-correlations (default)

See reference [1] for more information on the properties of biased and unbiased correlation estimates.

`c = xcorr(x,'option')` specifies one of the above normalization options for the autocorrelation.

`c = xcorr(x,y,maxlags)` returns the cross-correlation sequence over the lag range `[-maxlags:maxlags]`. Output `c` has length `2*maxlags+1`.

`c = xcorr(x,maxlags)` returns the autocorrelation sequence over the lag range `[-maxlags:maxlags]`. Output `c` has length `2*maxlags+1`. If `x` is an N-by-P matrix, `c` is a matrix with `2*maxlags+1` rows whose P^2 columns contain the autocorrelation sequences for all combinations of the columns of `x`.

`c = xcorr(x,y,maxlags,'option')` specifies both a maximum number of lags and a scaling option for the cross-correlation.

`c = xcorr(x,maxlags,'option')` specifies both a maximum number of lags and a scaling option for the autocorrelation.

Meta Platforms, Inc. - Exhibit 1009
Page 672 of 720

`[c,lags] = xcorr(...)` returns a vector of the lag indices at which `c` was estimated, with the range `[-maxlags:maxlags]`. When `maxlags` is not specified, the range of `lags` is `[-N+1:N-1]`.

In all cases, the cross-correlation or autocorrelation computed by `xcorr` has the 0-th lag in the middle of the sequence, at element or row `maxlags+1` (element or row `N` if `maxlags` is not specified).

Examples

The second output, `lags`, is useful for plotting the cross-correlation or autocorrelation. For example, the estimated autocorrelation of zero-mean Gaussian white noise $c_{ww}(m)$ can be displayed for $-10 \le m \le 10$ using

```
ww = randn(1000,1);
[c_ww,lags] = xcorr(ww,10,'coeff');
stem(lags,c_ww)
```

Swapping the `x` and `y` input arguments reverses (and conjugates) the output correlation sequence. For row vectors, the resulting sequences are reversed left to right; for column vectors, up and down. The following example illustrates this property (`mat2str` is used for a compact display of complex numbers).

```
x = [1,2i,3]; y = [4,5,6];
[c1,lags] = xcorr(x,y);
c1 = mat2str(c1,2), lags

c1 =

    [12-i*8.9e-016 15-i*8 22-i*10 5-i*12 6+i*8.9e-016]

lags =

    -2    -1     0     1     2

c2 = conj(fliplr(xcorr(y,x)));
c2 = mat2str(c2,2)

c2 =

    [12-i*8.9e-016 15-i*8 22-i*10 5-i*12 6+i*8.9e-016]
```

For the case where input argument `x` is a matrix, the output columns are arranged so that extracting a row and rearranging it into a square array

Meta Platforms, Inc. - Exhibit 1009
Page 673 of 720

produces the cross-correlation matrix corresponding to the lag of the chosen row. For example, the cross-correlation at zero lag can be retrieved by

```
randn('seed',0)
X = randn(2,2);
[M,P] = size(X);
c = xcorr(X);
c0 = zeros(P); c0(:) = c(M,:)   % Extract zero-lag row
```

```
c0 =

    1.7500    0.3079
    0.3079    0.1293
```

You can calculate the matrix of correlation coefficients that the MATLAB function `corrcoef` generates by substituting

```
c = xcov(X,'coef')
```

in the last example. The function `xcov` subtracts the mean and then calls `xcorr`.

Use `fftshift` to move the second half of the sequence starting at the zeroth lag to the front of the sequence. `fftshift` swaps the first and second halves of a sequence.

Algorithm

For more information on estimating covariance and correlation functions, see [1] and [2].

Diagnostics

There must be at least one vector input argument; otherwise, `xcorr` gives the following error message:

```
1st arg must be a vector or matrix.
```

The string '*option*' must be the last argument; otherwise, `xcorr` gives the following error message:

```
Argument list not in correct order.
```

If the second argument was entered as a scalar, it is taken to be `maxlags` and no succeeding input can be a scalar. When the second argument is a vector, the first must also be a signal vector. The third argument, when present, must be

Meta Platforms, Inc. - Exhibit 1009
Page 674 of 720

a scalar or a string. If they are not, `xcorr` gives the appropriate error message(s):

```
3rd arg is maxlag, 2nd arg cannot be scalar.
When b is a vector, a must be a vector.
Maxlag must be a scalar.
```

Normally the lengths of the vector inputs should be the same; if they are not, then the only allowable scaling option is `'none'`. If it is not, `xcorr` gives the following error message:

```
OPTION must be 'none' for different length vectors A and B.
```

See Also

`conv`	Convolution and polynomial multiplication.
`corrcoef`	Correlation coefficient matrix.
`cov`	Covariance matrix.
`xcorr2`	Two-dimensional cross-correlation.
`xcov`	Cross-covariance function estimate (equal to mean-removed cross-correlation).

References

[1] Bendat, J.S., and A.G. Piersol. *Random Data: Analysis and Measurement Procedures*. New York: John Wiley & Sons, 1971. Pg. 332.

[2] Oppenheim, A.V., and R.W. Schafer. *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1975. Pgs. 63-67, 746-747, 839-842.

Meta Platforms, Inc. - Exhibit 1009
Page 675 of 720

Purpose Two-dimensional cross-correlation.

Syntax

```
C = xcorr2(A)
C = xcorr2(A,B)
```

Description `C = xcorr2(A,B)` returns the cross-correlation of matrices `A` and `B` with no scaling. `xcorr2` is the two-dimensional version of `xcorr`. It has its maximum value when the two matrices are aligned so that they are shaped as similarly as possible.

`xcorr2(A)` is the autocorrelation matrix of input matrix `A`. It is identical to `xcorr2(A,A)`.

See Also

`conv2`	Two-dimensional convolution.
`filter2`	Two-dimensional digital filtering.
`xcorr`	Cross-correlation function estimate.

Meta Platforms, Inc. - Exhibit 1009
Page 676 of 720

Purpose

Cross-covariance function estimate (equal to mean-removed cross-correlation).

Syntax

```
v = xcov(x,y)
v = xcov(x)
v = xcov(x,'option')
[c,lags] = xcov(x,y,maxlags)
[c,lags] = xcov(x,maxlags)
[c,lags] = xcov(x,y,maxlags,'option')
```

Description

`xcov` estimates the cross-covariance sequence of random processes. Autocovariance is handled as a special case.

The true cross-covariance sequence is the mean-removed cross-correlation sequence

$$\phi_{xy}(m) = E\{(x_n - m_x)(y_{n+m} - m_y)^*\}$$

where m_x and m_y are the mean values of the two stationary random processes, and $E\{\}$ is the expected value operator. `xcov` estimates the sequence because, in practice, access is available to only a finite segment of the infinite-length random process.

`v = xcov(x,y)` returns the cross-covariance sequence in a length $2N$-1 vector, where `x` and `y` are length N vectors.

`v = xcov(x)` is the autocovariance sequence for the vector `x`. Where `x` is an N-by-P array, `v = xcov(X)` returns am array with $2N$-1 rows whose P^2 columns contain the cross-covariance sequences for all combinations of the columns of `X`.

By default, `xcov` computes raw covariances with no normalization. For a length N vector:

$$c_{xy}(m) = \begin{cases} \displaystyle\sum_{n=0}^{N-|m|-1} \left(x(n) - \frac{1}{N}\sum_{i=0}^{N-1} x_i \right)\left(y^*_{n+m} - \frac{1}{N}\sum_{=}^{-} y^*_i \right) & m \geq 0 \\ c^*_{yx}(-m) & m < 0 \end{cases}$$

The output vector `c` has elements given by `c`(m) = $c_{xy}(m\text{-}N)$, m=1,...,$2N$-1.

Meta Platforms, Inc. - Exhibit 1009
Page 677 of 720

The covariance function requires normalization to estimate the function properly.

`v = xcov(x,'option')` specifies a scaling option, where *option* is:

- `biased`, for biased estimates of the cross-covariance function
- `unbiased`, for unbiased estimates of the cross-covariance function
- `coeff`, to normalize the sequence so the auto-covariances at zero lag are identically 1.0
- `none`, to use the raw, unscaled cross-covariances (default)

See [1] for more information on the properties of biased and unbiased correlation and covariance estimates.

`[c,lags] = xcov(x,y,maxlags)` where `x` and `y` are length `m` vectors, returns the cross-covariance sequence in a length `2*maxlags+1` vector `c`. `lags` is a vector of the lag indices where `c` was estimated, that is, `[-maxlags:maxlags]`.

`[c,lags] = xcov(x,maxlags)` is the autocovariance sequence over the range of lags `[-maxlags:maxlags]`.

`[c,lags] = xcov(x,maxlags)` where `x` is an `m`-by-`p` array, returns array `c` with `2*maxlags+1` rows whose P^2 columns contain the cross-covariance sequences for all combinations of the columns of `x`.

`[c,lags] = xcov(x,y,maxlags,'option')` specifies a scaling option, where *option* is the last input argument.

In all cases, `xcov` gives an output such that the zeroth lag of the covariance vector is in the middle of the sequence, at element or row `maxlag+1` or at `m`.

Examples

The second output `lags` is useful when plotting. For example, the estimated autocovariance of uniform white noise $c_{ww}(m)$ can be displayed for $-10 \le m \le 10$ using

```
ww = randn(1000,1); % generate uniform noise with mean = 1/2
[cov_ww,lags] = xcov(ww,10,'coeff');
stem(lags,cov_ww)
```


Meta Platforms, Inc. - Exhibit 1009
Page 678 of 720

Algorithm

`xcov` computes the mean of its inputs, subtracts the mean, and then calls `xcorr`. For more information on estimating covariance and correlation functions, see [1] and [2].

Diagnostics

`xcov` does not check for any errors other than the correct number of input arguments. Instead, it relies on the error checking in `xcorr`, which it calls.

See Also

`conv`	Convolution and polynomial multiplication.
`corrcoef`	Correlation coefficient matrix.
`cov`	Covariance matrix.
`xcorr`	Cross-correlation function estimate.
`xcorr2`	Two-dimensional cross-correlation.

References

[1] Bendat, J.S., and A.G. Piersol. *Random Data: Analysis and Measurement Procedures*. New York: John Wiley & Sons, 1971. Pg. 332.

[2] Oppenheim, A.V., and R.W. Schafer. *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1975. Pgs. 63-67, 746-747, 839-842.

Meta Platforms, Inc. - Exhibit 1009
Page 679 of 720

Purpose Recursive digital filter design.

Syntax

```
[b,a] = yulewalk(n,f,m)
```

Description `yulewalk` designs recursive IIR digital filters using a least-squares fit to a specified frequency response.

`[b,a] = yulewalk(n,f,m)` returns row vectors `b` and `a` containing the `n+1` coefficients of the order `n` IIR filter whose frequency-magnitude characteristics approximately match those given in vectors `f` and `m`:

- `f` is a vector of frequency points, specified in the range between 0 and 1, where 1 corresponds to half the sample frequency (the Nyquist frequency). The first point of `f` must be 0 and the last point 1, with all intermediate points in increasing order. Duplicate frequency points are allowed, corresponding to steps in the frequency response.
- `m` is a vector containing the desired magnitude response at the points specified in `f`.
- `f` and `m` must be the same length.
- `plot(f,m)` displays the filter shape.

