Signal Processing Toolset
User Manual

NATIONAL INSTRUMENTS™

June 2001 Edition
Part Number 322142B-01

Important Information

Contents

About This Manual

Organization of this Manual

The *Signal Processing Toolset User Manual* is divided into six sections and is organized as follows:

Part I—Introduction

- Chapter 1, *Signal Processing Toolset Overview*, provides an overview of the Signal Processing Toolset, its components, and installation instructions.
- Chapter 2, *Analysis Beyond FFT*, provides basic information about signal processing, Fourier transform, Gabor expansion, Wigner-Ville Distribution, wavelet transform, time-frequency transform, and the role of the Signal Processing Toolset in signal analysis.

Part II—Joint Time-Frequency Analysis

- Chapter 3, *Joint Time-Frequency Analysis*, explains the need for and approaches to joint time-frequency analysis (JTFA).
- Chapter 4, *Joint Time-Frequency Analysis Algorithms*, describes the algorithms the JTFA virtual instruments (VIs) use. The JTFA algorithms implemented in this toolset fall into the following two categories: linear and quadratic.
- Chapter 5, *Joint Time-Frequency Analysis Applications*, describes the Online and Off-line JTFA examples included with the Signal Processing Toolset. These examples are designed to help you learn more about JTFA.

Part III—Super-Resolution Spectral Analysis

- Chapter 6, *Introduction to Model-Based Frequency Analysis*, introduces the basic concepts of model-based frequency analysis.
- Chapter 7, *Model-Based Frequency Analysis Algorithms*, outlines the theoretical background of model-based frequency analysis and describes the relationship among the model coefficients, power spectra, and parameters of damped sinusoids.
- Chapter 8, *Applying Super-Resolution Spectral Analysis and Parameter Estimation*, describes a super-resolution spectral analysis

example application included with the Signal Processing Toolset. This example is designed to help you learn about model-based analysis.

Part IV—Wavelet Analysis

- Chapter 9, *The Fundamentals of Wavelet Analysis*, describes the history of wavelet analysis, compares Fourier transform and wavelet analysis, and describes some applications of wavelet analysis.
- Chapter 10, *Wavelet Analysis by Discrete Filter Banks*, describes the design of two-channel perfect reconstruction filter banks and defines the types of filter banks used with wavelet analysis.
- Chapter 11, *Wavelet Analysis Applications*, describes the 1D and 2D Wavelet Transform examples included with the Signal Processing Toolset. These examples are designed to help you learn about wavelet analysis.

Part V—Digital Filter Design Application

- Chapter 12, *Digital Filter Design Application*, describes the digital filter design (DFD) application used to design infinite impulse response (IIR) and finite impulse response (FIR) digital filters.
- Chapter 13, *IIR and FIR Implementation*, describes the filter implementation equations for IIR and FIR filtering and the format of the IIR and FIR filter coefficient files.

Part VI—Signal Processing for LabWindows/CVI

- Chapter 14, *Joint Time-Frequency Analysis for LabWindows/CVI*, describes functions used to perform JTFA analysis in LabWindows/CVI.
- Chapter 15, *Super-Resolution Spectral Analysis for LabWindows/CVI*, describes functions used to perform super-resolution spectral analysis and parameter estimation in LabWindows/CVI.
- Chapter 16, *Wavelet Analysis for LabWindows/CVI*, describes functions used to perform wavelet analysis in LabWindows/CVI.
- Chapter 17, *Using Your Coefficient Designs with DFD Utilities*, describes the DFD utilities used for filtering applications in LabWindows/CVI.

Conventions Used in This Manual

The following conventions are used in this manual:

<>	Angle brackets enclose the name of a key on the keyboard—for example, <Shift>. Angle brackets that contain numbers separated by an ellipsis represent a range of values associated with a bit or signal name—for example, DBIO<3..0>.
»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
[ ]	Numbers enclosed by brackets denote references in Appendix C, *References*, of this manual—for example, [27] refers to entry number 27 in Appendix C. Empty brackets that follow a parameter type indicate that the parameter is an array.
📝	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.
`monospace italic`	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

Related Documentation

The following documents contain information that you might find helpful as you read this manual:

- *Signal Processing Toolset Help*, available by selecting **Help»Signal Processing Help**
- *LabVIEW Help*, available by selecting **Help»Contents and Index**
- *LabVIEW User Manual*
- *LabWindows/CVI User Manual*

Part I

Introduction

This section of the manual introduces the Signal Processing Toolset and presents background information about signal processing.

- Chapter 1, *Signal Processing Toolset Overview*, provides an overview of the Signal Processing Toolset, its components, and installation instructions.
- Chapter 2, *Analysis Beyond FFT*, provides basic information about signal processing, Fourier transform, Gabor expansion, Wigner-Ville Distribution, wavelet transform, time-frequency transform, and the role of the Signal Processing Toolset in signal analysis.

1 Signal Processing Toolset Overview

This chapter provides information on the analysis areas where you can use the Signal Processing Toolset, an overview of the toolset and its components, and installation instructions.

Analysis Areas

The Signal Processing Toolset is primarily intended for the analysis of signals whose frequency contents change over time. Examples of such signals include the sound or vibration created by engines, most biomedical signals, and seismological data records. Time-frequency and wavelet transforms are widely used in modern signal analysis. The Signal Processing Toolset provides tools to process those signals for which the classical Fourier transform is not suitable, such as transient signals and signals whose frequency contents change over time.

The Signal Processing Toolset can be divided into the following three major analysis areas:

- joint time-frequency analysis (JTFA)
- super-resolution spectral analysis
- wavelet analysis

JTFA is suitable for signals with narrow-band instantaneous frequencies. The wavelet transform is suitable for signals with short time durations. Super-resolution spectral analysis, a model-based frequency analysis method, is mainly applicable when there is a small number of data samples.

The relationship of the number of samples to the frequency resolution can be quantified by the following equation:

$$\Delta f = \frac{sampling\ frequency}{number\ of\ samples} \tag{1-1}$$

where Δf denotes the frequency resolution. The frequency resolution characterizes the minimum difference that can be distinguished between two sinusoids. This analysis method is effective when you have a small number of data samples. Refer to Chapter 6, *Introduction to Model-Based Frequency Analysis*, for more information about model-based frequency analysis and its role in super-resolution spectral analysis.

Toolset Components

This section describes the Signal Processing Toolset components for JTFA, super-resolution spectral analysis, wavelet analysis, and digital filter design.

Joint Time-Frequency Analysis

The Signal Processing Toolset provides several algorithms for applications with nonstationary signals, signals where the frequency content varies with time. These algorithms include the award-winning and patented Gabor spectrogram, as well as the Wigner-Ville Distribution, Choi–Williams Distribution, short-time Fourier transform, cone-shaped distribution, and adaptive spectrogram. Applications for JTFA include speech processing, sound analysis, sonar, radar, machine testing, vibration analysis, and dynamic signal monitoring.

The Signal Processing Toolset includes a stand-alone application you can use to test the algorithms named above. To access this stand-alone application, select **Start»Programs»National Instruments»Signal Processing Toolset»NI SPT Application 6.0**. On the **SP** palette, select either **Online JTFA** or **Off-line JTFA**.

Refer to Part II, *Joint Time-Frequency Analysis*, of this manual for more information about joint time-frequency analysis.

For LabVIEW users, the JTFA VIs are located on the **Functions»Signal Processing Toolset»Joint Time-Frequency Analysis** palette. Refer to the *Signal Processing Toolset Help*, available by selecting **Help»Signal Processing Toolset**, for more information about individual VIs.

For LabWindows/CVI users, the JTFA functions are available by selecting **Library»SPT»JTFA**. Refer to Chapter 14, *Joint Time-Frequency Analysis for LabWindows/CVI*, of this manual for more information about JTFA functions for LabWindows/CVI.

Super-Resolution Spectral Analysis

Super-resolution spectral analysis is a model-based analysis method and is especially powerful when the number of data samples is limited. The Signal Processing Toolset contains a stand-alone application you can use to test algorithms such as covariance, Prony's method, principle component auto-regression (PCAR), and the matrix pencil method for model-based analysis. To access this stand-alone application, select **Start»Programs»National Instruments»Signal Processing Toolset» NI SPT Application 6.0**. On the **SP** palette, select **Super-Resolution Spectral Analysis**.

Use the super-resolution spectral analysis VIs and functions to perform high-resolution spectral analysis and parameter estimation. The parameters include amplitude, phase, damping factor, and frequency of damped sinusoids. You can use the VIs and functions for other applications such as linear prediction, signal synthesis, data compression, and system identification. These tools have a diverse range of applications in areas such as biomedicine, economics, geophysics, noise and vibration, and speech analysis.

Refer to Part III, *Super-Resolution Spectral Analysis*, of this manual for more information about super-resolution spectral analysis.

For LabVIEW users, the super-resolution spectral analysis VIs are located on the **Functions»Signal Processing Toolset»Super-Resolution Spectral Analysis** palette. Refer to the *Signal Processing Toolset Help*, available by selecting **Help»Signal Processing Toolset**, for more information about individual VIs.

For LabWindows/CVI users, the super-resolution spectral analysis functions are available by selecting **Library»SPT»SRSA**. Refer to Chapter 15, *Super-Resolution Spectral Analysis for LabWindows/CVI*, of this manual for more information about super-resolution spectral analysis functions for LabWindows/CVI.

Wavelet Analysis

The Signal Processing Toolset provides an intuitive and interactive interface for designing filter banks and 1D and 2D wavelet transforms. To access the wavelet transform application, select **Start»Programs»National Instruments»Signal Processing Toolset»NI SPT Application 6.0**. On the **SP** palette, select either **1D Wavelet Transform** or **2D Wavelet Transform**.

You can use wavelets for feature extraction and data compression. By interactively selecting a wavelet prototype, such as equiripple or maxflat, and different finite impulse response combinations, you can easily find the best wavelet or filter bank for your application.

As you design the wavelets, you can apply them to 1D and 2D signals, or images, and immediately see the effect of the design on your signal. The Wavelet Analysis functions are especially powerful for signals that have short time duration and wide frequency bandwidth.

Refer to Part IV, *Wavelet Analysis*, of this manual for more information about wavelet analysis and filter bank design.

For LabVIEW users, the super-resolution spectral analysis VIs are located on the **Functions»Signal Processing Toolset»Wavelet Analysis** palette. Refer to the *Signal Processing Toolset Help*, available by selecting **Help»Signal Processing Toolset**, for more information about individual VIs.

For LabWindows/CVI users, the wavelet functions are available by selecting **Library»SPT»Wavelet**. Refer to Chapter 16, *Wavelet Analysis for LabWindows/CVI*, of this manual for more information about wavelet analysis functions for LabWindows/CVI.

Digital Filter Design Application

The Signal Processing Toolset contains a Digital Filter Design (DFD) application. The DFD application provides a general-purpose design tool for signal conditioning, control systems, digital signal processing, and virtual instrument applications. Use the application to design bandpass, bandstop, lowpass, and highpass filters and filters with an arbitrary magnitude response. Use the graphical user interface to design infinite impulse response (IIR) and finite impulse response (FIR) filters.

Interactively edit the magnitude response graph or the pole-zero plot in the z-plane to design filters. Test your design online with a built-in function generator, and analyze the filter using the step and impulse responses, magnitude and phase responses, and pole-zero plot. When you complete your design, you can save the filter coefficients to a file for use in other applications.

Complete the following steps to open the DFD application and locate more information about the DFD application.

1. Select **Start»Programs»National Instruments» Signal Processing Toolset»NI Digital Filter Design 6.0**.
2. Click the **Get Info** button in the **Main Menu** dialog box.
3. Select the specific filter design you want to see or select **All Information** from the **Digital Filter Design Toolkit** dialog box menu. Selecting a specific filter design provides information about just that particular filter. Selecting **All Information** provides information about the DFD application and the different filter designs.

System Requirements

To install the Signal Processing Toolset 6.0 for LabVIEW, your system must have the Full Development or Professional Development System of LabVIEW 6.0 or later.

To install the Signal Processing Toolset 6.0 for LabWindows/CVI, your system must have LabWindows/CVI 5.5 or later.

Installation

Complete the following steps to install the Signal Processing Toolset.

1. Insert the Signal Processing Toolset CD into your CD-ROM drive and double-click `setup.exe`.
2. Follow the instructions on your screen.

After you have completed the on-screen installation instructions, you are ready to run the Signal Processing Toolset.

2

Analysis Beyond FFT

Often, it is neither possible nor desirable to physically open up a system and study it. In many such instances, you can gain knowledge about a system by measuring and analyzing signals associated with the system. For example, physicists and chemists use the spectrum dispersed by a prism to distinguish between different types of matter. Astronomers determine distances between planets by examining spectra modified by Doppler Shifts. Physicians use the electrocardiograph (ECG), which traces the electrical activity of the heart, as a nonsurgical means of diagnosing heart problems. Analyzing signals can be an ideal way to examine closed systems.

This chapter provides basic information about signal processing, the Fourier transform, the Gabor expansion, the Wigner-Ville Distribution, the wavelet transform, the time-frequency transform, and the role of the Signal Processing Toolset in signal analysis.

Background

Prior to World War II, signal processing was primarily a part of physics, and scientists and engineers mainly dealt with analog signals. Then the sampling theorem, proved by the mathematician J. Whittaker [35] and applied to communication by Claude Shannon [29], led to a new era of signal processing.

Think of modern signal processing as the combination of physics and statistics. With the discovery of the sampling theorem and the advance of the digital computer, scientists are able to employ elegant mathematical approaches to process signals that our ancestors would never have been able to imagine. One such approach is the virtual prism, or Fourier transform. Applications of modern signal processing range from the control of the *Mars Pathfinder* spacecraft more than twenty million miles away from earth to the discovery of abnormal cells inside the human body.

One fundamental mathematical tool employed in signal processing is a transform. When asked to multiply the Roman numerals LXIV and XXXII, only a few of us can immediately give the correct answer. However, after you translate the Roman numerals into the Arabic numerals 64 and 32, you can solve to get 2048. The process of converting the unfamiliar Roman numerals into common Arabic numerals is a typical example of transforms [17]. Properly applying transforms can simplify calculations or make certain attributes of a signal explicit.

Fourier Transform

One of the most popular transforms known to scientists and engineers is the Fourier transform, which converts a signal from the time domain to the frequency domain. The Fourier transform is extremely useful when applied correctly. Two hundred years ago, during the study of heat propagation and diffusion, Jean Baptiste Joseph Fourier found a series of harmonically related sinusoids useful to represent the temperature distribution throughout a body. That method of computing the weight of each sinusoidal function is now known as the Fourier transform. The Fourier transform not only benefits the study of heat distribution, but it is also useful in many other mathematical operations, such as solving differential equations. The example that scientists and engineers are most familiar with is the convolution theory. You can apply the Fourier transform to convert time-consuming convolutions into more efficient multiplications.

The Fourier transform acts as a mathematical prism to break down a signal into a group of waveforms, or different frequencies, as a prism breaks up light into a color spectrum. With the help of the Fourier transform, scientists can interpret radiation from distant galaxies, diagnose illness in a developing fetus, and make inexpensive cellular phone calls. With the establishment of quantum mechanics, the significance of Fourier's discovery is even more obvious. For example, with the Fourier transform, scientists can quantitatively describe the Heisenberg uncertainty principle, a fundamental and inescapable property of the world. The Heisenberg uncertainty principle states that certain pairs of quantities, such as the position and velocity of a particle, cannot both be predicted with complete accuracy.

The Fourier transform is so powerful that people tend to apply it everywhere without noticing one fundamental difference between the mathematical prism and a real prism. A real prism produces instantaneous spectra. Spectra produced in the evening are different than spectra produced that morning. When using a real prism to examine spectra of light, you need no previous knowledge about the light to produce the spectra. However, to compute the Fourier transform, you need to examine information over time.

The spectrum computed by the Fourier transform is the spectrum averaged over an infinitely long period of time before the present to an infinitely long period of time after the present.

Figure 2-1 illustrates two linear chirp signals. Each is a time-reversed version of the other. Whereas frequencies of the signal on the left increase with time, frequencies of the signal on the right decrease with time. Although the frequency behavior of the two signals is obviously different, their frequency spectra computed by the Fourier transform, as shown in Figure 2-2, are identical. The Fourier transform preserves all information about the time waveform. Otherwise, the signal could not be reconstructed from the transform.

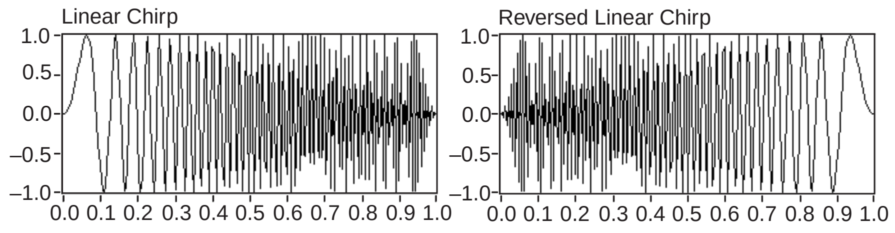


Figure 2-1. Time-Reversed Linear Chirp Signals

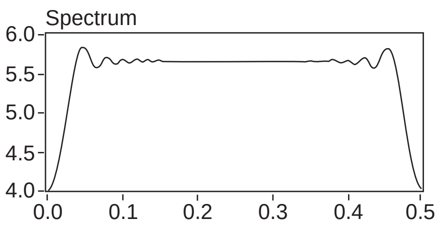


Figure 2-2. Frequency Spectra Computed by the Fourier Transform for Time-Reversed Linear Chirp Signals

Figure 2-3 depicts the spectrum of an engine sound, and the top plot of Figure 2-4 depicts the corresponding time waveform. If you could hear this signal, you could clearly identify several knocking sounds caused by out-phase firing inside the engine. As indicated by the wavelet transform, the second plot in Figure 2-4, the knocking sound is actually quite strong. To compute the Fourier transform, you have to include the signal before the knocking takes place and the signal after the knocking ends. The spectrum computed with the Fourier transform indicates that the frequencies contained in the entire time waveform, not the frequencies in a particular time instant. The Fourier transform provides the average signal characteristics. Although the amplitude of engine knock sounds can be large in a very short time period, the energy of the sound, compared to the total background noise, is negligible. Because the sound energy of engine knocking is relatively small, the presence of engine knocks is completely overwhelmed in the averaged spectra computed by the Fourier transform. Consequently, there are no obvious signatures in the spectrum to show the presence of engine knock. The Fourier transform smears the signal's local behavior globally.

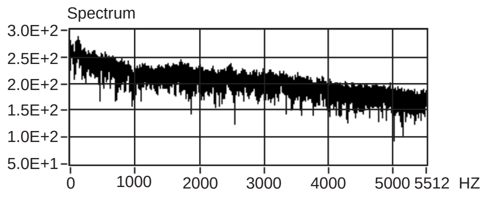


Figure 2-3. Engine Sound Spectrum

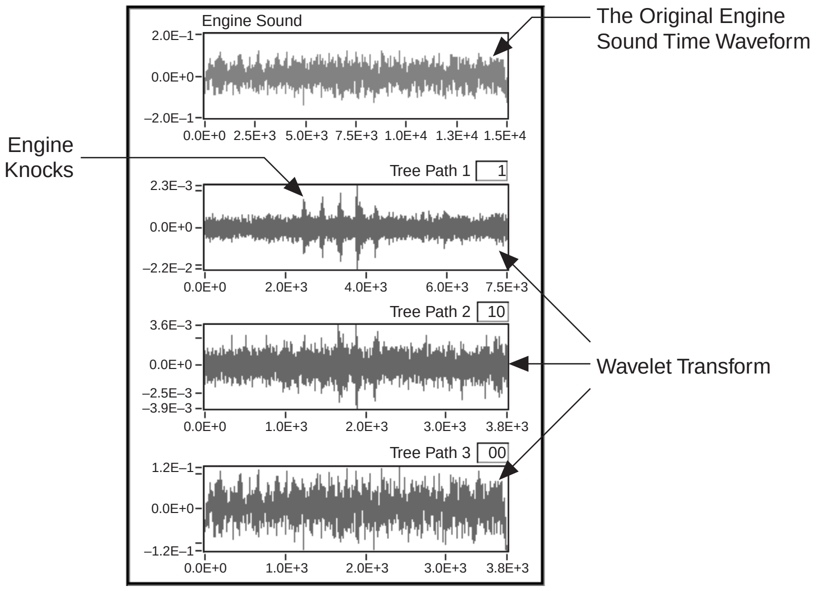


Figure 2-4. Engine Sound Time Waveform and Wavelet Transform

As Hubbard observed, “the Fourier transform is poorly suited to very brief signals, or signals that change suddenly and unpredictably; yet in signal processing, brief changes often carry the most interesting information” [11]. Although most natural spectra are time dependent, for example, morning and evening light, the Fourier transform makes “changing frequency” unthinkable. As Gabor once wrote, “even experts could not at times conceal an uneasy feeling when it came to the physical interpretation of results obtained by the Fourier method” [12].

The set of basic functions employed by Fourier, sine and cosine functions, is the mathematical model of the most fundamental natural phenomena, the wave, and a solution of differential equations. Unfortunately, this is not the case for time-frequency and wavelet transforms. Neither time-frequency nor wavelet transforms are likely to have the revolutionary impact upon science and engineering that the Fourier transform has had. However, the time-frequency and wavelet transforms do offer many interesting features that the Fourier transform does not possess.

Gabor Expansion and Wigner-Ville Distribution

The development of Fourier’s alternatives started at least a half century ago and has involved many people. The first two important articles dealing with the limitation of the Fourier transform appeared right after World War II, one written by Dennis Gabor [12] and the other by J. Ville [33]. Because Ville’s conclusion was similar to a process introduced by Eugene Wigner in quantum mechanics in 1932 [36], traditionally Ville’s method is known as the Wigner-Ville Distribution.

However, initial reactions to Gabor’s and Ville’s work was not enthusiastic. The difficulty with the Gabor expansion is that a set of elementary functions suitable for general time-frequency analysis does not form an orthogonal basis. The problem with the Wigner-Ville Distribution is the crossterm interference that makes the resulting presentation difficult to interpret. However, two sets of papers in the early 1980s triggered great interest in revisiting Gabor’s and Ville’s pioneering work [1] [3]. Since the early 1980s, scientists have made many developments, some of which are now mature applications.

Wavelet Transform

The recognition of the wavelet transform is much more recent, though a similar methodology can be traced to the early twentieth century [9]. Wavelets are not a new idea, and the concept has existed in other forms in many different fields. For example, the numerical implementation of the wavelet transform is nothing more than the well-established method of filter banks.

In addition to detecting engine knocks, the wavelet transform is also successfully used for train wheel diagnosis. Two of the main causes of train accidents are defective wheels and bearings. Hence, on-line train wheel and bearing diagnoses are an important part of avoiding potential catastrophes. The parameter that engineers believe can be used to effectively detect hidden flaws in wheels and bearings is the variation in railroad track vibration. The defective wheels and bearings usually generate impulse-like noise as the train moves on the track, making abnormal track vibrations. The wavelet transform can effectively filter out such noise.

Figure 2-5 illustrates a typical train wheel on-line testing result. When a wheel is far away from the accelerometer mounted beneath the track, the corresponding track vibration is weak. Track vibration increases as the train wheel approaches the accelerometer. The vibration reaches a maximum when a wheel is right above the accelerometer. The plot on the top of Figure 2-5 shows the vibration history during the time that eight wheels pass the accelerometer. The x-axis describes the time index and the y-axis indicates the magnitude of track vibration. Each bump corresponds to one wheel passing over the accelerometer. Obviously, there is no clear signature between the normal and abnormal wheels in the time waveform. However, in the wavelet transform domain, the plot on the bottom of Figure 2-5, you can identify a potential problem at the fifth wheel, between $x = 500$ and $x = 550$. The wavelet-transform-based on-line diagnosis system is expected to substantially reduce potential train accidents caused by defective wheels and bearings.

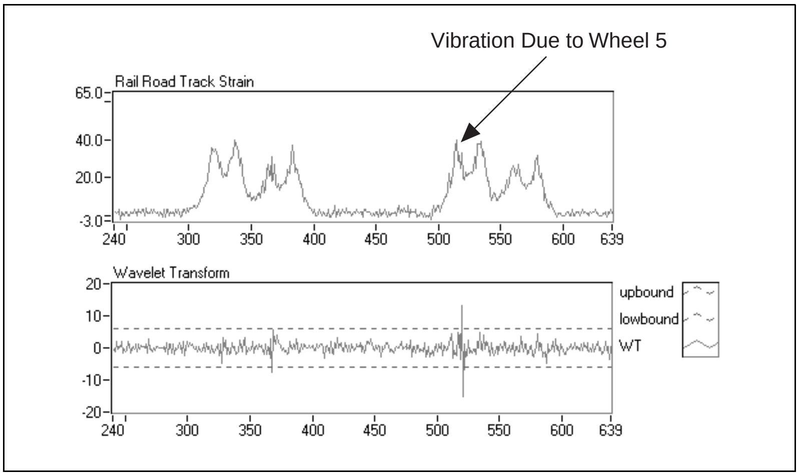


Figure 2-5. Train Wheel On-Line Testing Result

The Fourier transform compares a signal with a set of sine and cosine functions. Each sine and cosine function has a different oscillating frequency. Hence, the result of Fourier transform indicates magnitudes of the signal at each individual frequency. The wavelet transform compares a signal with a set of short waveforms called wavelets. Each wavelet has a different time duration, or scale. The shorter the time duration, the wider the frequency bandwidth, and vice versa. In mathematical terms, the process of stretching or compressing the fundamental wavelet, usually called the mother wavelet, is called dilating. As wavelets get narrower and narrower, they eventually become impulse-like functions, equivalent to wide frequency bands. Consequently, the wavelet transform can process impulse-like signals, such as engine knocks and noise created by defective train wheels and bearings. In those examples, the wavelet transform is superior to the Fourier transform.

Besides wide-band, or short time duration, signal detection, the wavelet transform is also widely used for 2D image processing. Figure 2-6 is the 2D wavelet transform of Gabor’s picture. In this example, the full image on the left is broken into four sub-images. The upper-left subimage, which is a quarter of the size of the original image, contains the major features of Gabor’s portrait. The remaining three sub-images have relatively less important information, and thus have less influence on the reconstruction. This suggests that instead of storing or transferring the entire large image, you only need to store or transfer the upper-left image and the number of prominent pixels in the other three sub-images. Because the number of pixels in each sub-image is a quarter of the number of pixels contained in the original image, by applying a 2D wavelet transform, you can save a lot of memory and communication bandwidth.

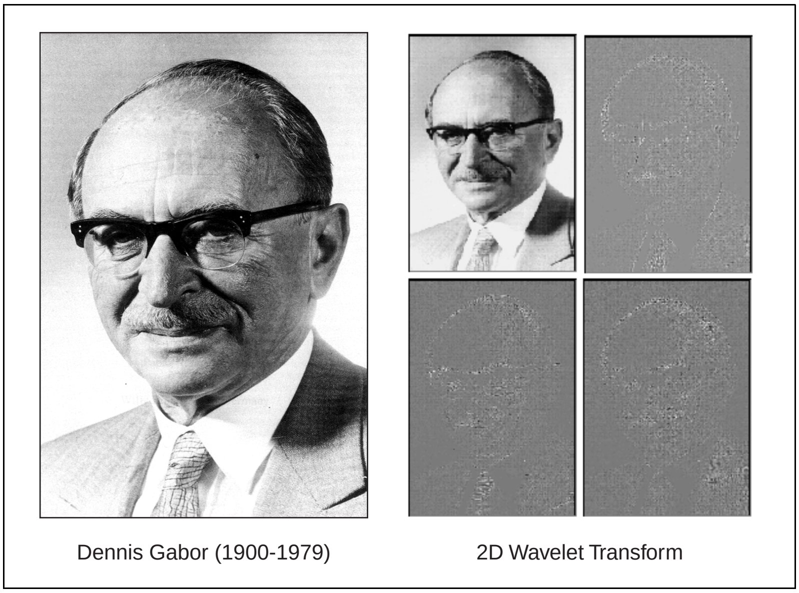


Figure 2-6. Gabor Portrait and Sub-Images

Time-Frequency Transform and the Gabor Spectrogram

While the wavelet transform is well-suited for the analysis of predominately nonstationary signals with sudden peaks or discontinuity, the time-frequency transform is effective for analyzing narrow-band signals or signals whose frequency changes slowly with time. The detection of impulse signals by low-orbit satellites is a good example of a time-frequency transform application.

The detection and estimation of impulse signals has been an important national security issue because nuclear weapon testing can cause impulse signals. Figure 2-7 depicts an impulse signal received by the U.S. Department of Energy ALEXIS/BLACKBEARD satellite. After passing through dispersive media, such as the ionosphere, the impulse signal turns into a non-linear chirp signal. While the time waveform is severely corrupted by random noise, the conventional spectrum is dominated by radio carrier signals that remain basically unchanged over time. As shown in Figure 2-7, neither the time waveform nor the power spectrum indicates the existence of the impulse signal. However, when looking at the amplitudes of the Gabor coefficients, computed by the short-time Fourier transform, you can identify the presence of the chirp-type signal arching across the joint time-frequency domain.

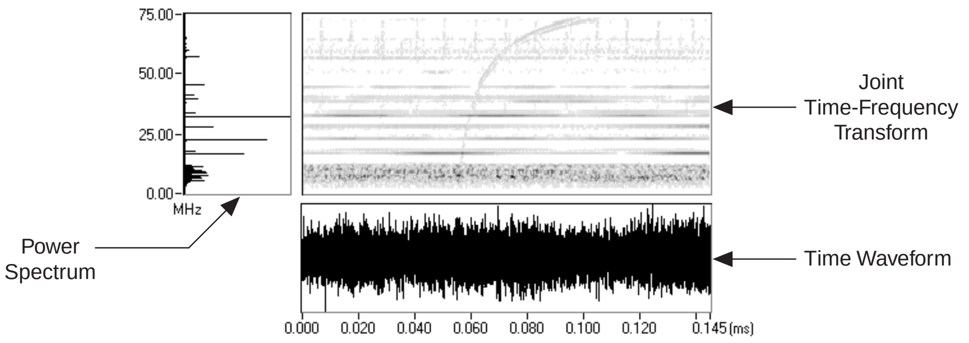


Figure 2-7. Impulse Signal Received by ALEXIS/BLACKBEARD Satellite (Data courtesy of Non-Proliferation & International Security Division, Los Alamos National Laboratory)

From the joint time-frequency domain, you can mask the Gabor coefficients that correspond to the desired signal, as shown in Figure 2-8. After the masking operation, apply the Gabor expansion to recover the original time waveform. Figure 2-9 compares the noisy and reconstructed signals.

As shown in Figure 2-7, the interesting signal in this example cannot be detected from either the time waveform or the conventional spectrum. When the signal-to-noise rate (SNR) is very low, as with many satellite signals, the short-time Fourier transform and Gabor expansion could be the only choices for detection and estimation.

The time-frequency transform describes how the spectrum of a signal changes with time, and that is its major advantage. Such an instantaneous spectrum is very useful in many applications. One such application is aneurysm research. An aneurysm is a bulge or sac formed by the ballooning of the wall of an artery or a vein. It can become the site of a blood clot that breaks away and lodges in vital organs, such as the heart or the brain, causing potentially fatal heart failure or brain damage. Except for Computer Tomography (CT) and Magnetic Resonance Imaging (MRI), at present, there is no simple and economical method to screen for aneurysms. The results obtained from the Gabor spectrogram could eventually lead to an economical aneurysm screening test.

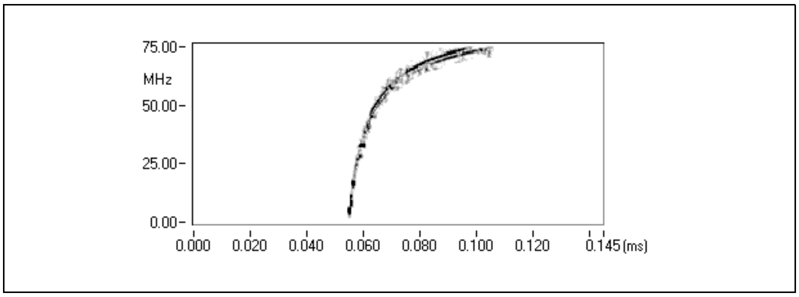


Figure 2-8. Signal Masked from Noisy Background

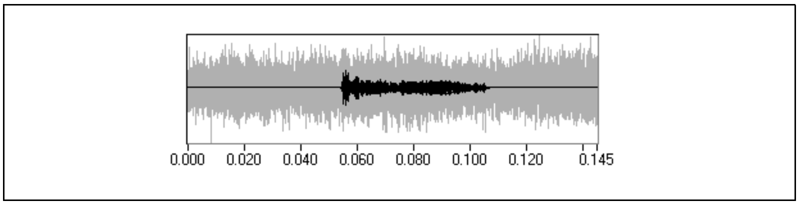


Figure 2-9. Reconstructed Signal

Some aneurysms emit specific resonant sounds of varying frequencies. So, the frequency of the sound created by blood flow is a potential feature for diagnosing an aneurysm. However, the sound associated with an aneurysm is generated by a complicated, dynamic fluid system involving the arterial wall, heart chamber, surrounding blood vessels, and moving blood, all under varying pressure. The sound recorded from an aneurysm is caused by vibration stimulated by the blood flow inside the aneurysm and nearby blood vessels. This vibrational system is non-linear and time-varying. In addition, the sound emitted by an aneurysm is non-stationary and is usually combined with the biological noises generated by the heart, respiratory system, and eye movements.

Figure 2-10 shows an aneurysm signal recorded directly from an antercranial aneurysm during surgery. This record has 512 data samples, corresponding to 300 milliseconds in time. The spectrum of the signal is illustrated on the left. It shows the range of resonant frequencies, 450 to 550 Hz, but provides no other useful information. The center plot depicts the corresponding Gabor spectrogram that describes the instantaneous spectrum of the signal.

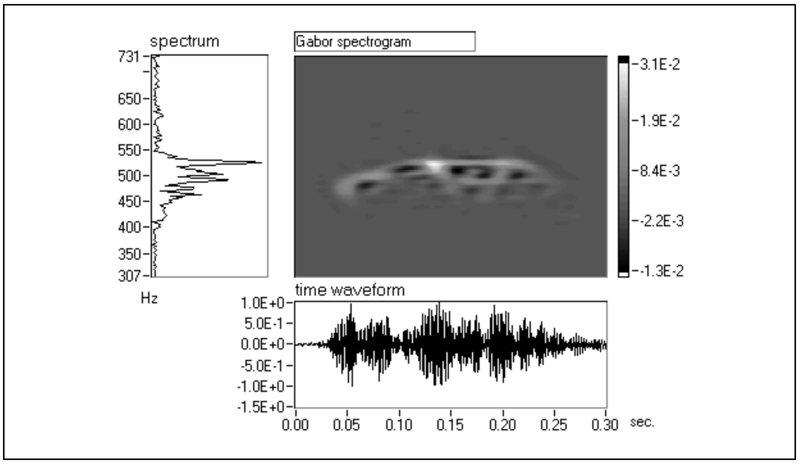


Figure 2-10. Gabor Spectrogram of Blood Flow Sound (Data courtesy of the Medical School at the University of Pittsburgh)

With the help of the Gabor spectrogram, the physical system that produces this signal could hypothetically be described as follows:

- The main arc is produced by the variation of blood pressure which changes the physical parameters of the resonant system. More specifically, at the very beginning of the vibration, in the vicinity of 0.05 seconds, the spectral component is relatively wide, corresponding to a low Q, center frequency versus frequency bandwidth, value of the vibrational system due to the existence of a large damping effect.
- As the stimulation increases, the pressure-induced vibration becomes stronger, and the spectral component narrows, which indicates an increasing Q value. As a result, between 0.1 to 0.15 seconds, the vibration tends to concentrate on a single frequency, yielding a sinusoidal-like waveform.
- As the vibration continues, another interesting phenomenon occurs. Between 0.15 to 0.25 seconds, when the vibrational magnitude increases considerably, you can observe several branches with harmonic-like patterns deviating from the main arc. However, after a short period of time, these branches merge back into the main arc again. This interesting behavior could indicate that, at the branching points, the vibrational system has reached the upper limit of its linear range. However, the stimulation is still present, which provides the system with additional energy.
- This over-stimulation causes the vibration to enter a nonlinear stage.

- Conversely, as the stimulation decreases, the vibrational system loses energy.
- Finally, the system ceases vibration when the damping effect becomes dominant again.

In this example, researchers use the Gabor spectrogram as an X-ray, to get a better understanding of the mechanism of an aneurysm system. As shown in Figure 2-10, information provided by the Gabor spectrogram is not available in either the time waveform or the conventional frequency spectrum. The resulting observation could eventually lead to an efficient means of diagnosing aneurysms that will not only be less expensive than CT and MRI, but also pain-free.

The Gabor spectrogram has also been successfully applied in earthquake engineering, with developments like the detection of soil liquefaction conducted at the University of Tokyo. Soil liquefaction is an earthquake-related phenomenon that takes place in saturated soils from the sub-surface soft layer. The cause of liquefaction is the rise of the water pore pressure under undrained conditions when the ground shakes. The increase of the pore pressure reduces the soil shear resistance to almost zero, causing the soil to behave as a liquid. Consequently, the energy of horizontal vibrations, seismic shear waves from depth, transferred by the soil is substantially reduced, particularly, the high frequency contents. Researchers recognize soil liquefaction as the main cause of the collapse of earth dams and slopes, failure of foundations, superstructures and lifelines, such as gas and electrical power supplies.

Since the 1964 Niigata earthquake, scientists have obtained and studied a number of seismic records from liquefied-soil sites. The records show that the horizontal ground acceleration alternates uniquely after the onset of liquefaction. The frequency of the acceleration abruptly drops off toward the 0.3 to 1 Hz range, and amplitudes decrease, whereas the vertical acceleration remains fairly stable. The decrease of the soil shear modulus, as a consequence of the water pore-pressure build up, triggers the alternation of the horizontal ground acceleration.

Researchers developed methods for liquefaction detection based on these seismic records. Recently, researchers from the International Center for Disaster-Mitigation Engineering at the University of Tokyo employed the seismic-signal instantaneous spectrum to quantify the alternation of the horizontal ground acceleration. Figure 2-11 shows the seismic-signal instantaneous spectrum from a site where extensive liquefaction occurred. Figure 2-12 shows the seismic-signal instantaneous spectrum from a site where no liquefaction occurred. The mean instantaneous frequency computed from the seismic-signal instantaneous spectrum can characterize the frequency changes, as shown in Figures 2-13 and 2-14. Occurrence of liquefaction is judged on the relative difference in the mean instantaneous frequency of the horizontal and vertical acceleration. Now, researchers from the University of Tokyo are able to remotely detect the occurrence of soil liquefaction. Figure 2-13 represents a typical liquefaction record, whereas Figure 2-14 indicates no liquefaction occurring.

Refer to Qian and Chen’s March 1999 article in the *IEEE Signal Processing Magazine* to learn about other successful signal processing applications [28].

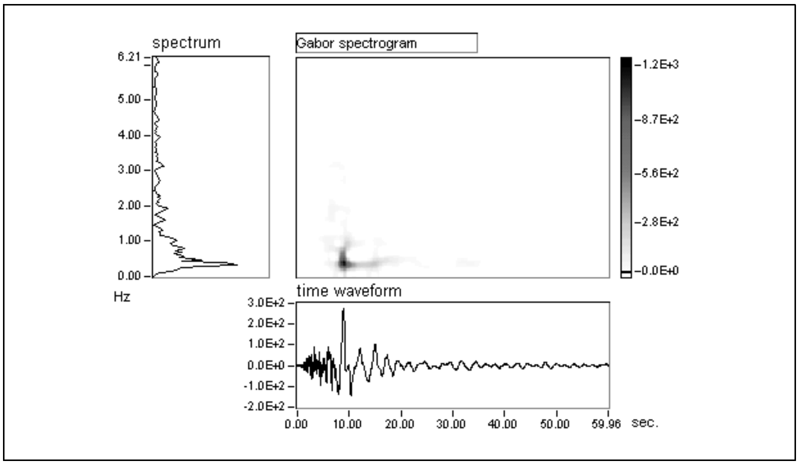


Figure 2-11. East-West Component of the Ground Acceleration Record at Higashi-Kobe Bridge from the 1995 Hyogoken-Nanbu Earthquake (Data courtesy of the International Center for Disaster-Mitigation Engineering at the Institute of Industrial Science, University of Tokyo)

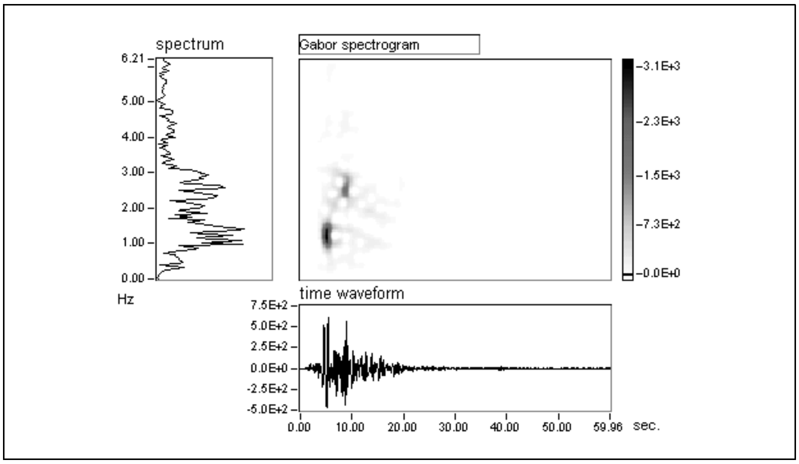


Figure 2-12. East-West Component of the Ground Acceleration Record at JMA Kobe Station from the 1995 Hyogoken-Nanbu Earthquake. (Data courtesy of the International Center for Disaster-Mitigation Engineering at the Institute of Industrial Science, University of Tokyo)

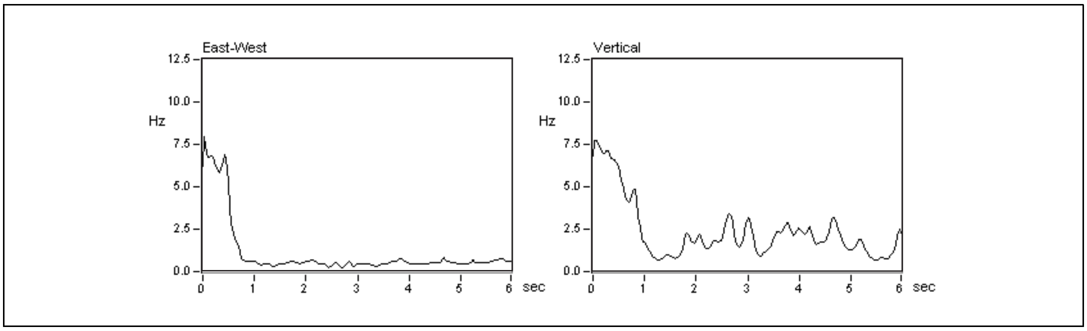


Figure 2-13. Mean Instantaneous Frequencies at the Figure 2-11 Site
The mean instantaneous frequency for East-West component, computed from Figure 2-11, is obviously lower than that for non-liquefaction case in Figure 2-14. (Data courtesy of the International Center for Disaster-Mitigation Engineering at the Institute of Industrial Science, University of Tokyo)

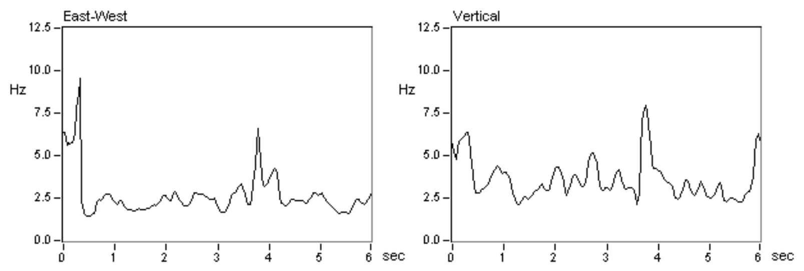


Figure 2-14. Mean Instantaneous Frequencies at the Figure 2-12 Site
The mean instantaneous frequency for East-West component is computed from Figure 2-12 (Data courtesy of the International Center for Disaster-Mitigation Engineering at the Institute of Industrial Science, University of Tokyo)

Role of the Signal Processing Toolset

There is no doubt that time-frequency and wavelet transforms have begun to pervade modern technology, as well as our everyday life. The Signal Processing Toolset provides you with the tools to process signals for which the classical Fourier transform is not suitable, such as the transient signal and the signal whose frequency contents evolve over time.