The output filter coefficients are ordered in descending powers of z,

$$\frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + \cdots + b(n+1)z^{-n}}{a(1) + a(2)z^{-1} + \cdots + a(n+1)z^{-n}}$$

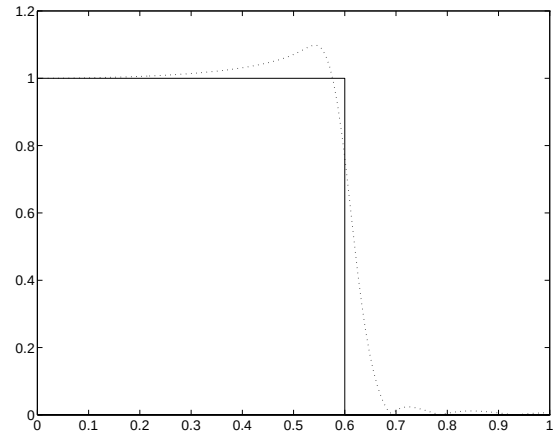
When specifying the frequency response, avoid excessively sharp transitions from passband to stopband. You may need to experiment with the slope of the transition region to get the best filter design.

Example Design an 8th-order lowpass filter and overplot the desired frequency response with the actual frequency response:

```
f = [0 0.6 0.6 1];
m = [1 1 0 0];
```


Meta Platforms, Inc. - Exhibit 1009
Page 680 of 720

```
[b,a] = yulewalk(8,f,m);
[h,w] = freqz(b,a,128);
plot(f,m,w/pi,abs(h),'--')
```



Algorithm

`yulewalk` performs a least-squares fit in the time domain. It computes the denominator coefficients using modified Yule-Walker equations, with correlation coefficients computed by inverse Fourier transformation of the specified frequency response. To compute the numerator, `yulewalk` takes the following steps:

1. Computes a numerator polynomial corresponding to an additive decomposition of the power frequency response.
2. Evaluates the complete frequency response corresponding to the numerator and denominator polynomials.
3. Uses a spectral factorization technique to obtain the impulse response of the filter.
4. Obtains the numerator polynomial by a least-squares fit to this impulse response.

Meta Platforms, Inc. - Exhibit 1009
Page 681 of 720

See Also

`butter`	Butterworth analog and digital filter design.
`cheby1`	Chebyshev type I filter design (passband ripple).
`cheby2`	Chebyshev type II filter design (stopband ripple).
`ellip`	Elliptic (Cauer) filter design.
`fir2`	Window-based finite impulse response filter design – arbitrary response.
`firls`	Least square linear-phase FIR filter design.
`maxflat`	Generalized digital Butterworth filter design.
`remez`	Parks-McClellan optimal FIR filter design.

References

[1] Friedlander, B., and B. Porat. “The Modified Yule-Walker Method of ARMA Spectral Estimation.” *IEEE Transactions on Aerospace Electronic Systems*. AES-20, No. 2 (March 1984). Pgs. 158-173.

Meta Platforms, Inc. - Exhibit 1009
Page 682 of 720

Purpose

Conversion of zero-pole-gain to second-order sections.

Syntax

```
[sos,g] = zp2sos(z,p,k)
[sos,g] = zp2sos(z,p,k,'order')
[sos,g] = zp2sos(z,p,k,'order','scale')
sos = zp2sos(...)
```

Description

zp2sos converts a zero-pole-gain representation of a given system to an equivalent second-order section representation.

[sos,g] = zp2sos(z,p,k) finds a matrix sos in second-order section form with gain g equivalent to the zero-pole-gain system represented by input arguments z, p, and k. Vectors z and p contain the zeros and poles of the system $H(z)$, not necessarily in any order:

$$H(z) = k\frac{(z-z_1)(z-z_2)\cdots(z-z_n)}{(z-p_1)(z-p_2)\cdots(z-p_m)}$$

where n and m are the lengths of z and p, respectively, and k is a scalar gain. The zeros and poles must be real or complex conjugate pairs. sos is an L-by-6 matrix:

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

whose rows contain the numerator and denominator coefficients b_{ik} and a_{ik} of the second-order sections of $H(z)$:

$$H(z) = g\prod_{k=1}^{L} H_k(z) = g\prod_{k=1}^{L}\frac{b_{0k}+b_{1k}z^{-1}+b_{2k}z^{-2}}{1+a_{1k}z^{-1}+a_{2k}z^{-2}}$$

The number of rows L of matrix sos is the maximum of the ceiling of $n/2$ and the ceiling of $m/2$.

Meta Platforms, Inc. - Exhibit 1009
Page 683 of 720

`[sos,g] = zp2sos(z,p,k,'order')` specifies the order of the rows in `sos`, where *`order`* is:

- `down`, to order the sections so the first row of `sos` contains the poles closest to the unit circle
- `up`, to order the sections so the first row of `sos` contains the poles farthest from the unit circle (default)

`[sos,g] = zp2sos(z,p,k,'order','scale')` specifies the desired scaling of the gain and the numerator coefficients of all second-order sections, where *`scale`* is:

- `none`, to apply no scaling (default)
- `inf`, to apply infinity-norm scaling
- `two`, to apply 2-norm scaling

Using infinity-norm scaling in conjunction with `up`-ordering minimizes the probability of overflow in the realization. Using 2-norm scaling in conjunction with `down`-ordering minimizes the peak round-off noise.

`sos = zp2sos(...)` embeds the overall system gain, `g`, in the first section, $H_1(z)$, so that

$$H(z) = \prod_{k=1}^{L} H_k(z)$$

Example

Find a second-order section form of a Butterworth lowpass filter:

```
[z,p,k] = butter(5,0.2);
sos = zp2sos(z,p,k);
```

Algorithm

`zp2sos` uses a four-step algorithm to determine the second-order section representation for an input zero-pole-gain system:

1. It groups the zeros and poles into complex conjugate pairs using the `cplxpair` function.
2. It forms the second-order section by matching the pole and zero pairs according to the following rules:

Meta Platforms, Inc. - Exhibit 1009
Page 684 of 720

a Match the poles closest to the unit circle with the zeros closest to those poles.

b Match the poles next closest to the unit circle with the zeros closest to those poles.

c Continue until all of the poles and zeros are matched.

`zp2sos` groups real poles into sections with the real poles closest to them in absolute value. The same rule holds for real zeros.

3 It orders the sections according to the proximity of the pole pairs to the unit circle. `zp2sos` normally orders the sections with poles closest to the unit circle last in the cascade. You can tell `zp2sos` to order the sections in the reverse order by specifying the `down` flag.

4 `zp2sos` scales the sections by the norm specified in the `'scale'` argument. For arbitrary $H(\omega)$, the scaling is defined by:

$$\|H\|_p = \left[\frac{1}{2\pi} \int_0^{2\pi} |H(\omega)|^p \, d\omega \right]^{\frac{1}{p}}$$

where p can be either ∞ or 2. See the references for details on the scaling.

This scaling is an attempt to minimize overflow or peak round-off noise in fixed point filter implementations.

See Also

`cplxpair`	Group complex numbers into complex conjugate pairs.
`sos2zp`	Conversion of second-order sections to zero-pole-gain.
`ss2sos`	Conversion of state-space to second-order sections.
`tf2sos`	Conversion of transfer function to second-order sections.
`zp2ss`	Conversion of zero-pole-gain to state-space.
`zp2tf`	Conversion of zero-pole-gain to transfer function.

Meta Platforms, Inc. - Exhibit 1009
Page 685 of 720

References

[1] Jackson, L.B. *Digital Filters and Signal Processing*. 3rd ed. Boston: Kluwer Academic Publishers, 1996. Chapter 11.

[2] Mitra, S.K. *Digital Signal Processing: A Computer-Based Approach*. New York: McGraw-Hill, 1998. Chapter 9.

[3] Vaidyanathan, P.P. "Robust Digital Filter Structures." *Handbook for Digital Signal Processing*. S.K. Mitra and J.F. Kaiser, ed. Chapter 7. New York: John Wiley & Sons, 1993.

Meta Platforms, Inc. - Exhibit 1009
Page 686 of 720

Purpose

Conversion of zero-pole-gain to state-space.

Syntax

```
[A,B,C,D] = zp2ss(z,p,k)
```

Description

`zp2ss` converts a zero-pole-gain representation of a given system to an equivalent state-space representation.

`[A,B,C,D] = zp2ss(z,p,k)` finds a single input, multiple output, state-space representation

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

given a system in factored transfer function form

$$H(s) = \frac{Z(s)}{P(s)} = k\frac{(s-z_1)(s-z_2)\cdots(s-z_n)}{(s-p_1)(s-p_2)\cdots(s-p_n)}$$

Column vector `p` specifies the pole locations, and matrix `z` the zero locations with as many columns as there are outputs. The gains for each numerator transfer function are in vector `k`. The `A`, `B`, `C`, and `D` matrices are returned in controller canonical form.

`Inf` values may be used as place holders in `z` if some columns have fewer zeros than others.

Algorithm

`zp2ss`, for single-input systems, groups complex pairs together into two-by-two blocks down the diagonal of the `A` matrix. This requires the zeros and poles to be real or complex conjugate pairs.

See Also

`sos2ss`	Conversion of second-order sections to state-space.
`ss2zp`	Conversion of state-space to zero-pole-gain.
`tf2ss`	Conversion of transfer function to state-space.
`zp2sos`	Conversion of zero-pole-gain to second-order sections.
`zp2tf`	Conversion of zero-pole-gain to transfer function.

Meta Platforms, Inc. - Exhibit 1009
Page 687 of 720

Purpose Conversion of zero-pole-gain to transfer function.

Syntax

```
[b,a] = zp2tf(z,p,k)
```

Description `zp2tf` forms transfer function polynomials from the zeros, poles, and gains of a system in factored form.

`[b,a] = zp2tf(z,p,k)` finds a rational transfer function:

$$\frac{B(s)}{A(s)} = \frac{b_1 s^{(nb-1)} + \cdots + b_{(nb-1)} s + b_{nb}}{a_1 s^{(na-1)} + \cdots + a_{(na-1)} s + a_{na}}$$

given a system in factored transfer function form:

$$H(s) = \frac{Z(s)}{P(s)} = k\frac{(s-z_1)(s-z_2)\cdots(s-z_m)}{(s-p_1)(s-p_2)\cdots(s-p_n)}$$

Column vector `p` specifies the pole locations, and matrix `z` the zero locations, with as many columns as there are outputs. The gains for each numerator transfer function are in vector `k`. The zeros and poles must be real or come in complex conjugate pairs. The polynomial coefficients are returned in vectors: the denominator coefficients in row vector `a` and the numerator coefficients in matrix `b`, with as many rows as there are columns of `z`.

`Inf` values can be used as place holders in `z` if some columns have fewer zeros than others.

Algorithm The system is converted to transfer function form using `poly` with `p` and the columns of `Z`.

See Also

`sos2tf`	Conversion of second-order sections to transfer function.
`ss2tf`	Conversion of state-space to transfer function.
`tf2zp`	Conversion of transfer function to zero-pole-gain.
`zp2sos`	Conversion of zero-pole-gain to second-order sections.
`zp2ss`	Conversion of zero-pole-gain to state-space.

Meta Platforms, Inc. - Exhibit 1009
Page 688 of 720

Purpose

Zero-pole plot.

Syntax

```
zplane(z,p)
zplane(b,a)
[hz,hp,ht] = zplane(z,p)
```

Description

This function displays the poles and zeros of discrete-time systems.

`zplane(z,p)` plots the zeros specified in column vector `z` and the poles specified in column vector `p` in the current figure window. The symbol `'o'` represents a zero and the symbol `'x'` represents a pole. The plot includes the unit circle for reference. If `z` and `p` are arrays, `zplane` plots the poles and zeros in the columns of `z` and `p`, respectively, in different colors.

You can override the automatic scaling of `zplane` using

```
axis([xmin xmax ymin ymax])
```

or

```
set(gca,'ylim',[ymin ymax])
```

or

```
set(gca,'xlim',[xmin xmax])
```

after calling `zplane`. This is useful in the case where one or a few of the zeros or poles have such a large magnitude that the others are grouped tightly around the origin and are hard to distinguish.

`zplane(b,a)` where `b` and `a` are row vectors, first uses `roots` to find the zeros and poles of the transfer function represented by numerator coefficients `b` and denominator coefficients `a`.

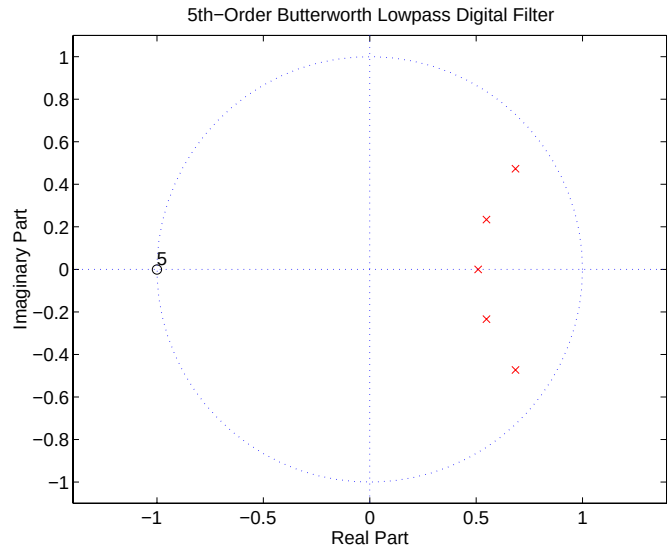
`[hz,hp,ht] = zplane(z,p)` returns vectors of handles to the zero lines, `hz`, and the pole lines, `hp`. `ht` is a vector of handles to the axes/unit circle line and to text objects, which are present when there are multiple zeros or poles. If there are no zeros or no poles, `hz` or `hp` is set to the empty matrix `[]`.

Meta Platforms, Inc. - Exhibit 1009
Page 689 of 720

Examples

Plot the poles and zeros of a 5th-order Butterworth lowpass digital filter with cutoff frequency of 0.2:

```
[z,p,k] = butter(5,0.2);
zplane(z,p);
title('5th-Order Butterworth Lowpass Digital Filter');
```



To generate the same plot with a transfer function representation of the filter:

```
[b,a] = butter(5,0.2);   % transfer function
zplane(b,a)
```

See Also

`freqz`	Frequency response of digital filters.

Meta Platforms, Inc. - Exhibit 1009
Page 690 of 720

Index

A

`abs` 6-2, **6-12**
`ac2poly` 6-3, **6-13**
`ac2rc` 6-3, **6-14**
algorithm, filtering in SPTool 5-18
aliased sinc function. *See* Dirichlet function
aliasing
  and impulse invariance 2-42
  preventing during resampling 4-21
  reducing with analytic signal 4-38
all-pole filter. *See* IIR filter
all-zero filter. *See* FIR filter
`am` 4-29
AM. *See* amplitude modulation
`amdsb–sc` 4-29, 6-236
`amdsb–tc` 4-29, 6-236
amplitude demodulation
  double side-band, suppressed carrier 6-114
  double side-band, transmitted carrier 6-114
  single side-band 6-114
amplitude modulation 4-29
  double side-band, suppressed carrier 6-236
  double side-band, transmitted carrier 6-236
  single side-band 6-236
`amssb` 4-30, 6-236
analog filter
  Bessel 6-22
  Butterworth 6-44
  Chebyshev type I 6-66
  Chebyshev type II 6-71
  converting to digital 2-41, 6-194
  design 2-7
    Bessel 2-11, 6-22
    Butterworth 6-43
    Chebyshev type I 6-65
    Chebyshev type II 6-71
    elliptic 6-125, 6-126
    inverse 6-203
  frequency response 1-26, 6-173
  order estimation
    Butterworth 6-49
    Chebyshev type I 6-56
    Chebyshev type II 6-61
    elliptic 6-133
  representational models 1-40
analog frequency xvii
analog prototype 2-38
  Bessel filter 2-11, 6-21
  Butterworth filter 2-8, 6-42
  Chebyshev type I filter 2-9, 6-54
  Chebyshev type II filter 2-10, 6-59
  conversion to bandpass 6-220
  conversion to bandstop 6-223
  conversion to highpass 6-225
  conversion to lowpass 6-227
  elliptic filter 6-131
  frequency response 2-12
  plotting 2-12
analog prototype design
  Bessel 2-38
  bilinear transformation 2-43
  Butterworth 2-38
  Chebyshev 2-38
  elliptic 2-38
  filter discretization 2-41
  frequency transformation 2-38
  impulse invariance 2-42

Meta Platforms, Inc. - Exhibit 1009
Page 691 of 720

See also IIR filter design
analog signal. *See* signal
analytic signal 2-26, 6-187
applications 4-38
properties 4-38
`angle` 6-2, **6-15**
anti-symmetric filter 2-25
Apply button, Spectrum Viewer 5-97
Apply Filter button 5-18
applying parameters with Apply button 5-100
`arburg` 4-11, **6-16**
`arcov` 4-11, **6-17**
ARMA filter 1-15
See also IIR filter
ARMA model 4-13, 4-15
Prony's method 4-13
Steiglitz-McBride method 4-15
`armcov` 4-11, **6-18**
array
display in Signal Browser 5-47, 5-51
in SPTool 5-14
Array Signals button, Signal Browser 5-47, 5-51
ARX model 4-13
`aryule` 4-11, **6-19**
ASCII file, importing 1-13
attenuation, stopband 5-70
attributes, instantaneous 6-187
autocorrelation 6-368
multiple channels 3-4
two-dimensional 6-372
autocorrelation sequence
converting from filter coefficients 6-262
converting from reflection coefficients 6-282
converting to filter coefficients 6-13
converting to reflection coefficients 6-14
autocovariance 6-373
multiple channels 3-4
autoregressive (AR) filter 1-15
See also IIR filter
autoregressive (AR) model
via Burg method 6-16, 6-239
via covariance method 6-17, 6-243
via modified covariance method 6-18, 6-247
via Yule-Walker AR method 6-19, 6-278
autoregressive moving average (ARMA) filter 1-15
See also IIR filter
auto-spectrum, in SPTool 5-15
averaging filter 1-14
axis labels, in Signal Browser 5-20, 5-23
axis parameters
in Filter Viewer 5-20
in Spectrum Viewer 5-20
axis scaling range
in Filter Viewer 5-25, 5-87
in Spectrum Viewer 5-24, 5-100, 5-102
axis scaling units
in Filter Viewer 5-25, 5-87
in Spectrum Viewer 5-24, 5-100, 5-102

B

band edges, prewarping 2-44
bandlimited interpolation 6-314
bandpass filter
analog prototype design 2-6
and impulse invariance 2-42
Bessel 6-22
Butterworth 6-43, 6-44
Chebyshev type I 6-65, 6-66
Chebyshev type II 6-70, 6-71
elliptic 6-125, 6-126
example, Chebyshev type I 2-40
FIR design, with window method 2-21, 6-154
transformation from lowpass to 6-220

Meta Platforms, Inc. - Exhibit 1009
Page 692 of 720

bandstop filter
analog prototype design 2-6
Bessel 6-22
Butterworth 6-44, 6-45
Chebyshev type I 6-66, 6-67
Chebyshev type II 6-71, 6-72
elliptic 6-126, 6-127
FIR design, with window method 2-21, 6-153
transformation from lowpass to 6-223
bandwidth 2-40
`bartlett` 4-2, 6-8, **6-20**
compared to `triang` 6-20
example 4-2
Bartlett window 4-2
coefficients 6-20
Bessel filter
analog 6-22
analog prototype 2-11, 2-38, 6-21
bandpass configuration, analog 6-22
bandstop configuration, analog 6-22
characteristics 2-11
highpass configuration, analog 6-22
limitations 6-24
lowpass configuration, analog 6-22
`besselap` 2-5, 6-10, **6-21**
example 2-11
`besself` 2-5, 2-6, 6-5, **6-22**
beta parameter, of Kaiser window 5-75
bias
correlation 3-3, 4-12
power spectral density 3-12
spectral density 3-12
variance trade-off 3-4
`bilinear` 2-5, 2-42, 2-43, 6-11, **6-25**
bilinear transformation 2-43, 6-25
defined 2-43
output representation 6-26
prewarping 2-44, 6-25
`blackman` 4-2, 6-8, **6-30**
Blackman window 4-4, 6-30
defined 4-4
`boxcar` 4-2, 6-8, **6-32**
example 4-2
boxcar window. *See* rectangular window
brackets, indicating closed interval xvii
buffering 6-33
Burg method 3-5, 3-6, 3-20
compared to Welch's method 3-22
defined 3-20
example 3-21
in Spectrum Viewer 5-104
`buttap` 2-5, 6-10, **6-42**
example 2-8
`butter` 2-5, 2-6, 6-5, **6-43**
Butterworth filter
analog 6-44
analog prototype 2-8, 2-38, 6-42
bandpass configuration
analog 6-44
digital 6-43
bandstop configuration
analog 6-45
digital 6-44
characteristics 2-8
design 6-43
digital 6-43
generalized 2-14
highpass configuration
analog 6-45
digital 6-44
limitations 6-47
lowpass configuration
analog 6-44
digital 6-43

Meta Platforms, Inc. - Exhibit 1009
Page 693 of 720

order estimation 2-7, 6-48
`buttord` 2-5, 6-5, **6-48**

C

canonical forms 1-17, 6-351
carrier frequency 4-29, 6-236, 6-365
carrier signal 4-29, 6-114
cascade, digital filter 1-37
Cauer filter. *See* elliptic filter
`cceps` 4-23, 6-9, **6-52**
example 4-23
center frequency 2-40
central features 1-2
cepstrum
applications 4-23
complex 4-23
inverse 4-23, 4-25
overview 4-23
real 4-23
See also real cepstrum, complex cepstrum
cepstrum analysis 4-23
`cheb1ap` 2-5, 6-10, **6-54**
example 2-9, 2-40
`cheb1ord` 2-5, 6-5, **6-55**
`cheb2ap` 2-5, 6-10, **6-59**
example 2-10
`cheb2ord` 2-5, 6-5, **6-60**
`chebwin` 4-2, 6-8, **6-64**
`cheby1` 2-5, 2-6, 6-5, **6-65**
example 2-45
`cheby2` 2-5, 2-6, 6-5, **6-70**
Chebyshev error minimization 2-22, 6-287
Chebyshev type I filter
analog 6-66
analog prototype 2-9, 2-38, 6-54
bandpass configuration
analog 6-66
digital 6-65
bandstop configuration
analog 6-67
digital 6-66
characteristics 2-9
design 6-65
digital 6-65
highpass configuration
analog 6-67
digital 6-66
lowpass configuration
analog 6-66
digital 6-65
order estimation 2-7, 6-55
Chebyshev type II filter
analog 6-71
analog prototype 2-38, 6-59
bandpass configuration
analog 6-71
digital 6-70
bandstop configuration
analog 6-72
digital 6-71
characteristics 2-10
design 6-70
digital 6-70
highpass configuration
analog 6-72
digital 6-71
limitations 6-69
lowpass configuration
analog 6-71
digital 6-70
order estimation 2-7, 6-60
Chebyshev window 4-9, 6-64