The remainder of this manual discusses the main areas of signal analysis, specifically joint time-frequency analysis (JTFA), super-resolution spectral analysis, and wavelet analysis, and how you can use the Signal Processing Toolset to perform these analyses. You will also find information about digital filter design. Part VI of the manual contains information specifically for LabWindows/CVI users.

Part II

Joint Time-Frequency Analysis

This section of the manual provides information on joint time-frequency analysis (JTFA) and the JTFA applications in the Signal Processing Toolset.

- Chapter 3, *Joint Time-Frequency Analysis*, explains the need for and approaches to JTFA.
- Chapter 4, *Joint Time-Frequency Analysis Algorithms*, describes the algorithms that the JTFA VIs and functions use.
- Chapter 5, *Joint Time-Frequency Analysis Applications*, introduces Signal Processing Toolset JTFA applications.

3 Joint Time-Frequency Analysis

This chapter explains the need for and approaches to joint time-frequency analysis (JTFA).

The Need for JTFA

Unlike traditional analysis in which you analyze a signal only in the time domain or frequency domain, JTFA allows you to analyze a signal in the time and frequency domains simultaneously. This enables you to better understand and process a particular signal. JTFA is used primarily to observe how the power spectrum of a signal changes over time. Whereas classical algorithms, such as the square of the Fourier transform, indicate only the average power spectrum of a signal, JTFA algorithms allow you to examine the instantaneous spectrum.

The upper-left plot in Figure 3-1 is a time-dependent spectrum which plots the energy of the signal as a function of both time and frequency. As shown, the time-dependent spectrum clearly reveals the pattern of the formants. From the formants, you can see how the frequency changes. The relative brightness levels of the plot show the intensity of the frequencies. In this example, the JTFA helps illustrate the mechanism of human speech.

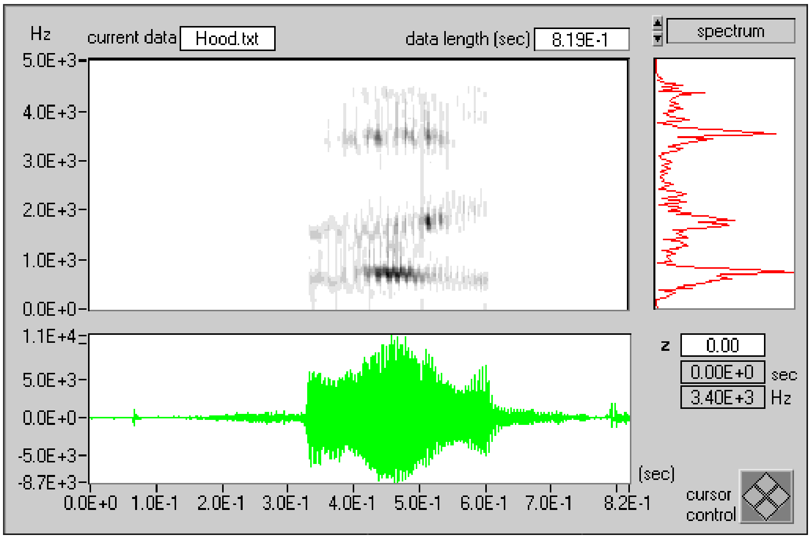


Figure 3-1. Speech Signal

Another important application with JTFA is the detection of noise-corrupted signals. In general, random noise tends to spread evenly across the time and frequency domains. However, the signal usually concentrates in a relatively short time period or a narrow frequency band. If you convert the noise-corrupted signal to the joint time-frequency domain, you can substantially improve the local or regional Signal-to-Noise Ratio (SNR).

Figure 3-2 depicts an impulse signal received by the U.S. Department of Energy ALEXIS/BLACKBEARD satellite. After passing through dispersive media, such as the ionosphere, the impulse signal becomes the nonlinear chirp signal. As shown in Figure 3-2, random noise dominates both the time waveform and the power spectrum. Neither indicates the existence of the impulse signal. However, from the time-dependent spectrum, you can immediately identify the presence of the chirp-type signal that arches across the joint time-frequency domain. The horizontal lines correspond to radio carrier signals that remain basically unchanged over time.

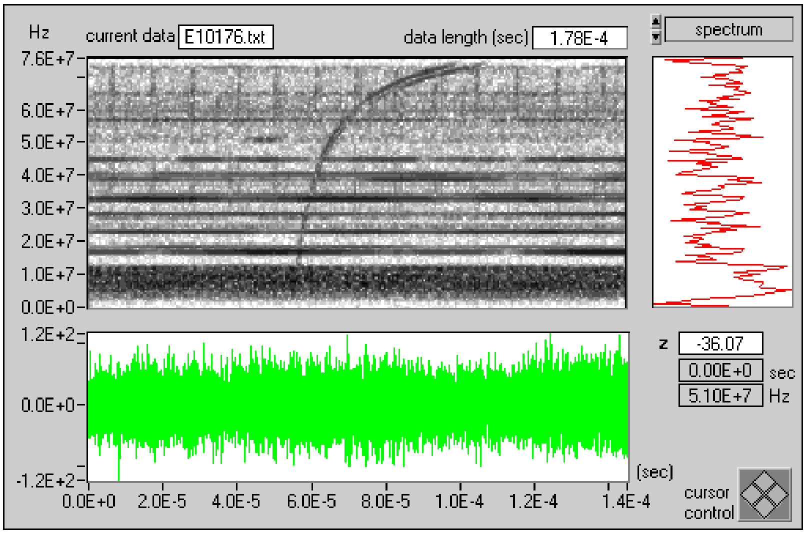


Figure 3-2. Ionized Impulse Signal (Data courtesy of Non-Proliferation and International Security Division, Los Alamos National Laboratory)

Based on the joint time-frequency representation, you can further mask the desired signal, as shown in the top plot in Figure 3-3. You can then apply the inverse transformation to recover the noiseless time waveform. The lower plot in Figure 3-3 illustrates the noisy and reconstructed signals. When the SNR is very low, as with many satellite signals, JTFA might offer the only opportunity to detect the signal of interest.

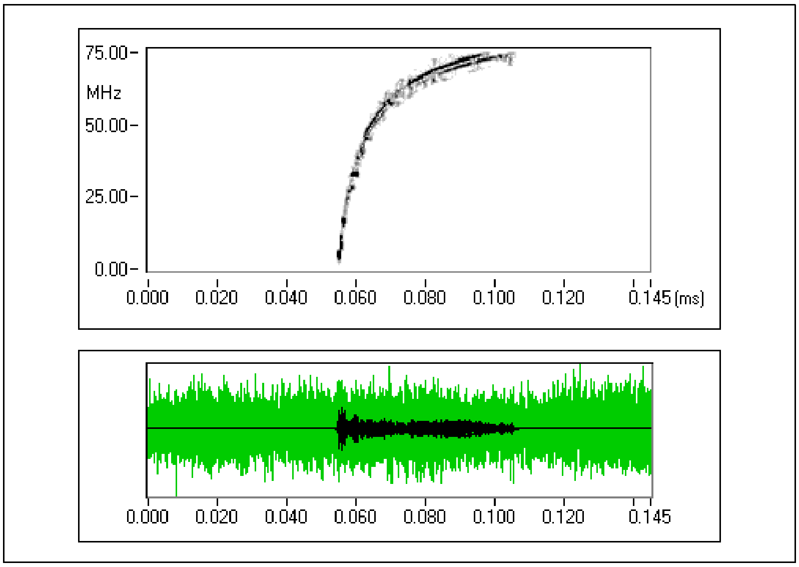


Figure 3-3. Reconstructed Signal

Basic Approaches to JTFA

The development of JTFA began more than a half century ago. The most straightforward approach to characterizing the frequency of a signal as a function of time is to divide the signal into several blocks that can be overlapped. Then the Fourier transform is applied to each data block to indicate the frequency contents of each data block. This process is known as the short-time Fourier transform (STFT) and roughly reflects how frequency contents change over time. The size of the blocks determines the time accuracy—the smaller the block, the better the time resolution. However, frequency resolution is inversely proportional to the size of a block. When the small block yields good time resolution, it also deteriorates the frequency resolution and vice versa. This phenomenon is known as the window effect.

From the concept of expansion and series, physicist Dennis Gabor suggested expanding a signal into a set of weighted frequency-modulated Gaussian functions. Because the Gaussian function is concentrated in both the time and frequency domains, the weights describe signal behavior in local time and frequency. The resulting presentation is known as the Gabor expansion. In fact, you can consider the Gabor expansion as the inverse of the STFT. However, this inverse relationship was not clear during Gabor's lifetime and not well understood until the 1980s. At present, both the theory and implementation of the Gabor expansion and STFT are mature enough to apply to real application problems.

As the linear JTFA develops, the quadratic JTFA, or time-dependent spectrum, is attracting attention. The simplest time-dependent spectrum is the square of the STFT, which is known as the STFT-based spectrogram, or the STFT spectrogram. However, the STFT spectrogram suffers from the window effect.

A more elegant method than the STFT spectrogram is the Wigner-Ville Distribution (WVD), which physicist Eugene P. Wigner originally developed in the context of quantum mechanics. The WVD gives high resolution and many other useful properties for signal analysis, but it suffers from crossterm interference. To reduce crossterm interference, you can use two proven algorithms, Cohen's class and the Gabor expansion-based spectrogram, also known as the Gabor spectrogram. Scientists at National Instruments developed the Gabor spectrogram in the early 1990s. Based on the conventional Gabor expansion and the WVD, scientists at National Instruments also introduced the adaptive representation-based spectrogram, or the adaptive spectrogram.

Unlike the linear JTFA method, the quadratic JTFA method is not unique. This toolset contains the following quadratic algorithms:

- adaptive spectrogram
- Cohen's class
  - Choi-Williams Distribution
  - cone-shaped distribution
  - STFT spectrogram
  - WVD
- Gabor spectrogram

Which method should you use? Often, the choice is application dependent. With these methods, you can process signals the conventional Fourier transform cannot handle.

4

Joint Time-Frequency Analysis Algorithms

This chapter describes the algorithms the joint time-frequency analysis (JTFA) VIs and functions use. The JTFA algorithms implemented in this toolset fall into two categories, linear and quadratic. Refer to the works of Qian and Chen [27] and Cohen [6] for more information about a particular algorithm.

Linear JTFA Algorithms

Linear JTFA includes the following methods:

- Gabor expansion, considered the inverse short-time Fourier transform (STFT)
- STFT, used for computing the Gabor coefficients
- adaptive representation, considered the inverse adaptive transform
- adaptive transform

Gabor Expansion and STFT

In Equation 4-1, the Gabor expansion represents a signal $s[i]$ as the weighted sum of the frequency-modulated and time-shifted function $h[i]$:

$$s[i] = \sum_{m} \sum_{n=0}^{N-1} C_{m,n} h[i - m\Delta M] e^{j2\pi ni/N} \tag{4-1}$$

where the Gabor coefficients $C_{m,n}$ are computed by the STFT in the following equation:

$$C_{m,n} = STFT[m\Delta M, n] = \sum_{i=0} s[i]\gamma^*[i - m\Delta M] e^{-j2\pi ni/N}$$

where N denotes the number of frequency bins and ΔM denotes the time sampling interval. You can use any function as $\gamma[i]$, as long as its dual function $h[i]$ exists. For the perfect reconstruction, the oversampling rate $N/\Delta M$, must be greater than or equal to one. For a given $h[i]$ or $\gamma[i]$, LabVIEW users can use the Fast Dual VI to compute the corresponding dual function, and LabWindows/CVI users can use the SPTFastDualFunction. Refer to the *Signal Processing Toolset Help*, available by selecting **Help»Signal Processing Toolset**, for more information about the Fast Dual VI. Refer to Chapter 14, *Joint Time-Frequency Analysis for LabWindows/CVI*, of this manual for more information about the SPTFastDualFunction.

If the STFT is not used for computing the Gabor coefficient $C_{m,n}$, there are no restrictions for $\gamma[i]$ or the ratio $N/\Delta M$.

Adaptive Representation and Adaptive Transform

In the Gabor expansion in Equation 4-1, the elementary functions $h[i-m\Delta M]e^{j2\pi ni/N}$ are time-shifted and frequency-modulated versions of the single prototype function $h[i]$. To better match the analyzed signal, the adaptive representation, shown in Equation 4-2, was developed to decompose the signal $s[i]$ as a sum of weighted linear adaptive modulated Gaussian functions:

$$s[i] = \sum_{k=0}^{D-1} A_k h_k[i] \tag{4-2}$$

where the adaptive Gaussian function $h_k[i]$ is defined by

$$h_k[i] = (\alpha_k \pi)^{-0.25} \exp\left\{-\frac{[i-i_k]^2}{2\alpha_k} + j(2\pi\theta_k[i-i_k])\right\}$$

which has three parameters: α_k, i_k, f_k. Therefore, the adaptive representation is more flexible than the elementary function used in the Gabor expansion.

The parameter D in Equation 4-2 denotes the total number of elementary functions used by $h_k[i]$. A_k is the weight of each individual $h_k[i]$, as computed by the adaptive transform.

Scientists at National Instruments [25] and Mallat and Zhang [16] independently developed the adaptive representation, also known as the matching pursuit.

Quadratic JTFA Algorithms

The quadratic JTFA algorithms include the following:

- STFT spectrogram
- Wigner-Ville Distribution (WVD)
- Pseudo Wigner-Ville Distribution (PWVD)
- Cohen's class
- Choi-Williams Distribution (CWD)
- cone-shaped distribution
- Gabor spectrogram
- adaptive spectrogram

STFT Spectrogram

The STFT-based spectrogram is defined as the square of the STFT, as shown in the following equation:

$$SP[m\Delta M, n] = \left|\sum_{i=0} s[i]\gamma[i - m\Delta M]e^{-j2\pi ni/N}\right|^2$$

where N denotes the number of frequency bins and ΔM denotes the time sampling interval. The STFT-based spectrogram is simple and fast but suffers from the window effect.

Figures 4-1 and 4-2 illustrate the window effect of the STFT spectrogram. In Figure 4-1, with a narrowband window, the time-dependent spectrum has high frequency resolution but poor time resolution. In Figure 4-2, with a wideband window, the time-dependent spectrum has poor frequency resolution but high time resolution.

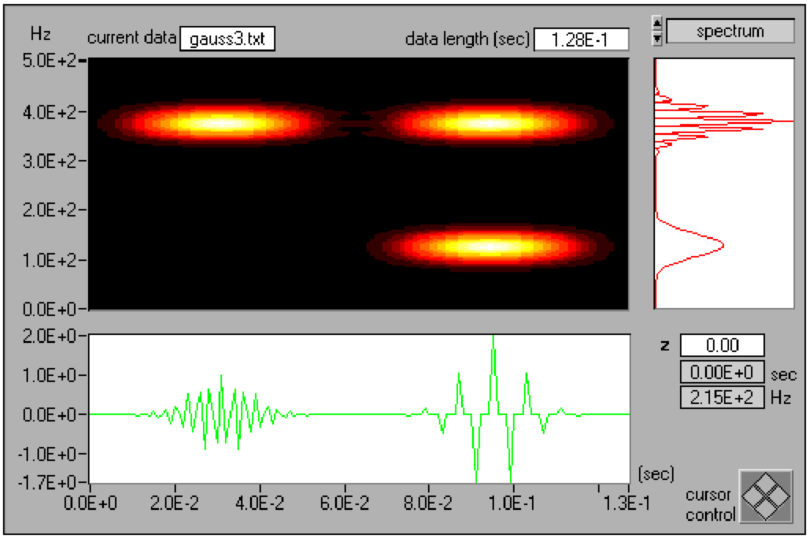


Figure 4-1. STFT-Based Spectrogram with a Narrowband Hanning Window for the Three-Tone Test Signal

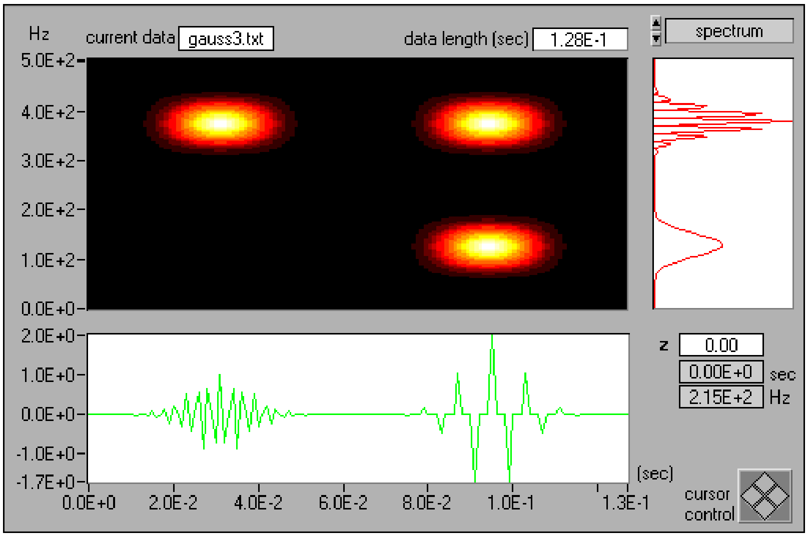


Figure 4-2. STFT-Based Spectrogram with a Wideband Hanning Window for the Three-Tone Test Signal

Wigner-Ville Distribution and Pseudo Wigner-Ville Distribution

For a signal $s[i]$, the Wigner-Ville Distribution (WVD) is

$$WVD[i, k] = \sum_{m=-L/2}^{L/2} R[i, m]e^{-j2\pi km/L}$$

where the function $R[i,m]$ is the instantaneous correlation given by

$$R[i, m] = z[i + m]z^*[i - m]$$

and $z[i]$ is the analytical, or interpolated, form of $s[i]$; see reference [27].

The WVD can also be computed by

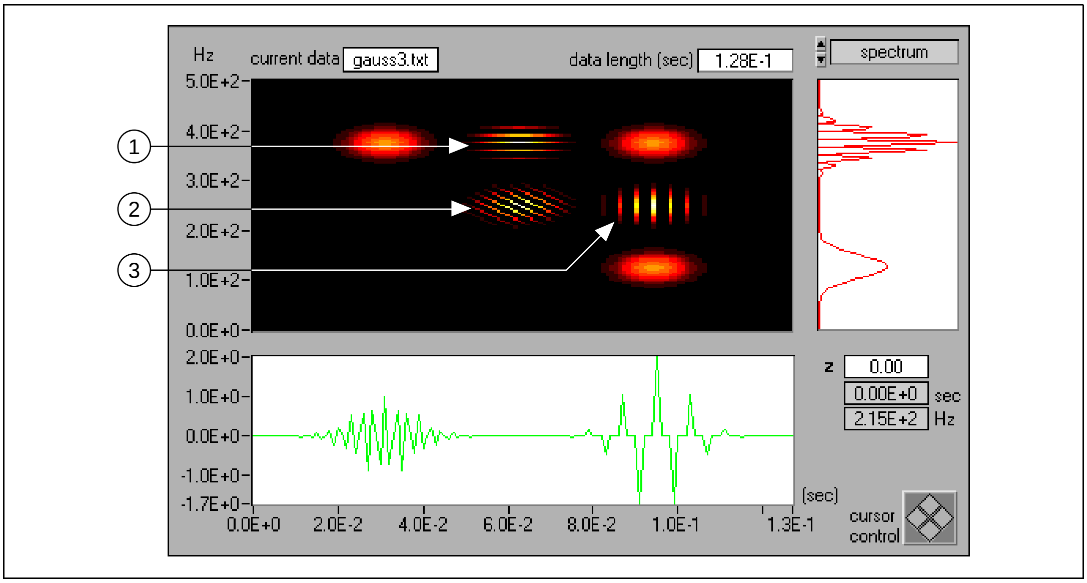
$$WVD[i, k] = \sum_{m=-L/2}^{L/2} \Re[i, m]e^{j2\pi km/L}$$

where $\Re[i, m] = Z[i + m]Z^*[i - m]$ and $Z[k]$ denotes the Fourier transform of $z[i]$.

The Wigner-Ville Distribution is simple and fast. It has the best joint time-frequency resolution of all known quadratic JTFA algorithms. However, if the analyzed signal contains more than one component, the WVD method suffers from crossterm interference.

Figure 4-3 depicts the WVD of the three-tone test signal. Three real signal terms are centered at (0.03 sec, 400 Hz), (0.09 sec, 100 Hz), and (0.09 sec, 400 Hz). Three crossterms exist and are labeled as 1, 2, and 3 in Figure 4-3.

A crossterm reflects the correlation between a pair of corresponding autoterms, always sits halfway between two corresponding autoterms, and oscillates frequently. Although the magnitude of a crossterm can be large, the average of a crossterm is usually limited.



1 crossterm 1	2 crossterm 2	3 crossterm 3

Figure 4-3. Wigner-Ville Distribution for the Three-Tone Test Signal

Autoterms at (0.03 sec, 400 Hz) and (0.09 sec, 400 Hz), which have different time centers, cause crossterm 1. Autoterms at (0.09 sec, 100 Hz) and (0.09 sec, 400 Hz), which have different frequency centers, cause crossterm 3. Autoterms at (0.03 sec, 400 Hz) and (0.09 sec, 100 Hz) create crossterm 2.

To alleviate the crossterm interference, you can assign different weights to the instantaneous correlation $R[i,m]$. Assigning different weights to $R[i,m]$ suppresses the less important parts and enhances the fundamental parts of the signal.

Two traditional methods exist for applying the weighting function to the instantaneous correlation $R[i,m]$. The first is in the time domain and known as the Pseudo Wigner-Ville Distribution (PWVD). The following equation represents the PWVD:

$$PWVD[i,k] = \sum_{m=-L/2}^{L/2} w[m]R[i,m]e^{-j2\pi km/L} \tag{4-3}$$

The PWVD effectively suppresses crossterms that correspond to a pair of autoterms with different time centers, such as crossterms 1 and 2 in Figure 4-3. Figure 4-4 illustrates the PWVD with the Gaussian window function *w*[*m*]. When compared with the WVD in Figure 4-3, the PWVD successfully eliminates crossterms 1 and 2.

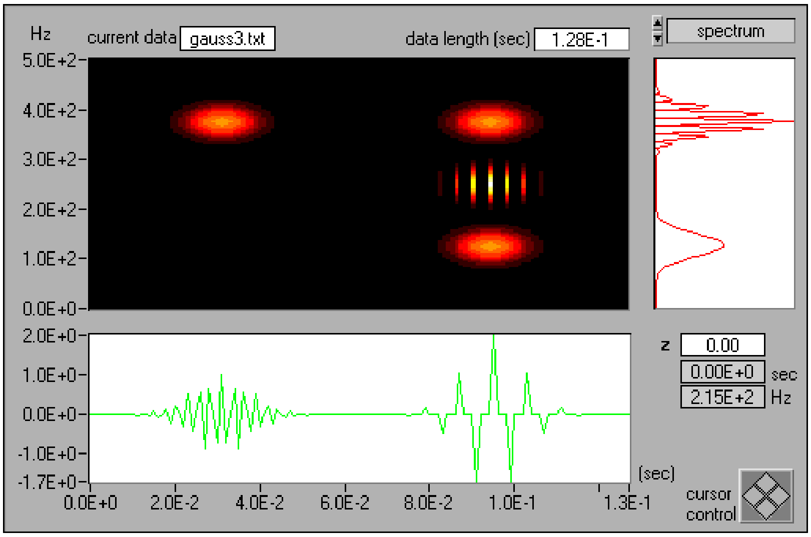


Figure 4-4. Pseudo Wigner-Ville Distribution with Gaussian Window *w*[*m*] for the Three-Tone Test Signal

In the second method, you assign weights to *R*[*i*,*m*] in the frequency domain. This method is represented by the following equation:

$$WVD[i,k] = \sum_{m=-L/2}^{L/2} H[m]\Re[i,m]e^{j2\pi km/L} \tag{4-4}$$

This weighting function effectively suppresses crossterms that correspond to a pair of autoterms with different frequencies, such as crossterms 2 and 3 in Figure 4-3. Figure 4-5 illustrates the PWVD with the Gaussian window function *H*[*m*]. Compared with the WVD in Figure 4-3, the PWVD successfully eliminates crossterms 2 and 3.

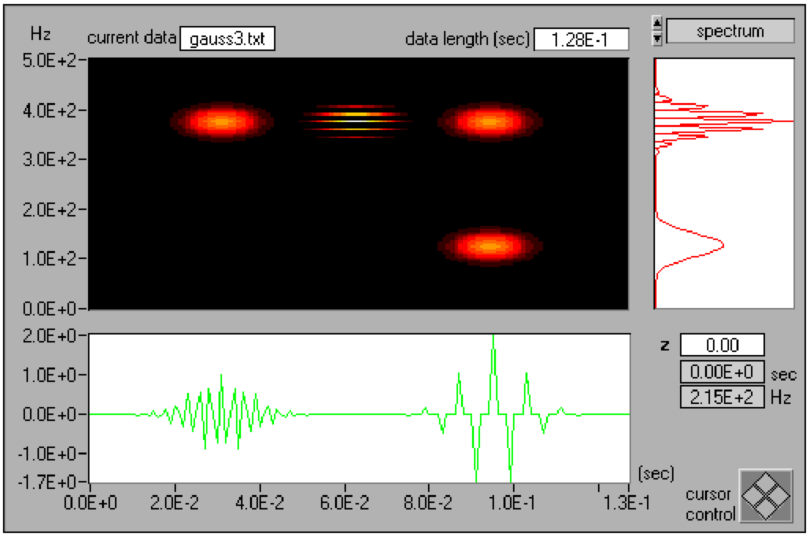


Figure 4-5. Pseudo Wigner-Ville Distribution for the Three-Tone Test Signal

Notice that Equation 4-4 is equivalent to

$$PWVD[i, k] = \sum_{m=-L/2}^{L/2} \left(\sum_{n} h[n]R[i-n, m] \right) e^{-j2\pi km/L} \tag{4-5}$$

where *h*[*n*] is the inverse Fourier transform of *H*[*m*] in Equation 4-4.

Cohen's Class

Because the crossterm often oscillates in the joint time-frequency domain, another intuitive way of reducing the crossterm interference is to perform 2D filtering to the Wigner-Ville Distribution. The result is described in the following equation:

$$C[i, k] = \sum_{m=-L/2}^{L/2} \sum_{n} \Phi[n, m]R[i-n, m]e^{-j2\pi km/L} \tag{4-6}$$

where Φ[*i*,*m*] denotes the kernel function. Notice that the window functions *w*[*m*] in Equation 4-3 and *h*[*m*] in Equation 4-5 are special cases of Φ[*i*,*m*] in Equation 4-6.

In 1966, Leon Cohen developed the representation $C[i,k]$ in Equation 4-6, so it is traditionally known as Cohen's class [4]. Compared with the PWVD in Equation 4-3 or 4-5, the Cohen's class method is more general and flexible. Most quadratic equations known so far, such as the STFT spectrogram, WVD, PWVD, Choi-Williams Distribution, and the cone-shaped distribution, belong to Cohen's class.

Choi-Williams Distribution

When the kernel function in Equation 4-6 is defined by

$$\Phi[i, m] = \sqrt{\frac{\alpha}{4\pi m^2}}\, e^{-\alpha i^2/(4m^2)} \tag{4-7}$$

the yield is the Choi-Williams Distribution (CWD). By adjusting the parameter α in Equation 4-7, you balance crossterm interference and time-frequency resolution; as α increases, the smoothing decreases. Figure 4-6 illustrates the CWD for the three-tone test signal where $\alpha = 1$. The CWD can effectively suppress the crossterm caused by two autoterms with different time and frequency centers, such as crossterm 2 in Figure 4-3. However, the CWD method cannot reduce crossterms that correspond to autoterms with the same time center or the same frequency center, such as crossterms 3 and 1, respectively, in Figure 4-3. Furthermore, the computation speed of the CWD is very slow.

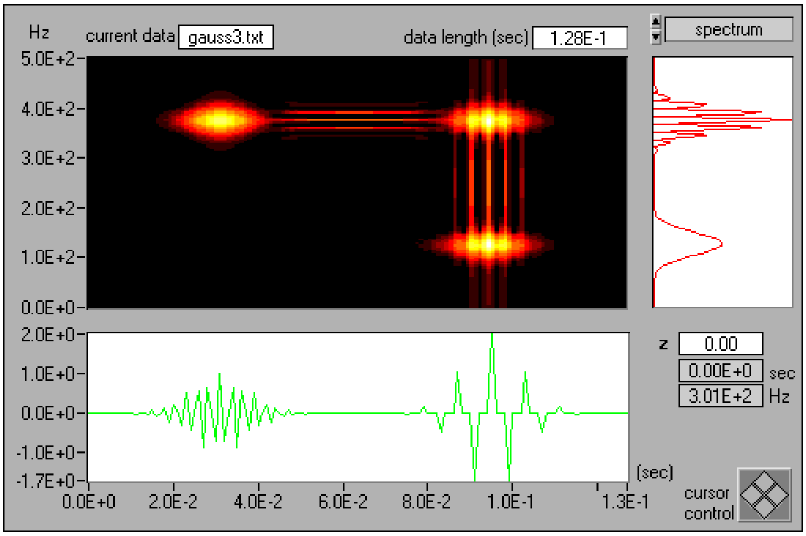


Figure 4-6. Choi-Williams Distribution ($\alpha = 1$) for the Three-Tone Test Signal

Cone-Shaped Distribution

When the kernel function in Equation 4-6 is defined by

$$\Phi[i, m] = \begin{cases} e^{-\frac{\alpha m^2}{c}} & \text{for } i < |m| \\ 0 & \text{otherwise} \end{cases} \tag{4-8}$$

the yield is the cone-shaped distribution. In the Signal Processing Toolset, the constant c is set to 500. By adjusting the parameter α in Equation 4-8, you can balance crossterm interference and time-frequency resolution; as α increases, the smoothing decreases. Figure 4-7 illustrates the cone-shaped distribution for the three-tone test signal where $\alpha = 1$. The cone-shaped distribution effectively suppresses crossterms 2 and 3 from Figure 4-3, but it cannot reduce the crossterms that correspond to autoterms with the same frequency center, such as crossterm 1 in Figure 4-3. The cone-shaped distribution is faster than the CWD method.

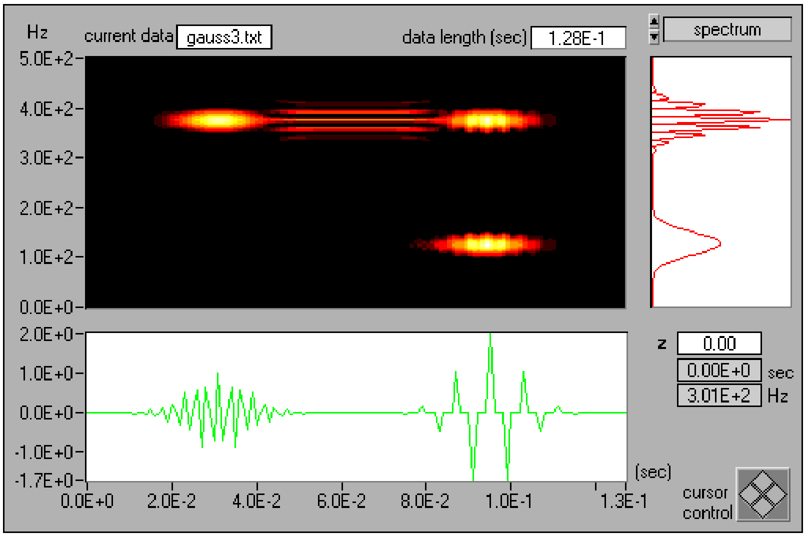


Figure 4-7. Cone-Shaped Distribution (α = 1) for the Three-Tone Test Signal

Gabor Spectrogram

In addition to applying the Pseudo Wigner-Ville Distribution window method, you can apply the Gabor expansion to a signal to identify the significance of each term to the energy of the signal at point $[i, k]$. You can then preserve those terms that have major contributions at point $[i, k]$ and remove those terms that have a negligible influence on the energy of the signal. Because this method is a Gabor expansion-based spectrogram, the resulting method is the Gabor spectrogram. The Gabor spectrogram is defined by the following equation:

$$GS_D[i, k] = \sum_{|m - m'| + |n - n'| \le D} C_{m, n} C_{m', n'} WVD_{h, h'}[i, k]$$

where $WVD_{h,h'}[i,k]$ denotes the cross-WVD of frequency-modulated Gaussian functions. The order of the Gabor spectrogram, D, controls the degree of smoothing. For $D = 0$, $GS_0[i,k]$ is non-negative and similar to the STFT spectrogram. As D moves toward infinity, the Gabor spectrogram converges to the WVD.

A lower order Gabor spectrogram has less crossterm interference but lower resolution. A higher order Gabor spectrogram has better resolution but more crossterms and a longer computation time. For best results, choose an order of three to five. The Gabor spectrogram has better resolution than the STFT spectrogram and much less crossterm interference than the cone-shaped, Choi-Williams, or Wigner-Ville Distributions.

Figure 4-8 illustrates the fourth-order Gabor spectrogram for the three-tone test signal. This Gabor spectrogram possesses high time-frequency resolution and does not have the crossterm interference that appears in the cone-shaped, Choi-Williams, and Wigner-Ville Distributions. The computational speed of the fourth-order Gabor spectrogram is slower than the STFT spectrogram and WVD but faster than the CWD and cone-shaped distribution.

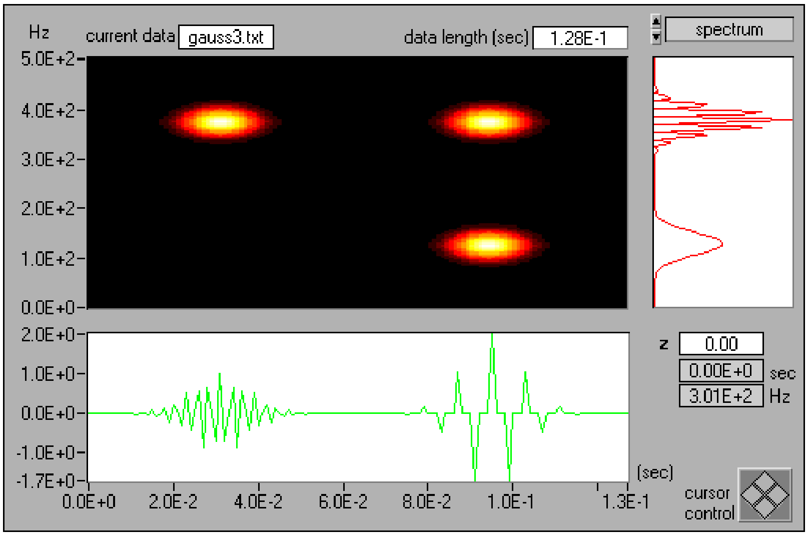


Figure 4-8. Gabor Spectrogram (Order Four) for the Three-Tone Test Signal

Adaptive Spectrogram

The adaptive spectrogram method is an adaptive representation-based spectrogram computed by the following equation:

$$AS[i, n] = 2\sum_{k=0}^{D-1}|A_k|^2\exp\left\{-\frac{[i-i_k]^2}{\alpha_k}-\alpha_k[n-\theta_k]\right\} \tag{4-9}$$

Refer to Equation 4-2 for the adaptive representation equation.

The adaptive spectrogram achieves the best joint time-frequency resolution if the analyzed signal is a sum of Gaussian functions. For example, Figure 4-9 shows that the adaptive spectrogram effectively describes the three-tone test signal. Unfortunately, the computation speed of the adaptive spectrogram increases exponentially with the analyzed data size.

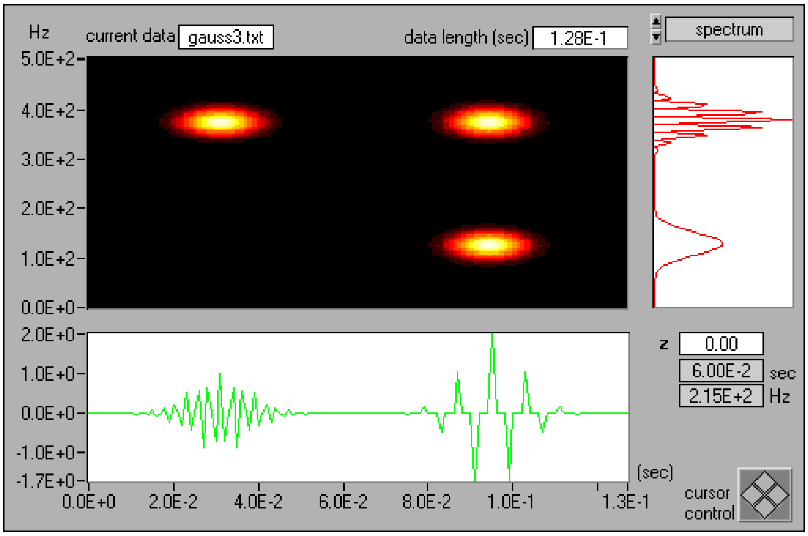


Figure 4-9. Adaptive Spectrogram for the Three-Tone Test Signal

Scientists at National Instruments [25] and Mallat and Zhang [16] independently developed the adaptive representation, also known as the matching pursuit. The adaptive methods in this toolset were implemented with the adaptive oriented orthogonal projective decomposition algorithm.

5

Joint Time-Frequency Analysis Applications

This chapter describes two comprehensive JTFA examples, Online JTFA and Off-line JTFA. With these examples, you can perform rather sophisticated, time-dependent spectrum analysis without needing to use any programming. Because each of the JTFA algorithms has advantages and disadvantages, select an algorithm that fits your application. The simplest and fastest algorithm is the STFT, which is suitable for online analysis and is used in the Online JTFA example. However, if the frequency contents of your signal change rapidly, consider using one of the algorithms in the Off-line JTFA example.

The Online JTFA and Off-line JTFA were designed for demonstration purposes only. For actual applications, use the VIs included in the Signal Processing Toolset to build your own JTFA instrument.

Accessing the Online and Off-line JTFA Examples

The Online JTFA and Off-line JTFA examples reside on the **SPT** palette, shown in Figure 5-1. Select **Start»Programs»National Instruments» Signal Processing Toolset»NI SPT Application 6.0** to open the **SPT** palette. On the **SPT** palette, click either the **Online JTFA** icon or the **Off-line JTFA** icon.

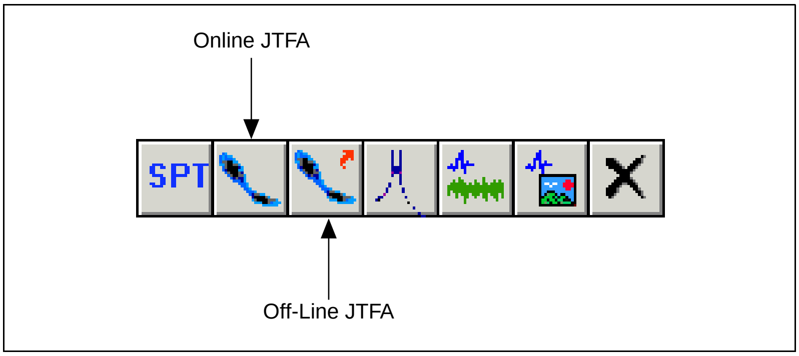
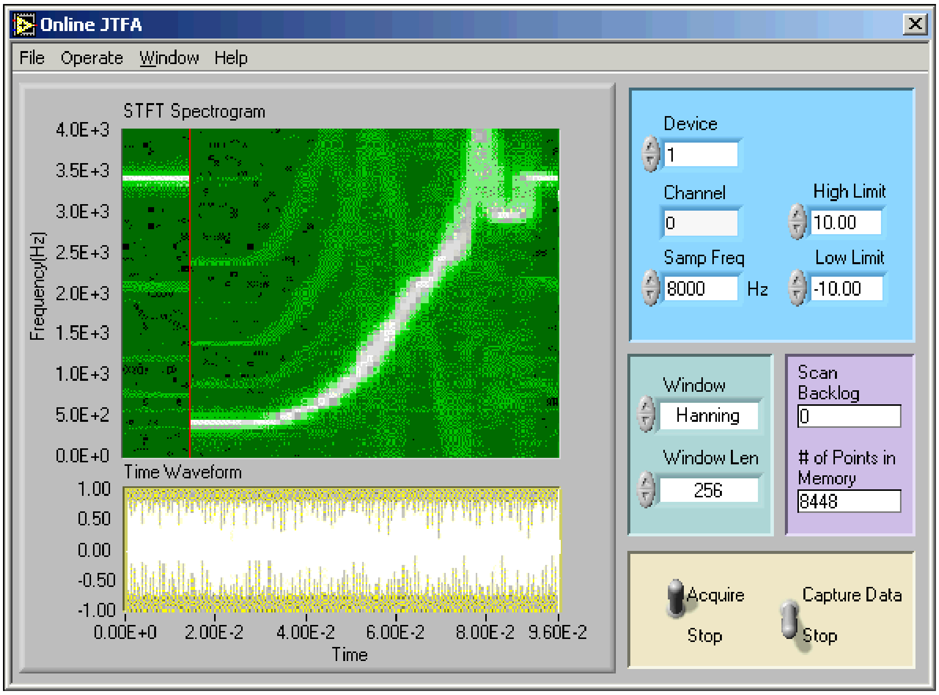


Figure 5-1. SPT Palette

Online JTFA

With the Online JTFA example, you can acquire data through NI-DAQ and perform an online STFT-spectrum analysis of that data. When you click the **Online JTFA** icon on the **SPT** palette, the Online JTFA front panel, shown in Figure 5-2, opens. The following sections describe the Online JTFA front panel controls and indicators. You also can select **Help» Show Context Help** or press <Ctrl-H> for information about controls and indicators.