Meta Platforms, Inc. - Exhibit 1009
Page 694 of 720

frequency response 4-9
`chirp` 1-9, 6-2, **6-75**
chirp signal 1-9
chirp *z*-transform (CZT) 4-34, 6-105
compared to discrete Fourier transform 4-34
execution time 4-35
for narrowband frequency analysis 6-105
classical IIR filter design 2-8
click-and-drag panning, in Signal Browser 5-48
coefficients
correlation 6-88
filter 1-15, 5-79, 6-13, 6-262, 6-263, 6-283
linear prediction 6-229
reflection 1-37, 6-14, 6-263, 6-282, 6-283
sequence 6-282
`cohere` 3-6, 3-16, 6-7, **6-79**
coherence 3-15, 6-79
defined 3-15
coherence function 3-15
Color button 5-35
color order
in Signal Browser 5-22
in Spectrum Viewer 5-22
Color Order text box 5-22
color, customizing in SPTool 5-20
column index vector, entering in Signal Browser 5-47
column, array 5-47
communications 4-10
communications simulation 4-29, 6-114, 6-236
See also modulation, demodulation, voltage controlled oscillation
Compact Disc standard 4-20
complex cepstrum, defined 4-23
complex conjugate 6-90
Complex Display mode 5-47
complex envelope 4-38
complex numbers, grouping by conjugate 6-90
complex signals, in Signal Browser 5-47
computation parameters, in Spectrum Viewer 5-103, 5-104
Conf. Int. check box 5-107
confidence interval
for cross spectral density 3-14
for power spectral density 3-14, 5-107
setting in Spectrum Viewer 5-107
conservation of total power, using `pmtm` 3-19
context sensitive help in SPTool 5-7
continuous signal. *See* signal
continuous-time filter. *See* analog filter
control systems 1-35
Control Systems Toolbox 1-35, 6-198
`conv` 1-14, 1-20, 6-2, **6-83**
`conv2` 1-14, 6-2, **6-84**
conversion
autocorrelation sequence to filter coefficients 6-13
autocorrelation sequence to reflection coefficients 6-14
filter coefficients to autocorrelation sequence 6-262
filter coefficients to reflection coefficients 6-263, 6-283
reflection coefficients to autocorrelation sequence 6-282
second-order sections to state-space 6-316
second-order sections to transfer function 6-318
second-order sections to zero-pole-gain 6-320
state-space to second-order sections 6-333
state-space to zero-pole-gain 6-338
transfer function to lattice 6-346
transfer function to second-order sections 6-347

Meta Platforms, Inc. - Exhibit 1009
Page 695 of 720

transfer function to state-space 6-350
zero-pole-gain to second-order section 6-379
zero-pole-gain to state-space 6-383
`convmtx` 1-39, 1-42, 6-3, **6-86**
convolution
and cross-correlation 3-3
and filtering 1-14, 6-148
convolution matrix 1-39, 6-86
defined 6-83
example 1-14
two-dimensional 6-84
obtaining subsection 6-84
convolution matrix 1-42, 6-86
defined 1-39
example 6-86
`corrcoef` 6-7, **6-88**
correlation 3-2
coefficient matrix 6-88
See also autocorrelation, cross-correlation
cosine window 4-4
`cov` 6-7, **6-89**
covariance 3-2
matrix 6-89
See also autocovariance, cross-covariance
covariance method 3-6, 3-22
defined 3-22
example 3-23
covariance method, in Spectrum Viewer 5-105
`cplxpair` 6-9, **6-90**
Create button, in Spectra panel 5-19, 5-97
`cremez` 2-17, 6-5, **6-91**
cross spectral density 3-14, 6-100
confidence interval 3-14
defined 3-5
cross-correlation 6-367
biased 3-3
multiple channels 3-4
normalization 3-4
two-dimensional 6-372
unbiased 3-3
cross-covariance 6-373
multiple channels 3-4
`csd` 3-6, 3-14, 6-7, **6-100**
CSD. *See* cross spectral density
cutoff frequency 2-38
defined 6-22
for Kaiser window filter 5-75
`czt` 4-35, 6-6, **6-105**
CZT. *See* chirp *z*-transform

D

data
duplicating in SPTool 5-16
editing in SPTool 5-13, 5-15
entering 1-13
exporting from SPTool 5-6
importing 1-13
importing into SPTool 5-3, 5-6, 5-7
measuring in SPTool 5-31
multichannel 1-4, 1-7
viewing in SPTool 5-31
data compression 4-10
data matrix 1-4, 1-7
data vector 1-4
`dct` 4-36, 6-6, **6-108**
example 4-37
`decimate` 6-9, **6-110**
decimation 6-110
FIR filter for 6-201
`deconv` 4-33, 6-9, **6-113**
example 4-33
deconvolution 4-33, 6-113
default plot, in Spectrum Viewer 5-102

Meta Platforms, Inc. - Exhibit 1009
Page 696 of 720

Default Session check box, in Preferences dialog box 5-28
Default SPTool session file 5-20, 5-28
default values, using empty matrix 3-11
delay
  adding to signal 2-25
  group 1-28
  noninteger 2-26
  phase 1-28
`demod` 4-29, 4-30, 6-9, **6-114**
  example 4-30
demodulation 4-30, 6-114
  example 4-30
  methods 4-30, 6-114
design, generalized filter 2-5
designed filter, in SPTool 5-14
DFT. *See* discrete Fourier transform
`dftmtx` 6-6, **6-117**
difference equation, relation to transfer function 1-32
differentiator 2-26, 6-167, 6-289
Digital Audio Tape standard 4-20
digital filter
  anti-causal 1-20
  as convolution matrix 1-40
  Butterworth 6-43
  cascade 1-37
  Chebyshev type I 6-65
  Chebyshev type II 6-70
  coefficients 1-15
  design 2-2
  elliptic 6-125
  FIR 2-16
    compared to IIR 2-16
  fixed-point implementation 1-37
  frequency response 1-24
  group delay 1-28, 6-182
identification from frequency data 6-207
IIR 2-4
  compared to FIR 2-4
implementation 1-14, 6-142, 6-145
  FFT-based (FIR) 6-142
  overlap-add method 1-22
  using convolution 1-14
  using `filter` function 1-16
impulse response 1-14, 1-23, 6-196
initial conditions 1-17
linear system models 1-32
names 1-15
order 1-15
  in state-space representation 1-34
order estimation
  Butterworth 6-48
  Chebyshev type I 6-55
  Chebyshev type II 6-60
  elliptic 6-132
  equiripple FIR 6-294
phase delay 1-28, 6-182
poles 1-30, 1-33
representational models 1-32
representing in MATLAB 1-32
second-order sections 1-37
specifications 2-7
startup transients 1-21, 1-22
structure
  lattice 1-37
  transposed direct form II 1-17
time-domain representation 1-16
transfer function representation 1-15
two-dimensional 6-148
zero-phase 1-20, 6-149
zeros 1-30, 1-33
zeros and poles 1-33
See also FIR filter, IIR filter

Meta Platforms, Inc. - Exhibit 1009
Page 697 of 720

digital filter design
FIR 2-16
IIR 2-4
digital frequency xvii
direct design 2-13
described 2-13
summary 2-5
`diric` 6-2, **6-118**
Dirichlet function 1-11, 6-118
defined 1-11
example 1-12
discrete cosine transform (DCT) 6-108
applications 4-36
energy compaction property 4-37
example 4-37
inverse 4-36, 6-190
reconstructing signal from few coefficients 4-37
discrete Fourier transform (DFT) 1-2, 1-43
algorithms 1-45
and IIR filter implementation 1-22
and spectral analysis 3-6
applications 6-137
dependence on signal length 1-45
example 1-44
execution time, using chirp *z*-transform 4-35
inverse 1-43, 6-192
matrix 6-117
two-dimensional 1-45, 6-193
matrix 6-117
two-dimensional 1-45, 6-141
See also fast Fourier transform (FFT), `fft`
discrete prolate spheroidal sequences (DPSSs) 3-19
discrete-time Fourier transform 3-5
discretization 2-41, 6-194
bilinear transformation 2-43
prewarping 2-44
impulse invariance 2-42
disk, loading variables from 5-8
`dpss` 6-9, **6-119**
`dpss.mat` 3-19
`dpssclear` 3-19, 6-9, **6-121**
`dpssdir` 3-19, 6-9, **6-122**
`dpssload` 3-19, 6-9, **6-123**
DPSSs. *See* discrete prolate spheroidal sequences
`dpsssave` 3-19, 6-9, **6-124**
duty cycle, specifying 1-8

E

echo detection 4-23
edge effects 1-22
edge frequencies, setting in Filter Designer 5-70
Edit Design button 5-18, 5-59
`eig`, in `pmusic` function 3-25
eigenanalysis
defined 3-24
frequency estimator functions 3-24
eigenvector method 3-5, 3-23
See also multiple signal classification method
`ellip` 2-5, 2-6, 6-5, **6-125**
`ellipap` 2-5, 6-10, **6-131**
example 2-10
`ellipord` 2-5, 6-5, **6-132**
elliptic filter
analog 6-126
analog prototype 2-38, 6-131
bandpass configuration
analog 6-126
digital 6-125

Meta Platforms, Inc. - Exhibit 1009
Page 698 of 720

bandstop configuration
analog 6-127
digital 6-126
characteristics 2-10
design 6-125
digital 6-125
highpass configuration
analog 6-127
digital 6-126
limitations 6-129
lowpass configuration
analog 6-126
digital 6-125
order estimation 2-7, 6-132
energy compaction 4-37
equiripple characteristics
Chebyshev type I filter (passband) 2-9
Chebyshev type II filter (stopband) 2-10
Chebyshev window 4-9
elliptic filter 2-10, 6-125, 6-131
from Parks-McClellan design 6-287
equiripple filter 2-22
error minimization
between desired and actual response 2-22
for equiripple filter 5-75
for least squares filter 5-75
integral of square 2-22
minimax 2-22
weighting in frequency bands 2-24
estimation
cross spectrum 3-14
power spectrum 3-6
transfer function 3-14
See also parametric modeling
estimation methods
in Spectrum Viewer 5-103, 5-104
nonparametric
FFT method 5-105
multiple signal classification method (MUSIC) 3-5, 5-106
multitaper method (MTM) 3-5, 5-105
Welch's method 3-5, 5-106
parametric 3-5
Burg method 5-104, 6-16
covariance method 6-17
modified covariance method 6-18
Yule-Walker AR method 5-107
Yule-Walker method 6-19
Export Filters as TF objects check box, in Preferences dialog box 5-29
Export menu item 5-6
Exporting Components check box, in Preferences dialog box 5-29
exporting data from MATLAB 1-13
extensions to SPTool 5-30

F

Factory Settings button, in Preferences dialog box 5-30
fast Fourier transform (FFT) 1-22, 1-43
and frequency response 1-24
`fft` 1-22
prime factor algorithm 1-45, 6-139
radix-2 algorithm 1-45, 6-139
role in signal processing 1-43
two-dimensional 6-141
`fft` 1-2, 1-22, 1-43, 6-6, **6-137**
complex inputs 1-45
example 1-44
execution time 1-45, 6-140
prime factor algorithm 1-45, 6-139
radix-2 algorithm 1-45, 6-139