Device 1
Channel 0
High Limit 10.00
Samp Freq 8000 Hz
Low Limit -10.00
Window Hanning
Window Len 256
Scan Backlog 0
# of Points in Memory 8448

Figure 5-2. Online JTFA Front Panel

Setting Data Acquisition Parameters

To perform online analysis properly, you must specify data acquisition parameters. Use the following Online JTFA front panel controls and indicators to set and monitor your data acquisition parameters:

- Use the **Device** control to specify the device number you assigned to your data acquisition (DAQ) hardware during its configuration.
- Use the **Channel** control to specify your data acquisition channel number.

- Use the **High Limit** and **Low Limit** controls to specify data acquisition input limits.
- Use the **Samp Freq** control to specify the sampling frequency. Base this frequency on your application needs. The maximum sampling frequency you can specify depends on the DAQ device and computer you use.
- Use the Scan Backlog indicator to judge whether your sampling frequency is too high. If the scan backlog value keeps increasing, reduce the sampling frequency.
- Use the **Window** control to specify your analysis window. You can set the control to one of the following window types:
  - Hanning (default)
  - Rectangular
  - Blackman
  - Hamming
- Use the **Window Length** control to specify the length of the analysis window. The maximum window length you can specify is 512.

Refer to your DAQ hardware and NI-DAQ user manuals for more information about configuring and using your DAQ hardware and NI-DAQ.

Acquiring Data

To begin data collection, toggle the **Acquire/Stop** switch to **Acquire**. Check the scan backlog indicator to be sure your system can keep up with the incoming data samples. If the value of scan backlog keeps increasing, decrease the **Samp Freq** setting. If you set the sampling rate to the lowest required by your system, but the scan backlog value continues to increase, you need to acquire your data through LabVIEW and save it as a text file. Then use the **Off-line JTFA** example to analyze the text file. Refer to the *Off-line JTFA* section of this chapter for more information about the Off-line JTFA example.

Displaying Data

The STFT Spectrogram display shows the STFT spectrogram of your acquired signal. The Time Waveform display shows the time waveform of your acquired signal.

Saving Data

To save your data, toggle the **Capture Data/Stop** switch to **Capture Data**. The # of Points in Memory indicator displays the number of data points in memory.

When you toggle the **Capture Data/Stop** switch to **Stop**, the total # of points saved indicator replaces the # of Points in Memory indicator. The Total # of Points Saved indicator displays the total number of data points saved in memory. The captured data remains in temporary memory until you stop acquiring data. To stop data acquisition, toggle the **Acquire/Stop** switch to **Stop**.

When you finish acquiring data, a dialog box prompts you to save the data to a text file. Select **Yes** to save the data in memory to a text file you designate. Select **Discard** to clear the data from memory without saving it.

Off-line JTFA

With the Off-line JTFA example, you can test all the quadratic JTFA algorithms included in the Signal Processing Toolset and find the one which best fits your application. When you click the **Off-line JTFA** icon on the **SPT** palette, the Off-line JTFA front panel, shown in Figure 5-3, opens. The following sections describe the Off-line JTFA front panel controls and indicators. You also can select **Help»Show Context Help** or press <Ctrl-H> for information about controls and indicators.

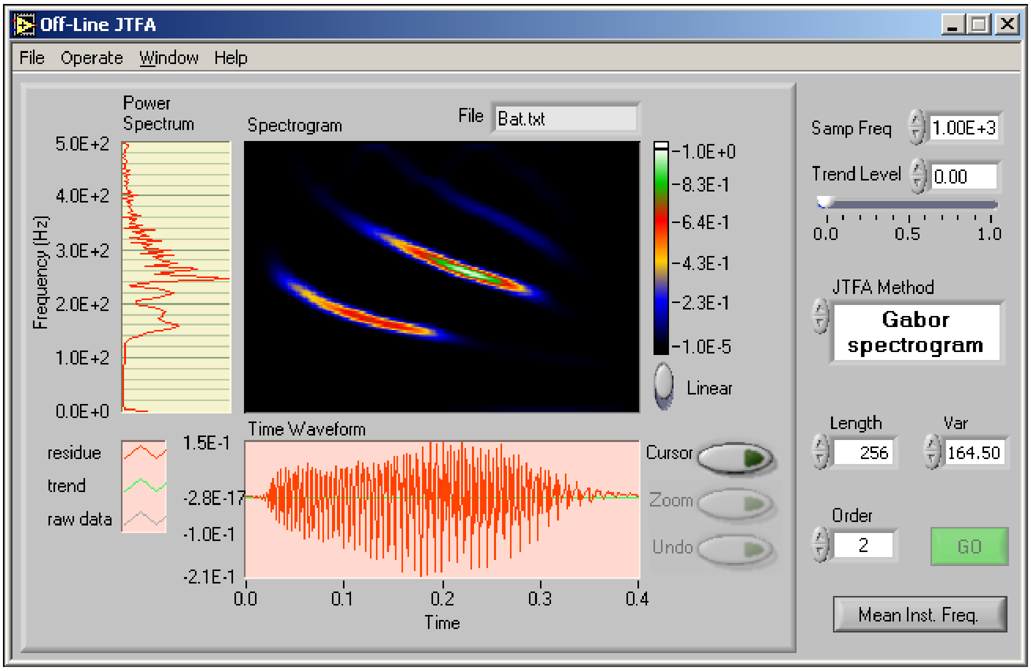


Figure 5-3. Off-line JTFA Front Panel

Loading Data

You can use the Off-line JTFA example to analyze either your own data file or one of the data files supplied with the Signal Processing Toolset. When you select **File»Open**, the application opens the data files supplied with the Signal Processing Toolset. You can select one of these data files for analysis or navigate to a data file of your own. Your data file must be a one-column or one-row spreadsheet text file. If your data file contains an x-index, use a word processor to remove the x-index before you analyze the data file.

Setting the Sampling Rate

Use the **Samp Freq** control to specify the sampling frequency. While the sampling frequency does not affect the computation results, it does affect the Power Spectrum display. The y-scale of the Power Spectrum display is the frequency. For the conventional power spectrum, as shown in Figure 5-3, the frequency ranges from DC to the Nyquist frequency, or half of the sampling frequency.

When the power spectrum is displayed as the instantaneous spectrum, the frequency range of the y-scale is determined by the zoom factor and the amount of area of the Spectrogram being analyzed. See Figure 5-4 and refer to the *Changing the Spectrogram Display* section of this chapter for more information about displaying the instantaneous spectrum.

Detrending

Detrending allows you to better analyze signals that contain DC offset or slow trend, for example, the stock index. Use the **Trend Level** control or the slider immediately below it to set the trend level. The higher you set the trend level, the more similar the trend of your analyzed signal is to the original signal.

Selecting a JTFA Method

With the Off-line JTFA example, you can choose one of the following quadratic algorithms as your data analysis method:

- STFT Spectrogram
- Gabor Spectrogram
- Adaptive Spectrogram
- Wigner-Ville Distribution
- Choi-Williams Distribution
- Cone-shaped distribution

Use the **JTFA Method** control to select the algorithm you want to use. If your signal processing experience is limited, start with the STFT Spectrogram because it is fast and simple. After choosing your analysis method, click **Go** to compute the spectrogram.

The following sections provide more information about each of the JTFA algorithms in the Signal Processing Toolset.

STFT Spectrogram

When you choose the STFT Spectrogram, adjust the **Analysis Window** and **Length** controls so that the resulting STFT spectrogram achieves the best balance between time and frequency resolution.

You can choose one of the following window types with the **Analysis Window** control:

- Hanning (default)
- Rectangular
- Blackman
- Hamming

The window length specified by the **Length** control must be less than 256. If you enter a length greater than 256, the length automatically truncates to 256. The default value for **Length** is 128. As the window length increases, the frequency resolution improves, but the time resolution becomes poorer and vice versa. Consider a long window as narrowband and a short window as wideband. You can use the window length that gives you the best balance between time and frequency resolution as a reference for the Gabor Spectrogram.

Use the **Freq Bins** control to specify the number of frequency lines in the Spectrogram display. Increasing the value of **Freq Bins** increases the amount of detail shown in the Spectrogram but lengthens computation time. The **Freq Bins** value must be a power of 2.

If you cannot achieve satisfactory resolution with the STFT Spectrogram, try the Gabor Spectrogram or one of the other JTFA methods included with the Off-line JTFA example.

Gabor Spectrogram

If the time-frequency resolution of the STFT Spectrogram is not satisfactory, try the Gabor Spectrogram next. The Gabor Spectrogram method requires more computation time than the STFT Spectrogram but achieves better time-frequency resolution.

When you choose the Gabor Spectrogram method, you need to specify values for the **Length**, **Var**, and **Order** controls. The analysis window for the Gabor Spectrogram is an optimal Gaussian window specified by length and variance. The **Length** control specifies the window length, and the **Var** control specifies the variance of the window.

The value you specify for the **Order** control determines resolution and crossterm interference. The higher the order, the better the time-frequency resolution becomes. As the order goes to infinity, the Gabor spectrogram converges to the Wigner-Ville Distribution. As the order increases, crossterms become more obvious. Also, computation time is proportional to the order selected. Set **Order** to a value between three and five to achieve the best compromise between resolution and crossterm interference.

Decrease the value of the **Var** control to reduce Gaussian window variance. Reducing Gaussian window variance eliminates the crossterm caused by a pair of autoterms separated in time but deteriorates the time-frequency resolution.

Adaptive Spectrogram

If you can consider a signal as the sum of sinusoidal functions with different Gaussian envelopes, or Gaussian Pulses, use the Adaptive Spectrogram to achieve the best time-frequency resolution.

Use the **# of Terms** control to specify the number of Gaussian Pulses used to approximate the analyzed signal. The higher you set the value of **# of Terms**, the more accurate the approximation becomes, and the smaller the residual becomes. However, increasing the value for **# of Terms** lengthens computation time. It is a good idea to start with a small number of terms and increase the value of **# of Terms** until the Residual(%) indicator returns a satisfactory reading. The residual is computed by the following equation:

$$residual = \frac{\sum_n |s[n] - a[n]|^2}{\sum_n |s[n]|^2}$$

where *a*[*n*] denotes the approximation. If the approximation is equal to the original signal *s*[*n*], the residual goes to zero.

Refer to the *STFT Spectrogram* section of this chapter for information about the **Freq Bins** control.

Wigner-Ville Distribution

The Wigner-Ville Distribution provides high time-frequency resolution at a rapid computation rate. However, the Wigner-Ville Distribution can suffer from severe crossterm interference if the analyzed signal consists of multiple components.

In order to lessen crossterm interference, place a checkmark in the **Analytical Signal?** checkbox. When you check the **Analytical Signal?** checkbox, the Off-line JTFA example converts the data samples into the corresponding analytical signal. Converting the samples into the analytical signal reduces the cross interference due to components from negative frequencies. However, the conversion can introduce distortion in the low frequency portion of the time-dependent spectrum of the signal, especially in the vicinity of DC.

Refer to the *STFT Spectrogram* section of this chapter for information about the **Freq Bins** control.

Choi-Williams Distribution

The Choi-Williams Distribution offers you a reduction in crossterm interference while preserving as many useful Wigner-Ville Distribution properties as possible. Like the Wigner-Ville Distribution, checking the **Analytical Signal?** checkbox reduces the cross interference due to components from negative frequencies but introduces distortion in the low frequency portion of the time-dependent spectrum.

You also can lessen crossterm interference by specifying a value for the **Alpha** control. In general, the smaller the **Alpha** value, the less crossterm interference you get, but the time-frequency resolution becomes poorer. The **Alpha** default value is 1.0E-6.

Refer to the *STFT Spectrogram* section of this chapter for information about the **Freq Bins** control.

Cone-Shaped Distribution

The cone-shaped distribution is another time-dependent spectrum designed to reduce crossterm interference. Like the Wigner-Ville Distribution and the Choi-Williams Distribution, place a checkmark in the **Analytical Signal?** checkbox to lessen crossterm interference.

Refer to the *Choi-Williams Distribution* section of this chapter for information about the **Alpha** control.

Refer to the *STFT Spectrogram* section of this chapter for information about the **Freq Bins** control.

Displaying Data

The Off-line JTFA example uses the following data displays:

- The Power Spectrum/Instantaneous display shows either the classical power spectrum or the instantaneous spectrum, depending on the settings you have chosen. Refer to the *Changing the Spectrogram Display* section of this chapter for more information about the Instantaneous display.
- The Spectrogram display shows the time-dependent spectrum for the chosen analysis method.
- The Time Waveform display of the Off-line JTFA front panel shows the time waveform of the analyzed signal.

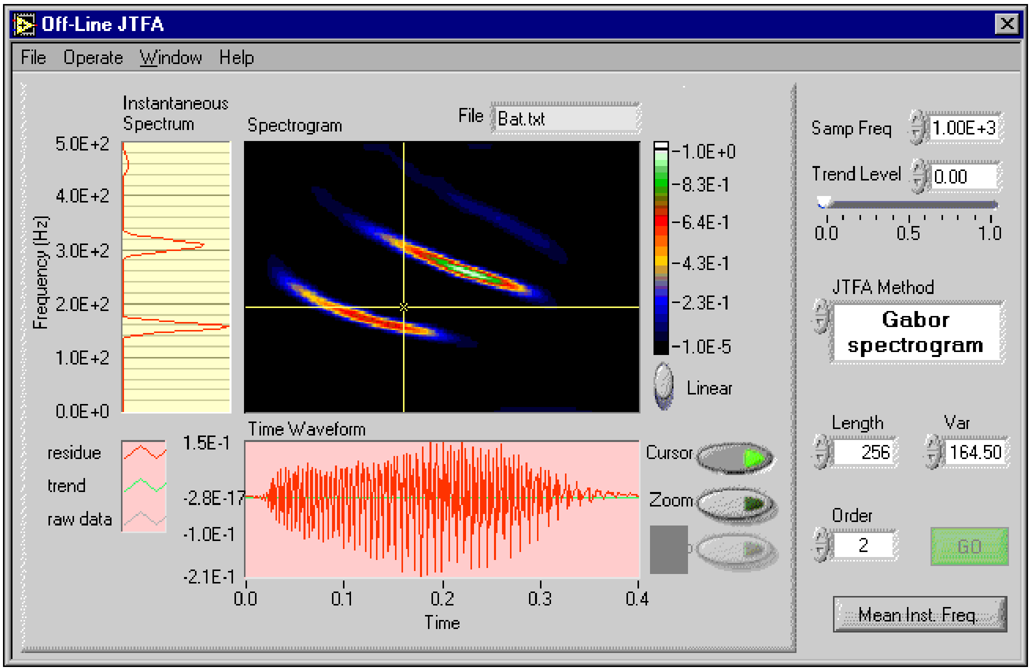
Changing the Spectrogram Display

You can change the Spectrogram data display by moving the **Linear/dB** switch and/or clicking the **Cursor** button. With the **Linear/dB** switch, you can choose to display your data in the Spectrogram as either a linear or decibel (dB) display.

Clicking the **Cursor** button turns the cursor on and off. When the cursor is on, as shown in Figure 5-4, the Power Spectrum display changes to the Instantaneous Spectrum. The Instantaneous Spectrum shows the instantaneous spectrum at the time indicated by the x-value of the cursor.

Frequency Zooming

With the cursor turned on, you can click the **Zoom** button to zoom in on the Spectrogram display in the frequency scale. The frequency range displayed equals $fs/(2 \times \text{zoom factor})$, where *fs* is the sampling frequency. The central frequency is determined by the y-value of the cursor. The zoom factor doubles every time you click the **Zoom** button. The maximum zoom factor is limited to 16, so the smallest frequency range is $fs/32$.



Samp Freq 1.00E+3
Trend Level 0.00
JTFA Method Gabor spectrogram
Length 256
Var 164.50
Order 2

Figure 5-4. Instantaneous Spectrum

Mean Instantaneous Frequency

After computing the spectrogram of your test data, click the **Mean Inst. Freq** button to calculate the mean instantaneous frequency. The Spectrogram display shows the profile of the mean instantaneous frequency, as shown in Figure 5-5.

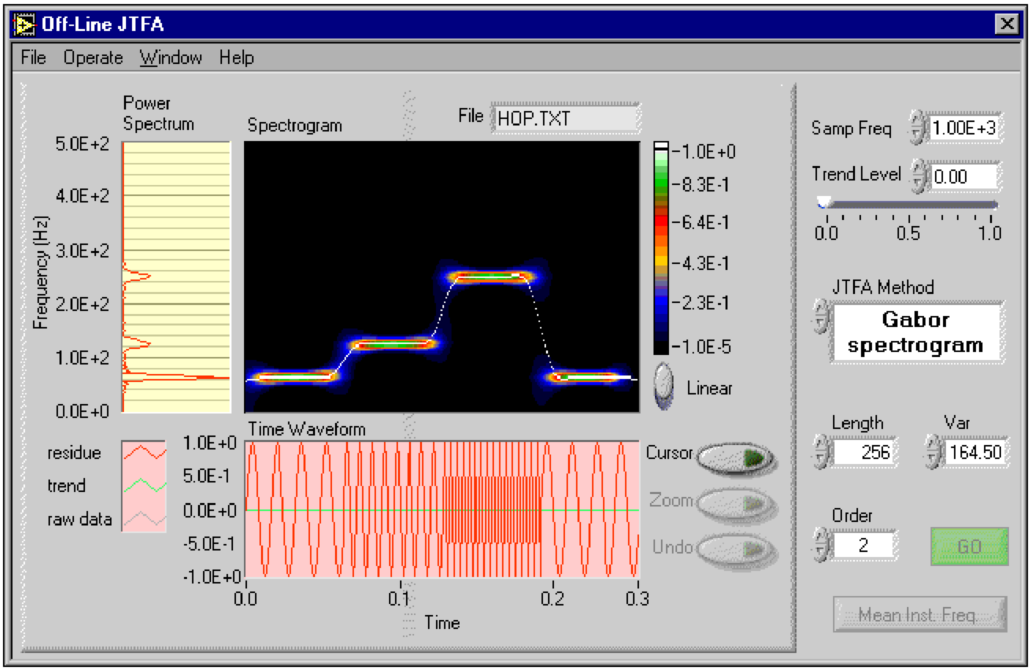


Figure 5-5. Mean Instantaneous Frequency

Saving Results

When you select **File»Save**, you can choose any of the following options from the pull-down menu and save your results as a text file:

- **Detrend** saves the detrended time waveform
- **Spectrum** saves the power spectrum
- **Spectrogram** saves the real spectrogram without truncating or normalizing it

All spectrograms display only the non-negative points. The Off-line JTFA example automatically truncates negative points to zero. If you set the **Linear/dB** switch to **dB**, the displayed spectrogram is further normalized. However, real spectrograms, except for the STFT Spectrogram and the Adaptive Spectrogram, sometimes contain negative values.

Part III

Super-Resolution Spectral Analysis

This section of the manual discusses model-based frequency analysis and the model-based frequency analysis algorithms used by the Signal Processing Toolset. Model-based frequency analysis enables you to obtain super-resolution spectra of the signals you are studying. This section of the manual also describes a super-resolution spectral analysis example included with the Signal Processing Toolset.

- Chapter 6, *Introduction to Model-Based Frequency Analysis*, introduces the basic concepts of model-based frequency analysis.
- Chapter 7, *Model-Based Frequency Analysis Algorithms*, outlines the theoretical background of model-based frequency analysis and describes the relationship among the model coefficients, power spectra, and parameters of damped sinusoids.
- Chapter 8, *Applying Super-Resolution Spectral Analysis and Parameter Estimation*, describes a comprehensive super-resolution spectral analysis example application included with the Signal Processing Toolset. This example is designed to help you learn about model-based analysis.

6

Introduction to Model-Based Frequency Analysis

This chapter introduces the basic concepts of model-based frequency analysis.

There are two methods usually employed to perform spectral analysis, non-model-based methods, such as the fast Fourier transform (FFT)-based methods, and model-based methods. Model-based methods need fewer data samples and are more accurate than the FFT-based methods if the model fits the analyzed data samples. When you employ model-based methods, you not only obtain super-resolution power spectra with a small data set, but you also can estimate the parameters of damped sinusoids. The model-based methods are an important alternative to classical FFT-based methods in many frequency analysis applications.

The Need for Model-Based Frequency Analysis

The term spectrum has been generalized for arbitrary signals and characterizes the frequency behavior of a signal. Examples of questions that spectral analysis can answer include whether most of the power of the signal resides at low or high frequencies, and whether there are resonances.

Spectral analysis is used widely in such diverse fields as biomedicine, economics, geophysics, noise and vibration, radar, sonar, speech, and other areas in which signals of unknown or questionable origin are of interest. By performing spectral analysis, you can often discover some important features of signals that are not obvious in the time waveform of the signal.

Over the last 30 years, a primary tool for spectral analysis has been the FFT. However, the frequency resolution of the FFT-based methods is bounded by the number of data samples. The relationship of the number of samples and frequency resolution can be quantified by the following equation:

$$\Delta f = \frac{sampling\ frequency}{number\ of\ samples} \quad (6\text{-}1)$$

where Δf denotes the frequency resolution. The frequency resolution characterizes the distinguishable minimum difference between two sinusoids. For a given sampling frequency, the more samples you have, the higher the frequency resolution.

Figure 6-1 illustrates a sum of two sinusoids. The frequencies of these two sinusoids are 0.11 Hz and 0.13 Hz, respectively. To separate the two sinusoids, the frequency resolution Δf has to be less than or equal to 0.02 Hz. Assume that the sampling frequency is 1 Hz. Based on Equation 6-1, you need at least 50 samples in order to separate the two sinusoids. Figure 6-2 uses the rectangular window and the Hamming window to depict the FFT-based power spectra. As long as you have enough samples, you can use either window to separate the two sinusoids.

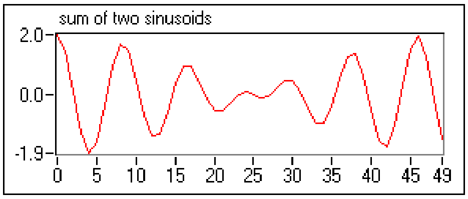


Figure 6-1. 50 Samples for a Sum of Two Sinusoids

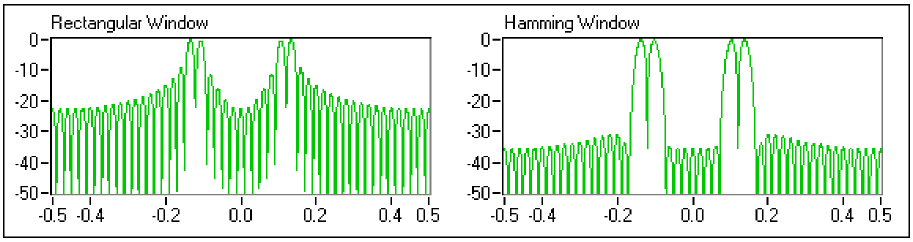


Figure 6-2. FFT-Based Power Spectra Based on 50 Samples

However, in many applications, the number of data samples is limited. The limited number of samples might be the result of a genuine lack of data, as in the seismic patterns of an erupting volcano. In other instances, it might be necessary to impose restrictions on the sample size to ensure that the spectral characteristics of a signal do not change over the duration of the data record, as in speech processing. When the data record is small, scientists often think that the frequency resolution of FFT-based power spectra is not adequate. For example, reduce the number of data samples for the two sinusoids in the example above to 15. Figure 6-3 shows the 15-sample data record. The resulting FFT-based power spectra are plotted in Figure 6-4. In this case, neither window yields a frequency resolution high enough to resolve the two close sinusoids.

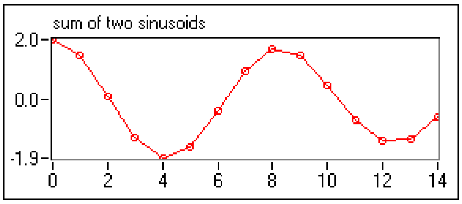


Figure 6-3. Two Sinusoids with 15 Samples

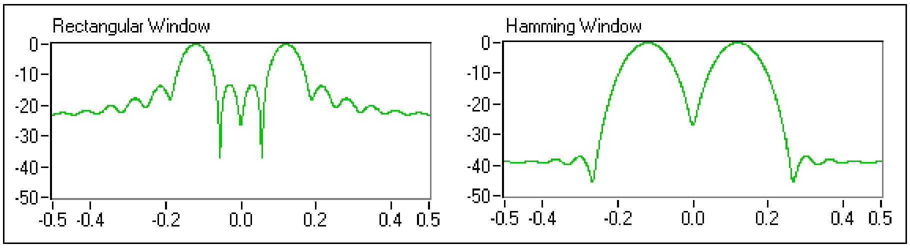


Figure 6-4. FFT-Based Power Spectra Based on 15 Samples

An alternative is the model-based method. By employing model-based analysis techniques, you can obtain super-resolution spectra. Once you assume a suitable signal model and determine its coefficients, you can predict the missing data based on the given finite data set. When you use the model-based method, it is as if you have an infinite number of data samples. Thus, you can substantially improve the frequency resolution.

Figure 6-5 depicts two model-based super-resolution power spectra for the sinusoids in Figure 6-3.

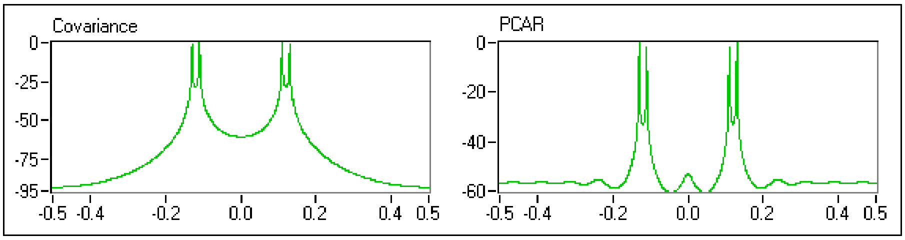


Figure 6-5. Super-Resolution Power Spectra Based on 15 Samples

Although the FFT-based methods need at least 50 samples, the model-based super-resolution power spectra detect two sinusoids satisfactorily with only 15 samples.

Another important application of model-based methods is the parameter estimation of damped sinusoids. The estimated parameters include amplitude, phase, damping factor, and frequency. You can compute the signal frequency and phase by applying the FFT, if the number of data samples is large enough. However, there is no indication of the signal damping behavior. In nature, the signal amplitude often changes with time, gradually decreasing or increasing until blowing out. The damping behavior is an important aspect of the signal that indicates whether the corresponding system is stable.

Figure 6-6 depicts a sum of two damped sinusoids in which the sampling frequency is 1 Hz.

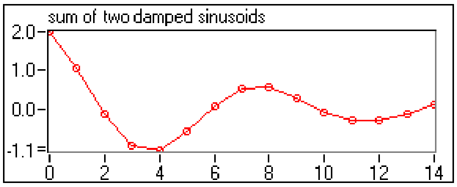


Figure 6-6. Damped Sinusoids

Table 6-1 lists the corresponding parameters, and Figure 6-7 plots the resulting FFT-based power spectra. Applying FFT-based methods provides no way to tell the complete information about the two damped sinusoids.

Table 6-1. Damped Sinusoids

Signal	Amplitude	Phase (rad)	Damping Factor	Frequency (Hz)
signal 1	1.0	0.20	–0.10	0.13
signal 2	1.0	0.10	–0.20	0.11

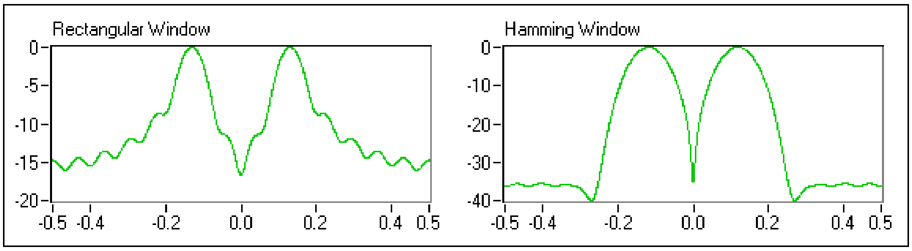
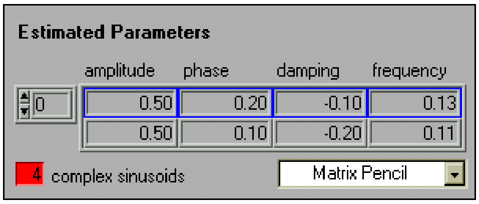


Figure 6-7. FFT-Based Power Spectra for Damped Sinusoids

Figure 6-8 illustrates the estimated result obtained by a model-based algorithm known as the matrix pencil. Notice that a real signal produces two imaginary, symmetrical complex sinusoids. The complex sinusoids indicator in the lower left corner of Figure 6-8 shows that there are a total of four complex sinusoids for the samples shown in Figure 6-6. Figure 6-8 also lists the parameters of the components with positive frequencies. Since the amplitude of the complex sinusoids is half that of the corresponding real sinusoids, the values in Figure 6-8 match those in Table 6-1 exactly.


Estimated Parameters

	amplitude	phase	damping	frequency
0	0.50	0.20	-0.10	0.13
	0.50	0.10	-0.20	0.11

4 complex sinusoids

Matrix Pencil

Figure 6-8. Parameter Estimation by Matrix Pencil Method

The precision of the FFT-based methods is accurate only at frequencies that are integer multiples of the frequency increment[1]:

$$\frac{sampling\ frequency}{number\ of\ FFT\ points} \tag{6-2}$$

There is no such limitation for the model-based methods. Therefore, the model-based methods are much more accurate than the FFT-based techniques.

Aside from the super-resolution power spectra, model-based analyses are also fundamental in many other signal processing applications. Although the focus of this part of the manual is on frequency analysis, the model-based applications in the Signal Processing Toolset can be easily tailored for many other applications, including the following applications:

- linear prediction, such as linear predict code
- signal synthesis
- data compression, such as speech compression
- system identification

[1] The number of samples in Equation 6-1 and the number of FFT points in Equation 6-2 might not be equal. For a given number of data samples, you always are able to increase the number of FFT points simply by zero-padding. Increasing the number of FFT points reduces the frequency increment but does not improve the ability to resolve two close sinusoids.

Applying Model-Based Methods Properly

Model-based methods can obtain super-resolution power spectra with a limited number of data samples or estimate the parameters of damped sinusoids. However, for best results, you need to consider the following three factors:

- The signal has to be a certain type of time series and the following equation should be able to generate the signal. The following equation is known as the recursive difference equation:

$$x[n] = -\sum_{k=1}^{p} a_k x[n-k] + w[n] \tag{6-3}$$

  where $w[n]$ denotes the error.
- You need to select the order of the model correctly, or you might obtain an incorrect spectrum or parameter estimate.
- The computation time of the super-resolution power spectra is much longer than that of FFT, JTFA, and wavelet transform. When the number of data samples is more than a few hundred, it is no longer appropriate to use model-based methods because of the computation time involved and the numerical inaccuracies that might result.

Figure 6-9 shows the plot of super-resolution power spectra for the sum of the two sinusoids in Figure 6-3. The spectra were computed by the same model-based methods as that in Figure 6-5. However, instead of choosing order four, the order is artificially increased to 10. Consequently, in addition to the four real components, several spurious peaks appear that do not actually exist. Blindly applying model-based techniques does not lead to a good estimation. A good estimation relies on selecting the proper signal model as well as the model order. The rest of this chapter and Chapters 7 and 8 deal with this central topic.

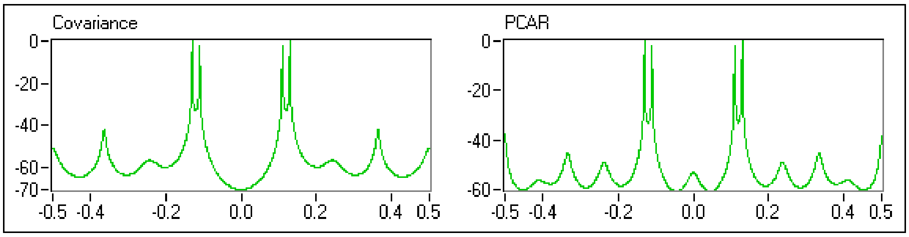


Figure 6-9. Super-Resolution Power Spectra with Order 10 for Sum Of Two Sinusoids

One reason for using a few samples to perform frequency analysis is to ensure that the spectral characteristics of a signal do not change over the duration of the data record. This is also a primary motivation of developing the JTFA and wavelet transform.

At this point, a natural question might be, “Which technique is the best?” The answer is that each method has advantages and disadvantages. None is superior to all others in every application.

Table 6-2 compares model-based methods with FFT, JTFA, and wavelets. There is no assumption about the analyzed signal for FFT, JTFA, and wavelet analysis, whereas the model-based methods work only for certain types of signals. Moreover, the performance of model-based frequency analysis is quite sensitive to noise, though there are some variations for different algorithms. For example, of the methods shown in Figure 6-5, the principle component auto-regressive (PCAR) method has better noise immunization than the covariance method.

Table 6-2. FFT, JTFA, Wavelets, and Model-Based Methods

Method	Signal Model	Stationary	Data Length	Frequency Resolution	Noise Sensitivity	Speed
FFT	arbitrary	yes	long	low	moderate	fast
JTFA	arbitrary	no	long	low	low	moderate
Wavelets	arbitrary	no	long	constant Q	low	fast
Model-based	not arbitrary	no	short	high	high	slow

The Toolset and Model-Based Methods

The Signal Processing Toolset contains VIs and functions for several effective model-based analysis methods. Using these VIs and functions, you can build your own applications to perform super-resolution spectral analysis and parameter estimation. In addition, there is an example to assist users who are not familiar with model-based frequency analysis.

Refer to the following documents for more information about the Signal Processing Toolset and model-based methods:

- Chapter 7, *Model-Based Frequency Analysis Algorithms*, for information about the algorithms used in the Signal Processing Toolset
- The *Signal Processing Toolset Help*, available by selecting **Help»Signal Processing Toolset**, for information about individual VIs
- Chapter 8, *Applying Super-Resolution Spectral Analysis and Parameter Estimation*, for information about the example application
- Chapter 15, *Super-Resolution Spectral Analysis for LabWindows/CVI*, for information about the model-based functions for LabWindows/CVI.

7

Model-Based Frequency Analysis Algorithms

This chapter outlines the theoretical background of model-based frequency analysis and describes the relationship among the model coefficients, power spectra, and parameters of damped sinusoids. In most cases, the conclusions are presented without justification. Refer to the works of Kay [14] and Marple [18] for more detailed background information about model-based frequency analysis.

Models, Power Spectra, and Damped Sinusoids

This section introduces the signal models used for model-based frequency analysis and explains the relationship among the model coefficients, power spectra, and parameters of damped sinusoids.

ARMA, MA, and AR Models

As discussed in the *Applying Model-Based Methods Properly* section of Chapter 6, *Introduction to Model-Based Frequency Analysis*, of this manual, model-based frequency analysis is suitable only for certain types of data. In general, the data has to be generated by exciting a linear shift-invariant causal pole-zero filter, or rational transfer function, with white noise. In other words, the data sample $x[n]$ has to fit the following model:

$$x[n] = -\sum_{k=1}^{p} a_k x[n-k] + \sum_{m=0}^{q} b_m w[n-m] \qquad \text{for } 0 \le n < N \qquad (7\text{-}1)$$

where $b_0 = 1$ and $w[n]$ is the white noise with zero mean and variance σ^2. Equation 7-1 is traditionally called the auto-regressive and moving average (ARMA) model.

There are two special cases of Equation 7-1, the first is $a_k = 0$ for all k. Consequently, the equation reduces to

$$x[n] = \sum_{m=0}^{q} b_m w[n-m] \qquad \text{for } 0 \le n < N \tag{7-2}$$

which is called a moving average (MA) model.

The second case is $b_m = 0$ for $m > 0$. In this case, the ARMA model in Equation 7-1 becomes

$$x[n] = -\sum_{k=1}^{p} a_k x[n-k] + w[n] \qquad \text{for } 0 \le n < N \tag{7-3}$$

which is called an auto-regressive (AR) model. According to Equation 7-3, you can use currently known data samples to predict the future data with error $w[n]$. Let the predicted data be $\hat{x}[n]$, then

$$\hat{x}[n] = -\sum_{k=1}^{p} a_k x[n-k] \qquad \text{for } p \le n < N \tag{7-4}$$

or

$$\begin{bmatrix} x[p-1] & x[p-2] & \dots & x[0] \\ x[p] & x[p-1] & \dots & x[1] \\ \vdots & \vdots & \vdots & \vdots \\ x[N-2] & x[N-1] & \dots & x[N-p+1] \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = -\begin{bmatrix} \hat{x}[p] \\ \hat{x}[p+1] \\ \vdots \\ \hat{x}[N-1] \end{bmatrix} \tag{7-5}$$

which is named a forward prediction. Alternatively, there is a backward prediction, which is explained later in this chapter.

The AR, MA, and ARMA models cover a wide range of signals in nature. In most applications, you can confidently apply model-based methods for frequency analysis. Usually, you can choose the appropriate model based on physical modeling. In practice, you might not know which of the given models is best for the problem at hand. An important result from the Wold decomposition [38] and Kolmogorov theorems [15] is that any AR or ARMA process can be represented by an MA process of possibly infinite order. Likewise, any MA or ARMA process can be represented by an AR process of possibly infinite order [15] [38]. If you choose the wrong model among the three, you can still obtain a reasonable approximation by using a high enough model order.

The next task is determining the model order. As shown in the *Applying Model-Based Methods Properly* section of Chapter 6, *Introduction to Model-Based Frequency Analysis*, of this manual, the wrong model order can lead to an incorrect result. To select the right order, you need some knowledge about the signal. Each complex sinusoid component counts as one order. Each real sinusoid component generates two complex sinusoids that correspond to two orders. If you are not sure what order you should use, you can use the minimum description length algorithm, introduced in the *Minimum Description Length* section of this chapter, to estimate the order.

Because AR-based algorithms are better understood and more popular than their counterparts, the next two sections limit discussion to AR-based methods.

Model Coefficients and Power Spectra

Taking the *z*-transform of Equation 7-1 yields a rational transfer function:

$$X(z) = \frac{B(z)}{A(z)} = \frac{1 + \sum_{m=1}^{q} b_m z^{-m}}{1 + \sum_{k=1}^{p} a_k z^{-k}} = H(z) \tag{7-6}$$

It can be proved that the power spectrum *P*(*f*) is *P*(*z*) evaluated along the unit circle, where

$$P(z) = H(z)H^*(1/z^*)\sigma^2 = \frac{B(z)B^*(1/z^*)}{A(z)A^*(1/z^*)}\sigma^2 \tag{7-7}$$

where * denotes the complex conjugate. For the AR model, the power spectrum is

$$P(f) = \frac{\sigma^2}{\left|1 + \sum_{k=1}^{p} a_k e^{-j2\pi fk}\right|^2} \tag{7-8}$$

which implies that once you compute the coefficients a_k of the AR model, you can obtain the power spectrum by taking the reciprocal of the fast Fourier transform (FFT) of a_k.

If *A*(*z*) is the *z*-transform of the coefficients a_k as shown in Equation 7-6, it can be shown that

$$A^*(1/z^*) = 1 + \sum_{k=1}^{p} (a_k)^* z^k$$

whereas *A*(*z*)*X*(*z*) forms the forward prediction, *A**(1/*z**)*X*(*z*) constitutes a backward prediction:

$$\hat{x}[n] = -\sum_{k=1}^{p} (a_k)^* x[n+k] \qquad \text{for } 0 \le n < N \tag{7-9}$$

which uses future data to predict the data that was sampled at *p* steps before. The formula of the backward prediction in Equation 7-9 can be written as

$$\begin{bmatrix} x[1] & x[2] & \dots & x[p] \\ x[2] & x[3] & \dots & x[p+1] \\ \vdots & \vdots & \vdots & \vdots \\ x[N-p] & x[N-p+1] & \dots & x[N-1] \end{bmatrix} \begin{bmatrix} (a_1)^* \\ (a_2)^* \\ \vdots \\ (a_p)^* \end{bmatrix} = - \begin{bmatrix} \hat{x}[0] \\ \hat{x}[1] \\ \vdots \\ \hat{x}[N-p-1] \end{bmatrix} \qquad (7\text{-}10)$$

The forward and backward predictions in Equation 7-8 are interchangeable. For example, let

$$A(z) = 1 + \sum_{k=0}^{p} a_k z^{-k} \qquad \text{or} \qquad A(z) = 1 + \sum_{k=0}^{p} (a_k)^* z^k$$

The resulting $P(z)$ in Equation 7-7 is the same.

AR Model and Damped Sinusoids

Damped sinusoids are common in applications such as noise and vibration. Many natural phenomena can be formulated as a linear combination of damped sinusoids:

$$x[n] = \sum_{k=1}^{p} C_k \exp\{(\alpha_k + j2\pi f_k)n\} = \sum_{k=1}^{p} C_k (z_k)^n \qquad \text{for } 0 \le n < N$$

(7-11)

where the parameter α_k indicates the damping factor and C_k denotes the complex amplitudes. Equation 7-11 also can be written in matrix form as

$$\begin{bmatrix} (z_1)^0 & (z_2)^0 & \dots & (z_p)^0 \\ (z_1)^1 & (z_2)^1 & \dots & (z_p)^1 \\ \vdots & \vdots & \vdots & \vdots \\ (z_1)^{N-1} & (z_2)^{N-1} & \dots & (z_p)^{N-1} \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_p \end{bmatrix} = \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix} \qquad (7\text{-}12)$$

where the matrix of the time-indexed *z* elements has a Vandermonde structure.

At first glance, Equation 7-12 does not seem to belong to any of the models described by Equations 7-1, 7-2, and 7-3. However, Equation 7-12 is closely related to the AR model in Equation 7-3[1]. In 1795, Baron de Prony discovered that z_k in Equation 7-12 actually are roots of the polynomial

$$A(z) = 1 + \sum_{k=1}^{p} a_k z^{-k} = \prod_{k=1}^{p} (1 - z_k z^{-1}) \tag{7-13}$$

where a_k are the coefficients of the regular AR model in Equation 7-3. Consequently, the procedure for finding the damped sinusoids parameters is to first compute the AR coefficients a_k. Then, solve the polynomial in Equation 7-13 to determine z_k. Finally, the solution of the linear system in Equation 7-12 gives the complex amplitudes C_k.