Meta Platforms, Inc. - Exhibit 1009
Page 699 of 720

real inputs 1-45
rearranging output 1-45, 6-144
specifying number of points 1-44
FFT length
in Filter Designer 5-20, 5-27
in Filter Viewer 5-25
FFT Length edit box, in Preferences dialog box 5-25, 5-27
FFT method, in Spectrum Viewer 5-105
FFT. *See* fast Fourier transform
`fft2` 1-45, 6-6, **6-141**
rearranging output 1-45
FFT-based filtering 1-22
`fftfilt` 1-19, 6-2, **6-142**
compared to `filter` 6-142
`fftshift` 1-45, 6-6, **6-144**
File Contents list 5-8
filter
analog prototype 2-8, 2-11, 6-21, 6-42, 6-54, 6-59, 6-131
analyzing in Filter Viewer 5-17
applying to a signal 5-18
Butterworth 2-7, 6-43
generalized 2-14
Chebyshev 2-7
Chebyshev type I 6-65
Chebyshev type II 6-70
coefficients 1-15, 2-17, 5-79
design
FIR 6-287
generalized 2-5
IIR 2-5
inverse 6-203, 6-207
discretization 2-41
elliptic 2-7, 6-125
equiripple 2-22
exporting as TF objects for Control System Toolbox 5-29
group delay 5-17
identification from frequency data 6-203
implementation 1-22, 6-142, 6-145
importing into SPTool 5-7, 5-8, 5-11
impulse response 5-17
linear time-invariant digital 1-2
magnitude response 5-17
measurements 5-37
median 4-28, 6-235
minimax 2-22
minimum phase 6-265
multiband FIR 2-22
names 1-15
naming in SPTool 5-16
order 1-15, 2-7, 6-48, 6-55, 6-60, 6-132
order selection 2-7
phase response 5-17
principal supported 1-2
Savitzky-Golay 6-309, 6-311
second-order sections 6-322
single band FIR 2-20
specifications 2-7
step response 5-17
transposed direct form II structure 1-17
two-dimensional 6-148
types 2-17
viewing in Filter Viewer 5-17
zeros and poles 5-17
See also FIR filter, IIR filter, digital filter, analog filter
`filter` 1-2, 1-16, 1-20, 6-2, **6-145**
compared to `fftfilt` 6-142
compared to `filtfilt` 1-21
final condition parameters 1-17
implementation 1-17

Meta Platforms, Inc. - Exhibit 1009
Page 700 of 720

initial condition parameters 1-17
initial conditions 6-150
filter coefficients
converting from autocorrelation sequence 6-13
converting to autocorrelation sequence 6-262
converting to reflection coefficients 6-263, 6-283
filter design
in Filter Designer 5-59, 5-73, 5-76
standard band configurations 5-59
using specification lines 5-70
Filter Designer 5-2, 5-17, 5-59, 6-329
activating 5-17, 5-59
changing plot properties 5-27
classical IIR filter design 5-76
closing 5-60
customizing 5-20
magnitude plot 5-63, 5-69, 5-78
measuring response characteristics 5-70
saving data to workspace 5-79, 5-110
setting edge frequencies 5-70
setting passband ripple 5-70
setting stopband attenuation 5-70
single band FIR filter design 5-73
window 5-60
filter parameters, in Filter Viewer 5-20
filter response, peaks and valleys 5-37
filter type
design 5-14
imported 5-14, 5-18
Filter Viewer 5-2, 5-17, 5-84, 6-330
activating 5-17, 5-84
customizing 5-20
default plot 5-85
plots 5-86
preferences 5-86
settings 5-86
subplots 5-86
viewing frequency response 5-82
viewing group delay 5-93
viewing impulse response 5-94
viewing magnitude response 5-89
viewing phase response 5-91
viewing step response 5-95
viewing zero-pole plot 5-94
window 5-85
`filter2` 6-3, **6-148**
filtering
and convolution 1-14
anti-causal 1-20
frequency domain 1-22
initial conditions 1-17
generating 1-18
zero-phase 1-20
filtering algorithm 5-18
`filtfilt` 1-19, 1-20, 2-4, 6-3, **6-149**
compared to `filter` 1-21
example 1-20
initial conditions 1-21
`filtic` 1-18, 6-3, **6-150**
FIR filter
arbitrary frequency response 6-157
compared to IIR 2-16
design 2-16
decimation 6-201
interpolation 6-201
least squares method 6-166
linear phase 6-166
multiband frequency response 6-157
Parks-McClellan method 6-287
window method 6-152
differentiator 2-26, 6-167, 6-289
Hilbert transformer 2-25, 6-167, 6-289

Meta Platforms, Inc. - Exhibit 1009
Page 701 of 720

implementation 1-17, 6-145
FFT-based 1-22, 6-142
overlap-add method 1-22, 6-142
linear phase 2-17, 6-287
order estimation, `remez` function 6-294
types 6-169, 6-292
FIR filter design 2-16
anti-symmetric 2-25
arbitrary responses 2-31
complex filters 2-17, 6-91
nonlinear phase 2-17, 6-91
reduced delay 2-34
constrained least squares 2-16, 2-27
linear phase 2-28
multiband 2-28, 2-29
weighted 2-30
equiripple 2-16, 2-22, 2-23, 5-73, 5-75
example 5-73, 5-113
in Filter Designer 5-59, 5-73
Kaiser window 5-73, 5-75
least squares 2-16, 2-22, 2-23, 5-73, 5-75
least squares compared to equiripple 2-23
linear phase filters 2-17, 2-22
multiband 2-16, 2-21, 2-22
order selection 5-75
parameters in Filter Designer 5-75
Parks-McClellan method 2-22
raised cosine method 2-17
role of Kaiser window 4-7
standard band 2-20
windowing method 2-16, 2-18
FIR filtering, in frequency domain 1-19
FIR lattice filter, implementation 1-38
`fir1` 2-16, 2-20, 6-5, **6-152**
accessing from Filter Designer 5-73, 5-75
`fir2` 2-16, 2-20, 6-5, **6-157**
example 2-21
`fircls` 2-16, 6-5, **6-160**
`fircls1` 2-16, 6-6, **6-163**
`firls` 2-16, 2-22, 6-6, **6-166**
accessing from Filter Designer 5-73, 5-75
compared to `remez` 2-23
filter characteristics 6-169
for differentiator design 2-26
weight vector 2-24
`firrcos` 2-17, 6-6, **6-171**
fixed-point implementation, digital filter 1-37
`fm` 4-30
FM. *See* frequency modulation
`fopen` 1-13
Fourier transform, eigenvector equivalent 3-24
Fourier transform, time dependent. *See* time-dependent Fourier transform
Fourier transform. *See* discrete Fourier transform, fast Fourier transform
`fread` 1-13
`freqs` 1-26, 6-3, **6-173**
`freqspace` **6-176**
frequency 6-287
analog xvii
angular 2-2
carrier 4-29, 6-236, 6-365
center 2-40
cutoff 2-38
digital xvii
normalization 2-2
Nyquist xvii, 2-2
prewarping 6-25
transformation 6-220, 6-223, 6-225, 6-227
vector 2-24, 6-157, 6-160, 6-376
frequency analysis
in Filter Viewer 5-84
time-dependent 6-323

Meta Platforms, Inc. - Exhibit 1009
Page 702 of 720

Frequency Axis Range pop-up menu, in Preferences dialog box 5-24, 5-25
Frequency Axis Scaling pop-up menu, in Preferences dialog box 5-24, 5-25
frequency axis scaling, in Spectrum Viewer 5-100
frequency demodulation 6-115
frequency domain
  duality with time domain 1-22
  FIR filtering 1-19
  for filter implementation 1-22
frequency domain based modeling. *See* parametric modeling
frequency estimator functions, in eigenanalysis 3-24
frequency estimator techniques
  eigenvector (EV) method 3-23
  multiple signal classification (MUSIC) method 3-23
frequency modulation 6-237
frequency points
  `freqz` 1-24, 1-26
  range 1-26
  spacing 1-26
Frequency Range pop-up menu, Spectrum Viewer 5-100
frequency response 1-24
  arbitrary 2-13, 6-157
  example 1-25
  in Filter Viewer 5-82, 5-84, 5-89
  inverse 6-203
  Kaiser window 4-6
  linear phase 2-17
  magnitude 1-26
  minimized error between desired and actual 2-22
  monotonic 2-9
  multiband 2-13
  of Bessel prototype 2-11
  of Butterworth prototype 2-8
  of Chebyshev type I prototype 2-9
  of Chebyshev type II prototype 2-10
  of Chebyshev window 4-9
  of elliptic prototype 2-10
  phase 1-26
    unwrapping 1-27
  plotting 1-25
  points at which evaluated 1-24
  spacing 6-176
  specifying sampling frequency 1-24
Frequency Scale pop-up menu, Spectrum Viewer 5-100
frequency transformation 2-38
  example 2-40
  lowpass to bandpass 6-220
  lowpass to bandstop 6-223
  lowpass to highpass 6-225
  lowpass to lowpass 6-227
frequency vector 6-287
`freqz` 1-24, 6-3, **6-177**
  frequency points 1-24
  sampling frequency 1-24
  spacing 6-176
From Disk radio button, in Import dialog box 5-8
From Workspace radio button, in Import dialog box 5-8
`fscanf` 1-13
Full View button 5-32

G

`gauspuls` 1-8, 1-10, 6-2, **6-180**
Gauss-Newton method 6-205, 6-209
generalized Butterworth filter 2-14
generalized cosine window 4-4

Meta Platforms, Inc. - Exhibit 1009
Page 703 of 720

Gibbs effect 2-19
reduced by window 4-2
graphical user interface (GUI) xii, 1-3
grid lines, in Filter Designer 5-20, 5-27
group delay 1-28, 5-17, 6-182
defined 1-28
example 1-29
of linear response filter 2-17
passband 2-11
viewing in Filter Viewer 5-93
Group Delay check box, Filter Viewer 5-86
group delay plot 5-86, 5-93
`grpdelay` 1-28, 6-3, **6-182**
GUI. *See* graphical user interface
GUI-based tools. *See* interactive tools

H

`hamming` 4-2, 6-8, **6-185**
Hamming window 2-20, 4-4, 6-185
`HandleVisibility` property 5-57, 5-110
`hanning` 4-2, 6-8, **6-186**
Hanning window 4-4, 6-186
highpass filter
analog prototype design 2-6
Bessel 6-22
Butterworth 6-44, 6-45
Chebyshev type I 6-66, 6-67
Chebyshev type II 6-71, 6-72
elliptic 6-126, 6-127
FIR design
with window method 2-21, 6-154
transformation from lowpass to 6-225
`hilbert` 2-26, 4-38, 6-6, **6-187**
example 4-39
Hilbert transform 4-34, 4-38, 6-187
and analytic signal 2-26
and instantaneous attributes 4-39
example 4-39
Hilbert transformer 6-167, 6-289
homomorphic systems 4-23
Horizontal button, for rulers 5-36, 5-38, 5-39

I

`icceps` 4-23, 4-25, 6-9, **6-189**
example 4-26
`idct` 4-36, 6-6, **6-190**
ideal lowpass filter 2-18
`ifft` 1-43, 6-6, **6-192**
specifying number of points 1-45
`ifft2` 1-45, 6-6, **6-193**
IIR filter
arbitrary frequency response 2-13
Bessel 2-11
Butterworth 2-8
Chebyshev type I 2-9
Chebyshev type II 2-10
compared to FIR 2-4
design 2-4
direct 2-13
Levinson-Durbin recursion 6-218
multiband 2-13
Prony's method 6-266
Steiglitz-McBride iteration 6-341
Yule-Walker 6-376
elliptic 2-10
implementation 6-145
frequency domain 1-22
zero-phase 1-20
IIR filter design 2-4, 2-5
analog prototype 2-5
Butterworth 2-7, 2-8, 5-76, 5-77
Chebyshev 2-7, 2-9, 2-10, 5-76, 5-77