Algorithms for Super-Resolution Spectral Analysis and Parameter Estimation

This section briefly introduces the algorithms included on the **Super-Resolution Spectral Analysis** palette. The covariance and PCAR methods are used to compute super-resolution power spectra. The matrix pencil and Prony's methods are mainly used for parameter estimation. The minimum description length algorithm is used to estimate the number of complex sinusoids.

Covariance Method

Assume that the future data is estimated by the forward prediction in Equations 7-4 and 7-5. The covariance method computes the coefficients a_k such that the error between $x[n]$ and $\hat{x}[n]$ is minimized:

$$min_{a_k} \sum_{n=p}^{N-1} |x[n] - \hat{x}[n]|^2$$

[1] Prony developed this method 13 years before the Fourier transform was introduced.

In Equation 7-5, the optimal coefficients a_k are the solution of the linear system of

$$\begin{bmatrix} x[p-1] & x[p-2] & \dots & x[0] \\ x[p] & x[p-1] & \dots & x[1] \\ \vdots & \vdots & \vdots & \vdots \\ x[N-2] & x[N-1] & \dots & x[N-p+1] \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = -\begin{bmatrix} x[p] \\ x[p+1] \\ \vdots \\ x[N-1] \end{bmatrix}$$

The covariance method is not difficult, but it is sensitive to noise.

Principle Component Auto-Regressive (PCAR) Method

The covariance method only minimizes the error between $x[n]$ and $\hat{x}[n]$ for $p \leq n < N$ (that is, $N - p$ points), even though there are N samples of $x[n]$. The PCAR method formulates the linear system as

$$\begin{bmatrix} X_f \\ X_b \end{bmatrix} \vec{a} = -\begin{bmatrix} \vec{x}_f \\ \vec{x}_b \end{bmatrix} \qquad (7\text{-}14)$$

where $\vec{a}$ denotes the data vector

$$\vec{a} = \begin{bmatrix} a_1 & a_2 & \dots & a_p \end{bmatrix}^T$$

$\vec{x}_f$ *and* $\vec{x}_b$ denote the right side vectors of the forward prediction in Equation 7-5 and the backward prediction in Equation 7-10. You also can write this relationship as

$$\begin{bmatrix} \vec{x}_f \\ \vec{x}_b \end{bmatrix} = \begin{bmatrix} x[p] & x[p+1] & \dots & x[N-1] & x[0] & x[1] & \dots & x[N-p-1] \end{bmatrix}^T$$

Similarly, the matrices ***X*f** and ***X*b** are the left side matrices of the forward prediction in Equation 7-5 and the backward prediction in Equation 7-10. You can write this relationship as

$$\begin{bmatrix} X_f \\ X_b \end{bmatrix} = \begin{bmatrix} x[p-1] & x[p-2] & \dots & x[0] \\ x[p] & x[p-1] & \dots & x[1] \\ \vdots & \vdots & \vdots & \vdots \\ x[N-2] & x[N-1] & \dots & x[N-p+1] \\ x[1] & x[2] & \dots & x[p] \\ x[2] & x[3] & \dots & x[p+1] \\ \vdots & \vdots & \vdots & \vdots \\ x[N-p] & x[N-p+1] & \dots & x[N-1] \end{bmatrix}$$

Consequently, the linear system in Equation 7-14 uses forward and backward prediction information. In this manner, you obtain extra data points and average more errors.

Moreover, you solve for the coefficients by

$$\vec{a} = \sum_{i=1}^{L} \frac{1}{\lambda_i} \vec{v}_i \vec{v}_i^{\,T} X^T \vec{x} \qquad (7\text{-}15)$$

where

$$X \equiv \begin{bmatrix} X_f \\ X_b \end{bmatrix} \qquad \text{and} \qquad \vec{x} \equiv \begin{bmatrix} \vec{x}_f \\ \vec{x}_b \end{bmatrix}$$

λ_i denote the *L* largest eigenvalues of the matrix **X**. $\vec{v}_i$ are *L* corresponding eigenvectors. The parameter *L* represents the number of complex sinusoids. Because you use only *L* principle components in Equation 7-15, the results obtained by PCAR are much less sensitive to noise than results obtained by the covariance method.

Prony's Method

Prony's method estimates the parameters of damped sinusoids. First, apply the covariance method to compute the AR coefficients a_k. Then, find the complex roots z_k of the polynomial in Equation 7-13. The phase of z_k indicates the frequency, and the amplitude is the damping factor. Finally, insert z_k into Equation 7-12 to solve C_k. The amplitude and phase of the sinusoid component z_k are equal to the amplitude and phase of C_k, respectively.

Matrix Pencil Method

The matrix pencil method is a modified Prony's method. It is faster and less sensitive to noise than Prony's method. However, the derivation is more involved. Refer to the work of Hua and Sarkar [10] for more information about the matrix pencil method.

Minimum Description Length

The minimum description length algorithm determines the number of sinusoids n by the following equation:

$$min_n\{N\ln\sigma^2 + 3n\ln N\}$$

where σ^2 is an estimation of the noise variance and N is the number of data samples. The optimal value n can be used as the AR order p for the covariance method or the number of complex sinusoids L for the PCAR and matrix pencil methods.

8 Applying Super-Resolution Spectral Analysis and Parameter Estimation

This chapter describes a comprehensive example based on the Super-Resolution Spectral Analysis (SRSA) VIs in the Signal Processing Toolset. The example is designed to help you learn about model-based analysis. With this example, you can try different algorithms for the data samples without needing to use any programming.

Accessing the SRSA Example

The SRSA example resides on the **SPT** palette, shown in Figure 8-1. Select **Start»Programs»National Instruments»Signal Processing Toolset» NI SPT Application 6.0** to open the **SPT** palette. On the **SPT** palette, click the **Super-Resolution Spectra Analysis** icon. When you click the icon, the Super-Resolution Spectral Analysis example front panel, shown in Figure 8-2, opens.

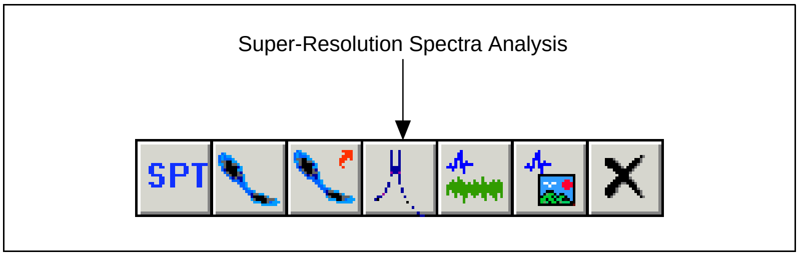


Figure 8-1. SPT Palette

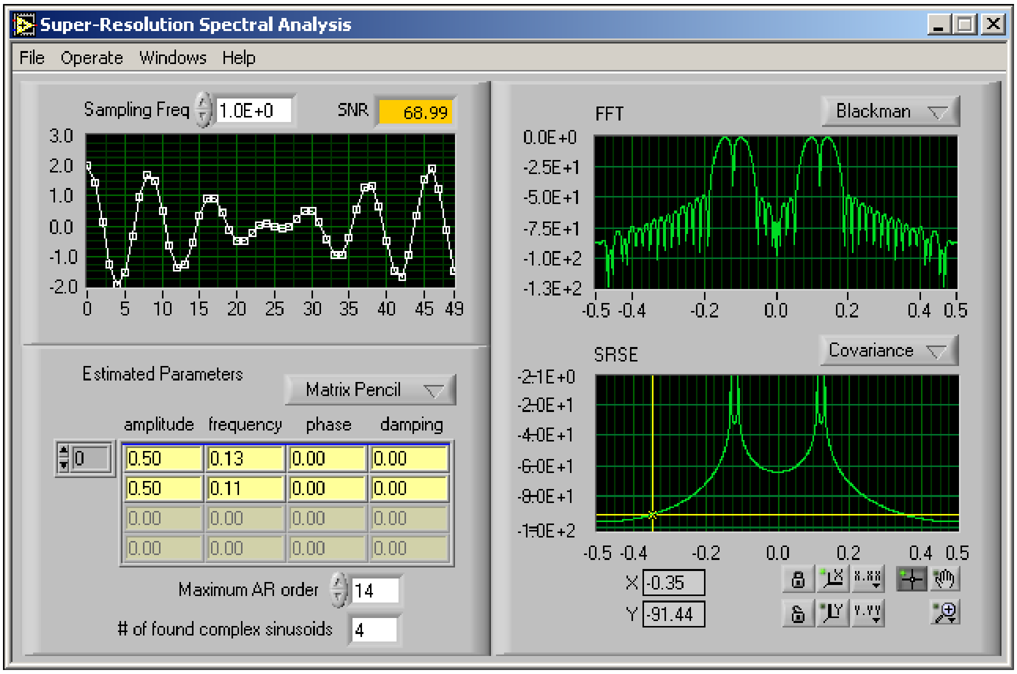
If you have LabVIEW 6.0 or later installed, you can access the source code for this example by selecting **Windows»Show Diagram** or by pressing <Ctrl-E>.

Performing Super-Resolution Spectral Analysis

Complete the following steps to perform super-resolution spectral analysis using the SRSA example.

1. Load data.
2. Set the sampling frequency with the **Sampling Freq** control.
3. Select the window type for the FFT-based spectrum.
4. Select the super-resolution spectrum method.
5. Select the damped sinusoid estimation method.
6. Set the number of complex sinusoids.

The following sections describe the above steps and the SRSA example front panel controls and indicators. You also can select **Help» Show Context Help** or press <Ctrl-H> for more information about controls and indicators.



Sampling Freq 1.0E+0 SNR 68.99

FFT: Blackman

SRSE: Covariance

Estimated Parameters: Matrix Pencil

0

amplitude	frequency	phase	damping
0.50	0.13	0.00	0.00
0.50	0.11	0.00	0.00
0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00

Maximum AR order 14

# of found complex sinusoids 4

X -0.35

Y -91.44

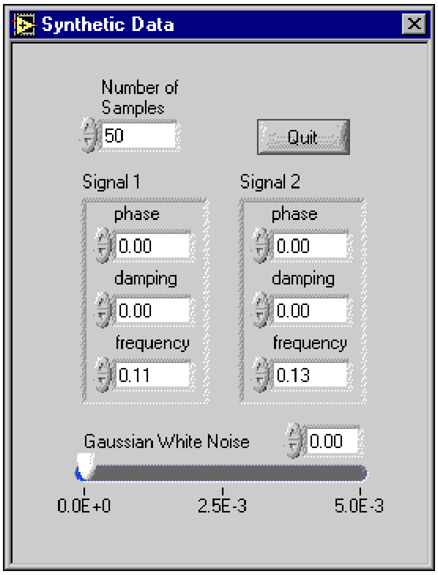
Figure 8-2. Super-Resolution Spectral Analysis Front Panel

Load Data

You can analyze either synthetic data supplied with the example or your own data text file. If you choose to use your own text file, the file must be a one-column or one-row spreadsheet text file. The synthetic data simulates two damped sinusoids plus Gaussian noise. If you are a first-time user, start with the synthetic data, which gives you a better idea of how to properly apply the model-based analysis.

Synthetic Data

To use the synthetic data, select **File»Load»Signal Generator**. When you select **Signal Generator**, the **Synthetic Data** front panel, shown in Figure 8-3, opens. The **Synthetic Data** front panel generates data samples containing two damped sinusoids corrupted with Gaussian white noise.



Number of Samples: 50

	Signal 1	Signal 2
phase	0.00	0.00
damping	0.00	0.00
frequency	0.11	0.13

Gaussian White Noise: 0.00

Figure 8-3. Synthetic Data Front Panel

The damped sinusoid has the following form:

$$s[n] = Ae^{-an}\cos(2\pi fn + \theta)$$

where	
	A is the real-valued amplitude
	a is the real-valued damping factor
	f is the normalized frequency
	θ is the phase

Use the **Number of Samples** control to specify the size of the data set. The SRSA example relies heavily on matrix computation, which requires more computer resources. National Instruments recommends that you limit the value for **Number of Samples** to less than a few hundred because of computing time and memory space considerations. The default value of **Number of Samples** is 50. Table 8-1 lists the default sinusoid parameters.

Table 8-1. Default Sinusoid Parameters

Sinusoid	Amplitude	Phase	Damping Factor	Frequency
sinusoid 1	1.0	0.0	0.0	0.11 Hz
sinusoid 2	1.0	0.0	0.0	0.13 Hz

When **Number of Samples** is set at 50, both the FFT-based and model-based methods separate two different frequencies well. If you reduce **Number of Samples** to 25, only the model-based spectra are able to distinguish between the two frequencies.

Use the **Gaussian White Noise** slide control to adjust the intensity of the additive Gaussian white noise. As mentioned in Chapter 6, *Introduction to Model-Based Frequency Analysis*, of this manual, the results of model-based analysis are sensitive to the intensity and type of noise. The performance of model-based analysis deteriorates substantially as the intensity of noise increases or the noise differs from Gaussian white noise.

Use the **phase**, **damping**, and **frequency** controls to see how the estimation results change.

Click the **Quit** button to close the **Synthetic Data** front panel.

Data Stored as Text Files

To use your own data files, select **File»Load»Data File** and navigate to the data file you want to analyze. The data file must be a one-column or one-row spreadsheet text file.

Set the Sampling Frequency

Use the **Sampling Freq** control, located above the time waveform plot in Figure 8-2, to specify the sampling frequency of the test data. The default value is 1 Hz.

Select the Window Type

Use the ring control above the FFT display, shown in Figure 8-2, to select the window type used to compute the FFT spectrum. You can choose from the following window types:

- Blackman (default)
- Hamming
- Hanning
- Rectangular

Use the FFT-based spectrum for comparison with the model-based spectrum in the SRSE display.

Select the Super-Resolution Spectrum Method

Use the ring control above the SRSE display, as shown in Figure 8-2, to select the super-resolution spectrum method. You can choose either the covariance method (default) or the principle component auto-regressive (PCAR) method.

The PCAR method is less sensitive to noise than that of the covariance method, but it requires more computing time and memory space.

Select the Damped Sinusoid Estimation Method

Use the ring control in the **Estimated Parameters** section, as shown in Figure 8-2, to specify the method used to estimate the parameters associated with damped sinusoids. You can choose either the matrix pencil method (default) or Prony's method. The matrix pencil method is more accurate and efficient than Prony's method. Refer to Chapter *7*, *Model-Based Frequency Analysis Algorithms*, of this manual for more information about the matrix pencil method and Prony's method.

Set the Number of Complex Sinusoids

Correctly specifying the number of complex sinusoids contained in the test data is one of the most important factors in effectively applying model-based analysis. Usually, each complex sinusoid counts as one order, and each real-valued sinusoid counts as two orders. You can have the SRSA example automatically count the number of complex sinusoids, or you can specify the number yourself. If the number of complex sinusoids contained in the test data is unknown, use the automatic method.

Automatic

Select **Operate»Sine Estimation»Automatic** to count the number of complex sinusoids automatically. The SRSA example uses the maximum description length algorithm to automatically estimate the number of complex sinusoids for the test data. # of Found Complex Sinusoids displays the result of this algorithm. To use the maximum description length algorithm, you need to define the upper boundary of the AR order. Use the **Maximum AR Order** control, shown in Figure 8-2, to specify the upper boundary. As the value of the **Maximum AR Order** increases, your results become more precise. However, computation time also increases as **Maximum AR Order** increases. The value of **Maximum AR Order** should be two to three times larger than the real order but cannot be larger than two-thirds of the number of samples. Refer to Chapter 7, *Model-Based Frequency Analysis Algorithms*, of this manual for more information about the maximum description length algorithm.

The indicators in the **Estimated Parameters** section of the front panel, shown in Figure 8-2, return values for amplitude, frequency, phase, and damping for each positive-frequency sinusoid.

Manual

Select **Operate»Sine Estimation»Manual** to specify the number of complex sinusoids yourself. When you select **Manual**, the **# of Complex Sinusoids** control replaces the **Maximum AR Order** control. Use the **# of Complex Sinusoids** control to specify the number of complex sinusoids contained in the test data. For a real-valued signal, if DC is presented, **# of Complex Sinusoids** should be an odd number. In all other case, **# of Complex Sinusoids** should be an even number.

Part IV

Wavelet Analysis

This section of the manual discusses the wavelet analysis and filter bank design.

- Chapter 9, *The Fundamentals of Wavelet Analysis*, describes the history of wavelet analysis, compares Fourier transform and wavelet analysis, and describes some applications of wavelet analysis.
- Chapter 10, *Wavelet Analysis by Discrete Filter Banks*, describes the design of two-channel perfect reconstruction filter banks and defines the types of filter banks used with wavelet analysis.
- Chapter 11, *Wavelet Analysis Applications*, describes the 1D and 2D Wavelet Transform examples and how to design a wavelet and filter bank.

9 The Fundamentals of Wavelet Analysis

Although Alfred Haar first mentioned the term wavelet in a 1909 thesis [9], wavelet analysis received little attention before the late 1970s. In the period between the 1970s and the present, scientists have carefully studied wavelet analysis and successfully applied it in many areas. Some people think current wavelet analysis is just the recasting and unifying of existing theories and techniques, however there is a wider range of potential applications for wavelet analysis than anyone anticipated.

This chapter describes the history of wavelet analysis, compares Fourier transform to wavelet analysis, and describes some applications of wavelet analysis.

Conventional Fourier Transform

The development of wavelet analysis originally was motivated by the desire to overcome the drawbacks of traditional Fourier analysis and short-time Fourier transform (STFT) processes. Fourier transform characterizes the frequency behaviors of a signal but not the frequency changes over time. STFT, or windowed Fourier transform, simultaneously characterizes a signal in time and frequency. After you select a window type, the signal time and frequency resolutions remain fixed, which can cause problems. However, signals encountered in nature always have a long time period at low frequency and a short time period at high frequency. This suggests that the window should have high time resolution at high frequency.

To understand the fundamentals of wavelet analysis, start with an artificial example. Figure 9-1 shows a signal *s(t)* that consists of two truncated sine waveforms. The first waveform spans 0 to 1 second, and the second waveform spans 1 to 1.5 seconds. In other words, the frequency of *s(t)* is 1 Hz for $0 \le t < 1$ and 2 Hz for $1 \le t < 1.5$.

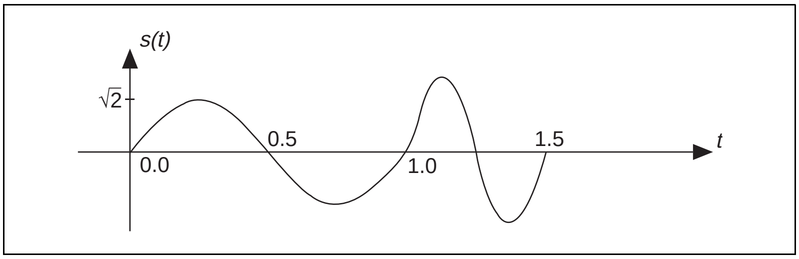


Figure 9-1. Sum of Two Truncated Sine Waveforms

When describing frequency behavior, you traditionally compare $s(t)$ with a group of harmonically-related complex sinusoidal functions, such as $\exp\{j2\pi kt/T\}$. Here, the term harmonically-related complex sinusoidal functions refers to the sets of periodic sinusoidal functions with fundamental frequencies that are all multiples of a single positive frequency $2\pi/T$.

You accomplish the comparison process with the following correlation, or inner product operation:

$$a_k = \int_T s(t)e_k^*(t)dt = \int_T s(t)\exp\left\{-j\frac{2\pi k}{T}t\right\}dt$$

where a_k is the Fourier coefficient and * denotes a complex conjugate.

The magnitude of a_k indicates the degree of similarity between the signal $s(t)$ and the elementary function $\exp\{j2\pi kt/T\}$. If the magnitude of a_k is large, it indicates a high degree of correlation between $s(t)$ and $\exp\{j2\pi kt/T\}$. If the magnitude of a_k is almost 0, it indicates a low degree of correlation between $s(t)$ and $\exp\{j2\pi kt/T\}$. Therefore, you can consider a_k to be the measure of similarity between the signal $s(t)$ and each complex sinusoidal function $\exp\{j2\pi kt/T\}$. Because $\exp\{j2\pi kt/T\}$ represents a distinct frequency $2\pi k/T$, a frequency tick mark, the Fourier coefficient a_k indicates the amount of signal present at the frequency $2\pi k/T$.

In Figure 9-1, $s(t)$ consists of two truncated sine waveforms. The inner product of such truncated signals and pure sine waveforms, which extends from minus infinity to plus infinity, never vanishes. In other words, a_k is not zero for all k. However, the dominant a_k, with the largest magnitude, corresponds to 1 Hz and 2 Hz elementary functions. This indicates that the primary components of $s(t)$ are 1 Hz and 2 Hz signals, but it is unclear, based on a_k alone, when the 1 Hz or the 2 Hz components exist in time.

There are many ways of building the frequency tick marks to measure the frequency behavior of a signal. With complex sinusoidal functions, not only can you analyze signals but you also can reconstruct the original signal with the Fourier coefficient a_k. For example, you can write $s(t)$ in terms of the sum of complex sinusoidal functions according to the following formula, traditionally known as Fourier expansion:

$$s(t) = \sum_{k=-\infty}^{\infty} a_k e_k(t) = \sum_{k=-\infty}^{\infty} a_k \exp\left\{ j\frac{2\pi t}{T}k \right\} \tag{9-1}$$

where a_k is the Fourier coefficient and $2\pi k/T$ is the frequency tick mark.

In this equation, because a_k is not zero for all k, you must use an infinite number of complex sinusoidal functions in Equation 9-1 to restore $s(t)$ in Figure 9-1.

Innovative Wavelet Analysis

Looking at $s(t)$ more closely, you find that to determine the frequency contents of $s(t)$, you need information regarding only one cycle, such as the time span of one cycle. With this information, you can compute the frequency with the following formula:

$$frequency = \frac{1}{time\ span\ of\ one\ cycle}$$

According to this equation, as the frequency becomes higher, the time span becomes shorter. Therefore, instead of using infinitely-long complex sinusoidal functions, you can use only one cycle of a sinusoidal waveform, or a wavelet, to measure $s(t)$. The wavelet $\psi(t)$ used to measure $s(t)$ is one cycle of sinusoidal waveform, as shown in Figure 9-2.

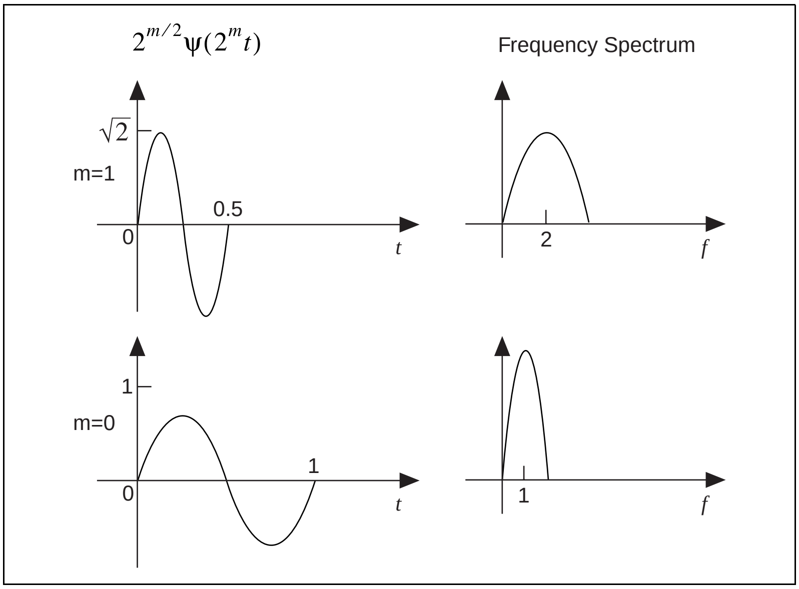


Figure 9-2. Wavelet

Because $\psi(t)$ spans 1 second, consider the frequency of $\psi(t)$ to be 1 Hz. As in the case of Fourier analysis, you can achieve the comparison process with the following correlation, or inner product, operation:

$$W_{m,n} = \int_T s(t)\psi_{m,n}(t)dt \tag{9-2}$$

where $W_{m,n}$ denotes the wavelet transform coefficients and $\psi_{m,n}(t)$ are the elementary functions of the wavelet transform. However, the structure of the elementary functions $\psi_{m,n}(t)$ differs from the Fourier transformations, which are the dilated and shifted versions of $\psi(t)$. The elementary functions of the wavelet transform, $\psi_{m,n}(t)$, can be computed by the following equation:

$$\psi_{m,n}(t) = 2^{m/2}\psi(2^m(t - n2^{-m})) \tag{9-3}$$

where m and n are integers.

When you increase n, you shift $\psi_{m,n}(t)$ forward in time. When you increase m, you compress the time duration, which increases the center frequency and frequency bandwidth of $\psi(t)$ [27]. Consider the parameter m as the scale factor and 2^{-m} as the sampling step. Therefore, as the time duration becomes shorter, the time sampling step becomes smaller, and vice versa.

If you assume that the center frequency of $\psi(t)$ is ω_0, the center frequency of $\psi_{m,n}(t)$ is $2^m\omega_0$. Consequently, you can systematically adjust the scale factor m to achieve different frequency tick marks to measure the signal frequency contents. In other words, as the scale factor m increases, the center frequency and bandwidth of the wavelet increases 2^m.

Figure 9-3 depicts the wavelet transform procedure. First, let $m = n = 0$ by aligning $\psi(t)$ and $s(t)$ at $t = 0$. As in Equation 9-3, compare $\psi(t)$ with $s(t)$ for $0 \le t < 1$. You get $W_{0,0} = 1$. Shift $\psi(t)$ to the next second, $n = 1$, and compare it with $s(t)$ for $1 \le t < 2$. You get $W_{0,1} = 0$.

Compress $\psi(t)$ into 0 seconds to 0.5 seconds, $m = 1$, and repeat the previous operations with the time-shift step 0.5. You get the following results, which are also displayed in the upper right corner of Figure 9-3:

$$W_{1,0} = 0 \qquad W_{1,1} = 0 \qquad W_{1,2} = 1 \qquad W_{1,3} = 0$$

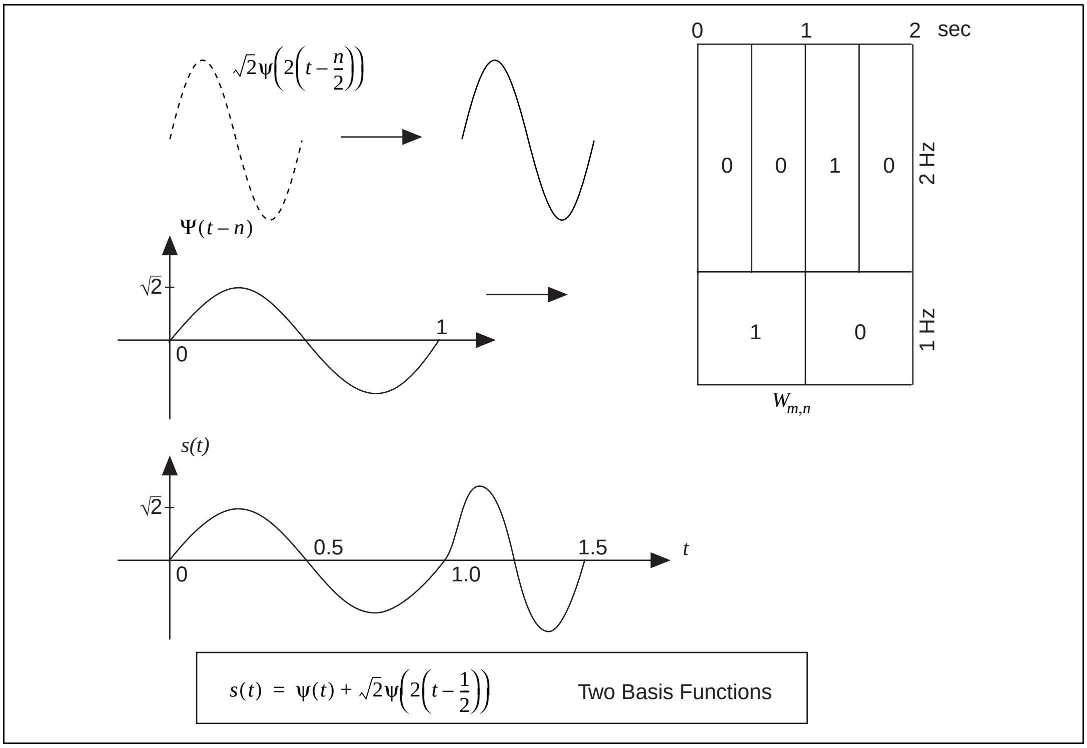

$\sqrt{2}\psi\left(2\left(t-\frac{n}{2}\right)\right)$
$\Psi(t-n)$
$\sqrt{2}$
0
1
$s(t)$
$\sqrt{2}$
0
0.5
1.0
1.5
t
0 1 2 sec
0 0 1 0
2 Hz
1 0
1 Hz
$W_{m,n}$
$s(t) = \psi(t) + \sqrt{2}\psi\left(2\left(t-\frac{1}{2}\right)\right)$
Two Basis Functions

Figure 9-3. Wavelet Analysis

You can continue to compress $\psi(t)$ by increasing the scale factor m and reducing the time-shift step 2^{-m} to test $s(t)$. This procedure is called wavelet transform. $\psi(t)$ is called the mother wavelet, because the different wavelets used to measure $s(t)$ are the dilated and shifted versions of this wavelet. The results of each comparison, $W_{m,n}$, are named wavelet coefficients. The index m and n are the scale and time indicators, respectively, which describe the signal behavior in the joint time-scale domain. As shown in Figure 9-5, you can convert the scale into frequency. Hence, $W_{m,n}$ also can be considered the signal representation in the joint time and frequency domain. In the example in Figure 9-3, when you check the wavelet coefficients, you find out that for $0 \le t < 1$, the frequency of $s(t)$ is 1 Hz, and for $1 \le t < 1.5$, the frequency of $s(t)$ is 2 Hz.

Unlike Fourier analysis, wavelet transform not only indicates what frequencies the signal $s(t)$ contains but also indicates when these frequencies occur. Moreover, the wavelet coefficients $W_{m,n}$ of a real-valued signal $s(t)$ are always real as long as you choose real-valued $\psi(t)$. Compared to Fourier expansion, you usually can use fewer wavelet functions to represent the signal $s(t)$. In the example in Figure 9-3, $s(t)$ can be completely represented by two terms, whereas an infinite number of complex sinusoidal functions would be needed in the case of Fourier expansion.

Wavelet Analysis versus Fourier Analysis

You can apply short-time Fourier transform (STFT) to characterize a signal simultaneously in both the time and frequency domains. However, you also can use wavelet analysis to perform the same operation because of its similarity to STFT. You compute both the STFT and the wavelet transform by the correlation, or inner product operation, but the main difference lies in how you build the elementary functions. Figure 9-6 shows a comparison of the transform processes.

Figure 9-4 illustrates the sampling grid for the STFT. For STFT, the elementary functions used to test the signal are time-shifted, frequency-modulated single window functions, all with some envelope. Because this modulation does not change the time or frequency resolutions, the time and frequency resolutions of the elementary functions employed in STFT are constant [27].

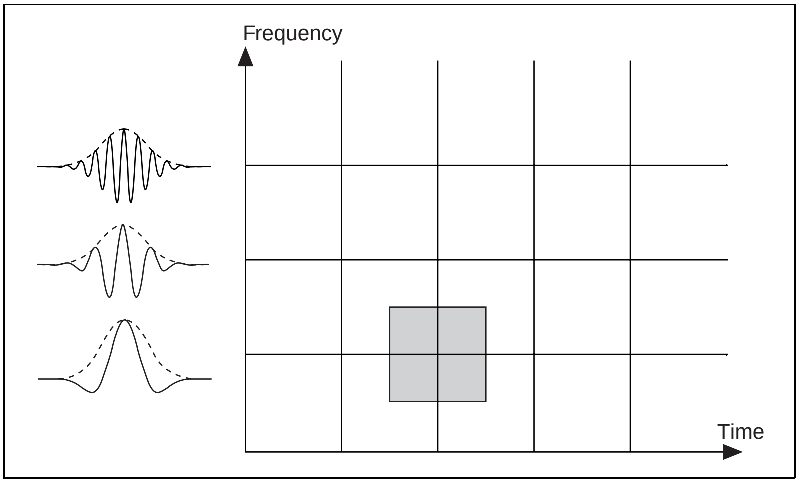


Figure 9-4. Short-Time Fourier Transform Sampling Grid

For wavelet transform, increasing the scale parameter *m* reduces the width of the wavelets. The time resolution of the wavelets increases, and the frequency resolution decreases as *m* becomes larger. This shows that wavelet analysis has good time resolution at high frequencies and good frequency resolution at low frequencies.

Figure 9-5 illustrates the sampling grid for wavelet transform. Suppose that the center frequency and bandwidth of the mother wavelet $\psi(t)$ are ω_0 and Δ_ω, respectively. For $\psi(2^m t)$, the center frequency is $2^m\omega_0$, and the bandwidth is $2^m\Delta_\omega$. Although the time and frequency resolutions change at different scales *m*, the ratio between the bandwidth and the center frequency remains constant. Therefore, wavelet analysis is also called constant Q analysis, where Q = center frequency⁄bandwidth.

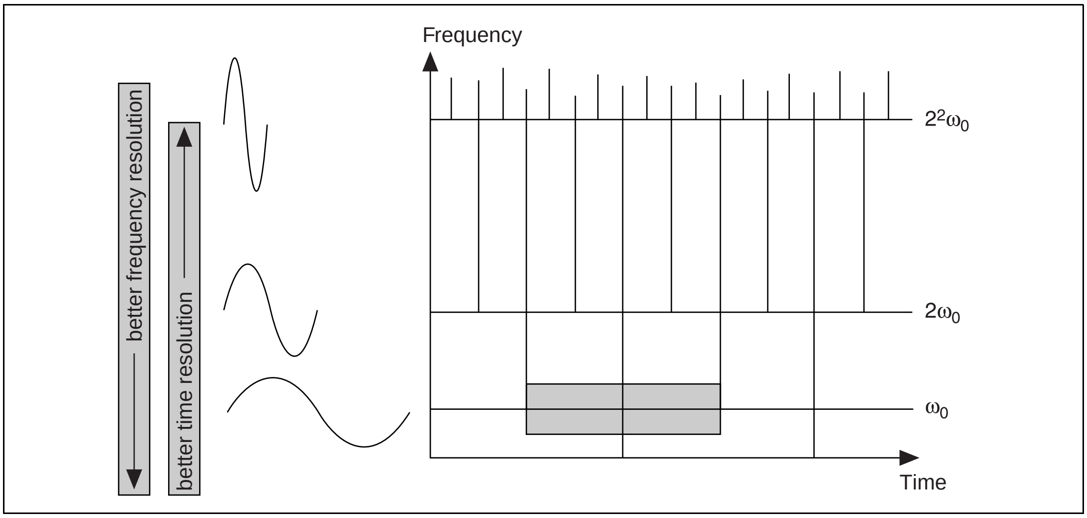


Figure 9-5. Wavelet Transform Sampling Grid

Wavelet transform is closely related to both conventional Fourier transform and short-time Fourier transform. As shown in Figure 9-6, all these transform processes employ the same mathematical tool, the correlation operation, or inner product, to compare the signal $s(t)$ to the elementary function $b_\alpha(t)$. The difference lies in the structure of the elementary functions $\{e_\alpha(t)\}$. In some cases, wavelet analysis is more natural because the signals always have a long time cycle at low frequency and a short time cycle at high frequency.

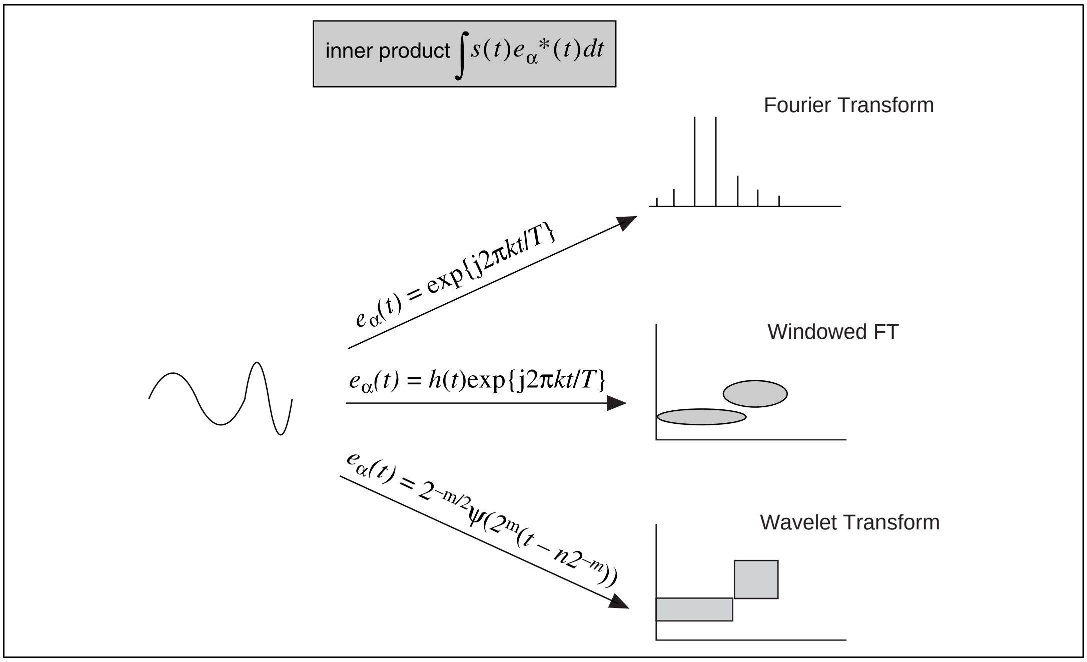

inner product $\int s(t)e_\alpha^*(t)dt$
Fourier Transform
$e_\alpha(t) = \exp\{j2\pi kt/T\}$
Windowed FT
$e_\alpha(t) = h(t)\exp\{j2\pi kt/T\}$
$e_\alpha(t) = 2^{-m/2}\psi(2^m(t - n2^{-m}))$
Wavelet Transform

Figure 9-6. Comparison of Transform Processes

Applications of Wavelet Analysis

You can use wavelet analysis in a variety of applications, including detecting the discontinuity of a signal, looking at a signal from different scales, removing the trend of a signal, suppressing noise, and compressing data.

Discontinuity Detection

Wavelet analysis detects signal discontinuity, such as jumps, spikes, and other non-smooth features. Ridding signals of noise is often much easier to identify in the wavelet domain than in the original domain

For example, the top plot of Figure 9-7 illustrates a signal *s*(*k*) made up of two exponential functions. The turning point, or the discontinuity, of the first derivative is at *k* = 500. The remaining plots are wavelet coefficients with different scale factors *m*. As the scale factor increases, you can pinpoint the location of the discontinuity.

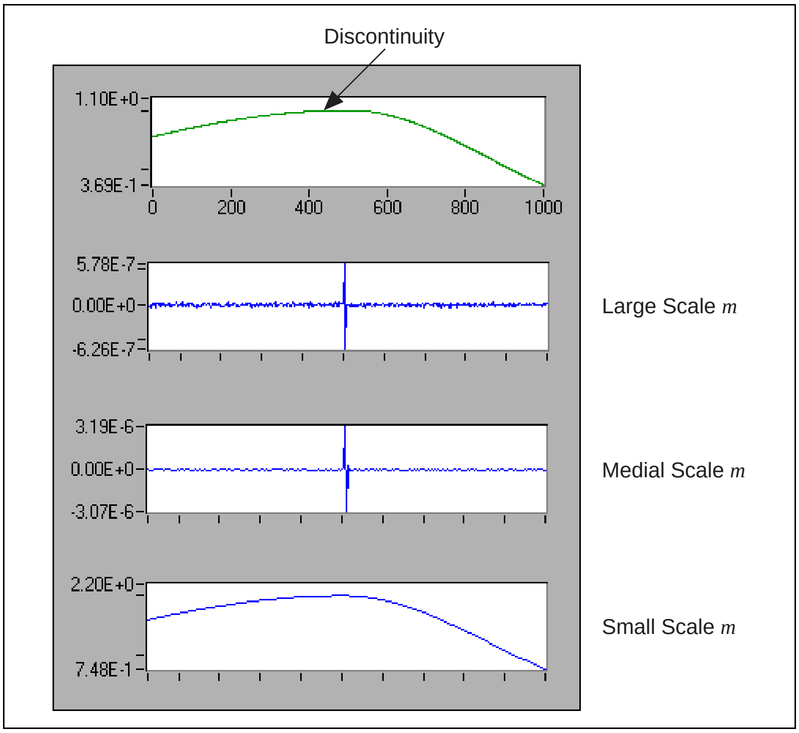


Figure 9-7. Detection of Discontinuity

Using wavelet analysis to detect the discontinuity, or break point, of a signal has helped to successfully repair scratches on old phonograph records. The procedure works by taking the wavelet transform on the signal, smoothing unwanted spikes, and inverting the transform to reconstruct the original signal minus the noise. In 1889, an agent of Thomas Edison used a wax cylinder to record Johannes Brahms performing his Hungarian Dance No. 1 in G minor. The recording was so poor that it was hard to discern the melody. By using wavelet transform, researchers improved the sound quality enough to distinguish the melody.

Multiscale Analysis

Multiscale analysis involves looking at a signal from different scales. Wavelet transform-based multiscale analysis helps you better understand the signal and provides a useful tool for selectively discarding undesired components, such as noise and trend, that corrupt the original signal.

Figure 9-8 illustrates a multiscale analysis of a Standard & Poor's (S&P) 500 stock index during the years 1947 through 1993. The top plot displays a monthly S&P 500 index and the bottom plot describes the long-term trend of the stock movement. The remaining two plots display the short-term fluctuation of the stock, at different levels, during this time. To better characterize the fluctuation that reflects the short-term behavior of the stock, you must remove the trend. To do this, first adjust the wavelet decomposition level until you obtain a desired trend. Then, set the corresponding wavelet coefficients to zero and reconstruct the original samples minus the trend.

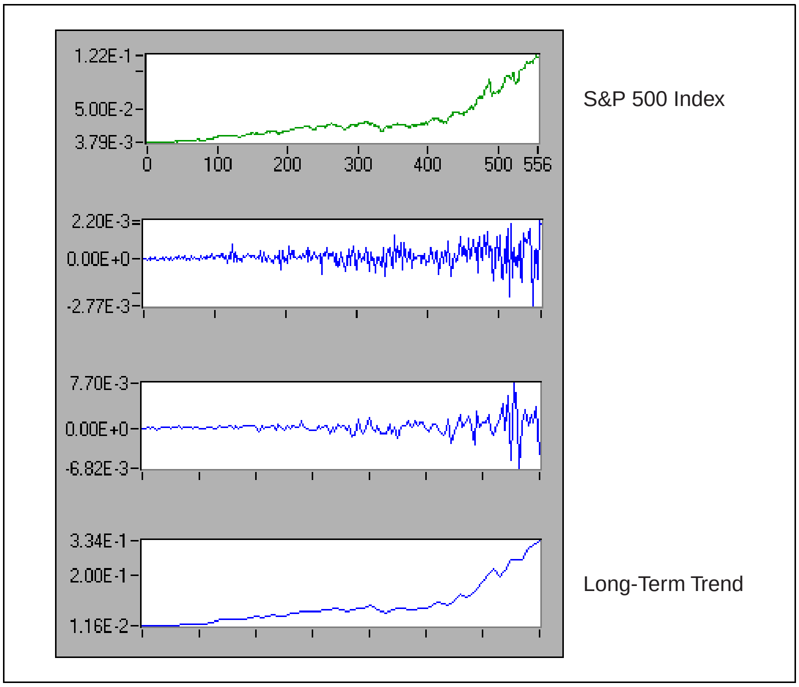


Figure 9-8. Multiscale Analysis

Detrending

The trend of a signal is often one of the least interesting aspects of the signal. Also, because the trend attaches to a strong DC component in the frequency spectrum, it blocks many other important signal features. Detrending involves removing the trend from a signal. How to remove the trend is one of the most important issues in the application of joint time-frequency analysis.

Traditional detrending techniques usually use lowpass filtering to remove the trend, which blurs sharp features in the underlying signal. Wavelet-based detrending is somewhat superior to this process because it preserves the important features of the original signal.

Figures 9-8 and 9-9 illustrate the same S&P 500 stock index information, but Figure 9-9 shows it as a joint time-frequency analysis. The top plot illustrates the S&P 500 stock index and its corresponding long-term trend, smooth curve. The center plot displays the residue between the original data and the trend, reflecting the short-term fluctuation. The bottom plot displays the joint time and frequency behavior of the residue. It shows that over the past 50 years, a four-year cycle dominates the S&P 500 index, which agrees with most economists’ assertions.

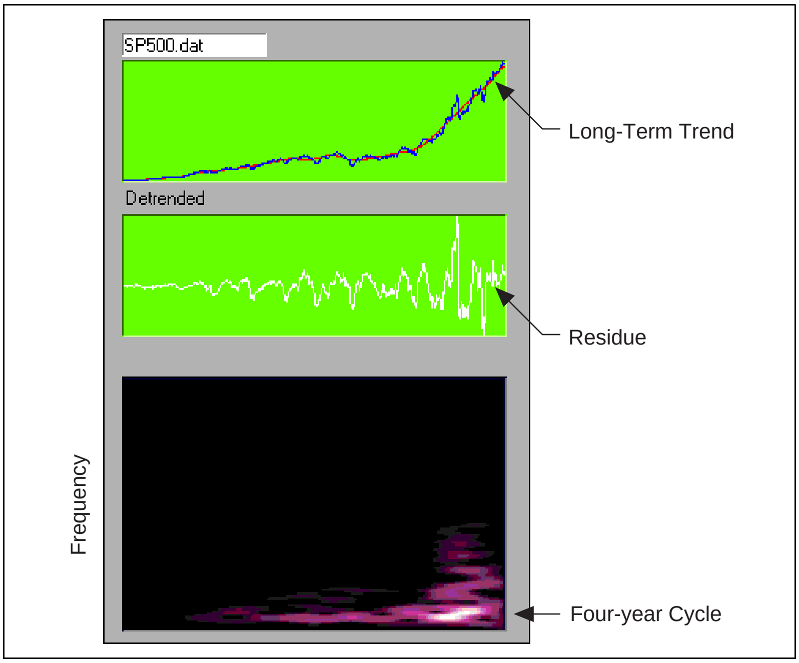


Figure 9-9. Detrend

Denoise

Unlike conventional Fourier transform, which uses only one basis function, wavelet transform provides an infinite number of mother wavelets to select. Consequently, you can select the wavelets that best match the signal. Once the wavelets match the signal, you can use a few wavelets as a basis from which to approximate the signal and achieve denoise.

Figure 9-10 illustrates denoise, one of the most successful applications of wavelet analysis. This application works by first taking the wavelet transform of the signal, setting the coefficients below a certain threshold to zero, and finally, inverting the transform to reconstruct the original signal. If the threshold is set properly, the resulting signal has less noise interference. Refer to Donoho's work for more information about wavelet transform-based denoising [8].

Although Figure 9-10 uses only 25% of the original data, the reconstruction preserves all important features contained in the original image. The left image is transformed into the wavelet basis with 75% of the wavelet components, those of the smallest magnitude, set to zero. The right image is reconstructed from the remaining 25% of the wavelet components.

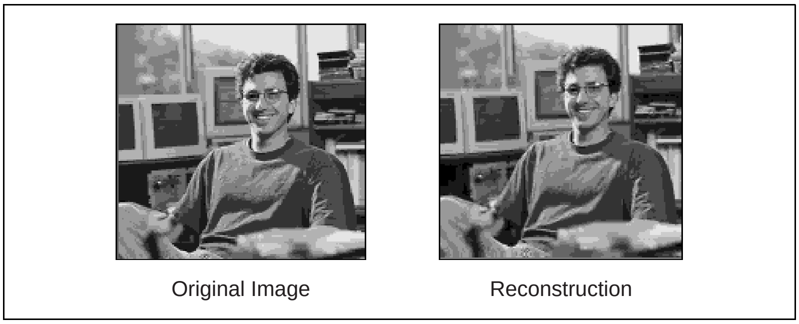


Figure 9-10. Denoise

Performance Issues

Although wavelet analysis possesses many attractive features, its numerical implementation is not as straightforward as that of its counterparts, such as conventional Fourier transform and STFT. The difficulty arises from the following two aspects:

- In order to reconstruct the original signal, the selection of the mother wavelet $\psi(t)$ is not arbitrary. Although any function can be used in Equation 9-2, you sometimes cannot restore the original signal based on the resulting wavelet coefficients $W_{m,n}$. $\psi(t)$ is a valid, or qualified, wavelet only if you can reconstruct the original signal from its corresponding wavelet coefficients. The selection of the qualified wavelet is subject to certain restrictions. On the other hand, it is not unique. Unlike the case of conventional Fourier transform, in which the basis functions must be complex sinusoidal functions, you can select from an infinite number of mother wavelet functions. Therefore, the biggest issue of applying wavelet analysis is how to choose a desired mother wavelet $\psi(t)$. It is generally agreed that the success of the wavelet transform application depends on a proper wavelet function selection.
- Because the scale factor m could go from negative infinity to positive infinity, it is impossible to make the time index of the wavelet function, $2^m(t - n2^{-m})$, an integer number simply by digitizing t as $i\Delta_t$, where Δ_t denotes the time sampling interval. This problem prohibits using digital computers to evaluate wavelet transform.

Fortunately, researchers discovered a relationship between wavelet transform and the perfect reconstruction filter bank, a type of digital filter bank. You can implement wavelet transform with specific types of digital filter banks known as two-channel perfect reconstruction filter banks. Chapter 10, *Wavelet Analysis by Discrete Filter Banks*, of this manual describes the basics of two-channel perfect reconstruction filter banks and the types of digital filter banks used with wavelet analysis.

10

Wavelet Analysis by Discrete Filter Banks

This chapter describes the design of two-channel perfect reconstruction filter banks and defines the types of filter banks used with wavelet analysis.

Two-Channel Perfect Reconstruction Filter Banks

Two-channel perfect reconstruction (PR) filter banks have been recognized as useful in signal processing for a long time, particularly after researchers discovered that two-channel PR filter banks are closely related to wavelet transform. Now, two-channel PR filter banks are a common technique for computing wavelet transform.

Figure 10-1 illustrates a typical two-channel filter bank system. The signal $X(z)$ is first filtered by a filter bank constituted by $G_0(z)$ and $G_1(z)$.

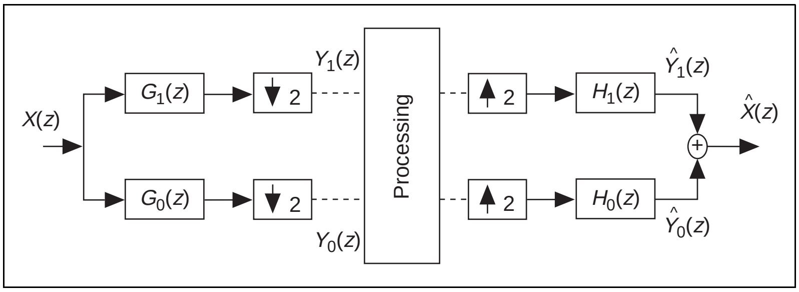


Figure 10-1. Two-Channel Filter Bank

Note For a finite impulse response (FIR) digital filter $g[n]$, the z-transform is defined as

$$G(z) = \sum_{n=0}^{N} g[n]z^{-n} = G(e^{j\omega}) = G(\omega) = \sum_{n=0}^{N} g[n]e^{-j\omega n}$$

where N denotes the filter order. Consequently, the filter length is equal to $N + 1$. Thus, $\omega = 0$ is equivalent to $z = 1$ and $\omega = \pi$ is equivalent to $z = -1$.

That is, *G(0)* and *G*(*p*) in the frequency domain correspond to *G*(*1*) and *G*(*–1*) in the z-domain.

The outputs of $G_0(z)$ and $G_1(z)$ are downsampled by two to obtain $Y_0(z)$ and $Y_1(z)$. After some processing, the modified signals are upsampled and filtered by another filter bank constituted by $H_0(z)$ and $H_1(z)$. If no processing takes place between the two filter banks, $Y_0(z)$ and $Y_1(z)$ are not altered. The sum of outputs of $H_0(z)$ and $H_1(z)$ is identical to the original signal $X(z)$, except for the time delay. Such a system is commonly referred to as a two-channel PR filter bank. $G_0(z)$ and $G_1(z)$ form an analysis filter bank, whereas $H_0(z)$ and $H_1(z)$ form a synthesis filter bank.

Note $G(z)$ and $H(z)$ can be interchanged. For instance, you can use $H_0(z)$ and $H_1(z)$ for analysis and $G_0(z)$ and $G_1(z)$ for synthesis. $H_0(z)$ and $H_1(z)$ are usually considered as the dual of $G_0(z)$ and $G_1(z)$, and vice versa.

Traditionally, $G_0(z)$ and $H_0(z)$ are lowpass filters, whereas $G_1(z)$ and $H_1(z)$ are highpass filters, where the subscripts 0 and 1 represent lowpass and highpass filters, respectively. Because two-channel PR filter banks process $Y_0(z)$ and $Y_1(z)$ at half the sampling rate of the original signal $X(z)$, they are used in many signal processing applications.

If you assume the conventions shown in Figure 10-2,

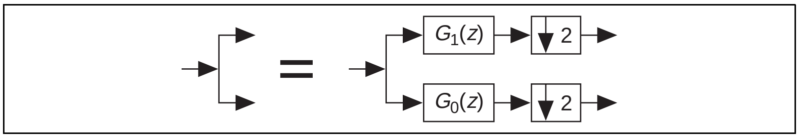


Figure 10-2. Conventions for Wavelet Transform

then the relationship between two-channel PR filter banks and wavelet transform can be illustrated by Figure 10-3.

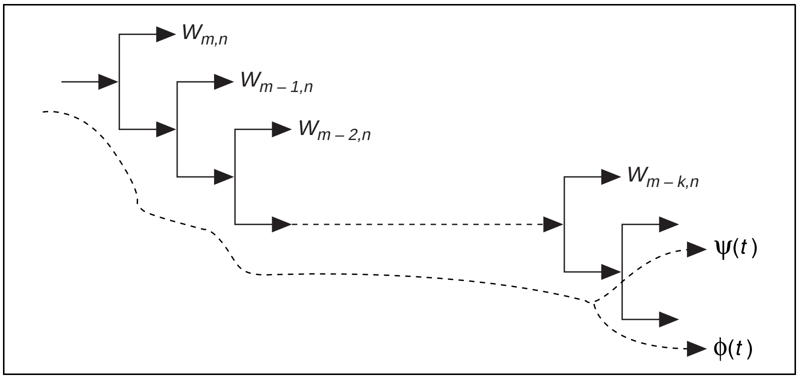


Figure 10-3. Relationship of Two-Channel PR Filter Banks and Wavelet Transform

Under certain conditions, two-channel PR filter banks are related to wavelet transform in the following two ways:

- The impulse response of the lowpass filters converges to the scaling function $\phi(t)$. Once you obtain $\phi(t)$, you can compute the mother wavelet function $\psi(t)$ by highpass $\phi(t)$, as shown in Figure 10-3.
- The outputs of each of the highpass filters are approximations of the wavelet transform. You can accomplish wavelet transform with a tree of two-channel PR filter banks. The selection of a desirable mother wavelet becomes the design of two-channel PR filter banks [27].

Figure 10-4 illustrates the relationship of filter banks and wavelet transform coefficients.

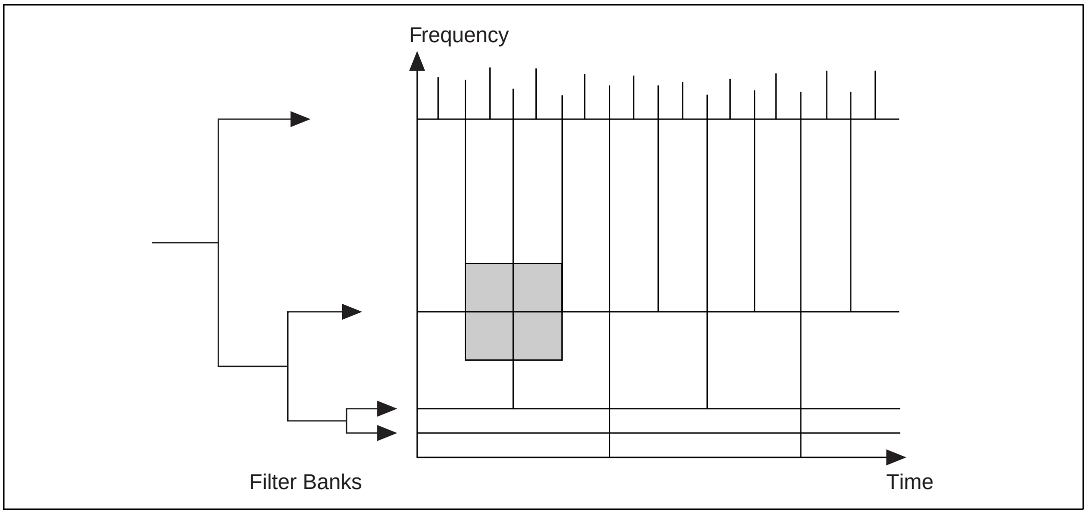


Figure 10-4. Filter Bank and Wavelet Transform Coefficients

The following sections describe the design fundamentals for two types of two-channel PR filter banks, biorthogonal and orthogonal. In most equations, you receive the results without justification. Refer to the works of Oppenheim and Schafer [19], Parks and Burrus [20], and Parks and McClellan [21], [22] for more information about the mathematical bases used in the following sections.

Biorthogonal Filter Banks

In Figure 10-1, you can define the output of the low-channel as [30] [31]

$$\hat{Y}_0(z) = \frac{1}{2}H_0(z)[G_0(z)X(z) + G_0(-z)X(-z)]$$

Similarly, you can define the output of the high channel as

$$\hat{Y}_1(z) = \frac{1}{2}H_1(z)[G_1(z)X(z) + G_1(-z)X(-z)]$$

Add the outputs of the two channels together to obtain

$$\frac{1}{2}[H_0(z)G_0(z) + H_1(z)G_1(z)]X(z) + \frac{1}{2}[H_0(z)G_0(-z) + H_1(z)G_1(-z)]X(-z) \tag{10-1}$$

One term involves $X(z)$, and the other involves $X(-z)$. For perfect reconstruction, the term with $X(-z)$, traditionally called the alias term, must be zero. To achieve this, you want

$$H_0(z)G_0(-z) + H_1(z)G_1(-z) = 0 \tag{10-2}$$

which you accomplish by letting

$$H_0(z) = G_1(-z) \qquad \text{and} \qquad H_1(z) = -G_0(-z) \tag{10-3}$$

The relationship in Equation 10-3 implies that you can obtain $h_0[n]$ by alternating the sign of $g_1[n]$, as shown in the following equation:

$$h_0[n] = (-1)^n g_1[n]$$

Similarly,

$$h_1[n] = (-1)^{n+1} g_0[n] \tag{10-4}$$

Therefore, $g_1[n]$ and $h_1[n]$ are the highpass filters, if $g_0[n]$ and $h_0[n]$ are the lowpass filters. For perfect reconstruction, you also want the first term in Equation 10-1, called the distortion term, to be a constant or a pure time delay. For example,

$$H_0(z)G_0(z) + H_1(z)G_1(z) = 2z^{-l} \tag{10-5}$$

where l denotes a time delay.

If you satisfy both Equations 10-2 and 10-5, the output of the two-channel filter bank in Figure 10-1 is a delayed version of the input signal:

$$\hat{X}(z) = z^{-l}X(z)$$

However, there remains a problem computing $G_0(z)$ and $G_1(z)$, or $H_0(z)$ and $H_1(z)$. Once you determine $G_0(z)$ and $G_1(z)$, you can find the rest of the filters with Equation 10-3. You also can write Equation 10-3 as

$$G_1(z) = H_0(-z) \qquad \text{and} \qquad H_1(z) = -G_0(-z)$$

which, when substituted into Equation 10-5, yields

$$G_0(z)H_0(z) - G_0(-z)H_0(-z) = P_0(z) - P_0(-z) = 2z^{-l} \tag{10-6}$$

where $P_0(z)$ denotes the product of two lowpass filters, $G_0(z)$ and $H_0(z)$, from the following equation:

$$P_0(z) = G_0(z)H_0(z) \tag{10-7}$$

Equation 10-6 indicates that all odd terms of the product of the two lowpass filters, $G_0(z)$ and $H_0(z)$, must be zero except for order l, where l must be odd. But, even order terms are arbitrary. You can summarize these observations with the following formula:

$$p_0[n] = \begin{cases} 0 & n \text{ odd and } n \neq l \\ 2 & n = l \\ \text{arbitrary} & n \text{ even} \end{cases} \tag{10-8}$$

This reduces the design of two-channel PR filter banks to two steps.

1. Design a filter $P_0(z)$ that satisfies Equation 10-8.
2. Factorize $P_0(z)$ into $G_0(z)$ and $H_0(z)$. Then use Equation 10-3 to compute $G_1(z)$ and $H_1(z)$.

The following two types of filters are frequently used for $P_0(z)$:

- an equiripple halfband filter [32]
- a maximum flat filter

In the equiripple halfband filter, halfband refers to a filter in which $\omega_s + \omega_p = \pi$, where ω_s denotes the stopband frequency and ω_p denotes the passband frequency, as shown in Figure 10-5.

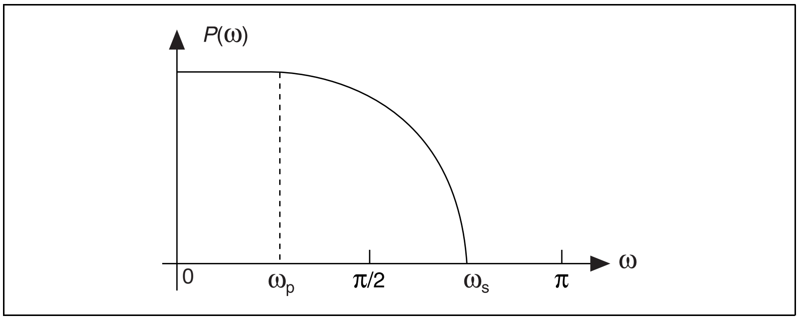


Figure 10-5. Halfband Filter

The form of the maximum flat filter is defined by the following formula:

$$P_0(z) = (1 + z^{-1})^{2p} Q(z) \tag{10-9}$$

which has $2p$ zeros at $z = -1$ or $\omega = \pi$. If you limit the order of the polynomial $Q(z)$ to $2p - 2$, then $Q(z)$ is unique.

Note The maximum flat filter, here, differs from the Butterworth filter. The low-frequency asymptote of the Butterworth filter is a constant. The maximum flat filter is not.

In all cases, the product of lowpass filter $P_0(z)$ is a type I filter:

$$p_0[n] = p_0[N - n] \qquad N \text{ even}$$

where N denotes the filter order. Consequently, the number of coefficients $p_0[n]$ is odd, $N + 1$.

Figure 10-6 plots the zeros distribution of a maximum flat filter $P_0(z)$ for $p = 3$.

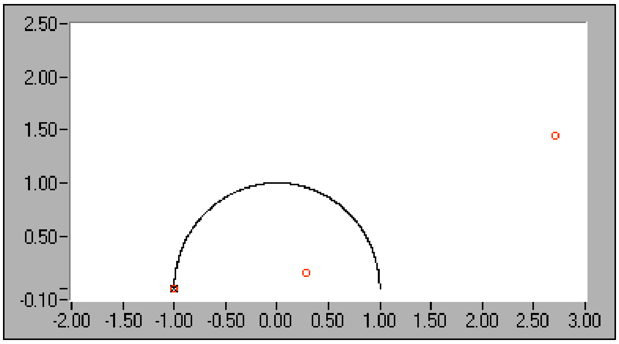


Figure 10-6. Zeros Distribution for $(1 - z^{-1})^6 Q(z)$

There are six zeros at $\omega = \pi$. In this case, the order of the unique polynomial $Q(z)$ is four, which contributes another four zeros that are not on the unit circle. If you let three zeros at $\omega = \pi$ go to $G_0(z)$ according to the formula

$$G_0(z) = (1 + z^{-1})^3$$

and the rest of the zeros go to $H_0(z)$, you obtain B-spline filter banks. The coefficients of $g_0[n]$ and $g_1[n]$ and the corresponding scaling function and mother wavelet are plotted in Figure 10-7. Both the scaling function and mother wavelet generated by $g_0[n]$ and $g_1[n]$ are smooth.

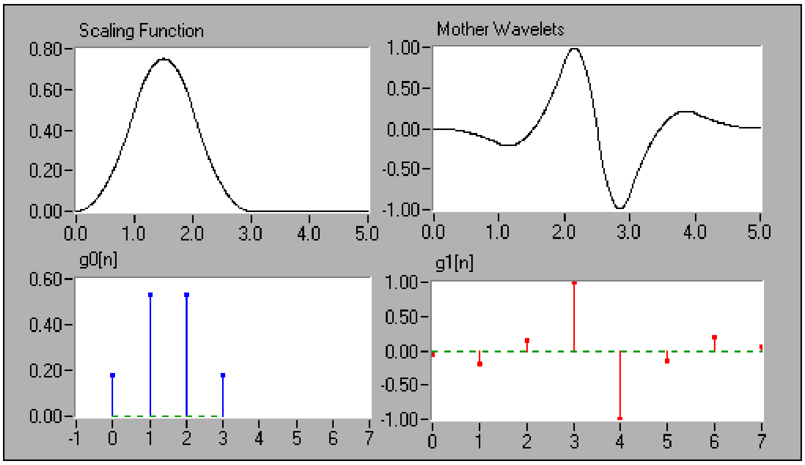


Figure 10-7. B-Spline Filter Bank

Figure 10-8 depicts the dual filter bank $h_0[n]$ and $h_1[n]$ and the corresponding scaling function and mother wavelet. You also can use $h_0[n]$ and $h_1[n]$ for analysis. In Figure 10-8, the tree filter banks constituted by $h_0[n]$ and $h_1[n]$ do not converge.

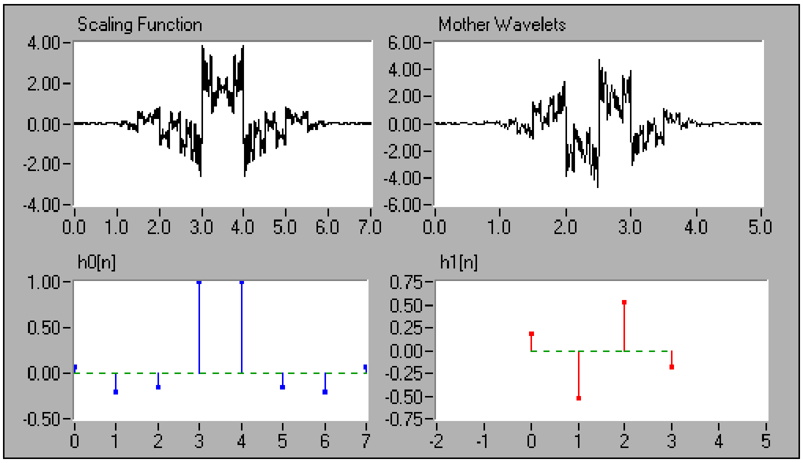


Figure 10-8. Dual B-Spline Filter Bank

Remember that two-channel PR filter banks do not necessarily correspond to the wavelet transform. The wavelet transformations are special cases of two-channel PR filter banks. The conditions of two-channel PR filter banks are more moderate than those for the wavelet transform.

Finally, the analysis filter banks and synthesis filter banks presented in this section are orthogonal to each other:

$$\sum_n g_i[n-2k]h_i[n] = \delta(k) \tag{10-10}$$

and

$$\sum g_i[n-2k]h_l[n] = 0 \qquad i \neq l, \forall k$$

The filters banks that satisfy Equation 10-10 are traditionally called biorthogonal filter banks. In addition to Equation 10-10, if the analysis filter banks also satisfy the following equations:

$$\sum_{n} g_i[n-2k]g_i[n] = \delta(k) \qquad (10\text{-}11)$$

and

$$\sum g_i[n-2k]g_l[n] = 0 \qquad i \neq l, \forall k$$

the resulting filter banks are called orthogonal filter banks. Orthogonal filter banks are special cases of biorthogonal filter banks.

Orthogonal Filter Banks

As shown in the preceding section, once you determine $P_0(z)$, the product of two lowpass filters, you must factorize $P_0(z)$ into $G_0(z)$ and $H_0(z)$. The combinations of zeros are not unique. Different combinations lead to different filter banks. Sometimes $G_0(z)$ and $G_1(z)$ work well, but $H_0(z)$ and $H_1(z)$ might not; refer to Figure 10-7 and Figure 10-8. One way to make this process easier is to limit the selections to a subset. The most effective approach is to require $G_0(z)$ and $G_1(z)$, and thereby $H_0(z)$ and $H_1(z)$, to be orthogonal, as described by Equation 10-11.

These constraints reduce the filter bank design to one filter design. Once you select $G_0(z)$, you can find the other filters. The constraints imposed by Equation 10-11 guarantee that both filter banks have the same performance and provide other advantages, as well. For example, many applications demonstrate that the lack of orthogonality complicates quantization and bit allocation between bands, eliminating the conservation of energy.

To achieve Equation 10-11, let

$$G_1(z) = -z^{-N}G_0(-z^{-1}) \qquad (10\text{-}12)$$

which implies that $g_1[n]$ is the alternating flip of $g_0[n]$

$$(g_1[0], g_1[1], g_1[2], \ldots) = (g_0[N], -g_0[N-1], g_0[N-2], \ldots)$$

Equation 10-12 implies that, for orthogonal wavelets and filter banks,

$$H_0(z) = z^{-N}G_0(z^{-1})$$

where you use the relation in Equation 10-3. Consequently, Equation 10-7 can be written as

$$P_0(z) = z^{-N}G_0(z)G_0(z^{-1})$$

If $G_0(z)G_0(z^{-1}) = P(z)$, then

$$P(e^{j\omega}) = \sum_{n=-N}^{N} p[n]e^{-j\omega n} = \left|\sum_{n=0}^{N} g_0[n]e^{-j\omega n}\right|^2 \tag{10-13}$$

which implies that $P(z)$ is non-negative.

Similar to biorthogonal cases, the selection of $P_0(z)$ in orthogonal cases is dominated by maximum flat and equiripple halfband filters. However, because of constraints imposed by Equation 10-13, $P_0(z)$ must be the time-shifted, non-negative function $P(z)$. Although the maximum flat filter in Equation 10-9 ensures this requirement, special care must be taken when $P_0(z)$ is an equiripple halfband filter.

Figure 10-9 plots the third-order Daubechies filter banks and wavelets. It is derived from the same maximum flat filter as that depicted in Figure 10-6. In this case, however, $G_0(z)$ contains three zeros at $\omega = \pi$ and all zeros inside the unit circle, therefore possessing minimum phase. Because of the orthogonality, its dual filter bank has the same convergence property. Compared to the B-spline cases in Figures 10-7 and 10-8, the third-order Daubechies wavelet and scaling function is not as smooth as that of $G_0(z)$ and $G_1(z)$ in Figure 10-7 but is much smoother than that of $H_0(z)$ and $H_1(z)$ in Figure 10-8.

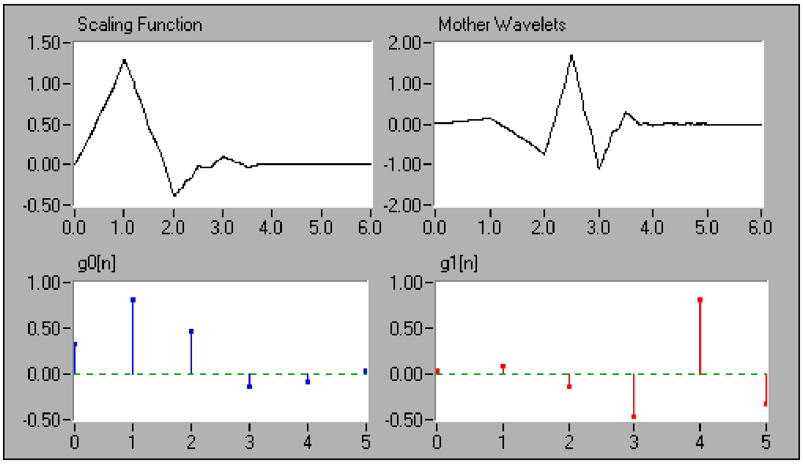


Figure 10-9. Third-Order Daubechies Filter Banks and Wavelets

2D Signal Processing

The preceding sections introduced two-channel PR filter banks for 1D signal processing. In fact, two-channel PR filter banks also can be used for 2D signals, as shown in Figure 10-10. In this case, you process rows first and then columns. Consequently, one 2D array splits to the following four 2D sub-arrays:

- low-low
- low-high
- high-low
- high-high

Each sub-array is a quarter the size of the original 2D signal.

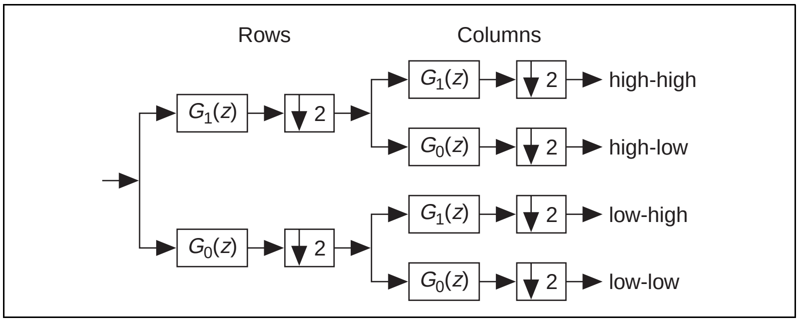
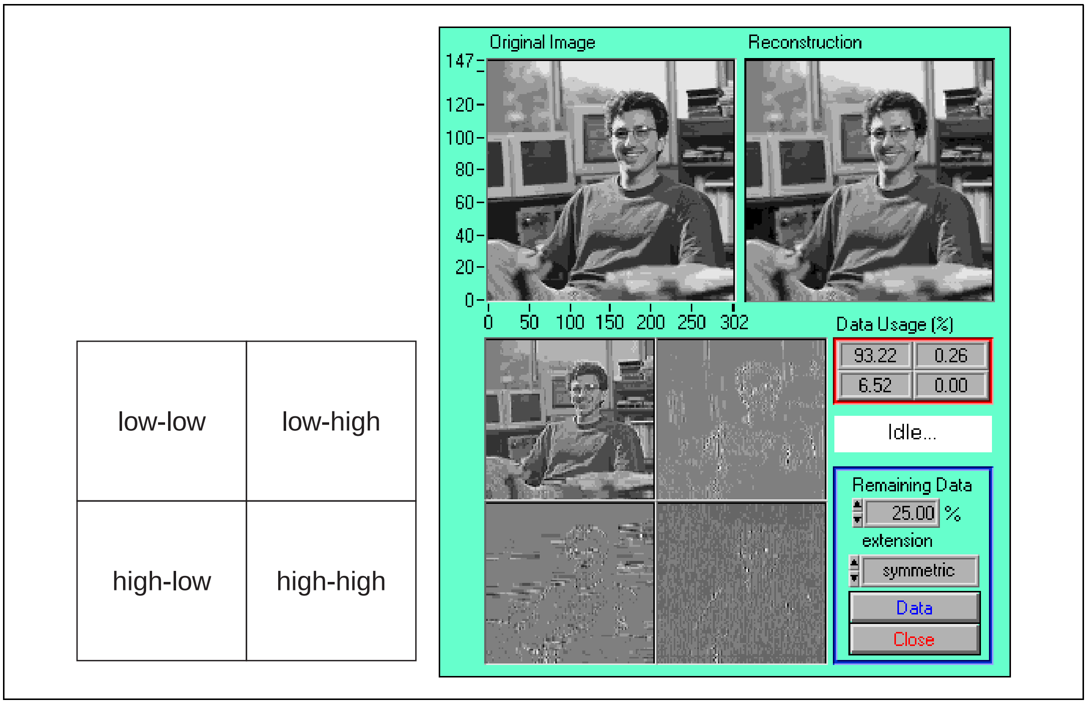


Figure 10-10. 2D Signal Processing

Figure 10-11 illustrates 2D image decomposition by two-channel PR filter banks. In this case, the original 128-by-128 2D array is decomposed into four 64-by-64 sub-arrays. The total size of the four sub-arrays is the same as the original 2D array. For example, the total number of elements in the four sub-arrays is 16,384, which equals 128 × 128. However, if you select the filters properly, you can make sub-arrays such that the majority elements are small enough to be neglected. Consequently, you can use a fraction of the entire wavelet transform coefficients to recover the original image and achieve data compression. In this example, you use the largest 25% wavelet transform coefficients to rebuild the original image. Among them, the majority, 93.22%, are from the low-low sub-array. The remaining three sub-arrays contain limited information. If you repeat the wavelet transform to the low-low sub-array, you can further reduce the compression rate.



Data Usage (%)

93.22	0.26
6.52	0.00

Remaining Data 25.00 %

extension symmetric

Figure 10-11. 2D Image Decomposition

11 Wavelet Analysis Applications

This chapter describes the 1D and 2D Wavelet Transform examples and how to design a wavelet and filter bank to meet your application needs.

With the wavelet transform examples, you can apply wavelet transform to 1D and 2D signals and design wavelets without using any programming. Although you can use this example without understanding the fundamentals of wavelets and filter banks, you should review Chapter 9, *The Fundamentals of Wavelet Analysis*, and Chapter 10, *Wavelet Analysis by Discrete Filter Banks*, before running the examples. Review those two chapters to help you attain the best results.

Accessing the Wavelet Analysis Examples

The wavelet transform examples reside on the **SPT** palette, shown in Figure 11-1. Select **Start»Programs»National Instruments» Signal Processing Toolset»NI SPT Application 6.0** to open the **SPT** palette. On the **SPT** palette, click either the **1D Wavelet Transform** icon or the **2D Wavelet Transform** icon.

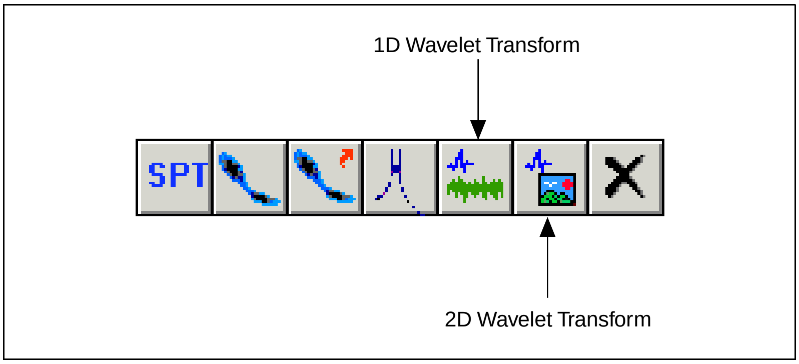


Figure 11-1. SPT Palette

If you have LabVIEW 6.0 or later installed, you can access the source code for an example by opening the front panel of the example and selecting **Windows»Show Diagram** or by pressing <Ctrl-E>.

1D Wavelet Transform

You can use the 1D Wavelet Transform example to compute the wavelet packet for 1D test data. A wavelet packet is a generalized wavelet composition. Refer to the *Specify the Tree Path* section of this chapter for more information about wavelet packets. You also can use the 1D Wavelet Transform example to test a wavelet and filter bank of your own design.

When you click the **1D Wavelet Transform** icon on the **SPT** palette, the 1D Wavelet Transform front panel, shown in Figure 11-2, and the Wavelets and Filters front panel, shown in Figure 11-3, open. Refer to the *Select the Wavelet* section of this chapter for more information about the Wavelets and Filters front panel.

The 1D Wavelet Transform front panel includes four plots. The upper plot displays the original signal. The other three plots display the wavelet transform results for different tree paths.

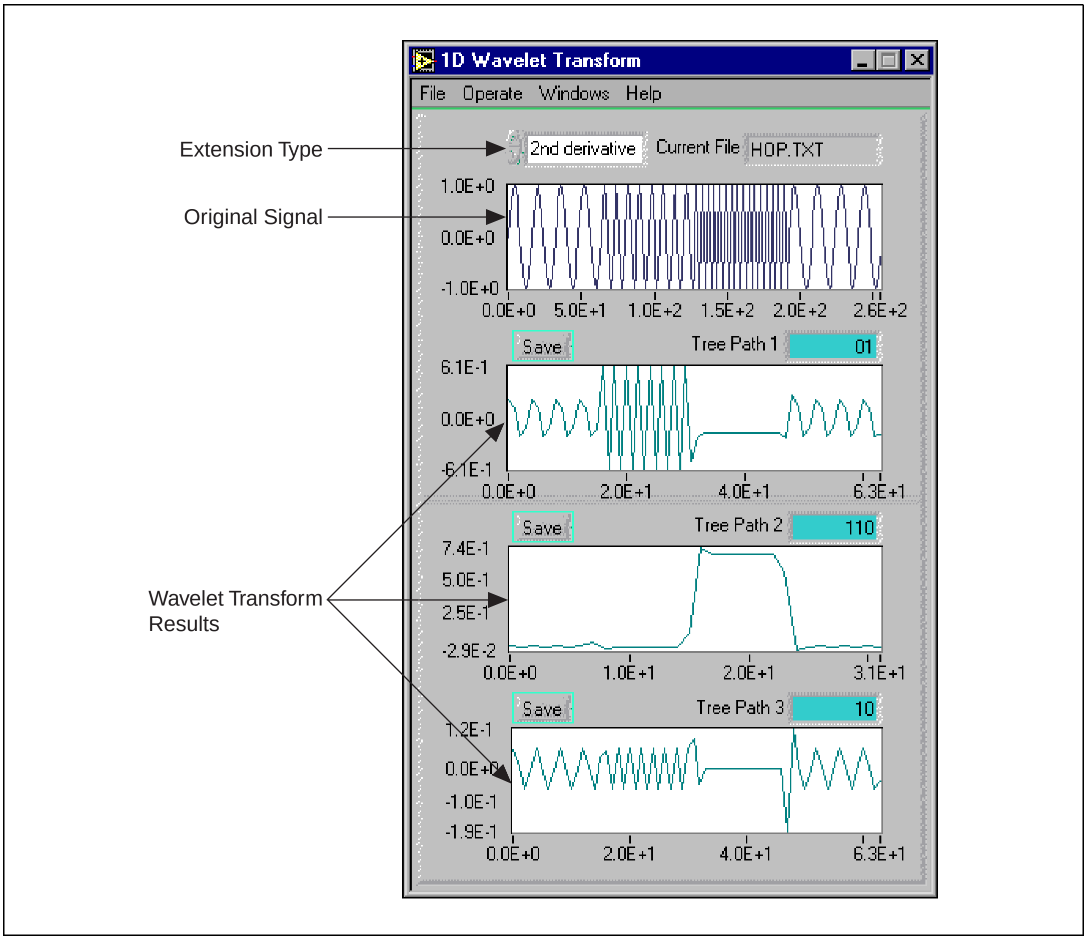


Figure 11-2. 1D Wavelet Transform Front Panel

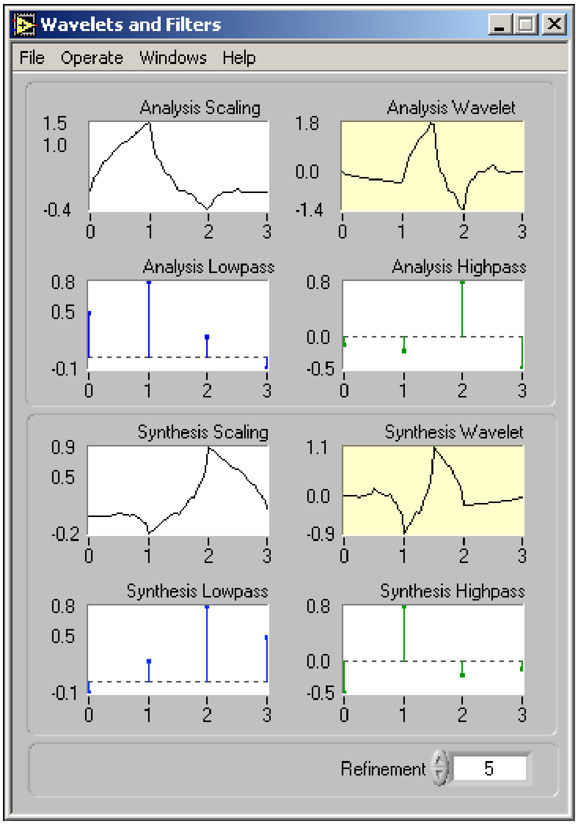


Figure 11-3. Wavelets and Filters Front Panel

Using the 1D Wavelet Transform Example

Complete the following steps to use the 1D Wavelet Transform example to compute the wavelet packet for 1D test data.

1. Load the test data.
2. Select the wavelet.
3. Specify the extension type.
4. Specify the tree path.
5. Specify the display method.
6. Save the wavelet transform results.

The following sections describe the above steps and the 1D Wavelet Transform example front panel controls and indicators. You also can select **Help»Show Context Help** or press <Ctrl-H> for more information about controls and indicators.

Load the Test Data

Select **File»Load»Data File** to open the **Choose file to read** dialog box. In the **Choose file to read** dialog box, navigate to the location of the data file you want to analyze.