Meta Platforms, Inc. - Exhibit 1009
Page 704 of 720

classical (analog prototype) 2-5, 2-8
comparison of filter types 2-8
general steps 2-37
illustration 2-37
in Filter Designer 5-76
order estimation 2-7
plotting prototypes 2-12
single step 2-6
single step order estimation 2-7
system model 2-7
direct methods 2-13
Yule-Walker 2-13
elliptic 2-7, 2-10, 5-76, 5-77
example 5-76, 5-77
generalized Butterworth 2-14
in Filter Designer 5-59, 5-76
maximally flat 2-14
parameters in Filter Designer 5-77
to specifications 2-7
See also direct design, parametric modeling
IIR lattice filter, implementation 1-38
image processing 6-84
with `fft2` and `ifft2` 1-45
`impinvar` 2-5, 2-42, 6-11, **6-194**
Import As pop-up menu, in Import dialog box 5-9
Import menu item 5-4, 5-6
imported filter, in SPTool 5-14
impulse invariance 2-42, 6-194
limitations 2-42
impulse response 1-23, 5-17, 6-196
and impulse invariance 2-42
computing with `filter` 1-23
computing with `impz` 1-23
defined 1-23
example 1-23
of ideal lowpass filter 2-18
viewing in Filter Viewer 5-94
Impulse Response check box, Filter Viewer 5-86
impulse response plot 5-86, 5-94
`impz` 6-3, **6-196**
example 1-23
indexing, of vectors 1-15
Inherit from pop-up menu, Spectrum Viewer 5-100
inheriting parameters 5-100
initial conditions 1-17, 1-21, 6-150
generating 1-18
Initial Type pop-up menu, in Preferences dialog box 5-21
instantaneous attributes 4-39, 6-187
interactive tools 5-2
extended example 5-113
Filter Designer 5-2, 5-59, 6-329
Filter Viewer 5-2, 5-84, 6-330
Signal Browser 5-2, 5-43, 6-328
Spectrum Viewer 5-2, 5-97, 6-331
SPTool 5-2, 6-328
`interp` 6-9, **6-199**
interpolation 6-199
FIR filter design 6-201
interval notation xvii
`intfilt` 6-6, **6-201**
inverse complex cepstrum 4-25
inverse discrete cosine transform 6-190
accuracy of signal reconstruction 4-38
inverse discrete Fourier transform 1-43, 6-192
`ifft` 1-43
matrix 6-117
two-dimensional 1-45, 6-193
inverse filter design 6-207, 6-266
analog 6-203
digital 6-207
inverse Fourier transform, continuous. *See* `sinc` function

Meta Platforms, Inc. - Exhibit 1009
Page 705 of 720

`invfreqs` 2-5, 4-11, 4-16, 6-8, **6-203**
`invfreqz` 2-5, 4-11, 4-16, 6-8, **6-207**

K

`kaiser` 4-2, 6-8, **6-210**
  accessing from Filter Designer 5-73
  example 4-5
Kaiser window 4-4, 6-210
  and FIR filter design 4-7, 5-73
  beta parameter 4-4, 6-210
  example 4-5
  frequency response 4-6
`kaiserord` 2-16, 6-6, **6-211**
  accessing from Filter Designer 5-75

L

ladder coefficients 1-38
Lagrange interpolation filter 6-201
Laplace transform 1-41
  equivalent to state-space representation 1-41
`latc2tf` 1-39, 1-42, 6-3, **6-216**
`latcfilt` 1-19, 1-39, 6-3, **6-217**
lattice coefficients 1-38
lattice filter 1-42
  implementation 1-38
  implementation with `latcfilt` 1-39
lattice form
  converting from transfer function 6-346
lattice structure 1-37
lattice/ladder filter
  implementation 1-38
  implementation with `latcfilt` 1-39
least squares method, FIR filter design 6-166
  filter characteristics 6-169
`levinson` 4-11, 6-8, **6-218**, **6-305**
  and parametric modeling 4-12
Levinson-Durbin recursion 4-12, 6-218, 6-305
line color
  in Filter Viewer 5-35
  in Signal Browser 5-35
  in Spectrum Viewer 5-35
line selection
  in Filter Viewer 5-34, 5-35
  in Signal Browser 5-34, 5-35
  in Spectrum Viewer 5-34, 5-35
line style
  customizing in SPTool 5-20
  in Filter Viewer 5-22, 5-35
  in Signal Browser 5-22, 5-35
  in Spectrum Viewer 5-22, 5-35
Line Style Order edit box, in Preferences dialog box 5-22
linear phase 2-16, 2-17, 6-166
  filter design 6-287
  related characteristics 2-17
linear prediction coefficients 6-229
linear prediction modeling 4-12
linear predictive coding
linear swept-frequency cosine. *See* chirp
linear system models 1-32
linear system transformations 1-41
  conversion chart 1-41
linear time-invariant differential equations, represented in state-space form 1-40
`load` 1-13
lowpass filter
  analog prototype design 2-6
  and impulse invariance 2-42
  Bessel 6-22
  Butterworth 6-43, 6-44
  Chebyshev type I 6-65, 6-66
  Chebyshev type II 6-70, 6-71

Meta Platforms, Inc. - Exhibit 1009
Page 706 of 720

elliptic 6-125, 6-126
FIR design, with window method 2-21
for decimation 6-110
for interpolation 6-199
ideal impulse response 2-18
translation of cutoff frequency 6-227
`lp2bp` 2-5, 2-39, 6-10, **6-220**
example 2-40
`lp2bs` 2-5, 2-39, 6-10, **6-223**
`lp2hp` 2-5, 2-39, 6-10, **6-225**
`lp2lp` 2-5, 2-39, 6-10, **6-227**
`lpc` 2-5, 4-11, 6-8, **6-229**
See also linear predictive coding, Prony's method
LPC. *See* linear prediction coefficients

M

magnitude
of Fourier transform of sequence 1-44
of frequency response 1-26
viewing in Filter Viewer 5-89
of transfer function estimate 3-15
vector 2-24, 6-157, 6-160, 6-376
Magnitude Axis Scaling pop-up menu, in Preferences dialog box 5-24, 5-25
Magnitude check box, Filter Viewer 5-86
magnitude plot 5-78
magnitude plot, in Filter Designer 5-69, 5-78
magnitude response 5-17
magnitude response plot 5-86, 5-89
in Filter Designer 5-63
Magnitude Scale pop-up menu, Spectrum Viewer 5-100
magnitude scale, in Spectrum Viewer 5-100
manufacturing 4-10
Marker Size edit box, in Preferences dialog box 5-21
match frequency (for prewarping) 6-25
MAT-file
`dpss.mat` 3-19
importing 1-13
importing into SPTool 5-4, 5-6
loading into SPTool 5-8
MAT-file format, converting to 1-13
matrices
convolution 1-39, 6-86
correlation coefficient 6-88
covariance 6-89
data 1-4, 1-7
discrete Fourier transform 6-117
for second-order sections form 1-37
inverse discrete Fourier transform 6-117
matrix form. *See* state-space form
`maxflat` 2-5, 2-14, 6-5, **6-233**
maxima, local 5-37
maximally flat 2-14
measurement lines 5-70
measurements
in Filter Viewer 5-37
in Signal Browser 5-37
in Spectrum Viewer 5-37, 5-101
saving in Filter Viewer 5-37
saving in Signal Browser 5-37
saving in Spectrum Viewer 5-37
`medfilt1` 4-28, 6-9, **6-235**
median filter 4-28, 6-235
message signal 4-29, 6-236
Method pop-up menu, Spectrum Viewer 5-103
MEX-file 1-13
M-files 1-3
creating xii, 1-3
modifying xii

Meta Platforms, Inc. - Exhibit 1009
Page 707 of 720

viewing xii
minima, local 5-37
minimax method, FIR filter design 2-22
See also Parks-McClellan method
minimum phase filter 6-265
models
autoregressive 6-16, 6-17, 6-18, 6-19
via Burg method 6-239
via covariance method 6-243
via modified covariance method 6-247
via Yule-Walker AR method 6-278
models, system representation 1-32
modified covariance method 3-6, 3-22
defined 3-22
example 3-23
modified periodogram 3-9
`modulate` 4-29, 6-9
example 4-30
method flags 4-29
modulation 6-236
amplitude
defined 4-29
example 4-30
frequency
methods 4-29, 6-236
phase 4-30
pulse time 4-30
pulse width 4-30
quadrature amplitude 4-30
mouse zoom 5-32
in Filter Designer 5-20
in Filter Viewer 5-20, 5-88
in Signal Browser 5-20
in Spectrum Viewer 5-20
turning off 5-33
Mouse Zoom button 5-32
moving average (MA) filter 1-15
See also FIR filter
MTM. *See* multitaper method
multiband filter
FIR 2-21
FIR, with transition bands 2-22
IIR 2-13
multichannel data 1-4, 1-7
multichannel signal 3-4
multiple signal classification method (MUSIC) 3-5, 3-6, 3-23
defined 3-23
in Spectrum Viewer 5-106
multiplicity, of zeros and poles 5-71
multirate filter bank, implementation 1-19
multirate filtering 1-19
multitaper method (MTM) 3-5, 3-6, 3-16
compared to Welch's method 3-19
defined 3-16
example 3-17
in Spectrum Viewer 5-105
MUSIC. *See* multiple signal classification method

N

New Design button 5-17, 5-59
noninteger delay 2-26
nonrecursive filter. *See* FIR filter
normalization 3-3
correlation 3-4, 6-367, 6-368
power spectral density 3-12
Nyquist frequency xvii, 2-2

O

objects, editing in SPTool 5-15
one-time mouse zooming 5-32
Open Session menu item 5-6

Meta Platforms, Inc. - Exhibit 1009
Page 708 of 720

order estimation 2-7, 6-294
Butterworth 6-48
Chebyshev type I 6-55
Chebyshev type II 6-60
elliptic 6-132
in Filter Designer 5-75, 5-78
order selection 2-7
in Filter Designer 5-75, 5-78
order, of filter 1-15, 2-7
orthogonal windows, in PSD estimates 3-16
oscillator, voltage controlled 6-365
overlap-add method, FIR filter implementation 1-22, 6-142

P

Page Setup dialog box 5-45, 5-99
panner 5-52
in Signal Browser 5-20, 5-23, 5-48
Panner check box, in Preferences dialog box 5-23
parameters
in Spectrum Viewer 5-99, 5-100, 5-103
inheriting in Spectrum Viewer 5-100
parametric modeling 4-10, 6-207
applications 4-10
frequency domain based 4-16
summary 2-5
techniques 4-10
time domain based
linear predictive coding 4-12, 4-13
Steiglitz-McBride method 4-15
time-domain based 4-11
Burg method 6-16
covariance method 6-17
modified covariance method 6-18
Yule-Walker method 6-19
parentheses, indicating open interval xvii
Parks-McClellan method, FIR filter design 2-22, 6-287
partial fraction 1-42
partial fraction expansion 1-40
defined 1-35
determining with `residue` 1-41
example 1-35
partial fraction form 1-35, 6-302
passband
equiripple 2-9, 2-10
group delay 2-11
passband ripple, setting in Filter Designer 5-70
passband zoom 5-33
Passband Zoom button 5-33
`pburg` 3-6, 3-21, 6-7, **6-239**
example 3-21
`pcov` 3-6, 3-22, 6-7, **6-243**
example 3-23
Peaks button, Signal Browser 5-37
periodic sinc function 6-118
See also Dirichlet function
periodogram 3-7
modified 3-9
persistent mouse zooming 5-32
phase
computing with `angle` 6-15
of Fourier transform of sequence 1-44
of frequency response 1-26
viewing in Filter Viewer 5-91
of transfer function estimate 3-15
unwrapping 1-27, 6-360
Phase check box, Filter Viewer 5-86
phase delay 1-28, 6-182
defined 1-28
example 1-29
of linear response filter 2-17
phase demodulation 6-115

Meta Platforms, Inc. - Exhibit 1009
Page 709 of 720

phase distortion
eliminating
during filtering 1-19
example 1-20
using `filtfilt` 1-20
in FIR filters 1-20
nonlinear
in IIR filters 1-20
phase modification
data dependent, using `>cceps` 4-25
phase modulation 4-30, 6-237
phase response 5-17
phase response plot 5-86, 5-91
Phase Units pop-up menu, in Preferences dialog box 5-25
phase units, in Filter Viewer 5-25
Play menu item, Signal Browser 5-46
playing a signal 5-46
plot
analog prototypes 2-12
coherence function 3-16
complex cepstrum 4-24
DFT 1-44
frequency response 1-25
magnitude 1-26
phase 1-26
group delay 1-29, 5-86, 5-93
impulse response 5-86, 5-94
in Filter Viewer 5-84, 5-86, 5-88, 5-89
magnitude response 5-78, 5-86, 5-89
modified periodogram 3-10
multitaper estimate 3-17, 3-18
periodogram 3-7
phase delay 1-29
phase response 5-86, 5-91
power spectral density 3-11
spectral density 5-101
step response 5-86, 5-95
strip plot 6-344
tiling in Filter Viewer 5-88
transfer function 3-15
zero-pole 1-30, 5-86, 5-94, 6-385
plug-ins 5-20, 5-30
`pm` 4-30
`pmcov` 3-6, 3-22, 6-7, **6-247**
example 3-23
p-model. *See* parametric modeling
`pmtm` 3-6, 6-7, **6-251**
example 3-17
`pmusic` 3-6, 3-23, 6-7, **6-255**
pole-zero filter. *See* IIR filter
`poly` 1-33, 1-42
`poly2ac` 6-4, **6-262**
`poly2rc` 6-4, **6-263**
polynomial
division 4-33, 6-113
multiplication 6-83
roots 1-33
stabilization 6-265
polyphase filtering techniques 1-19
`polystab` 6-9, **6-265**
power spectral density 6-273
bias 3-12
computation parameters 5-103, 5-104
confidence interval 3-14
default plot 5-102
defined 3-5
estimation by Burg method 3-6, 3-20, 5-104, 6-239
estimation by covariance method 3-6, 3-22, 5-105, 6-243
estimation by FFT method 5-105
estimation by modified covariance method 3-6, 6-247

Meta Platforms, Inc. - Exhibit 1009
Page 710 of 720

estimation by multitaper method 3-6, 3-16, 5-105
estimation by MUSIC method 3-6, 3-23, 5-106
estimation by Welch's method 3-6, 3-10, 5-106
estimation by Yule-Walker AR method 3-6, 3-19, 5-107, 6-278
estimation methods 5-103, 5-104
in SPTool 5-13
normalization 3-12
units of 3-11
viewing in Spectrum Viewer 5-97, 5-102, 5-104
preferences
rulers 5-21
saving in Signal Browser 5-45
preferences file
in SPTool 5-31
`sigprefs.mat` 5-31
Preferences menu item 5-6, 5-20, 5-30
prewarping 6-25
Print dialog box 5-45, 5-55, 5-99, 5-108, 5-120, 5-124
Print Preview window
Handle Graphics 5-57, 5-110
Signal Browser 5-45, 5-55, 5-56
preferences 5-56, 5-120
Spectrum Viewer 5-99, 5-108, 5-124
preferences 5-109
printing
from Signal Browser 5-2, 5-21, 5-22, 5-31, 5-33, 5-38, 5-117
from Spectrum Viewer 5-2, 5-21, 5-22, 5-31, 5-33, 5-38, 5-98
prolate-spheroidal window 4-4
`prony` 2-5, 4-11, 4-13, 6-8, **6-266**
Prony's method 4-13, 6-266
modeling 4-13
prototype
Bessel filter 2-11, 6-21
Butterworth filter 2-8, 6-42
Chebyshev type I filter 6-54
Chebyshev type II filter 6-59
elliptic filter 6-131
PSD. *See* power spectral density
`psd`. *See* `pwelch` 6-268
`ptm` 4-30
pulse time demodulation 6-115
pulse time modulation 4-30, 6-237
pulse train generator 6-269
pulse trains
generating 1-9
`pulstran` 1-9
pulse width demodulation 6-115
pulse width modulation 4-30, 6-237
`pulstran` 1-9, 6-2, 6-75, 6-119, 6-121, 6-122, 6-123, 6-124, **6-269**, 6-286, 6-346, 6-359
`pwelch` 3-6, 3-10, 6-7, **6-273**
example 3-12
`pwm` 4-30
`pyulear` 3-6, 3-20, 6-7, **6-278**
example 3-20, 3-21

Q

`qam` 4-30
quadrature amplitude demodulation 6-115
quadrature amplitude modulation 4-30, 6-237

R

radar applications 4-27
radix-2 algorithm 1-45
raised cosine filter design 6-171
`randn` xiv

Meta Platforms, Inc. - Exhibit 1009
Page 711 of 720

random number, generation xiv
range notation xvii
Range pop-up menu, Filter Viewer 5-87
`rc2poly` 6-4, **6-283**
`rceps` 4-23, 4-25, 6-9, **6-285**
real cepstrum 6-285
  defined 4-24
  reconstructing signal (minimum-phase) 4-25
rebuffering 6-33
rectangular window 2-18, 4-2, 6-32
`rectpuls` 6-2, **6-286**
recursive filter. *See* IIR filter
references 1-46, 3-27, 4-40
reflection coefficients 1-37, 1-39
  converting from autocorrelation sequence 6-14
  converting from filter coefficients 6-263, 6-283
  converting to autocorrelation sequence 6-282
`remez` 2-16, 2-22, 6-6, **6-287**
  accessing from Filter Designer 5-73, 5-75
  compared to `firls` 2-23
  filter characteristics 6-292
  for differentiator design 2-26
  for Hilbert transformer design 2-25
  order estimation 6-294
  weight vector 2-24
Remez exchange algorithm 2-22, 6-287
`remezord` 2-16, 6-6, **6-294**
  accessing from Filter Designer 5-75
`resample` 6-10, **6-298**
resampling 4-20, 6-298
  in FIR filtering 1-19
  See also decimation, interpolation
`residue` 1-41, 1-42
residue form. *See* partial fraction form
`residuez` 1-42, 6-4, **6-302**
Revert panel 5-31
ripple, passband 5-70
`rlevinson` 6-9, **6-305**
roots
  of Bessel filter 6-21
  polynomial 1-33
`roots` 1-33, 1-42
ruler color 5-21
Ruler Color edit box, in Preferences dialog box 5-21
Ruler Marker pop-up menu, in Preferences dialog box 5-21
ruler markers 5-21, 5-36
ruler type
  in Signal Browser 5-21
  in Spectrum Viewer 5-21
rulers
  bringing to center 5-35
  customizing in SPTool 5-20
  dragging 5-36
  find ruler buttons 5-35
  horizontal 5-36
  horizontal mode 5-40
  in Filter Viewer 5-26, 5-33
  in Signal Browser 5-20, 5-21, 5-23, 5-33
  in Spectrum Viewer 5-20, 5-21, 5-24, 5-33
  parameters 5-36
  positioning 5-38
  preferences 5-21
  saving measurements 5-37
  slope 5-36
  slope mode 5-42
  track 5-36
  track mode 5-41
  vertical 5-36
  vertical mode 5-39

Meta Platforms, Inc. - Exhibit 1009
Page 712 of 720

Rulers check box, in Preferences dialog box 5-23, 5-24, 5-26

S

sampling frequency
  changing in SPTool 5-9
  in SPTool 5-16
Sampling Frequency edit box, in Import dialog box 5-4, 5-9
Sampling Frequency menu item 5-16
sampling rate
  changing by noninteger factor 4-20, 6-298
  changing for irregularly spaced data 4-22
  changing with `upfirdn` 1-19
  decreasing by integer factor 6-110
  increasing by integer factor 6-199
Save Rulers button 5-37
Save Session menu item 5-6
saving changes in SPTool 5-30
saving data
  from Filter Designer 5-79, 5-110
  from Signal Browser 5-57
saving data, from Filter Designer 5-79, 5-110
saving data, from Signal Browser 5-57
saving settings, in Filter Viewer 5-86
Savitzky-Golay
  filter design 6-309
  filtering 6-311
`sawtooth` 1-8, 6-2, **6-308**
sawtooth wave 1-8
scalar
  for state-space form 1-34
  representing gain 1-33
Scale pop-up menu, Filter Viewer 5-87
Search for Plug-Ins at start-up check box, in Preferences dialog box 5-30
second-order section form
  converting from transfer function 6-347
  converting from zero-pole-gain 6-379
second-order sections 1-42
second-order sections form 1-36
  converting from state-space form 6-333
  converting to state-space 6-316
  converting to transfer function 6-318
  converting to zero-pole-gain 6-320
  defined 1-36
  filtering 6-322
  specifying in SPTool 5-12
selecting data objects in SPTool 5-15
sequence
  autocorrelation 6-13, 6-14, 6-262
settings
  restoring in SPTool 5-30
  rulers 5-21
  saving in Signal Browser 5-45
  saving in SPTool 5-31
`sgolay` **6-309**, 6-311
`sgolayfilt` **6-311**
signal
  adding noise 1-6
  analytic 4-38, 6-187
  buffering 6-33
  carrier 4-29, 6-114
  complex 5-47
  continuous (analog) 1-2
  differentiation 2-26
  discrete (digital) 1-2
  generating 1-7
  importing into SPTool 5-3, 5-7, 5-8, 5-10
  linking to spectrum 5-100
  measurements 5-37
  measurements in Signal Browser 5-36
  message 4-29, 6-236