You can choose to analyze your own data file or one of the example data files supplied with the Signal Processing Toolset. Your data file must be either a one-column or one-row spreadsheet text file.

Select the Wavelet

Select **File»Load»Existing Wavelets** to open the **Choose file to read** dialog box. In the **Choose file to read** dialog box, navigate to the location of the data file you want to analyze.

You can choose to use wavelets supplied with the Signal Processing Toolset or wavelets from your own data file, or you can design wavelets with the Wavelets Designer. Your wavelet file must contain the filter coefficients of $G_0(z)$, $G_1(z)$, $H_0(z)$, and $H_1(z)$ in four lines consecutively. You can create your data file with the Wavelets Designer or another application. Refer to the *Wavelets Designer* section of this chapter for information about the Wavelets Designer.

The wavelets supplied with the Signal Processing Toolset are stored with the global variables Analysis Filters and Synthesis Filters. The mother wavelet, scaling functions, and filter coefficients for the global variables are displayed by the following Wavelets and Filters front panel indicators, shown in Figure 11-3:

- Analysis Scaling displays the scaling function of the wavelet transform.
- Analysis Wavelet displays the mother wavelet of the wavelet transform.
- Analysis Lowpass displays the coefficients of the analysis lowpass filter $G_0(z)$.
- Analysis Highpass displays the coefficients of the analysis highpass filter $G_1(z)$.

- Synthesis Scaling displays the scaling function of the inverse wavelet transform.
- Synthesis Wavelet displays the mother wavelet of the inverse wavelet transform.
- Synthesis Lowpass displays the coefficients of the synthesis lowpass filter $H_0(z)$.
- Synthesis Highpass displays the coefficients of the synthesis highpass filter $H_1(z)$.

Use the **Refinement** control on the Wavelets and Filters front panel, shown in Figure 11-3, to specify how many levels to go through to compute the wavelet and scaling function. A proper wavelet usually converges after four or five levels.

Specify the Extension Type

Use **Extension Type** on the 1D Wavelet Transform front panel, shown in Figure 11-2, to specify the padding method for the data. You can choose from the following padding methods:

- zero padding adds zeros at the beginning and end of the original data.
- symmetric ext. symmetrically adds the input data at the beginning and end of the original data.
- 1st derivative adds smooth padding of order 1 at the beginning and end of the original data.
- 2nd derivative (default) adds smooth padding of order 2 at the beginning and end of the original data.

The number of points padded at either the beginning or end of the original signal is determined by the following equation:

$$\left\lfloor \frac{[Maximum(N_0, N_1) + 1]}{2} \right\rfloor$$

where N_0 is the number of coefficients of filter $G_0(z)$, and N_1 is the number of coefficients of filter $G_1(z)$.

Specify the Tree Path

Enter a value in **Tree Path** on the 1D Wavelet Transform front panel, shown in Figure 11-2, to specify a path for the wavelet packet. The **Tree Path** value must be a string composed of 0 or 1, where 0 represents passing a lowpass filter $G_0(z)$ and 1 represents passing a highpass filter $G_1(z)$. Figure 11-4 illustrates an example tree path.

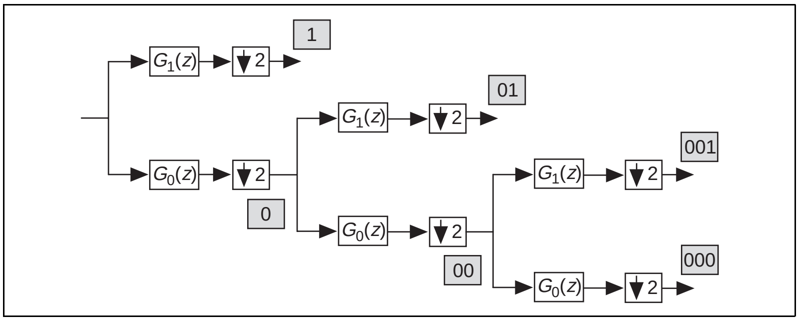


Figure 11-4. Example of Tree Path

You can define any tree path for your application. Figure 11-5 illustrates the full path for a three-level decomposition.

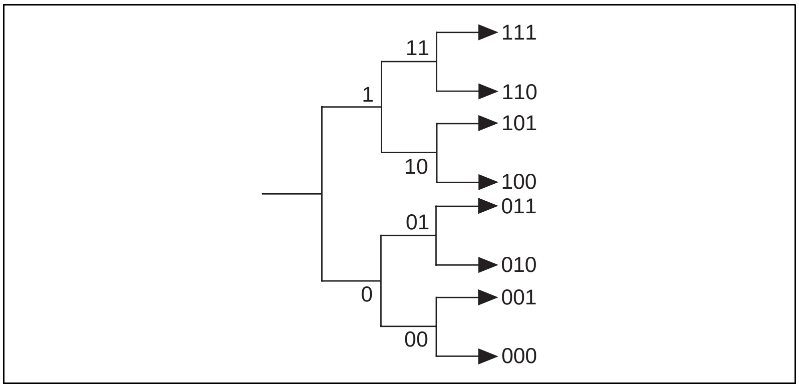


Figure 11-5. Full Path of a Three-Level Perfect Reconstruction Tree

For example, you can decompose the signal *X* as 0, 100, 101, and 11. Then, you use those coefficients to reconstruct the original signal with synthesis filter banks as shown in Figure 11-6.

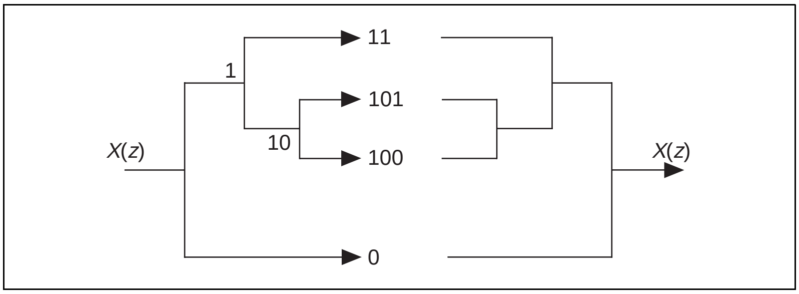


Figure 11-6. Wavelet Packet

Although, in this case, you do not follow the ordinary wavelet decomposition scheme discussed in the earlier chapters, you can still fully recover the original signal *X*, if the coefficients are not altered. This generalized wavelet decomposition is called a wavelet packet and offers a wider range of possibilities for signal processing.

You should let your application determine the tree path you specify. One common technique used to determine the tree path is called entropy-based criterion. In this technique, you check each node of the decomposition tree and quantify the information. Then, you continue to decompose those nodes that contain more information.

Specify the Display Method

The plot below **Tree Path**, as shown in Figure 11-2, displays the wavelet transform result for that path. You can display the result as either a waveform or a histogram. Select **Windows»Show Waveform** or **Show Histogram** to specify the display method.

Save the Wavelet Transform Result

Click the **Save** button above the wavelet transform result plot, as shown in Figure 11-2, to save that particular result as a text file. You can use the wavelet transform result to fully reconstruct the original signal.

2D Wavelet Transform

As discussed in Chapter 10, *Wavelet Analysis by Discrete Filter Banks*, of this manual, by applying wavelet transform, you can break one image into four subimages: low-low, low-high, high-low, and high-high. With the 2D Wavelet Transform example, you can apply 2D wavelet transform to an image and use part of the subimage data to reconstruct the image.

When you click the **2D Wavelet Transform** icon on the **SPT** palette, shown in Figure 11-1, the Wavelets and Filters front panel, shown in Figure 11-3, and the 2D Wavelet Transform front panel open. Figure 11-7 shows the 2D Wavelet Transform front panel with a data file already loaded.

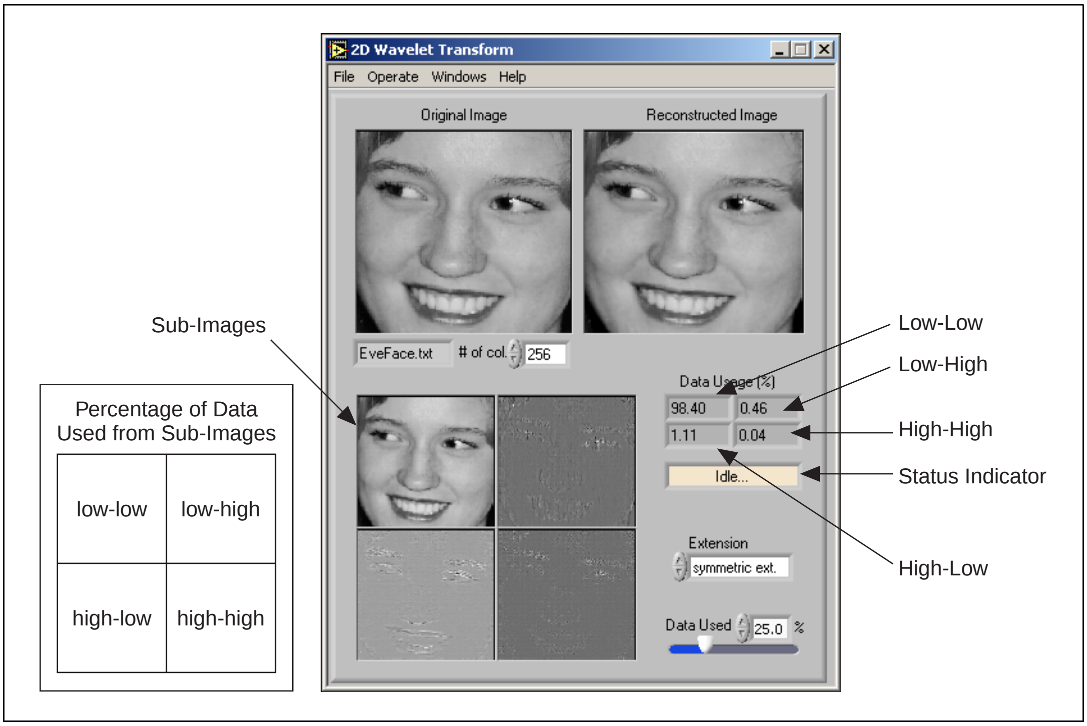


Figure 11-7. 2D Wavelet Transform Front Panel

Using the 2D Wavelet Transform Example

Complete the following steps to use the 2D Wavelet Transform example to apply 2D wavelet transform to an image and then reconstruct the image.

1. Load the image.
2. Specify the number of columns of the image.
3. Specify the extension type used for reconstruction.
4. Specify the data percentage used for reconstruction.
5. Select a wavelet.

The following sections describe the above steps and the 2D Wavelet Transform example front panel controls and indicators. You also can select **Help»Show Context Help** or press <Ctrl-H> for more information about controls and indicators.

Load the Image

Your image file can be a byte stream file of unsigned byte integer, or U8, type, a `.tif` file, or a `.bmp` file. You can load an image file in the following two ways:

- Select **File»Load»2D Image File** to open the **Choose file to read** dialog box. In the **Choose file to read** dialog box, navigate to the location of the image file you want to use.
- With no image loaded into the 2D Wavelet Transform example, change the value of **# of col.** This opens the **Choose file to read** dialog box. In the **Choose file to read** dialog box, navigate to the location of the image file you want to use.

Specify the Number of Columns of the Image

Use the **# of col.** control, shown in Figure 11-7, to specify the number of columns of the image. The value of **# of col.** must be exactly the same as the column number of the test image. Otherwise, the image cannot be displayed correctly.

Specify the Extension Type Used for Reconstruction

Use the **Extension** control, shown in Figure 11-7, to specify the padding method for the data. Refer to the *1D Wavelet Transform* section of this chapter for information about specifying an extension.

Specify the Data Percentage Used for Reconstruction

Use the **Data Used** control, shown in Figure 11-7, to specify the data percentage used for reconstruction. The value of **Data Used** represents the percentage of the largest wavelet coefficients used from all the subimages to restore the image. The Data Usage (%) indicator shows the percentage of coefficients from every subimage used to reconstruct the image.

In Figure 11-7, the original image size is 256 × 256, or 65,536 data samples. The reconstruction uses 25% of the largest wavelet transform coefficients. As shown in Data Usage (%), 98.44%, or 64,513 coefficients, are from the low-low subimage; 0.55%, or 360 coefficients, are from the low-high subimage; 1.23%, or 806 coefficients, are from the high-low subimage; and 0.05%, or 33 coefficients, are from the high-high subimage. From these figures, you can see that the low-low subimage contains more information than the others.

Select a Wavelet

Refer to the *1D Wavelet Transform* section of this chapter for information about selecting a wavelet.

Wavelets Designer

With the Wavelets Designer, you can design the filters $G_0(z)$ and $H_0(z)$, which can be used to derive $G_1(z)$ and $H_1(z)$. The mother wavelet and the scaling function can then be computed by $G_0(z)$, $G_1(z)$, $H_0(z)$, and $H_1(z)$. Refer to Chapter 10, *Wavelet Analysis by Discrete Filter Banks*, for more information about the wavelet and filter banks.

On the front panel of either the 1D or 2D Wavelet Transform example, select **File»Load»Wavelets Designer** to open the Wavelets Designer front panel, shown in Figure 11-8.

If you have LabVIEW 6.0 or later installed, you can access the source code for the Wavelets Designer by selecting **Windows»Show Diagram** or by pressing <Ctrl-E>.

Wavelets Designer Front Panel Displays

The upper plot displays the frequency response of $G_0(z)$ and $G_1(z)$.

The lower plot illustrates the zero distribution of $G_0(z)$ and $H_0(z)$. Because all the zeros are symmetrical with respect to the x-axis, only the upper half of the plane is displayed. The ° represents the zeros in $G_0(z)$, and the ×

represents the zeros in $H_0(z)$. To select a zero, place the cursor on the zero that you want to choose and click the left mouse button. This switches the zeros from $G_0(z)$ to $H_0(z)$ and vice versa. If two zeros are too close to choose, use the **Zoom Tool** palette to zoom in on the zeros until you can identify the them.

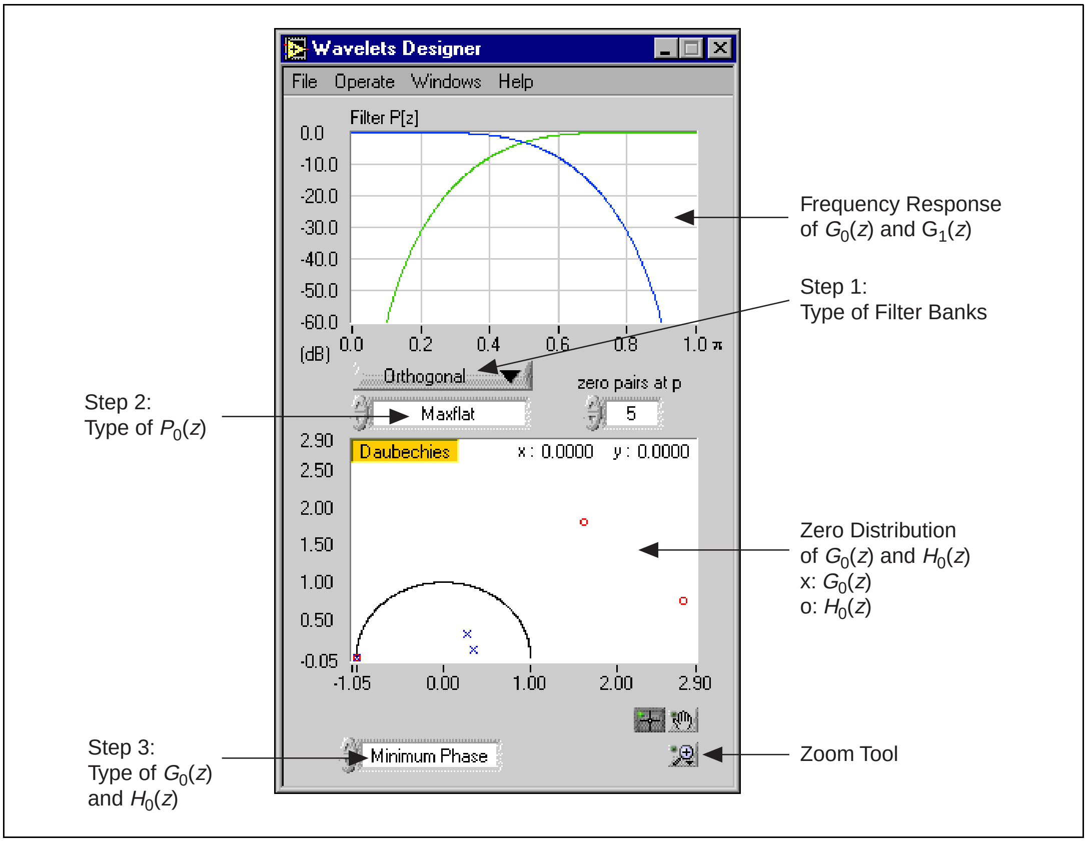


Figure 11-8. Wavelets Designer

Designing Wavelets

Figure 11-9 illustrates the wavelet design process, and Figure 11-8 shows the Wavelets Designer controls to use when designing wavelets. Complete the following steps to design wavelets.

1. Select the type of filter bank.
2. Find the product of $G_0(z)$ and $H_0(z)$, $P_0(z)$.
3. Factorize $P_0(z)$ into $G_0(z)$ and $H_0(z)$.

The following sections describe each of the steps in the wavelet design process.

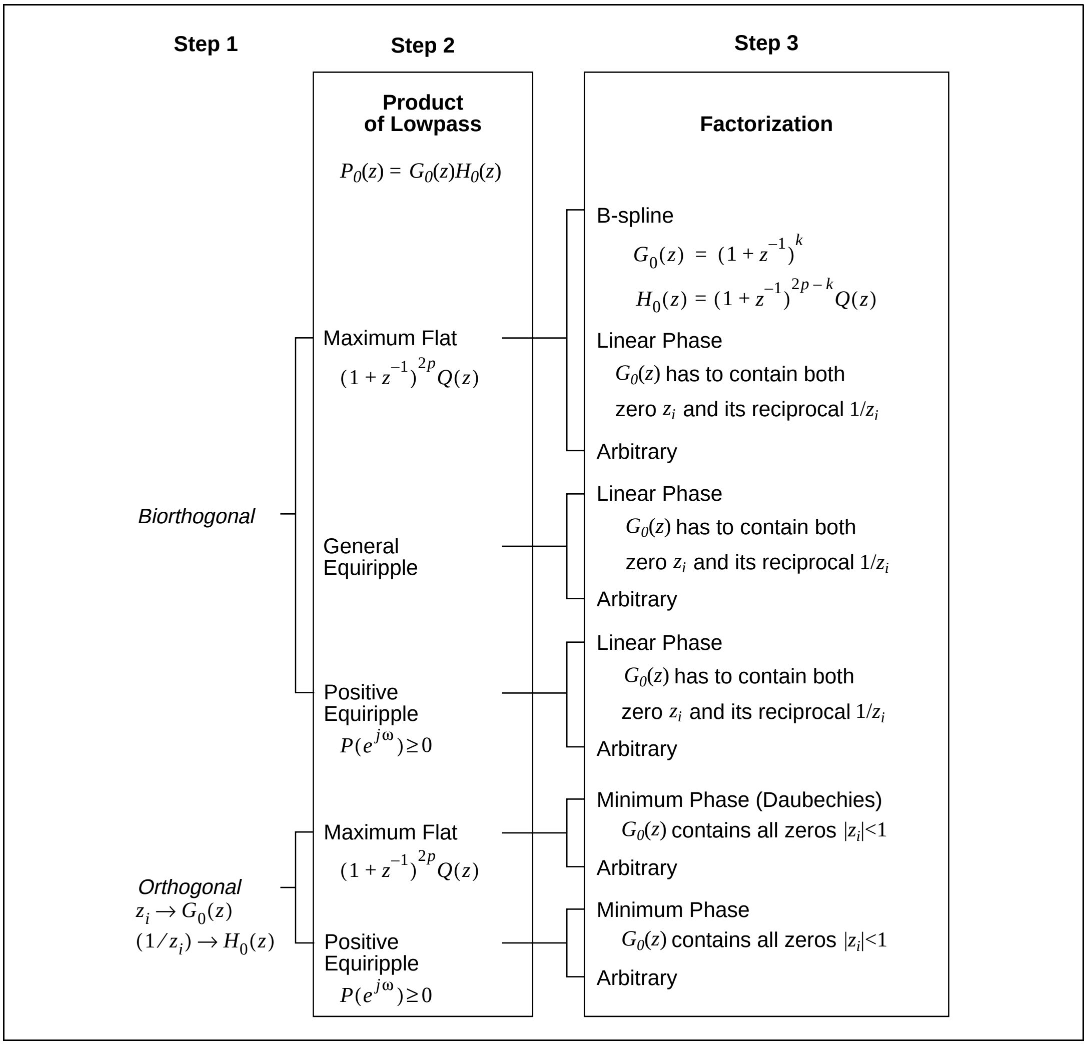


Figure 11-9. Design Procedure for Wavelets and Filter Banks

Select the Type of Filter Bank

Use the **Filter Bank** ring control, shown in Figure 11-8 as Step 1, to select the filter bank type. You can choose from two types of wavelets and filter banks, Orthogonal (default) and Biorthogonal. The orthogonal filters and wavelets are easier to design because they involve fewer parameters, but the orthogonal filter banks cannot be linear phase.

Find the Product $P_0(z)$

$P_0(z)$ denotes the product of $G_0(z)$ and $H_0(z)$ as shown in the following equation:

$$P_0(z) = G_0(z)H_0(z)$$

Use the **P_0(z)** control, shown in Figure 11-8 as Step 2, to specify the $P_0(z)$ type. When **Filter Bank** is set to Orthogonal, you can set **P_0(z)** to either Maxflat (default), for a maximum flat filter, or to Positive Equiripple. When **Filter Bank** is set to Biorthogonal, you can set **P_0(z)** to Maxflat (default), General Equiripple, or Positive Equiripple.

Because all filters in the Wavelets Designer act as real-valued finite impulse response (FIR) filters, the zeros of $P_0(z)$, $G_0(z)$, and $H_0(z)$ are symmetrical in the z-plane. This implies that for any zero z_i, there always exists z_i*. If z_i is complex, as shown in Figure 11-10, you only need to deal with half of the z-plane. Once you select z_i, the Wavelets Designer automatically includes its complex conjugate z_i*.

The maximum flat filters has the form

$$P_0(z) = (1 + z^{-1})^{2p}Q(z).$$

It differs from the Butterworth filter. The maximum flat filters have good frequency attenuation but wider transition band. The parameter p is controlled by the **zero pairs at** π control, as shown in Figure 11-12. $Q(z)$ is a $2p - 2$ order polynomial, which you can uniquely determine if p is decided. Therefore, the total number of coefficients of $P_0(z)$ is $4p - 1$. For maximum flat filters, there are multiple zeros at $z = 0$. Use the **zeros at** π control to determine how many zeros at $z = 0$.

The equiripple is further divided into the general equiripple and positive equiripple filters. However, you can select only positive equiripple filters for orthogonal filter banks. Although both are halfband filters, the sum of the normalized passband and stopband frequencies equals 0.5, the Fourier transform of the positive equiripple filter $p_0[n]$ is always real and non-negative, as shown in Figure 11-10.

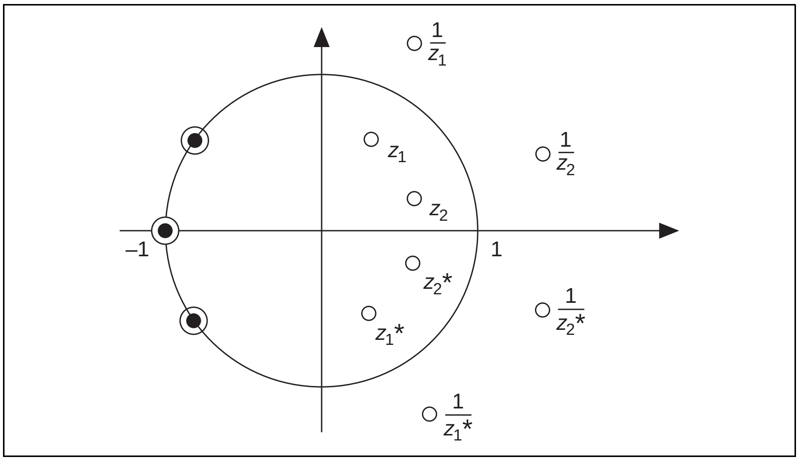


Figure 11-10. Non-Negative Equiripple Halfband Filter

There are two parameters associated with equiripple filters, **# of taps** and **passband**, as shown in Figure 11-11. Use the **# of taps** control to define the number of coefficients of $P_0(z)$. Because $P_0(z)$ is a type I FIR filter, the length of $P_0(z)$ must be $4p - 1$, where $p = 2,3,\ldots$. Use the **passband** control to define the normalized cutoff frequency of $P_0(z)$, which must be less than 0.5.

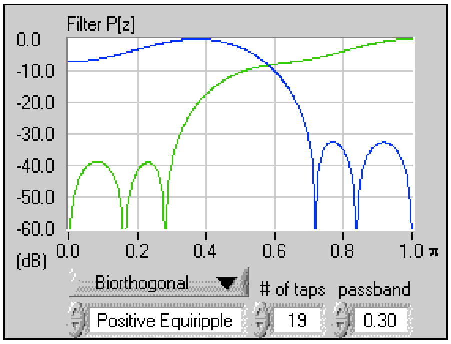


Figure 11-11. Equiripple Filter

Factorize $P_0(z)$ into $G_0(z)$ and $H_0(z)$

Once you determine $P_0(z)$, you must factorize it into the lowpass filters, $G_0(z)$ and $H_0(z)$. Use **Type of G0(z) and *H*0(z)**, shown in Figure 11-8 as Step 3, to accomplish the factorization. The combination of $G_0(z)$ and $H_0(z)$ is not unique. For a given $P_0(z)$, you have the following four choices for $G_0(z)$ and $H_0(z)$:

- Linear Phase—Any zero and its reciprocal must belong to the same filter as shown in Figure 11-12.
- Minimum Phase—$G_0(z)$ contains all the zeros inside the unit circle as shown in Figure 11-13. When $P_0(z)$ is maximum flat and $G_0(z)$ is minimum phase, the resulting wavelets are traditionally known as Daubechies wavelets.
- B-Spline—This choice is only available when the filter is biorthogonal and maximum flat. In this case

  $$G_0(z) = (1+z^{-1})^k \qquad H_0(z) = (1+z^{-1})^{2p-k}Q(z)$$

  where k is specified with the **zeros at** π control. p is decided by the **zeros at** π control, as mentioned in the *Find the Product P0(z)* section of this chapter. Figure 11-14 shows an example of B-Spline factorization.
- Arbitrary—No specific constraints are associated with this filter. Figure 11-15 shows an example of arbitrary factorization.

After you decide the type of $G_0(z)$ and $H_0(z)$, the Wavelets Designer automatically computes the constraints. For example, once you select a zero, the reciprocal of the zero is automatically included if you choose $G_0(z)$ for linear phase. Figure 11-9 summarizes all possible design combinations provided by the Wavelets Designer.

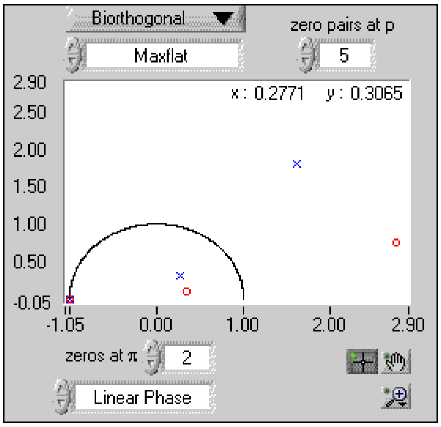


Figure 11-12. Linear Phase Filter

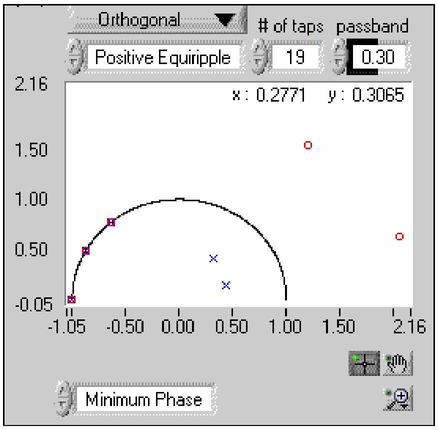


Figure 11-13. Minimum Phase Filter

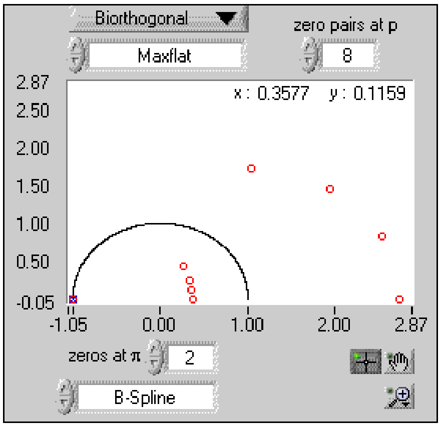


Figure 11-14. B-Spline Filter

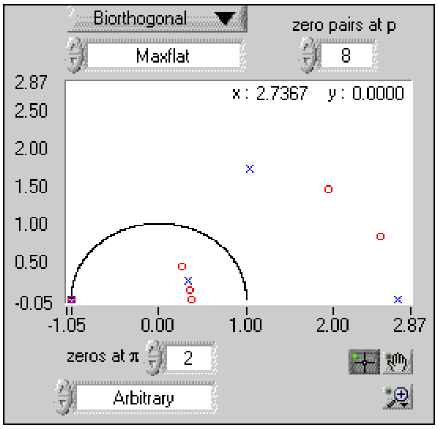


Figure 11-15. Arbitrary Filter

Note The conditions for linear phase and orthogonality are contradictory. In general, you cannot achieve linear phase and orthogonality simultaneously.

The **Wavelets Designer** also provides the following additional utilities:

- Select **File»Load»Design Spec** to load a saved design file.
- Select **File»Save»Wavelet** to save the designed analysis filter coefficients and synthesis filter coefficients in a text file.
- Select **File»Save»Design Spec** to save your design information in a binary file.

- Select **Windows»Show Filter Coef** to display a table listing the designed analysis and synthesis filter coefficients.
- Select **Windows»Show Wavelets** to open the Wavelets and Filters front panel, shown in Figure 11-3. Refer to the *1D Wavelet Transform* section of this chapter for information about the Wavelets and Filters front panel.

To assist you with testing your own applications, the Wavelets Designer saves the filter coefficients as the following global variables in the Wavelet Global VI:

- Analysis Filter Coefficients contains coefficients $G_0(z)$ and $G_1(z)$.
- Synthesis Filter Coefficients contains coefficients $H_0(z)$ and $H_1(z)$.

These variables simultaneously change as you change the design. If you incorporate those parameters into your own application, you can see the effect of the different design.

Part V

Digital Filter Design

This section of the manual describes the Digital Filter Design (DFD) application. Refer to the works of Jackson [13], Oppenheim and Schafer [19], Parks and Burrus [20], Parks and McClellan [21] [22], and Williams and Taylor [37] for more information about the theory and algorithms implemented in the DFD application.

- Chapter 12, *Digital Filter Design Application*, describes the DFD application used to design infinite impulse response (IIR) and finite impulse response (FIR) digital filters.
- Chapter 13, *IIR and FIR Implementation*, describes the filter implementation equations for IIR and FIR filtering and the format of the IIR and FIR filter coefficient files.

12 Digital Filter Design Application

This chapter describes the Digital Filter Design (DFD) application, including all required filter coefficient forms and implementation equations. The DFD application provides complete filter design and analysis tools you can use to design digital filters to meet your precise filter specifications. You can graphically design impulse response (IIR) and finite impulse response (FIR) digital filters, interactively review filter responses, save your filter design work, and load your design work from previous sessions.

You can save digital-filter coefficients for later implementation from within LabVIEW and LabWindows/CVI. Also, you can call Windows DFD dynamic link libraries (DLLs) from other applications, or other applications can load the filter-coefficient files directly.

If you have a National Instruments data acquisition (DAQ) device, you can perform real-world filter testing in the DFD application. You can view the time waveforms or the spectra of the input signal and the filtered signal while you simultaneously redesign your digital filters.

Figure 12-1 illustrates the interaction between the DFD application and related applications.

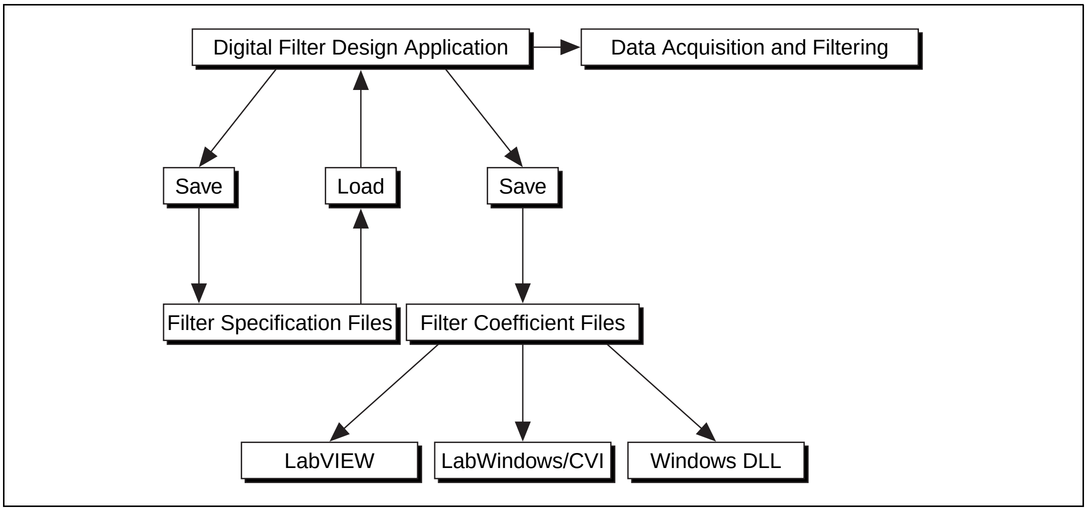

Digital Filter Design Application
Data Acquisition and Filtering
Save
Load
Save
Filter Specification Files
Filter Coefficient Files
LabVIEW
LabWindows/CVI
Windows DLL

Figure 12-1. Conceptual Overview of the Digital Filter Design Application

Main Menu

Select **Start»Programs»National Instruments»Signal Processing Toolset»NI Digital Filter Design 6.0** to open the **Main Menu** dialog box, shown in Figure 12-2.

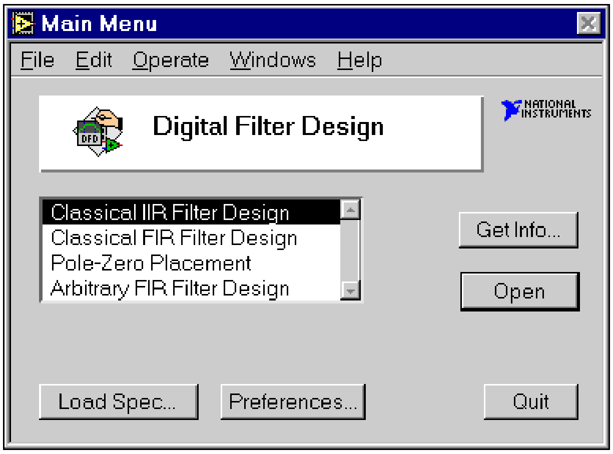


Figure 12-2. DFD Main Menu

Opening the Filter Design Front Panels

From the **Main Menu** dialog box, you can open any one of the following digital filter design front panels:

- Classical IIR Filter Design
- Classical FIR Filter Design
- Pole-Zero Placement
- Arbitrary FIR Filter Design.

Refer to the *Digital Filter Design Front Panels* section later in this chapter for more information about each design front panel.

Directly Loading a Filter Specification File

You can load a previously designed filter specification file directly from the **Main Menu** dialog box. Click the **Load Spec** button. The DFD application prompts you to select the filter specification file that you saved during previous design work. After you select the file, you can open the appropriate design front panel for that specification file. Then, resume work on an ongoing design project.

Editing the DFD Preferences

Click the **Preferences** button on the **Main Menu** dialog box to customize your DFD application preferences. You can edit your DFD application preferences for future design sessions.

Quitting the DFD Application

Click the **Quit** button in the **Main Menu** dialog box to quit the DFD application.

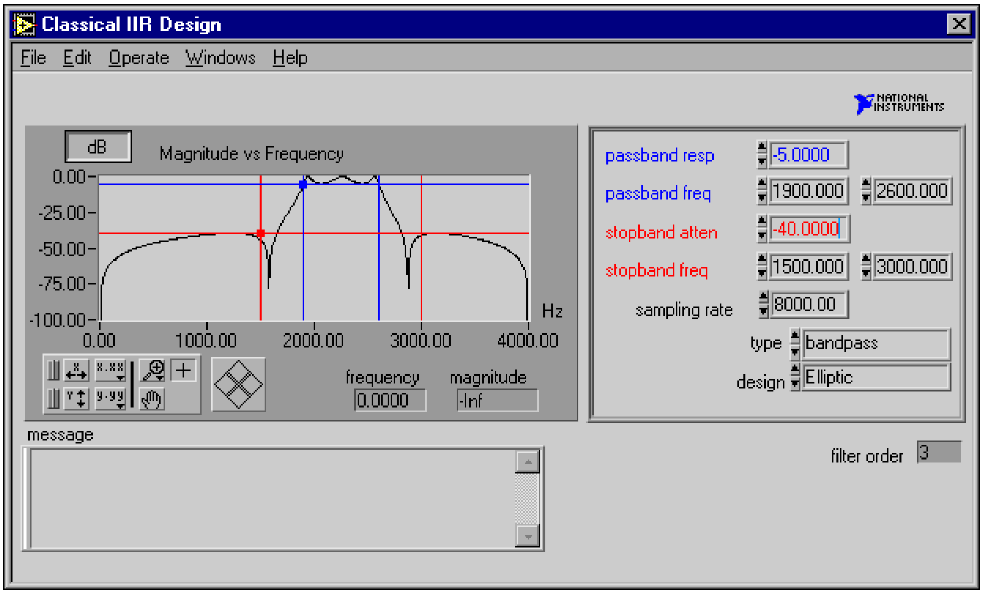
Digital Filter Design Front Panels

When you double-click one of the four design selections in the **Main Menu** dialog box, the DFD application loads and runs the selected design front panel. You can use these design front panels to design IIR or FIR filters, save your design work and filter coefficients, or load previous filter designs.

After designing your filter, you can move from the design front panels to the Analysis of Filter Design front panel to view various frequency domain and time domain filter responses. You can save these responses to text files for use in other applications. You also can perform real-world testing of your filter designs by moving to the DAQ and Filter front panel, which performs data acquisition and filtering in parallel with your filter designing. Refer to the *Analysis of Filter Design Front Panel* and the *DAQ and Filter Front Panel* sections of this chapter for more information about these two front panels.

Common Controls and Features

The following sections describe the controls and features of the DFD application. Figure 12-3 shows the Classical IIR Filter Design front panel, which is representative of the filter design front panels.



frequency 0.0000
magnitude -Inf
passband resp -5.0000
passband freq 1900.000 2600.000
stopband atten -40.0000
stopband freq 1500.000 3000.000
sampling rate 8000.00
type bandpass
design Elliptic
filter order 3

Figure 12-3. Classical IIR Filter Design Front Panel

Saving Filter Specifications

Select **File»Save Spec** to save all your specifications for the present filter design front panel. The DFD application prompts you for the name of the filter-specification file to save. Name your specification files appropriately for a given filter design. For example, if you design a lowpass IIR filter, name the file `lowpass.iir` or `lowp1.iir` if this design is the first of many lowpass IIR designs.

Table 12-1 lists suggested filename extensions for the four filter design front panels. These names have no effect on how the DFD application interprets the file contents.

Table 12-1. Suggested DFD Filename Extensions

Design Front Panel	Filename
Classical IIR Design	*filename*.iir
Classical FIR Design	*filename*.fir
Pole-Zero Placement	*filename*.pz
Arbitrary FIR Design	*filename*.arb

Loading Filter Specifications

Select **File»Load Spec** to load a filter specification file into the present filter design front panel. The DFD application prompts you for the location of the filter specification file to load. If the selected specification file is the same type design as the present design front panel, the DFD application loads the specification from the selected file into the present design front panel for viewing, editing, or analysis.

If you designed the selected specification file in a different design front panel than the present front panel, the DFD application prompts you to open the appropriate design front panel for that specification file. For example, if you are using the Pole-Zero Placement front panel and you load a specification file saved in the Classical FIR Design front panel, the DFD application prompts you to open the Classical FIR Design front panel to resume work on the loaded filter specifications.

Saving Filter Coefficients

Select **File»Save Coeff** to save your filter coefficients to a file. The DFD application first prompts you for the format, either text or log, of the coefficient file.

Click the **text** button to select the text format. The text format allows you to view or print the coefficient file or to use the coefficients in other non-LabVIEW filtering applications.

Click the **log** button to select the log format for LabVIEW-only filtering applications. However, LabVIEW filtering utilities read both text-formatted and log-formatted coefficient files.

After you select the format of the coefficient file, the DFD application prompts you for the name of the filter coefficient file to save. Name your coefficient files appropriately for a given filter design. For example, if you save bandpass IIR filter coefficients, name the file `bpiir.txt` or `bpiir.log`, depending on the coefficient file type.

Analyzing Filter Designs

Select **File»Analysis** to analyze your filter design. The DFD application loads and runs the Analysis of Filter Design front panel. From this analysis front panel, you can view the filter magnitude response, phase response, impulse response, step response, and pole-zero plot. You also can view and print full-screen plots of each response. From the full-screen views, you can save the analysis results to a text file. Refer to the *Analysis of Filter Design Front Panel* section of this chapter for more information about analyzing filter designs.

Testing Filter Designs

If you have a National Instruments DAQ device, you can test the present filter design on real-world signals by selecting **File»DAQ and Filter**. The DFD application loads and runs the DAQ and Filter front panel. From this front panel, you can configure your DAQ device and then acquire real signals. The acquired data passes through the currently designed filter, and the DFD application plots the input and output waveforms and spectrums.

You also can test your filter designs using a built-in simulated function generator. Select **File»DAQ and Filter** and configure the DAQ source to simulated DAQ. You then can click the **Function Generator** button on the DAQ and Filter front panel to view and edit settings that include signal type, frequency, amplitude, and noise level.

Refer to the *DAQ and Filter Front Panel* section of this chapter for more information about the DAQ and Filter front panel.

Transferring Filter Designs

You can transfer some filter design specifications from one design front panel to another. For example, you can configure your passband and stopband requirements while you design an FIR filter and find the IIR filter that meets your design specifications. Not all design front panels can share specifications. Table 12-2 shows the transfers you can perform and the corresponding command path.