Meta Platforms, Inc. - Exhibit 1009
Page 713 of 720

multichannel 3-4
naming in SPTool 5-16
peaks 5-37
playing 5-46
plotting 1-6
rebuffering 6-33
reconstruction
from DCT coefficients 4-37
minimum phase 4-25, 6-285
representing
in MATLAB 1-4
multichannel 1-4
single channel 1-4
selecting in Signal Browser 5-48
valleys 5-37
viewing in Signal Browser 5-17, 5-48
See also waveform
Signal Browser 5-2, 5-17, 5-43, 6-328
activating 5-17, 5-43
customizing 5-20
printing from 5-23, 5-43, 5-44, 5-54
saving data to workspace 5-57
window 5-44
Signal Processing Toolbox 1-2
signal type
array 5-14
vector 5-14
`sigprefs.mat` 5-31
`sinc` 1-10, 6-2, **6-313**
bandlimited interpolation example 6-314
sinc function 1-10, 6-313
and bandlimited interpolation 6-314
basic example 1-10
defined 1-10
sinusoidal wave 1-8
Slepian sequences. *See* discrete prolate spheroidal sequences
Slope button, for rulers 5-36
Slope control 5-38, 5-41
sonar applications 4-27
`sos2ss` 1-42, 6-4, **6-316**
`sos2tf` 1-42, 6-4, **6-318**
`sos2zp` 1-42, 6-4, **6-320**
`specgram` 4-27, 6-10, **6-323**
example 4-27, 6-365
specification lines 5-78
dragging to edit filter 5-70
specifications for filter design 2-7
spectral analysis 3-5
cross spectral density 3-14
defined 3-5
power spectral density 3-5
using Spectrum Viewer 5-97
spectral density 3-5
See also power spectral density, cross spectral density
spectral density plot
in Spectrum Viewer 5-101
spectrogram 4-27, 6-323
example 4-27, 6-365
spectrum
computing in SPTool 5-19
importing into SPTool 5-7, 5-8, 5-13
linking to signal 5-100
measurements 5-37
measurements in Spectrum Viewer 5-36, 5-101
naming in SPTool 5-16
peaks 5-37
updating in SPTool 5-19
valleys 5-37
viewing in Spectrum Viewer 5-97
viewing in SPTool 5-19
spectrum type, auto 5-15
Spectrum Viewer 5-2, 5-19, 5-97, 6-331

Meta Platforms, Inc. - Exhibit 1009
Page 714 of 720

activating 5-19, 5-97
changing plot properties 5-102
customizing 5-20
default plot 5-102
printing from 5-24, 5-97, 5-98, 5-107, 5-108
setting confidence intervals 5-107
viewing power spectral density plots 5-102
window 5-98
speech processing 4-10, 4-21
`spline` 4-22
`spt` extension 5-6
SPTool 5-2, **6-328**
activating from Signal Browser 5-54
closing 5-6
customizing 5-6, 5-20
loading 5-3
preferences 5-6
window 5-5
`sptool` command 6-11, **6-328**
`square` 1-8, 6-2, **6-332**
square wave 1-8
`ss2sos` 1-42, 6-4, **6-333**
`ss2tf` 1-42, **6-337**
`ss2zp` 1-42, 6-4, **6-338**
stabilization, polynomial 6-265
standards
Compact Disc 4-20
Digital Audio Tape 4-20
startup transients 1-22
reducing 1-21, 6-149
state-space form 1-40, 1-42
converting from second-order sections 6-316
converting from transfer function 6-350
converting from zero-pole-gain 6-383
converting to second-order section 6-333
converting to zero-pole-gain 6-338
defined 1-34
representing in MATLAB 1-34
specifying in SPTool 5-11
statistical operations 3-2
Stay in Zoom-mode After Zoom check box, in Preferences dialog box 5-23, 5-24, 5-26, 5-27
Steiglitz-McBride method 4-15, 6-341
step response 5-17
viewing in Filter Viewer 5-95
Step Response check box, Filter Viewer 5-86
step response plot 5-86, 5-95
`stmcb` 2-5, 4-11, 4-15, 6-9, **6-341**
stopband
attenuation, setting in Filter Designer 5-70
equiripple 2-10
strip plot 6-344
defined 6-344
`strips` 6-2, **6-344**
structure, digital filter
lattice 1-37
transposed direct form II 1-17
subplots 5-84
in Filter Viewer 5-86
subspace thresholds, controlling in `pmusic` function 3-25
`svd`, in `pmusic` function 3-25
swept-frequency cosine generator. *See* chirp
system identification 4-13
system models 1-32
and bilinear transformation 2-44
and filter design functions 2-7
and frequency transformation functions 2-39

T

tapers, in PSD estimates 3-16
taps 2-17

Meta Platforms, Inc. - Exhibit 1009
Page 715 of 720

texts, related 1-46
`tf2latc` 1-39, 1-42, 6-4, **6-346**
`tf2ss` 1-42, 6-4, **6-350**
`tf2zp` 1-42, 6-339, **6-352**
`tfe` 3-6, 3-14, 6-7, **6-354**
`thresh` 3-25
tiling 5-88
tiling display, in Filter Viewer 5-26
tiling preferences, in Filter Viewer 5-20
Time Response Length edit box, in Preferences dialog box 5-25
time response length, in Filter Viewer 5-25
time vector 1-6
  returned by `modulate` 4-30
time-dependent Fourier transform 4-27
time-domain analysis, in Filter Viewer 5-84
time-domain based modeling. *See* parametric modeling
toolbox
  Control Systems Toolbox 1-35, 6-198, 6-340
  Image Processing Toolbox 6-109, 6-190, 6-235
  Signal Processing Toolbox 1-2
  Symbolic Math Toolbox 6-21
  System Identification Toolbox 6-357
Track button, for rulers 5-36, 5-38, 5-40
transfer function 1-32, 1-35, 1-40, 1-42
  coefficients 1-15, 5-79
  defined 1-15
  derivation 1-15
  estimate from input and output 6-354
  estimating using Welch's method 3-14
  exporting from SPTool 5-79
  factored form 1-33
  for analog filter 1-41
  representing in MATLAB 1-32
  zero-pole-gain form 1-33
transfer function form
  converting from second-order sections 6-318
  converting to lattice 6-346
  converting to second-order sections 6-347
  converting to state-space 6-350
  specifying in SPTool 5-11
transform 4-34
  chirp *z*-transform (CZT) 4-34, 6-105
  discrete cosine 6-108
  discrete Fourier 1-43
  Hilbert 4-38, 6-187
  inverse discrete cosine 4-36, 6-190
  inverse discrete Fourier 6-192
transformations
  between system models 1-41
  bilinear 2-43, 6-25
  frequency 2-38, 6-220, 6-223, 6-225, 6-227
transition band 2-23
transposed direct form II 6-145
  initial conditions 6-150
`triang` 4-2, 6-8, **6-358**
  compared to `bartlett` 6-20
  example 4-2
triangular window 6-358
`tripuls` 6-2, **6-359**
two-dimensional operations
  autocorrelation 6-372
  convolution 6-84
    obtaining subsection 6-84
  cross-correlation 6-372
  discrete Fourier transform 1-45, 6-141
  filtering 6-148
  inverse discrete Fourier transform 1-45, 6-193
two-dimensional signal processing, with `fft2` and `ifft2` 1-45

Meta Platforms, Inc. - Exhibit 1009
Page 716 of 720

U

unit circle 6-265
unit impulse function 1-7
unit ramp function 1-7
unit sample, multichannel representation 1-7
unit step function 1-7
units of power spectral density (PSD) function 3-11
`unwrap` 1-27, 6-3, **6-360**
Update button 5-19, 5-97
`upfirdn` 1-19, 4-22, 6-10, **6-361**

V

Valleys button, Signal Browser 5-37
variance
  of correlation sequence estimate 3-4
  of power spectrum estimate 3-8
`vco` 4-29, 4-31, 6-10, **6-365**
vector
  data 1-4
  display, in Signal Browser 5-50
  for filter coefficients 1-16, 1-32
  frequency 2-24, 6-157, 6-160, 6-287, 6-376
  in SPTool 5-14
  indexing xvii, 1-15
  magnitude 2-24, 6-157, 6-160, 6-376
  time 1-6
  weighting 2-24, 6-167, 6-288
Vertical button, for rulers 5-36, 5-38
View button 5-17, 5-19
voltage controlled oscillator 4-31, 6-365

W

waveform
  aperiodic 1-8
  chirp 1-9
  chirp, example 1-9
  from sinusoids 1-6
  generating with `diric` function 1-11
  generating with `pulstran` 1-9
  generating with `sinc` function 1-10
  linear swept-frequency cosine. *See* chirp
  periodic 1-7
  sawtooth 1-8, 6-308
    example 1-8
  sinusoidal pulse, Gaussian-modulated 1-8
  square 1-8, 6-332
  triangle 6-308
Welch's method 3-6
  bias 3-12
  compared to the Burg method 3-22
  compared to the MTM method 3-19
  compared to the Yule-Walker AR method 3-20
  for cross spectral density estimation 3-14, 6-102
  for nonparametric system identification 3-14
  for power spectral density estimation 3-5, 3-10, 6-81, 6-275
  in Spectrum Viewer 5-106
  normalization 3-12
white noise 1-6
window
  applied to periodogram 3-9
  Bartlett 4-2, 6-20
  Blackman 4-4, 6-30
  boxcar 2-18
  Chebyshev 4-9, 6-64
  for filter design 2-18
  generalized cosine 4-4
  Hamming 2-20, 4-4, 6-185
  Hanning 4-4, 6-186
  Kaiser 4-4, 6-210

Meta Platforms, Inc. - Exhibit 1009
Page 717 of 720

rectangular 2-18, 6-32
shapes, overview 4-2
specifying for `fir1` 2-21
triangular 6-358
window method
FIR filter design 2-18
multiband design 2-21
single band design 2-20
window method, FIR filter design
bandpass configuration 6-152
bandstop configuration 6-152
highpass configuration 6-152
lowpass configuration 6-152
Workspace Contents list, in Import dialog box 5-7
workspace, loading variables from 5-7

X

X Label edit box, in Preferences dialog box 5-23
`xcorr` 3-2, 6-7, **6-367**
and parametric modeling 4-12
`xcorr2` 6-7, **6-372**
`xcov` 3-2, 6-7, **6-373**

Y

Y Label edit box, in Preferences dialog box 5-23
`yulewalk` 2-5, 2-13, 6-5, **6-376**
example 2-14
Yule-Walker AR method 3-5, 3-6, 3-19
compared to Welch's method 3-20
defined 3-19
example 3-20, 3-21
Yule-Walker AR method, in Spectrum Viewer 5-107
Yule-Walker equations 2-13
Yule-Walker filter design 6-376

Z

zero frequency component, centering with `fftshift` 1-45
zero-order hold. *See* averaging filter
zero-phase filtering 6-149
zero-pole analysis
example 1-30
zero-pole plots 6-385
zero-pole gain 1-42
zero-pole plot 5-86, 5-94
viewing in Filter Viewer 5-94
zero-pole-gain form 1-40
converting from second-order sections 6-320
converting from state-space form 6-338
converting to second-order section 6-379
converting to state-space 6-383
defined 1-33
representing in MATLAB 1-33
specifying in SPTool 5-11
zeros and poles 5-17
in transfer function 1-33
multiplicity of 5-71
Zeros and Poles check box, Filter Viewer 5-86
zoom controls
in Filter Designer 5-31
in Filter Viewer 5-31
in Signal Browser 5-31
in Spectrum Viewer 5-31
in SPTool 5-31
Zoom In-X button 5-32
Zoom In-Y button 5-32
Zoom Out-X button 5-32
Zoom Out-Y button 5-32
zoom persistence 5-32

Meta Platforms, Inc. - Exhibit 1009
Page 718 of 720

changing 5-32
in Filter Designer 5-27
in Filter Viewer 5-26, 5-87
in Signal Browser 5-23, 5-46
in Spectrum Viewer 5-24
zooming
in Filter Designer 5-77
in Filter Viewer 5-87
in Signal Browser 5-46
in Spectrum Viewer 5-103
one-time 5-32
persistent 5-32
`zp2sos` 1-42, 6-4, **6-379**
`zp2ss` 1-42, 6-4, **6-383**
`zp2tf` 1-42, **6-384**
`zplane` 1-30, 6-3, **6-385**
z-transform 1-15, 1-32
chirp *z*-transform (CZT) 4-34, 6-105
discrete Fourier transform 1-43

Meta Platforms, Inc. - Exhibit 1009
Page 719 of 720

Meta Platforms, Inc. - Exhibit 1009
Page 720 of 720