Table 12-2. Filter Specification Transfers

Design Transfer	Command Path
Filter specs from the Classical IIR to the Classical FIR	**File»Xfer Classical FIR**
Filter specs from the Classical FIR to the Classical IIR	**File»Xfer Classical IIR**
Poles and zeros from Classical IIR to the Pole-Zero Placement	**File»Xfer Pole Zero**

Returning to the Main Menu

Select **File»Main Menu** to return to the **Main Menu** dialog box.

Panning and Zooming Options

The graphs on the filter design front panels include the **Graph** palette. Use the controls on this palette to pan the display area of a graph and zoom in and out of graph sections. Figure 12-4 shows a graph and its **Graph** palette.

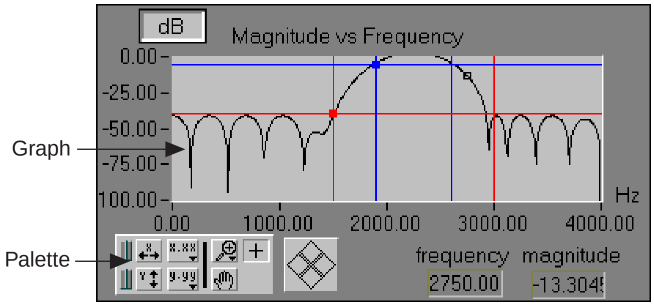


Figure 12-4. Example of Graph Palette

Autoscale X Button

Click the **Autoscale X** button to autoscale the x-data of the graph.

Autoscale Y Button

Click the **Autoscale Y** button to autoscale the y-data of the graph.

Lock Switch

Click the lock switch next to the desired autoscale button to autoscale that particular scale continuously.

Scale Format Buttons

The **Scale Format** buttons give you run-time control over the format of the x-scale and y-scale markers. When you click a **Scale Format** button, a menu appears. From the shortcut menu, you can choose the **Format**, **Precision**, and **Mapping Mode** of the graph.

Zoom Tool

Click the **Zoom Tool** to open the **Zoom** palette, shown in Figure 12-5. After choosing your zooming method, you can zoom in on a section of the graph by dragging a selection rectangle around that section.

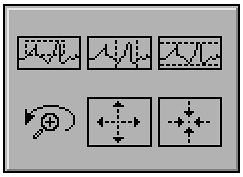


Figure 12-5. Zoom Palette

The top row of the **Zoom** palette contains the following buttons from left to right:

- **Zoom by rectangle**
- **Zoom by rectangle, with zooming restricted to x-data**
- **Zoom by rectangle, with zooming restricted to y-data**

The bottom row of the **Zoom** palette contains the following buttons from left to right:

- **Undo last zoom** resets the graph to its previous setting.
- **Zoom in about a point** allows you to zoom in on a specific point. If you click and hold the mouse button on a specific point, the graph continuously zooms in until you release the mouse button. Shift-click to zoom in the opposite direction.
- **Zoom out about a point** allows you to zoom out on a specific point. If you click and hold the mouse button on a specific point, the graph continuously zooms out until you release the mouse button. Shift-click to zoom in the opposite direction.

Panning Tool

Click the **Panning Tool**, represented by the hand icon in Figure 12-4, to switch to a mode in which you can scroll the visible data by clicking and dragging sections of the graph.

Plus/Crosshatch Button

Click the **Plus** button. This puts the graph in operate mode. In operate mode, you can click in the graph to move the cursors.

Graph Cursors

The graphs on the Classical IIR and Classical FIR design front panels have movable cursors. Figure 12-6 shows two cursors on a graph. When the graph is in operate mode, you can move a cursor by clicking on it and dragging it with the **Operating Tool**.

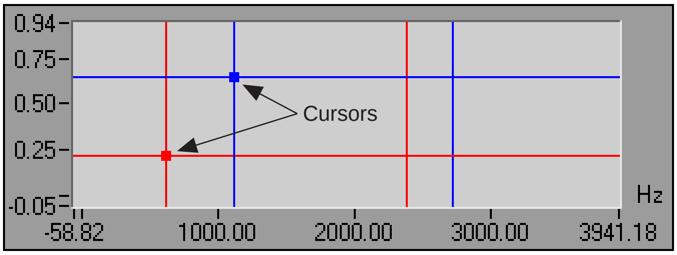
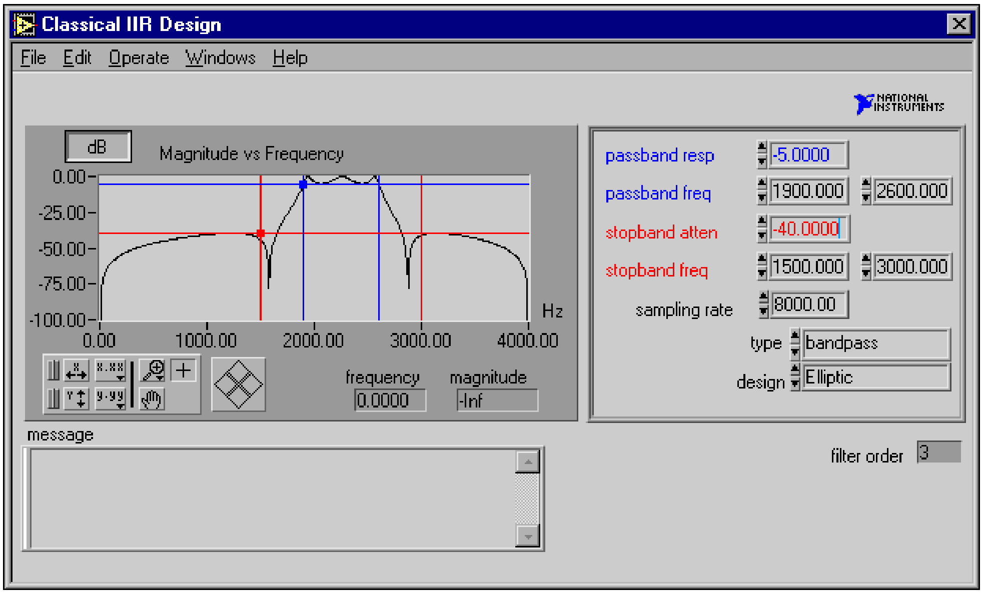


Figure 12-6. Example of Two Cursors on a Graph

You also can click the direction diamonds on the **Cursor Movement** control to move selected cursors in the specified direction. You select cursors by first moving them on the graph with the **Operating Tool**.

Classical IIR Filter Design

Figure 12-7 shows the Classical IIR Design front panel. This front panel includes a graphical interface with the Magnitude vs. Frequency cursors and plot on the left side and a text-based interface with digital controls on the right side.



frequency 0.0000
magnitude -Inf

passband resp -5.0000
passband freq 1900.000 2600.000
stopband atten -40.0000
stopband freq 1500.000 3000.000
sampling rate 8000.00
type bandpass
design Elliptic
filter order 3

Figure 12-7. Classical IIR Design Front Panel

Use this front panel to design classical IIR digital filters. These filters include the classic types of lowpass, highpass, bandpass, and bandstop as well as the classic designs of Butterworth, Chebyshev, Inverse Chebyshev, and Elliptic.

To design classical IIR filters, adjust the filter specifications. The passband and stopband requirements define a filter specification. You can define these requirements by using either the text-based interface or the cursors in the Magnitude vs. Frequency graph. As you use the mouse to click and drag the cursors, the panel updates the text-based entries. Similarly, as you enter new specifications in the text-base interface, the panel updates cursors.

The lower passband frequency fp_1, upper passband frequency fp_2, and the passband response Gp define the passband specification. For the bandpass filter, the passband ranges from fp_1 to fp_2. The passband is the region in the frequency domain with a response near 1.0. Gp is the minimum allowable passband gain or filter magnitude response. In Figure 12-7, the passband is specified as having a minimum gain of –5 dB between the frequencies of fp_1 = 1900 Hz and fp_2 = 2600 Hz.

The following ranges define the passband:

lowpass	$0 \le f \le fp_1$
highpass	$fp_1 \le f \le f_{samp}/2$
bandpass	$fp_1 \le f \le fp_2$
bandstop	$0 \le f \le fp_1$, $fp_2 \le f \le f_{samp}/2$

where	fp_1 is passband frequency 1
	fp_2 is passband frequency 2
	f_{samp} is the sampling rate

The lower stopband frequencies fs_1 and fs_2 and the stopband attenuation *Gs* define the stopband specification. For the bandpass filter, the stopband ranges from 0.0, DC to the lower stopband frequency fs_1, and from the upper stopband frequency fs_2 to half of the sampling rate, or the Nyquist rate. The stopband is the region in the frequency domain with a response near 0.0. *Gs* is the minimum acceptable stopband attenuation or filter magnitude response.

In Figure 12-7, the stopband specification has a minimum attenuation of –40 dB between the frequencies of 0 and fs_1 = 1500 Hz and between the frequencies of fs_2 = 3000 Hz and 4000 Hz.

The following ranges define the stopband:

lowpass	$fs_1 \le f \le f_{samp}/2$
highpass	$0 \le f \le fs_1$
bandpass	$0 \le f \le fs_1$, $fs_2 \le f \le f_{samp}/2$
bandstop	$fs_1 \le f \le 2$

where	fs_1 is passband frequency 1
	fs_2 is passband frequency 2
	f_{samp} is the sampling rate

The Classical IIR Design front panel estimates the minimum filter order required for the selected type and design to meet or exceed the modified filter specifications. The DFD application automatically computes other appropriate filter parameters and designs, and plots the IIR filter. You see immediate graphical feedback to help you determine whether the filter meets your specifications.

Classical IIR Design Front Panel Controls and Displays

Use the design front panel **File** menu to complete the following tasks:

- Save your filter specifications and coefficients.
- Load filter designs from previous work.
- Open the Analysis or DAQ and Filter front panels.
- Transfer the IIR design specifications to the Classical FIR Design front panel.
- Transfer the poles and zeros to the Pole-Zero Placement front panel.
- Return to the **Main Menu** dialog box.

Magnitude-Frequency Plot

The graph in Figure 12-7 plots the frequency response $H(f)$ as magnitude of the designed digital filter. The magnitude, y-axis, is in linear or decibel units, depending on how you set the button in the upper-left corner of the graph. The frequency, x-axis, is in hertz. The full scale ranges from 0.0 to Nyquist, or half the sampling rate.

Cursors

When you move the blue cursors, you control the passband response, or horizontal lines, and the passband frequencies, or vertical lines.

When you move the red cursors, you control the stopband attenuation, or horizontal lines, and the stopband frequencies, or vertical lines.

These cursors represent the filter design specifications for the selected classical IIR filter. In the passband, the filter has a gain greater than or equal to the specified passband response. In the stopband, the filter has a gain less than or equal to the specified stopband attenuation.

Linear/dB Button

Use the **linear/dB** button to control the display units, either linear or dB, of all magnitude and gain controls and displays. These controls and displays include Magnitude vs. Frequency plot (*y*-axis), passband response, stopband attenuation, and tracking cursor magnitude.

Frequency and Magnitude Indicators

The frequency and magnitude indicators display the location of the square, transparent tracking cursor. This cursor is locked to the frequency response $H(f)$, so moving this cursor updates the frequency and magnitude digital displays with data points from $H(f)$.

Text-Based Interface

You can enter the complete filter specifications using the text-based portion of the design front panel. Refer to the right-hand side of Figure 12-7 to see the text-based interface.

Use **passband resp** to define the minimum gain in the passband. The horizontal blue cursor line represents this response in the Magnitude vs. Frequency plot.

In the passband, the filter gain is guaranteed to be at least as high as the specified passband response (Gp). That is, $|H(f)| \geq Gp$.

When **type** is set to **bandpass** or **bandstop**, use the first value in **passband freq** to define one frequency edge of the passband. The first vertical blue cursor line represents this frequency in the Magnitude vs. Frequency plot. Use the second value in **passband freq** to define the second frequency edge of the passband. The second vertical blue cursor line represents this frequency in the Magnitude vs. Frequency plot.

Use **stopband atten** to define the minimum attenuation in the stopband. The horizontal red cursor line represents this attenuation in the Magnitude vs. Frequency plot.

In the stopband, the filter gain is guaranteed to be no higher than the specified stopband attenuation (Gs). That is, $|H(f)| \leq Gs$.

When **type** is set to **bandpass** or **bandstop**, use the first value in **stopband freq** to define one frequency edge of the stopband. The first vertical red cursor line represents this frequency in the Magnitude vs. Frequency plot. Use the second value in **stopband freq** to define the second frequency edge of the stopband. The second vertical red cursor line represents this frequency in the Magnitude vs. Frequency plot.

Sampling Rate Control

Use the **sampling rate** control to specify the sampling rate in samples per second, hertz.

Type Control

Use the **type** control to specify one of the following filter types:

- **lowpass**
- **highpass**
- **bandpass**
- **bandstop**

Design Control

Use the **design** control to specify one of the following filter design algorithms:

- Butterworth
- Chebyshev
- Inverse Chebyshev
- Elliptic

Filter Order Indicator

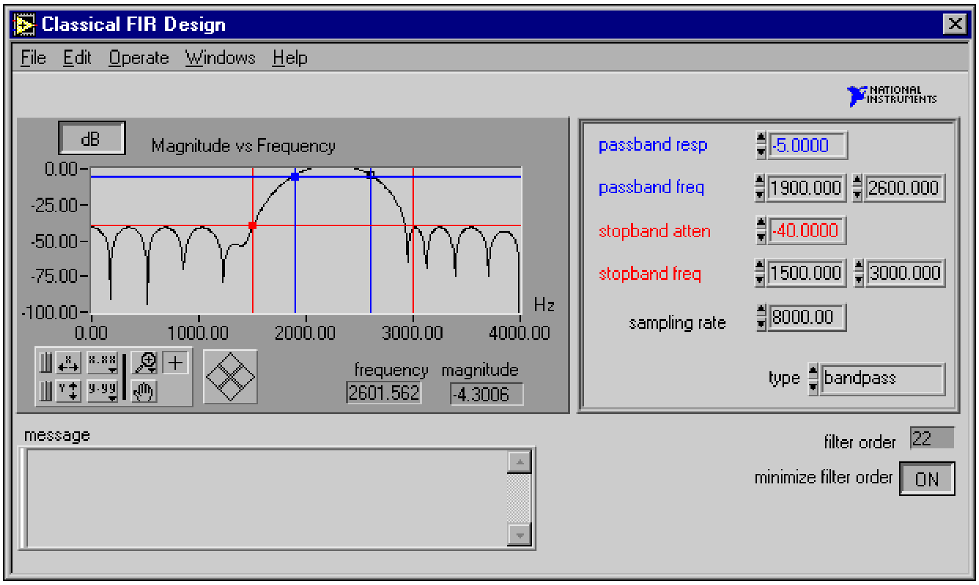
The filter order indicator displays the estimated filter order of the classical IIR filter. The DFD application automatically estimates the filter order as the lowest possible order that meets or exceeds the desired filter specifications.

Message Window

The **message** window displays errors that occur during the IIR design procedure. These errors occur when the filter specifications are inconsistent with the chosen filter type.

Classical FIR Design

Figure 12-8 shows the Classical FIR Design front panel. This front panel functions similarly to the Classical IIR Design front panel. The front panel includes a graphical interface with the Magnitude vs. Frequency cursors and plot on the left side and a text-based interface with digital controls on the right side.



frequency 2601.562
magnitude -4.3006
passband resp -5.0000
passband freq 1900.000 2600.000
stopband atten -40.0000
stopband freq 1500.000 3000.000
sampling rate 8000.00
type bandpass
message
filter order 22
minimize filter order ON

Figure 12-8. Classical FIR Design Front Panel

Use the Classical FIR Design front panel to design classical FIR digital filters. These filters include the classic types of lowpass, highpass, bandpass, and bandstop and use the Parks-McClellan equiripple FIR filter design algorithm.

To design classical FIR filters, adjust the desired filter specifications. The passband and stopband requirements define a filter specification. You can define these requirements by using either the text-based interface or the cursors in the Magnitude vs. Frequency graph. As you use the mouse to click and drag the cursors, the text entries update. Similarly, as you enter new specifications in the text-based interface, the cursors update.

The lower passband frequency fp_1, upper passband frequency fp_2, and the passband response Gp define the passband specification. For the bandpass filter, the passband ranges from fp_1 to fp_2. The passband is the region in the frequency domain with a response near 1.0. *Gp* is the minimum allowable passband gain or filter magnitude response.

In Figure 12-8, the passband specification is a minimum gain of –5 dB between the frequencies of fp_1 = 1900 Hz and fp_2 = 2600 Hz.

The following ranges define the passband:

lowpass	$0 \le f \le fp_1$
highpass	$fp_1 \le f \le f_{samp}/2$
bandpass	$fp_1 \le f \le fp_2$
bandstop	$0 \le f \le fp_1\ ,\ fp_2 \le f \le f_{samp}/2$

where	fp_1 is passband frequency 1
	fp_2 is passband frequency 2
	f_{samp} is the sampling rate

The stopband frequencies fs_1 and fs_2 and the stopband attenuation *Gs* define the stopband specification. For the bandpass filter, the stopband ranges from 0.0, DC to the lower stopband frequency $fs_{1,}$ and from the upper stopband frequency fs_2 to half of the sampling rate, or Nyquist.
The stopband is the region in the frequency domain with a response near 0.0. *Gs* is the minimum acceptable stopband attenuation or filter magnitude response.

In Figure 12-8, the stopband specification has a minimum attenuation of –40 dB between the frequencies of 0 and fs_1 = 1500 Hz and between the frequencies of fs_2 = 3000 Hz and 4000 Hz.

The following ranges define the stopband:

lowpass	$fs_1 \le f \le f_{samp}/2$
highpass	$0 \le f \le fs_1$
bandpass	$0 \le f \le fs_1\ ,\ fs_2 \le f \le f_{samp}/2$
bandstop	$fs_1 \le f \le 2$

where	fs_1 is passband frequency 1
	fs_2 is passband frequency 2
	fp_{samp} is the sampling rate

The Classical FIR Design front panel estimates the minimal filter order required for the selected type and design to meet or exceed the modified filter specifications. The DFD application automatically computes other appropriate filter parameters and designs, and plots the FIR filter. You see immediate graphical feedback to help you determine whether the filter meets your specifications.

Classical FIR Design Front Panel Controls and Displays

The Classical FIR Design front panel controls and displays are similar to those on the Classical IIR Design front panel but with two exceptions. The Classical FIR Design front panel has a **minimize filter order** button but does not have a **design** control.

Use the **minimize filter order** button to specify whether the DFD application minimizes the estimated filter order. If this button is **OFF**, the DFD application uses a fast formula to estimate the filter order to meet or exceed the desired filter specifications. If this button is **ON**, the DFD application iteratively adjusts the filter order until it finds the minimum order that meets or exceeds the filter specifications.

Use the Classical FIR Design front panel **File** menu to complete the following tasks:

- Save your filter specifications and coefficients
- Load filter designs from previous work
- Open the Analysis or the DAQ and Filter front panels
- Transfer the FIR design specifications to the Classical IIR Design front panel
- Return to the **Main Menu** dialog box

Refer to the *Classical IIR Design Front Panel Controls and Displays* section of this chapter for information about the same controls and displays on the Classical FIR Design front panel.

Pole-Zero Placement Filter Design

Figure 12-9 shows the Pole-Zero Placement filter design front panel. The front panel includes a graphical interface with the *z*-plane pole and zero cursors and the Magnitude vs. Frequency plot on the left side and a text-based interface with digital controls on the right side.

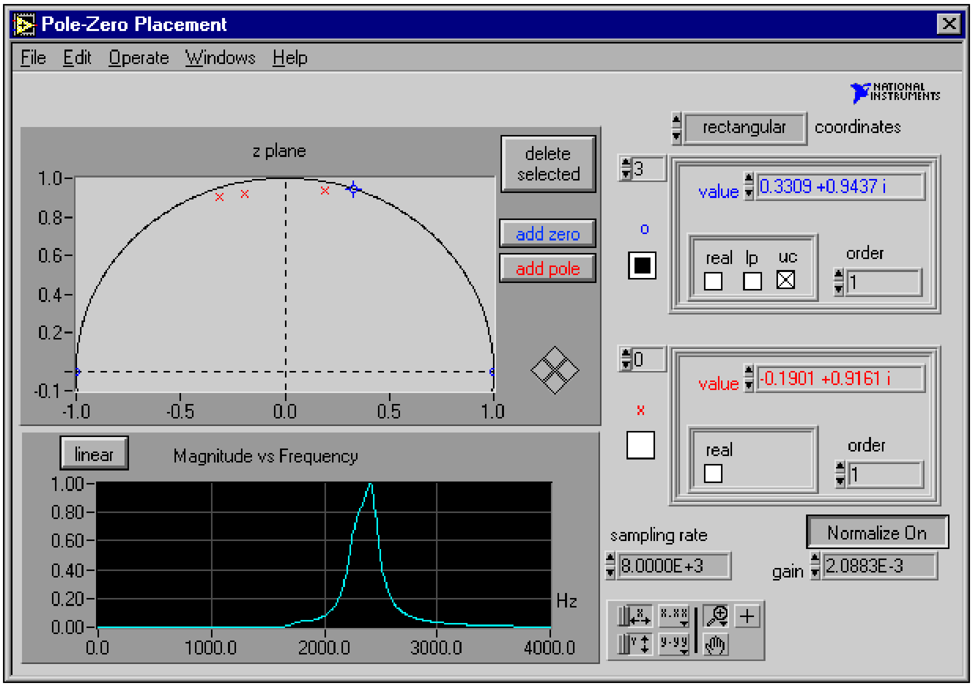


Figure 12-9. Pole-Zero Placement Filter Design Front Panel

Use the Pole-Zero Placement filter design front panel to design IIR digital filters by manipulating the filter poles and zeros in the z-plane. The poles and zeros initially could have originated from classical IIR designs. Use this front panel to move existing poles and zeros directly on the z-plane plot. You can add and delete poles and zeros and accurately control their important characteristics.

You can describe the poles and zeros by using either the text-based interface or the cursors in the z-plane plot. As you use the mouse to click and drag the cursors, the text entries update. Similarly, as you enter new specifications in the text entries, the pole and zero cursors update.

The following specifications describe pole-zero filter designs:

- pole and zero locations in the *z*-plane
- characteristics of each pole and zero
- gain
- sampling rate

Any change in these parameters corresponds to a change in the filter coefficients. The DFD application matches the poles and zeros and creates stable second-order stages for IIR filter coefficients. The DFD application then uses these coefficients to compute the filter magnitude response. For immediate graphical feedback to your pole-zero filter designs, the Magnitude vs. Frequency plot updates automatically when you change the poles or zeros.

Pole-Zero Placement Front Panel Controls and Displays

Use the design front panel **File** menu to complete the following tasks:

- Save your filter specifications and coefficients
- Load filter designs from previous work
- Open the Analysis or the DAQ and Filter front panels
- Return to the Main Menu front panel

Z-Plane Plot

In the z-plane plot, you can move each pole, represented by a red ×, anywhere within the unit circle along and above the *x*-axis. You can move each zero, represented by a blue o, anywhere along and above the *x*-axis.

Click the **delete selected** button to delete the selected pole or zero. Click poles and zeros to select them.

Click the **add pole** button to add a pole to the **z**-plane. The new pole is located at the origin.

Click the **add zero** button to add a zero to the z-plane. The new zero is located at the origin.

Coordinates Control

Use the **coordinates** control to specify how the DFD application displays the poles and zeros, either in **rectangular** or **polar** coordinates.

Array of Zeros in Rectangular Coordinates

Immediately below the **coordinates** control and labeled with a blue o is the array of zeros in rectangular coordinates indicator. The complex value of each zero represents its rectangular position on the z-plane. The integer, 3, in the upper-left box is the **index** of the displayed zero. By changing this **index** value, you can display a particular zero of the array of zeros. When you select a particular zero in the z-plane plot, the DFD application sets the index value of the array to the selected zero.

If you select the **real** checkbox, the zero becomes purely real and is limited to real-axis movement.

When you select the **lp** checkbox, the zero has linear phase. If the zero is not real or on the unit circle, the DFD application matches it with another zero at a radius of 1/*r*, where *r* is the radius of the original zero. The radius is the distance from the origin. Linear-phase zeros are important in linear-phase FIR filters. If your z-plane plot contains only zeros and all the zeros have linear phase, the FIR filter you designed has an overall linear phase response.

If you select the **uc** checkbox, the zero must be located on the unit circle, with a radius of 1.0 The zero is limited to movement along the unit circle.

The **order** text entry is the order of the zero or the number of actual zeros at this location in the *z*-plane.

An *Mth*-order zero at $z = b$ has a *z*-transform of $H(z) = (z - b)^M$.

Array of Poles in Rectangular Coordinates

Immediately below the array of zeros in rectangular coordinates and labeled with a red × is the array of poles in rectangular coordinates.

The complex value of each pole represents its rectangular position on the z-plane. The integer 0 in the upper-left box is the **index** of the displayed pole. By changing this index value, you can display a particular pole of the array of poles. When you select a particular pole in the z-plane plot, the DFD application sets the index value of the array to the selected pole.

Whether poles are real is the only special characteristic that applies to poles. If you select the **real** checkbox, the pole becomes purely real and is limited to real-axis movement.

The **order** text entry specifies the pole order or the number of actual poles at this location in the *z*-plane.

An *Mth*-order pole at $z = a$ has a z-transform of $H(z) = (z-a)^{-M}$.

If you change the **coordinates** to **polar**, the DFD application displays the poles and zeros in polar coordinates, as shown in Figure 12-10.

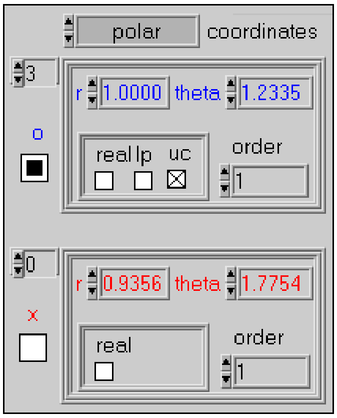


Figure 12-10. Array of Zeros and Poles in Polar Coordinates

Magnitude-Frequency Plot

The Magnitude vs. Frequency graph in Figure 12-9 plots the frequency response, $H(f)$, and magnitude of the designed digital filter. The magnitude, y-axis, is in linear or decibel units, depending on how you set the **linear/dB** button in the upper-left corner of the graph. The frequency, x-axis, is in hertz. The full scale ranges from 0.0 to Nyquist, or half the sampling rate.

Sampling Rate Control

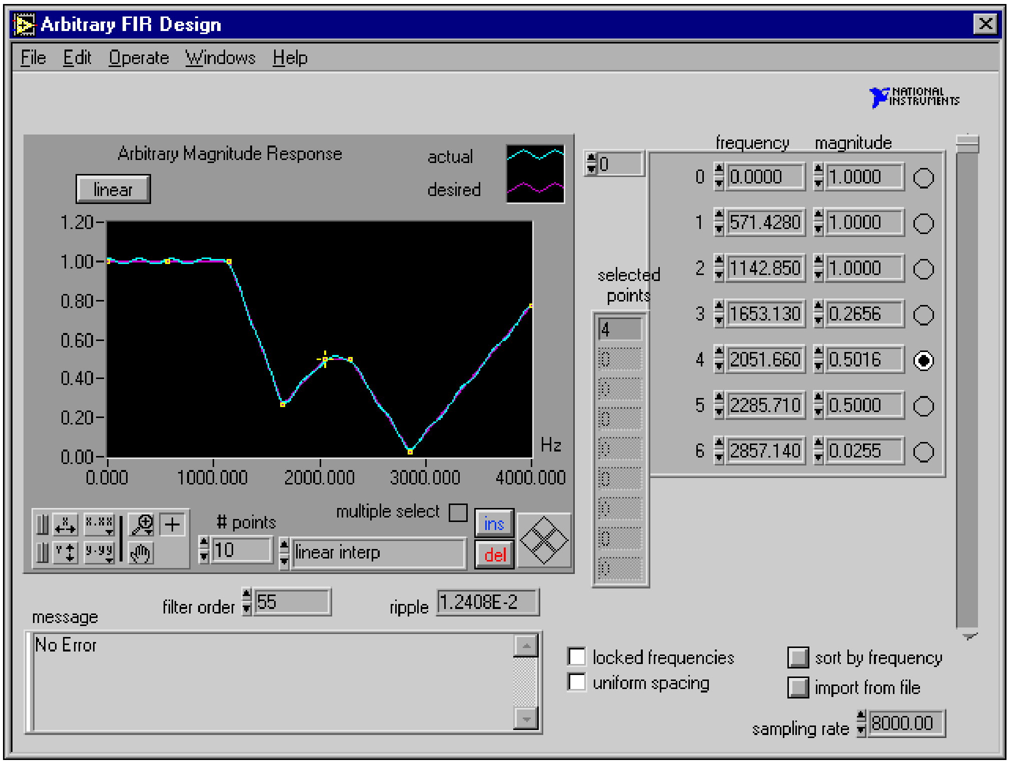
Use the **sampling rate** control to specify the sampling rate in samples per second, hertz.

Gain Control

Use the **gain** control to specify the gain constant for the designed filter. Increasing **gain** increases the overall gain of the designed filter. Setting the **Normalize** button to **Normalize On** adjusts the filter gain so that the maximum response is 1.0, 0 dB. If you set this button to **Normalize On**, you cannot adjust the gain control manually. Setting the **Normalize** button to **Normalize off** allows you to manually adjust the gain control but does not guarantee a maximum response of 1.0.

Arbitrary FIR Design

Figure 12-11 shows the Arbitrary FIR Design front panel. The front panel includes a graphical interface with the Magnitude vs. Frequency cursors and plot on the left side and a text-based interface with digital controls on the right side.



	frequency	magnitude	
0	0.0000	1.0000	○
1	571.4280	1.0000	○
2	1142.850	1.0000	○
3	1653.130	0.2656	○
4	2051.660	0.5016	●
5	2285.710	0.5000	○
6	2857.140	0.0255	○

points 10 linear interp
multiple select ☐
selected points 4
filter order 55
ripple 1.2408E-2
message
No Error
☐ locked frequencies
☐ uniform spacing
☐ sort by frequency
☐ import from file
sampling rate 8000.00

Figure 12-11. Arbitrary FIR Design Front Panel

Use this front panel to design arbitrary-magnitude FIR digital filters. Enter or modify the array magnitude response points, **frequency** and **magnitude**. From these points, the DFD application forms a desired magnitude response that covers the entire frequency range from 0.0 to Nyquist, or half the sampling rate. The DFD application then processes this desired response, along with the filter order, and uses the Parks-McClellan algorithm to design an optimal equiripple FIR filter. The Parks-McClellan algorithm minimizes the difference between the desired and actual filter response across the entire frequency range.

To design arbitrary-magnitude FIR filters, enter or modify the desired frequency-magnitude points and choose an interpolation type to generate the desired response between your specified points. The DFD application automatically designs and plots the equiripple FIR filter. You receive immediate graphical feedback to help you determine whether the filter meets your specifications.

Arbitrary FIR Filter Design Front Panel Controls and Displays

Use the design front panel **File** menu to complete the following tasks:

- Save your filter specifications and coefficients
- Load filter designs from previous work
- Open the Analysis or the DAQ and Filter front panels
- Return to the **Main Menu** dialog box

Arbitrary Magnitude Response Graph

The Arbitrary Magnitude Response graph in Figure 12-11 plots the desired and actual magnitude response of the designed FIR filter. The magnitude, y-axis, is in linear or decibel units, depending on how you set the **linear/dB** button in the upper-left corner of the graph. The frequency, x-axis, is in hertz. The full scale ranges from 0.0 to Nyquist, or half the sampling rate.

Linear/dB Button

Use the **linear/dB** button to control the display units, either linear or dB, of all magnitude and gain controls and displays. These controls and displays include the y-axis of the Arbitrary Magnitude Response plot, passband response, stopband attenuation, and tracking cursor magnitude.

# Points Control

Use **# points** to specify the number of frequency-magnitude points the DFD application uses to create the desired filter magnitude response. Reducing this number deletes points from the end of the frequency-magnitude array. Increasing this number inserts the additional number of points to the right of the selected point.

Multiple Select Checkbox

Check the **multiple select** checkbox to select more than one frequency-magnitude point on the response graph. Clicking a selected point removes that point from the selection list.

Interpolation Control

Use the **interpolation** control to select the type of interpolation the DFD application uses to generate the desired response from the array of frequency-magnitude points. Choose **linear interp** to create flat filters, such as lowpass, highpass, bandpass, and bandstop. Choose **spline interp** to create smoothly varying filters.

Ins Button

Click the **ins** button to insert a frequency-magnitude point between the selected point and the next point. If the selected point is the last point in the frequency-magnitude array, the DFD application inserts the new point between the last two points of the array. The DFD application inserts new points halfway along the line connecting the two outer points.

Del Button

Click the **del** button to delete the selected frequency-magnitude points. The DFD application deletes all selected points.

Selected Points Indicator

The selected points indicator displays the selected frequency-magnitude points. You can select points on the Arbitrary Magnitude Response graph by clicking the point. You also can select points directly from the frequency-magnitude array by clicking the circle to the right of each point.

Array of Frequency-Magnitude Points

The right side of the Arbitrary FIR Design front panel displays the array of frequency-magnitude points the DFD application uses to construct the desired filter magnitude response. The DFD application forms the desired filter response by interpolating between these points.

The frequency of each point is in hertz. The magnitude is in linear or decibel units of gain, depending on the setting of the **linear/dB** button in the upper-left corner of the Arbitrary Magnitude Response graph.

You can select points in this array by clicking the circle to the right of each point. Then, you can click the **del** button to delete the selected points. To move selected points, click the desired direction diamonds in the **cursor movement** control in the lower-right corner of the Arbitrary Magnitude Response graph.

Filter Order Control

Use **filter order** to specify the total number of coefficients in the digital FIR filter.

Ripple Indicator

The ripple indicator displays the largest absolute error, linear, between the desired and actual filter responses.

Message Window

The **message** window displays errors that occurred during the FIR design procedure.

Locked Frequencies Checkbox

Select the **locked frequencies** checkbox to lock the present frequency values of the frequency-magnitude points. If you select this checkbox, you can alter only the magnitude, the y-value, of the frequency-magnitude points.

Uniform Spacing Checkbox

Select the **uniform spacing** checkbox to space the frequency values of the frequency-magnitude points. The DFD application spaces the frequency-magnitude points uniformly from 0.0 to half the sampling rate, inclusive. Before spacing the frequency-magnitude points, the DFD application displays a dialog box. Click the **Uniform Spacing** button to continue and the **Cancel** button to cancel the spacing operation.

Sort by Frequency Button

Click the **sort by frequency** button to sort the frequency-magnitude points in both the response graph and the array by ascending frequency. The value of each frequency-magnitude point remains unchanged. Only the order of the points can change.

Import from File Button

Click the **import from file** button to import frequency-magnitude points from a text file. The imported file format consists of the following tab-delimited columns:

1st line:	sampling rate	dB/linear setting (0 for **linear**, 1 for **dB**)
2nd line:	frequency 1	magnitude 1
3rd line:	frequency 2	magnitude 2
4th line:	frequency 3	magnitude 3
.	.	.
.	.	.
.	.	.
last line:	last frequency	last magnitude

For example, a file with five frequency-magnitude points appears as

```
8000.0         1
   0.0     -60.0
1000.0     -40.0
2000.0     -20.0
3000.0       0.0
4000.0     -60.0
```

Sampling Rate Control

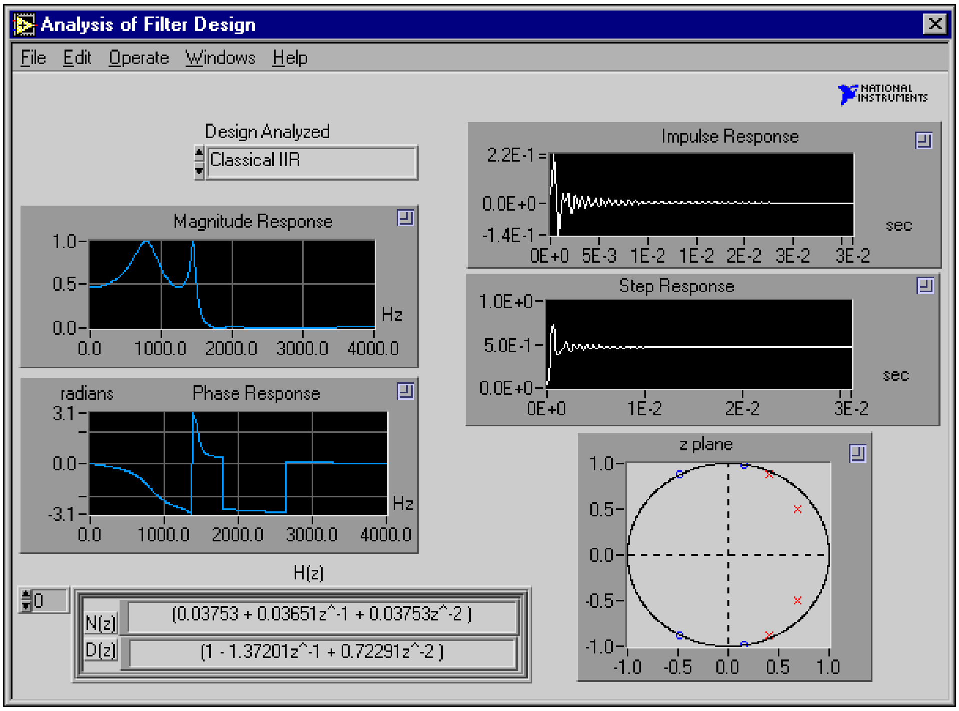
Use the **sampling rate** control to specify the sampling rate in samples per second, hertz.

Analysis of Filter Design Front Panel

Select **File»Analysis** from a filter design front panel toolbar to access the Analysis of Filter Design front panel, shown in Figure 12-12. Use the Analysis of Filter Design front panel to complete the following tasks:

- View the filter magnitude response, phase response, impulse response, step response, and pole-zero plot.
- View and print full-screen plots of each response.
- Save the analysis results in the full-screen views to text files.

The Analysis of Filter Design front panel uses the particular filter design specified in the filter design front panel in which it is opened to compute the various filter responses. You also can analyze any of the four filter designs from the **Design Analyzed** ring control. The **Analysis of Filter Design** front panel uses the filter parameters from the selected filter design.



N(z): (0.03753 + 0.03651z^-1 + 0.03753z^-2)

D(z): (1 - 1.37201z^-1 + 0.72291z^-2)

Figure 12-12. Analysis of Filter Design Front Panel

File Menu

Use the **File** menu to complete the following tasks:

- Load filter designs from previous work.
- Open the DAQ and Filter front panel.
- Go to the selected filter design front panel.
- Return to the **Main Menu** dialog box.

Design Analyzed Control

Use the **Design Analyzed** control to select the filter control to analyze. Select **File»Go to Design** to load and run the filter design front panel selected with the **Design Analyzed** control. If you continue to modify the same filter design that the DFD application is analyzing, the application recomputes all filter responses.

Analysis Displays

Each of the five filter plots has a **zoom box** control in the upper-right corner. Click in the **zoom box** to display a full-screen version of the plot. You can change the units in the full-screen versions of the following plots:

- In the Magnitude Response plot, you can change the units from linear to decibel.
- In the Phase Response plot, you can change the units from radians to degrees.
- In the Impulse and Step Responses plots, you can change the units from seconds to samples.

At each full-screen view, you can save the response data to text files.

Magnitude Response Plot

The Magnitude Response displays the magnitude of the filter response $H(f)$ as frequency varies from zero to half the sampling rate.

Phase Response Plot

The Phase Response displays the phase of the filter response $H(f)$ as frequency varies from zero to the sampling rate.

Impulse Response Plot

The Impulse Response displays the output of the digital filter when the input is a unit sample sequence, such as 1, 0, 0, … The input before the unity sample is also zero.

Step Response Plot

The Step Response displays the output of the digital filter when the input is a unit step sequence, such as 1, 1, 1, … The input samples before the step sequence are defined as zero.

Z-Plane Plot

In the z-plane plot of the filter poles and zeros, each pole is represented by a red ×, and each zero is represented by a blue o.

H(z) for IIR Filters

H(*z*) is the *z*-transform of the designed digital filter. For an IIR filter, *H*(*z*) can be represented by a product of fractions of second-order *z*-polynomials:

$$H(z) = \prod_{k=1}^{N_s} \frac{N_k(z)}{D_k(z)}$$

where $N_k(z)$ is the numerator for stage *k*

$D_k(z)$ is the denominator for stage *k*

N_s is the number of second-order stages

You can view the *N*(*z*) and *D*(*z*) polynomials for other stages by incrementing the index shown in the upper-left side of the *H*(*z*) display.

H(z) for FIR Filters

H(*z*) is the *z*-transform of the designed digital filter. You can scroll through *H(z)* using the scroll bar. For an FIR filter, *H*(*z*) can be represented as a polynomial in z^{-1}:

$$H(z) = \sum_{j=0}^{order-1} h_j z^{-j}$$

where $j = 0, 1, \ldots, order - 1$,

h_j represents the FIR filter coefficients, and

order is the number of FIR coefficients.

DAQ and Filter Front Panel

Select **File»DAQ and Filter** from a filter design front panel toolbar to access the DAQ and Filter front panel, shown in Figure 12-13. Use the DAQ and Filter front panel to accomplish the following tasks:

- Check the performance of your filter design on your own signals, if you have a National Instruments DAQ device.
- Check the performance of your filter design with a simulated signal.
- Configure your DAQ device and acquire your own signals.

When filtering signals, the DAQ and Filter front panel uses the particular set of filter coefficients specified in the filter design front panel in which it is opened. The signal data passes through the designed filter, and the DFD application plots the input and output waveforms and spectrums. You also can use any of the four filter designs from the **Filter Design** ring control. The DAQ and Filter front panel uses the filter parameters from the selected design specifications.

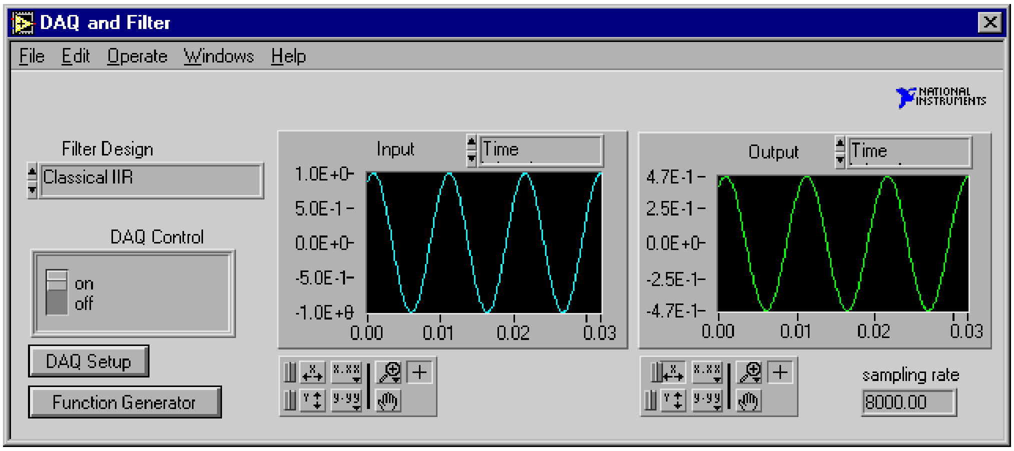


Figure 12-13. DAQ and Filter Front Panel

File Menu

Use the **File** menu to complete the following tasks:

- Load and test filter designs from previous work.
- Open the Analysis of Filter Design front panel.
- Go to the selected filter design front panel.
- Return to the **Main Menu** dialog box.

Filter Design Control

Use the **Filter Design** control to designate the filter design to use in filtering the acquired signal. Then, select **File»Go to Design** to load and run the filter design front panel selected with the **Filter Design** control.

On/Off Switch

Use the **on**/**off** switch to specify whether you want the DFD to acquire blocks continuously or on demand. Set the switch to **on** to continuously acquire blocks of data. Set the switch to **off** to acquire data when the **Acquire Once** button is clicked. The **Acquire Once** button only appears when the switch is set to **off**.

Function Generator Button

If you configure **source** on the DAQ Setup front panel, shown in Figure 12-14, to **simulated DAQ**, a built-in simulated function generator provides signals to the DAQ and Filter front panel. On the DAQ and Filter front panel, click the **Function Generator** button to view and edit settings, including signal type, frequency, amplitude, and noise level.

Spectrum/Time Ring Control

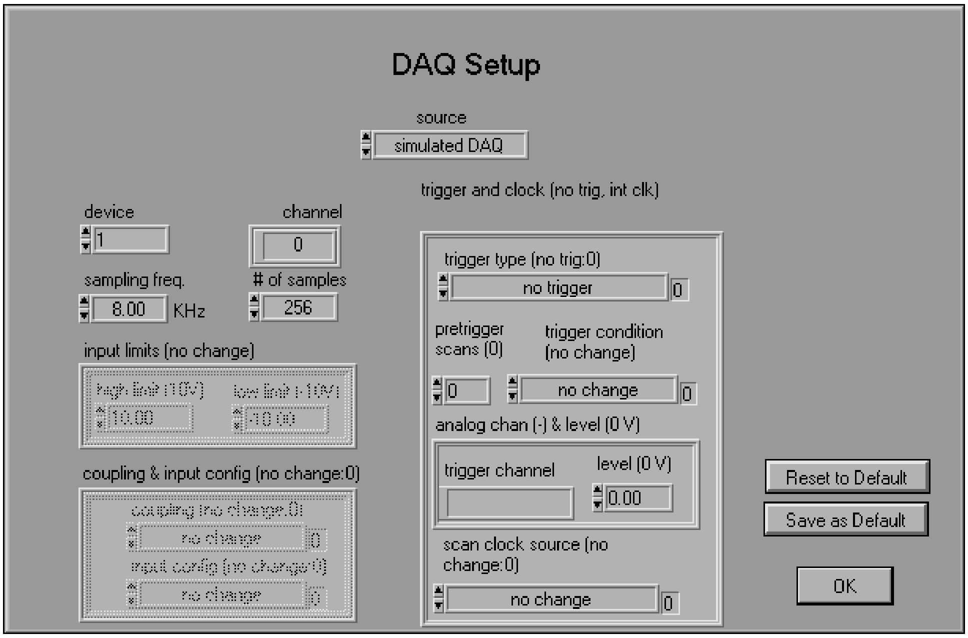
To change the view of a response plot, use the ring control above the plot. Select either **Time** or **Spectrum** for the input acquired signal or the filtered signal.

Sampling Rate Indicator

The actual sampling rate appears in an indicator in the lower right-hand corner of the DAQ and Filter front panel.

DAQ Setup Button and DAQ Setup Front Panel

Click the **DAQ Setup** button on the DAQ and Filter front panel to open the DAQ Setup front panel, shown in Figure 12-14. With the DAQ Setup front panel, you can change the data acquisition settings, such as the device number, number of samples to acquire, triggering parameters, or sampling rate. You also can set the source to either **DAQ Device** or **simulated DAQ** with the **source** ring control.



DAQ Setup

source: simulated DAQ

device: 1
channel: 0
sampling freq.: 8.00 KHz
# of samples: 256

input limits (no change)
high limit (10V): 10.00
low limit (-10V): -10.00

coupling & input config (no change:0)
coupling (no change:0): no change 0
input config (no change:0): no change 0

trigger and clock (no trig, int clk)
trigger type (no trig:0): no trigger 0
pretrigger scans (0): 0
trigger condition (no change): no change 0
analog chan (-) & level (0 V)
trigger channel:
level (0 V): 0.00
scan clock source (no change:0): no change 0

Figure 12-14. DAQ Setup Front Panel

13

IIR and FIR Implementation

This chapter describes the filter implementation equations for IIR and FIR filtering and the format of the IIR and FIR filter coefficient files.

Infinite Impulse Response Filters

Infinite impulse response (IIR) filters are digital filters with impulse responses that, theoretically, can be infinite in length or duration.
The general difference equation characterizing IIR filters is shown in the following equation:

$$y_i = \frac{1}{a_0}\left(\sum_{j=0}^{N_b-1} b_j x_{i-j} - \sum_{k=1}^{N_a-1} a_k y_{i-k} \right) \tag{13-1}$$

where N_b is the number of forward coefficients b_j, and N_a is the number of reverse coefficients a_k.

In most IIR filter designs, coefficient a_0 is 1. The output sample at the present sample index i consists of the sum of scaled present and past inputs (x_i and x_{i-j} when $j \neq 0$) and scaled past outputs (y_{i-k}).

The response of the general IIR filter to an impulse where $x_0 = 1$ and $x_i = 0$ for all $i \neq 0$ is called the impulse response of the filter. The impulse response of the filter described by Equation 13-1 has an infinite length for nonzero coefficients. In practical filter applications, however, the impulse response of stable IIR filters decays to near zero in a finite number of samples.

The advantage of digital IIR filters over finite impulse response (FIR) filters is that IIR filters usually require fewer coefficients to perform similar filtering operations. Therefore, IIR filters execute much faster and do not require extra memory, because they execute in place.

The disadvantage of IIR filters is that the phase response is nonlinear.
If the application does not require phase information, such as simple signal monitoring, IIR filters might be appropriate. Use FIR filters for applications that require linear phase responses.

IIR filters are also known as recursive filters or autoregressive moving-average (ARMA) filters. Refer to the works of Jackson [13], Oppenheim and Schafer [19], Parks and Burrus [20], and Parks and McClellan [21] [22] for more information about digital filter design.

Cascade-Form IIR Filtering

Filters implemented using the structure which Equation 13-1 directly defines are known as direct-form IIR filters. Direct form implementations often are sensitive to errors introduced by coefficient quantization and by computational precision limits. Additionally, a filter designed to be stable can become unstable with increasing coefficient length, which is proportional to filter order.

You can obtain a less-sensitive structure by dividing the direct-form transfer function into lower-order sections, or filter stages. The direct-form transfer function of the filter given by Equation 13-1, with $a_0 = 1$, can be written as a ratio of z transforms in the following equation:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_{N_b-1} z^{-(N_b-1)}}{1 + a_1 z^{-1} + \ldots + a_{N_a-1} z^{-(N_a-1)}} \qquad (13\text{-}2)$$

When you factor Equation 13-2 into second-order sections, the transfer function of the filter becomes a product of second-order filter functions:

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

where $N_s = \lfloor N_a/2 \rfloor$ is the largest integer less than or equal to $N_a/2$ and $N_a \geq N_b$. This new filter structure can be described as a cascade of second-order filters, as shown in Figure 13-1.

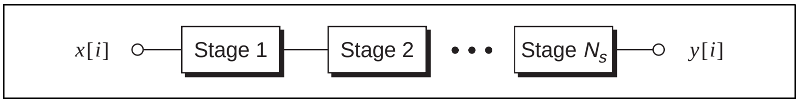


Figure 13-1. Cascaded Filter Stages

You can implement each second-order stage using the direct-form filter equations:

$$y[i] = b_0 x[i] + b_1 x[i-1] + b_2 x[i-2] - a_1 y[i-1] - a_2 y[i-2]$$

The illustration in Figure 13-2 shows the graphical representation of these direct-form equations.

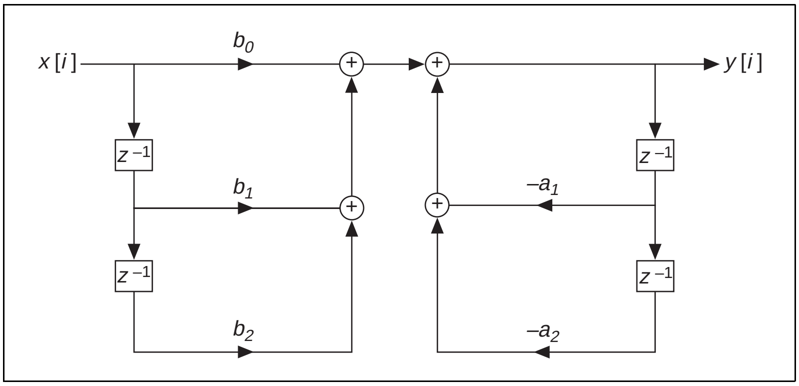


Figure 13-2. Direct Form Structure

For each stage, you must maintain two past inputs ($x[i-1]$, $x[i-2]$) and two past outputs ($y[i-1]$, $y[i-2]$).

A more efficient implementation of each second-order stage is known as the direct-form II. You can implement each second-order stage using the direct-form II filter equations:

$$s[i] = x[i] - a_1 s[i-1] - a_2 s[i-2]$$

$$y[i] = b_0 s[i] + b_1 s[i-1] + b_2 s[i-2]$$

The illustration in Figure 13-3 shows the graphical representation of these direct-form II equations.

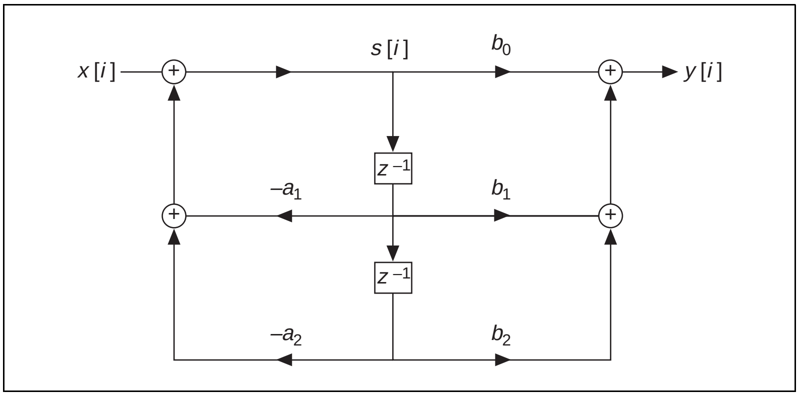


Figure 13-3. Direct Form II Structure

Finite Impulse Response Filters

FIR filters are digital filters with finite impulse responses. FIR filters are also known as nonrecursive filters, convolution filters, or moving-average (MA) filters, because you can express the output of an FIR filter as a finite convolution:

$$y_i = \sum_{k=0}^{n-1} h_k x_{i-k} \tag{13-3}$$

where x_i represents the input sequence to be filtered, y_i represents the output filtered sequence, and h_k represents the FIR filter coefficients.

FIR filters have the following characteristics:

- They can be designed to have linear phase by ensuring coefficient symmetry.
- They are always stable.
- You can perform the filtering function using the convolution. A delay generally is associated with the output sequence:

$$delay = \frac{n-1}{2}$$

where n is the number of FIR filter coefficients.

You design FIR filters by approximating a specified desired-frequency response of a discrete-time system. The most common techniques approximate the desired-magnitude response while maintaining a linear-phase response.

Format of the Filter-Coefficient Text Files

When you save your filter coefficients to a text file, the DFD application generates a readable text file that contains all the information you need to implement the designed FIR or IIR digital filter. This section describes the format for both FIR and IIR filter-coefficient files.

FIR-Coefficient File Format

Table 13-1 provides example FIR-coefficient text files and descriptions. You can use Equation 13-3 directly to implement the FIR filter.

Table 13-1. FIR-Coefficient Text Files and Descriptions

Coefficient File Example	Description
FIR filter coefficients	type of file
Sampling rate	sampling rate label
8.000000E+3	sampling rate in Hz
N	filter order label
22	filter order
h[0..21]	coefficients label
6.350871E–3	1st coefficient, *h*[0]
–8.833535E–3	2nd coefficient, *h*[1]
–2.847674E–2	.
4.626607E–2	.
4.103986E–2	.
–1.114579E–1	
–1.412791E–2	
1.810791E–1	
–5.984635E–2	

Table 13-1. FIR-Coefficient Text Files and Descriptions (Continued)

Coefficient File Example	Description
–2.002337E–1	
1.516199E–1	
1.516199E–1	
–2.002337E–1	
–5.984635E–2	
1.810791E–1	
–1.412791E–2	
–1.114579E–1	
4.103986E–2	
4.626607E–2	.
–2.847674E–2	.
–8.833535E–3	.
6.350871E–3	last coefficient, h[N – 1]

IIR Coefficient File Format

IIR coefficient files are slightly more complex than FIR coefficient files. IIR filters are usually described by two sets of coefficients, *a* and *b* coefficients. The total number of existing *a* coefficients equals $M \times S$, and the total number of existing *b* coefficients equals $(M + 1) \times S$, where M is the stage order, usually two, and S is the number of stages. An IIR filter with three second-order stages has two *a* coefficients per stage, for a total of six *a* coefficients, and three *b* coefficients per stage, for a total of nine *b* coefficients.

You can use Equation 13-1 to implement the IIR filter in cascade stages, and maintain two past inputs and two past outputs for each stage, or you can use the direct form II equations and maintain two past internal states.

Table 13-2 provides example IIR-coefficient text files and descriptions.

Table 13-2. IIR-Coefficient Text Files and Descriptions

Coefficient File Example	Description
IIR filter coefficients	coefficient type
Sampling rate	sampling rate label
8.000000E+3	sampling rate in Hz
Stage order	stage order label
2	order of each stage
Number of stages	number of stages label
3	number of stages
a coefficients	*a* coefficients label
6	number of coefficients
3.801467E–1	a_1 for stage 1
8.754090E–1	a_2 for stage 1
–1.021050E–1	a_1 for stage 2
9.492741E–1	a_2 for stage 2
8.460304E–1	a_1 for stage 3
9.450986E–1	a_2 for stage 3
b coefficients	*b* coefficients label
9	number of *b* coefficients
1.514603E–2	b_0 for stage 1
0.000000E+0	b_1 for stage 1
1.514603E–2	b_2 for stage 1
1.000000E+0	b_0 for stage 2
6.618322E–1	b_1 for stage 2
1.000000E+0	b_2 for stage 2

Table 13-2. IIR-Coefficient Text Files and Descriptions (Continued)

Coefficient File Example	Description
1.000000E+0	b_0 for stage 3
1.276187E+0	b_1 for stage 3
1.000000E+0	b_2 for stage 3

Part VI

Signal Processing for LabWindows/CVI

This section of the manual describes functions used for signal processing with LabWindows/CVI.

- Chapter 14, *Joint Time-Frequency Analysis for LabWindows/CVI*, describes functions used to perform JTFA analysis.
- Chapter 15, *Super-Resolution Spectral Analysis for LabWindows/CVI*, describes functions used to perform super-resolution spectral analysis and parameter estimation.
- Chapter 16, *Wavelet Analysis for LabWindows/CVI*, describes functions used to perform wavelet analysis.
- Chapter 17, *Using Your Coefficient Designs with DFD Utilities*, describes the Digital Filter Design utilities used for filtering applications.

14 Joint Time-Frequency Analysis for LabWindows/CVI

This chapter describes functions used to perform JTFA analysis.

SptRealSTFT

```
long SptRealSTFT(double x[ ], long len, SptWindowInfo * info,
SptExtension extension, double InitialCondition[ ], double
finalCondition[ ], ComplexNum * CxCoefficient);
```

Purpose

Computes short-time Fourier transform (STFT), also known as windowed Fourier transform, for real data samples.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
info	SptWindowInfo *	The analysis window information, which includes the window function **win**, window length **w_len**, time interval **dM**, and the number of frequency bins **N**. **N** must be a power of two.

Name	Type	Description
extension	SptExtension	Selects the extension method that decreases the artificial jumps at the beginning and end of data samples. Select from the following extension methods: • zero padding: 0 • symmetrical: 1 • 1st derivative (smooth padding of order 1): 2 • 2nd derivative (smooth padding of order 2): 3 • user defined: 4
initialCondition	double[ ]	The samples that decrease the artificial jumps at the beginning of the spectrogram. If you select **user defined** for **extension**, **initialCondition** is padded before the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the initial condition. The size of **initialCondition** must be **info.w_len** / 2.
finalCondition	double[ ]	The samples that decrease the artificial jumps at the end of the spectrogram. If you select **user defined** for **extension**, **finalCondition** is padded after the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the final condition. The size of **finalCondition** should be the same as the size of **initialCondition**.

Output

Name	Type	Description
CxCoefficient	ComplexNum *	The array for the resulting STFT. You should pre-allocate memory for this parameter. The size of **CxCoefficient** should be: `floor` (**len** / **info.dM**) × **info.N**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxSTFT

```
long SptCxSTFT(ComplexNum CxData[ ], long len, SptWindowInfo * info,
SptExtension extension, ComplexNum initialCondition[ ], ComplexNum
finalCondition[ ], ComplexNum * CxCoefficient);
```

Purpose

Computes STFT for complex data samples.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
info	SptWindowInfo *	The analysis window information, which includes the window function **win**, window length **w_len**, time interval **dM**, and the number of frequency bins **N**. **N** must be a power of two.
extension	SptExtension	Selects the extension method that decreases the artificial jumps at the beginning and end of data samples. Select from the following extension methods: • zero padding: 0 • symmetrical: 1 • 1st derivative (smooth padding of order 1): 2 • 2nd derivative (smooth padding of order 2): 3 • user defined: 4

Name	Type	Description
initialCondition	ComplexNum[ ]	The samples that decrease the artificial jumps at the beginning of the spectrogram. If you select **user defined** for **extension**, **initialCondition** is padded before the signal, **CxData**. Otherwise, the function ignores this parameter and automatically sets the initial condition. The size of **initialCondition** must be **info.w_len** / 2.
finalCondition	ComplexNum[ ]	The samples that decrease the artificial jumps at the end of the spectrogram. If you select **user defined** for **extension**, **finalCondition** is padded after the signal, **CxData**. Otherwise, the function ignores this parameter and automatically sets the final condition. The size of **finalCondition** should be the same as the size of **initialCondition**.

Output

Name	Type	Description
CxCoefficient	ComplexNum *	The array for the resulting STFT. You should pre-allocate memory for this parameter. The size of **CxCoefficient** should be: `floor` (**len** / **info.dM**) × **info.N**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealDGT

```
long SptRealDGT(double x[ ], long len, SptWindowInfo * info,
ComplexNum * CxCoefficient);
```

Purpose

Computes discrete Gabor transform for real data samples. If the dual function of the analysis function exists, `SptRealGaborExpansion` reverses this process.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
info	SptWindowInfo *	The analysis window information, which includes the window function **win**, window length **w_len**, Gabor time interval **dM,** and the number of frequency bins **N**. **w_len** must be evenly divisible by **dM** and **N. N** must be a power of two. The oversampling rate, ratio of **N** to **dM**, must be greater than or equal to one.

Output

Name	Type	Description
CxCoefficient	ComplexNum *	The resulting complex Gabor Coefficients. You should pre-allocate memory for this parameter. The size of **CxCoefficient** should be **size0** × **size1** elements, where **size0** = `ceil` [`ceil` (**len** / **info. N**) × **info.** N /**info. dM**]; **size1** = **info. N**;

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxDGT

```
long SptCxDGT(ComplexNum CxData[ ], long len, SptWindowInfo * info,
ComplexNum * CxCoefficient);
```

Purpose

Computes discrete Gabor transform for complex data samples. If the dual function of the analysis function exists, `SptCxGaborExpansion` reverses this process.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
info	SptWindowInfo *	The analysis window information, which includes the window function **win**, window length **w_len**, Gabor time interval **dM**, and the number of frequency bins **N**. **w_len** must be evenly divisible by **dM** and **N**. **N** must be a power of two. The oversampling rate, ratio of **N** to **dM**, must be greater than or equal to one.

Output

Name	Type	Description
CxCoefficient	ComplexNum *	The resulting complex Gabor coefficients. You should pre-allocate memory for this parameter. The size of **CxCoefficient** should be **size0** × **size1** elements, where **size0** = `ceil` [`ceil` (**len** / **info. N**) × **info. CxData** / **info. dM**]; **size1** = **info. N**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealGaborExpansion

```
long SptRealGaborExpansion(ComplexNum * CxCoefficient, long rows,
long cols, long signalLength, SptWindowInfo * info, double
waveform[ ]);
```

Purpose

Reconstructs the real data samples using Gabor coefficients and synthesis window.

Parameters

Input

Name	Type	Description
CxCoefficient	ComplexNum *	Complex Gabor coefficients.
rows	long	Determines the number of rows in **CxCoefficient**.
cols	long	Determines the number of columns in **CxCoefficient**.
signalLength	long	Determines the length of the reconstructed signal.
info	SptWindowInfo *	The synthesis window information, which includes the window function **win**, window length **w_len**, Gabor time interval **dM,** and the number of frequency bins **N**. **w_len** must be evenly divisible by **dM** and **N**. **N** must be a power of two. The oversampling rate, ratio of **N** to **dM**, must be greater than or equal to one.

Output

Name	Type	Description
waveform	double[ ]	The reconstructed time waveform. You should pre-allocate memory for this parameter. The size of **waveform** should be **signalLength**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxGaborExpansion

```
long SptCxGaborExpansion(ComplexNum * CxCoefficient, long rows, long
cols, long signalLength, SptWindowInfo * info, ComplexNum
CxWaveform[ ]);
```

Purpose

Reconstructs the complex data samples using Gabor coefficients and synthesis window.

Parameters

Input

Name	Type	Description
CxCoefficient	ComplexNum *	Complex-valued Gabor Coefficients.
rows	long	Determines the number of rows in **CxCoefficient**.
cols	long	Determines the number of cols in **CxCoefficient**.
signalLength	long	Determines the length of the reconstructed signal.
info	SptWindowInfo *	The synthesis window information, which includes the window function **win**, window length **w_len**, Gabor time interval **dM**, and the number of frequency bins **N**. **w_len** must be evenly divisible by **dM** and **N**. **N** must be a power of two. The oversampling rate, ratio of **N** to **dM**, must be greater than or equal to one.

Output

Name	Type	Description
CxWaveform	ComplexNum[ ]	The reconstructed time waveform. You should pre-allocate memory for this parameter. The size of **CxWaveform** should be **signalLength**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

Spt2DDiscreteGaborTransform

```
long Spt2DDiscreteGaborTransform(double * x, long rows, long cols,
SptWindowInfo * info1, SptWindowInfo * info2, ComplexNum * CxOutput);
```

Purpose

Computes the separable two-dimensional STFT. **info1** and **info2** determine the analysis windows for row and column data samples. If the dual functions for both analysis windows exist, `Spt2DDiscreteGaborExpansion` reverses this process.

Parameters

Input

Name	Type	Description
x	double *	An array of real 2D signals.
rows	long	Determines the number of rows in **x**.
cols	long	Determines the number of columns in **x**.
info1	SptWindowInfo *	The analysis window information for rows, which includes the window function **win**, window length **w_len**, Gabor time interval **dM**, and the number of frequency bins **N**. **w_len** must be evenly divisible by **dM** and **N. N** must be a power of two. The oversampling rate, ratio of **N** to **dM**, must be greater than or equal to one.
Info2	SptWindowInfo *	The analysis window information for columns, which includes the window function **win**, window length **w_len**, Gabor time interval **dM**, and the number of frequency bins **N**. **w_len** must be evenly divisible by **dM** and **N. N** must be a power of two. The oversampling rate, ratio of **N** to **dM**, must be greater than or equal to one.

Output

Name	Type	Description
CxOutput	ComplexNum *	The resulting complex Gabor coefficients. You should pre-allocate memory for this parameter. The size of **CxOutput** should be **size0** × **size1** × **size2** × **size3**, where **size0** = `ceil` [`ceil` (**rows** / **info1. N**) × **info1. N** / **info1. dM**]; **size1** = **info1.N**; **size2** = `ceil` [`ceil` (**cols** / **info2. N**) × **info2. N** / **info2. dM**]; **size3** = **info2. N**

Return Value

Name	Type	Description
status	`long`	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

Spt2DDiscreteGaborExpansion

```
long Spt2DDiscreteGaborExpansion(ComplexNum * CxData, long size0,
long size1, long size2, long size3, long rows, long cols,
SptWindowInfo * info1, SptWindowInfo * info2, double * output);
```

Purpose

Computes the separable two-dimensional Gabor expansion, the inverse of 2D Gabor Transform.

Parameters

Input

Name	Type	Description
CxData	`ComplexNum *`	The Gabor coefficients.
size0	`long`	The number of elements in the first index of **CxData**.
size1	`long`	The number of elements in the second index of **CxData**.
size2	`long`	The number of elements in the third index of **CxData**.
size3	`long`	The number of elements in the fourth index of **CxData**.
rows	`long`	Determines the number of rows in the 2D reconstructed signal.
cols	`long`	Determines the number of columns in the 2D reconstructed signal

Name	Type	Description
info1	`SptWindowInfo *`	The synthesis window information for rows, which includes the window function **win**, window length **w_len**, Gabor time interval **dM**, and the number of frequency bins **N**. **w_len** must be evenly divisible by **dM** and **N**. **N** must be a power of two. The oversampling rate, ratio of **N** to **dM**, must be greater than or equal to one.
info2	`SptWindowInfo *`	The synthesis window information for columns, which includes the window function **win**, window length **w_len**, Gabor time interval **dM**, and the number of frequency bins **N**. **w_len** must be evenly divisible by **dM** and **N**. **N** must be a power of two. The oversampling rate, ratio of **N** to **dM**, must be greater than or equal to one.

Output

Name	Type	Description
output	double *	The reconstructed 2D signal. You should pre-allocate memory for this parameter. The size of **output** should be **rows** × **cols**.

Return Value

Name	Type	Description
status	`long`	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealSTFTSpectrogram

```
long SptRealSTFTSpectrogram (double x[ ], long len, SptWindowInfo *
info, SptExtension extension, double initialCondition[ ], double
finalCondition[ ], double * spectrogram);
```

Purpose

Computes the STFT-based spectrogram of real data samples. This is the fastest and most robust algorithm among its counterparts.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
info	SptWindowInfo *	The analysis window information, which includes the window function **win**, window length **w_len**, time interval **dM**, and the number of frequency bins **N**. **N** must be a power of two.
extension	SptExtension	Selects the extension method that decreases the artificial jumps at the beginning and end of data samples. Select from the following extension methods: • zero padding: 0 • symmetrical: 1 • 1st derivative (smooth padding of order 1): 2 • 2nd derivative (smooth padding of order 2): 3 • user defined: 4

Name	Type	Description
initialCondition	double[ ]	The samples that decrease the artificial jumps at the beginning of the spectrogram. If you select **user defined** for **extension**, **initialCondition** is padded before the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the initial condition. The size of **initialCondition** must be **info.w_len** / 2.
finalCondition	double[ ]	The samples that decrease the artificial jumps at the end of the spectrogram. If you select **user defined** for **extension**, **finalCondition** is padded after the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the final condition. The size of **finalCondition** should be the same as the size of **initialCondition**.

Output

Name	Type	Description
spectrogram	double *	The STFT-based spectrogram. You should pre-allocate memory for this parameter. The size of **spectrogram** should be `floor` (**len** / **info.dM**) × **info.N**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxSTFTSpectrogram

```
long SptCxSTFTSpectrogram(ComplexNum CxData[ ], long len, SptWindowInfo *
      info, SptExtension extension, ComplexNum InitialCondition[ ],
      ComplexNum finalCondition[ ], double * spectrogram);
```

Purpose

Computes the STFT-based spectrogram for complex data samples. This is the fastest and most robust algorithm among its counterparts.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
info	SptWindowInfo *	The analysis window information, which includes the window function **win**, window length **w_len**, time interval **dM**, and the number of frequency bins **N**. **N** must be a power of two.
extension	SptExtension	Selects the extension method that decreases the artificial jumps at the beginning and end of data samples. Select from the following extension methods: • zero padding: 0 • symmetrical: 1 • 1st derivative (smooth padding of order 1): 2 • 2nd derivative (smooth padding of order 2): 3 • user defined: 4

Name	Type	Description
initialCondition	ComplexNum[ ]	The samples that decrease the artificial jumps at the beginning of the spectrogram. If you select **user defined** for **extension**, **initialCondition** is padded before the signal, **CxData**. Otherwise, the function ignores this parameter and automatically sets the initial condition. The size of **initialCondition** must be **info.w_len** / 2.
finalCondition	ComplexNum[ ]	The samples that decrease the artificial jumps at the end of the spectrogram. If you select **user defined** for **extension**, **finalCondition** is padded after the signal, **CxData**. Otherwise, the function ignores this parameter and automatically sets the final condition. The size of **finalCondition** should be the same as the size of **initialCondition**.

Output

Name	Type	Description
spectrogram	double *	The STFT-based spectrogram. You should pre-allocate memory for this parameter. The size of **spectrogram** should be `floor` (**len** / **info.dM**) × **info.N**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealFastGaborSpectrogram

```
long SptRealFastGaborSpectrogram(double x[ ], long len, long
timeInterval, long order, SptWindowInfo * windowInfo, SptFreqInfo *
freqInfo, SptExtension extension, double initialCondition[ ], double
finalCondition[ ], double * gaborSpectrogram);
```

Purpose

Combines Gabor expansion and Wigner-Ville distribution to compute the instantaneous spectra of the real data samples.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
timeInterval	long	Determines the time interval of the resulting 2D spectrogram. The smaller the **timeInterval** is, the larger the 2D spectrogram output array, and the greater the computation time. **timeInterval** is limited to a power of two.
order	long	Balances the resolution and crossterm interference. **order** must be greater than or equal to zero. The higher the **order** is, the better the resolution, but the more severe the interference, and vice versa. When **order** is set to zero, the Gabor spectrogram is non-negative. As **order** increases, the Gabor spectrogram converges to the Wigner–Ville distribution, and computation time increases. For most applications, choose an **order** between three and five.

Name	Type	Description
windowInfo	SptWindowInfo *	The analysis window information, which includes the window function **win**, window length **w_len**, Gabor time interval **dM**, and the number of frequency bins **N**. **w_len** must be evenly divisible by **dM** and **N**. **w_len**, **dM** and **N** must be a power of two. The oversampling rate, ratio of **N** to **dM**, must be greater than or equal to one.
freqInfo	SptFreqInfo *	The data structure for frequency parameters, which includes the following three parts: **zoomFactor** controls the frequency zoom in. **zoom factor** is limited to a power of two. **minFreq** determines the lowest frequency. **numOfBins** controls the number of columns in the spectrogram.
extension	SptExtension	Selects the extension method that decreases the artificial jumps at the beginning and end of data samples. Select from the following extension methods: • zero padding: 0 • symmetrical: 1 • 1st derivative (smooth padding of order 1): 2 • 2nd derivative (smooth padding of order 2): 3 • user defined: 4

Name	Type	Description
initialCondition	double[ ]	The samples that decrease the artificial jumps at the beginning of the spectrogram. If you select **user defined** for **extension**, **initialCondition** is padded before the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the initial condition. The size of **initialCondition** must be **windowInfo.w_len** / 2.
finalCondition	double[ ]	The samples that decrease the artificial jumps at the end of the spectrogram. If you select **user defined** for **extension**, **finalCondition** is padded after the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the final condition. The size of **finalCondition** should be the same as the size of **initialCondition**.

Output

Name	Type	Description
gaborSpectrogram	double *	The array for the resulting instantaneous power spectra. You should pre-allocate memory for this parameter. The size of **gaborSpectrogram** should be {[`floor` (**len** / **dM**) × **dM**] / **timeInterval**} × **freqInfo. numOfBins**, where **dM** = **windowInfo.dM**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxFastGaborSpectrogram

```
long SptCxFastGaborSpectrogram(ComplexNum CxData[ ], long len, long
timeInterval, long order, SptWindowInfo * windowInfo, SptFreqInfo *
freqInfo, SptExtension extension, ComplexNum initialCondition[ ],
ComplexNum finalCondition[ ], double * gaborSpectrogram);
```

Purpose

Combines Gabor expansion and Wigner-Ville distribution to compute the instantaneous spectra of the complex data samples.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
timeInterval	long	Determines the time interval of the resulting 2D spectrogram. The smaller the **timeInterval** is, the larger the 2D spectrogram output array, and the greater the computation time. **timeInterval** is limited to a power of two.
order	long	Balances the resolution and crossterm interference. **order** must be greater than or equal to zero. The greater the **order** is, the better the resolution, but the more severe the interference, and vice versa. When **order** is set to zero, the Gabor spectrogram is non-negative. As **order** increases, the Gabor spectrogram converges to the Wigner–Ville distribution, and computation time increases. For most applications, choose an **order** between three and five.

Name	Type	Description
windowInfo	SptWindowInfo *	The analysis window information, which includes the window function **win**, window length **w_len**, Gabor time interval **dM**, and the number of frequency bins **N**. **w_len** must be evenly divisible by **dM** and **N**. **w_len**, **dM**, and **N** must be powers of two. The oversampling rate, ratio of **N** to **dM**, must be greater than or equal to one.
freqInfo	SptFreqInfo *	The data structure for frequency parameters, which includes the following three parts: **zoomFactor** controls the frequency zoom in. **zoom factor** is limited to power of two. **minFreq** determines the lowest frequency. **numOfBins** controls the number of columns in the spectrogram.
extension	SptExtension	Selects the extension method that decreases the artificial jumps at the beginning and end of data samples. Select from the following extension methods: • zero padding: 0 • symmetrical: 1 • 1st derivative (smooth padding of order 1): 2 • 2nd derivative (smooth padding of order 2): 3 • user defined: 4

Name	Type	Description
initialCondition	ComplexNum[ ]	The samples that decrease the artificial jumps at the beginning of the spectrogram. If you select **user defined** for **extension**, **initialCondition** is padded before the signal, **CxData**. Otherwise, the function ignores this parameter and automatically sets the initial condition. The size of **initialCondition** must be **windowInfo.w_len** / 2.
finalCondition	ComplexNum[ ]	The samples that decrease the artificial jumps at the end of the spectrogram. If you select **user defined** for **extension**, **finalCondition** is padded after the signal, **CxData**. Otherwise, the function ignores this parameter and automatically sets the final condition. The size of **finalCondition** should be the same as the size of **initialCondition**.

Output

Name	Type	Description
gaborSpectrogram	double *	The array for the resulting instantaneous power spectra. You should pre-allocate memory for this parameter. The size of **gaborSpectrogram** should be {[`floor` (**len** / **dM**) × **dM**] / **timeInterval**} × **freqInfo. numOfBins**, where **dM** = **windowInfo.dM**.

Return Value

Name	Type	Description
status	`long`	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealEasyGaborSpectrogram

```
long SptRealEasyGaborSpectrogram(double x[ ], long len, long
gaborWindowLen, double var, long timeInterval, long order, double *
gaborSpectrogram)
```

Purpose

Computes the Gabor Expansion-based spectrogram for real data samples. Unlike the `SptRealFastGaborSpectrogram`, in which the user needs to compute the dual function in advance, `SptRealEasyGaborSpectrogram` combines two steps, computing the dual function of Gaussian window and Gabor spectrogram in one single function. Compared to `SptRealFastGaborSpectrogram`, `SptRealEasyGaborSpectrogram` is easier to use but has more overhead.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
gaborWindowLen	long	Determines the length of the Gaussian window.
var	double	Determines the variance of the Gaussian window.

Name	Type	Description
timeInterval	long	Determines the time interval of the resulting 2D spectrogram. The smaller the **timeInterval** is, the larger the 2D spectrogram output array, and the greater the computation time. **timeInterval** is limited to a power of two.
order	long	Balances resolution and crossterm interference. **order** must be greater than or equal to zero. The greater the **order** is, the better the resolution but the more severe the interference, and vice versa. When **order** is set to zero, the Gabor spectrogram is non-negative. As **order** increases, the Gabor spectrogram converges to the Wigner-Ville distribution, and computation time increases. For most applications, choose an **order** between three and five.

Output

Name	Type	Description
gaborSpectrogram	double *	The Gabor spectrogram. You should pre-allocate memory for this parameter. The size of **gaborSpectrogram** should be: **{[**`floor` **(len** / **dM)** × **dM]** / **timeInterval}** × **(gaussianWindowLen** / 2), where $\mathrm{dM} = 2^{\lfloor \log_2(\sqrt{\pi \mathrm{var}/2}) + 0.5 \rfloor}$

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxEasyGaborSpectrogram

```
long SptCxEasyGaborSpectrogram(ComplexNum CxData[ ], long len, long
gaborWindowLen, double var, long timeInterval, long order, double *
gaborSpectrogram)
```

Purpose

Computes the Gabor Expansion-based spectrogram for complex data samples. Unlike the `SptCxFastGaborSpectrogram`, in which the user needs to compute the dual function in advance, `SptCxEasyGaborSpectrogram` combines two steps, computing the dual function of Gaussian window and Gabor spectrogram in one single function. Compared to `SptCxFastGaborSpectrogram`, `SptCxEasyGaborSpectrogram` is easier to use but has more overhead.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
gaborWindowLen	long	Determines the length of the Gaussian window.
var	double	Determines the variance of the Gaussian window.

Name	Type	Description
timeInterval	long	Determines the time interval of the resulting 2D spectrogram. The smaller the **timeInterval** is, the larger the 2D spectrogram output array, and the greater the computation time. **timeInterval** is limited to a power of two.
order	long	Balances resolution and crossterm interference. **order** must be greater than or equal to zero. The greater the **order** is, the better the resolution but the more severe the interference, and vice versa. When **order** is set to zero, the Gabor spectrogram is non-negative. As **order** increases, the Gabor spectrogram converges to the Wigner-Ville distribution, and computation time increases. For most applications, choose an **order** between three and five.

Output

Name	Type	Description
gaborSpectrogram	double *	The Gabor spectrogram. You should pre-allocate memory for this parameter. The size of **gaborSpectrogram** should be: **{[**`floor` **(len** / **dM**) × **dM**] / **timeInterval**} × **gaussianWindowLen**, where $\mathrm{dM} = 2^{\left\lfloor \log_2(\sqrt{\pi \mathrm{var}/2}) + 0.5 \right\rfloor}$

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealCohen

```
long SptRealCohen(double x[ ], long len, double * kernel, long rows,
long cols, long timeInterval, long analyticalSignal, double * result);
```

Purpose

Computes the general Cohen’s class for the real input data samples.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
kernel	double *	The first quarter of a user-defined 2D kernel function.
rows	long	The number of rows of **kernel**. **rows** determines the number of distinct frequencies, or number of columns, of the resulting spectrogram. **rows** must be a power of two.
cols	long	The number of columns of **kernel**.
timeInterval	long	Determines the time interval of the resulting 2D spectrogram. The number of rows of the resulting 2D spectrogram is equal to **len** / **timeInterval**. The smaller the **timeInterval** is, the larger the 2D spectrogram output array, and the greater the computation time. **timeInterval** is limited to a power of two.
analyticalSignal	long	Determines whether to convert the input data samples to the corresponding analytical signal. Converting to the analytical signal can reduce the crossterm interference, although it can introduce distortion, particularly at low frequencies. It can be one of the two values: True: one; False: zero.

Output

Name	Type	Description
result	double *	The resulting Cohen's class. You should pre-allocate memory for this parameter. The size of **result** should be: `floor` (**len** / **timeInterval**) × (**rows** / 2).

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxCohen

```
long SptCxCohen(ComplexNum CxData[ ], long len, double * kernel, long
rows, long cols, long timeInterval, double * result);
```

Purpose

Computes the general Cohen’s class for complex input data samples.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
kernel	double *	The first quarter of a user-defined 2D kernel function.
rows	long	The number of rows in **kernel**. **rows** determines the number of distinct frequencies, or number of columns, in the resulting spectrogram. **rows** must be a power of two.
cols	long	The number of columns in **kernel**.
timeInterval	long	Determines the time interval of the resulting 2D spectrogram. The number of rows of the resulting 2D spectrogram is equal to **len** / **timeInterval**. The smaller the **timeInterval** is, the larger the 2D spectrogram output array, and the greater the computation time. **timeInterval** is limited to a power of two.

Output

Name	Type	Description
result	double *	The resulting Cohen’s class. You should pre-allocate memory for this parameter. The size of **result** should be: `floor` (**len** / **timeInterval**) × **rows**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealWignerVilleDistribution

```
long SptRealWignerVilleDistribution(double x[ ], long len, long
numOfFreqBins, long timeInterval, long analyticalSignal, double *
result);
```

Purpose

Computes the discrete Wigner-Ville distribution for real data samples.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
numOfFreqBins	long	Determines the number of columns in the spectrogram. It must be a power of two.
timeInterval	long	The time interval of the resulting 2D spectrogram. The number of rows of the resulting 2D spectrogram is equal to **len** / **timeInterval**. The smaller the **timeInterval** is, the larger the 2D spectrogram output array, and the greater the computation time. **timeInterval** is limited to a power of two.
analyticalSignal	long	Determines whether to convert the input data samples to the corresponding analytical signal. Converting to the analytical signal can reduce the crossterm interference, although it also can introduce distortion, particularly at low frequencies. It can be one of the two values: True: one; False: zero.

Output

Name	Type	Description
result	double *	The resulting Wigner-Ville distribution. You should pre-allocate memory for this parameter. The size of **result** should be `floor` (**len** / **timeInterval**) × **numOfFreqBins** / 2.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxWignerVilleDistribution

```
long SptCxWignerVilleDistribution(ComplexNum CxData[ ], long len,
long numOfFreqBins, long timeInterval, double * result);
```

Purpose

Computes the discrete Wigner-Ville distribution for complex data samples.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
numOfFreqBins	long	Determines the number of columns in the spectrogram. It must be a power of two.
timeInterval	long	Determines the time interval of the resulting 2D spectrogram. The number of rows of the resulting 2D spectrogram is equal to **len** / **timeInterval**. The smaller the **timeInterval** is, the larger the 2D spectrogram output array, and the greater the computation time. **timeInterval** is limited to a power of two.

Output

Name	Type	Description
result	double *	The resulting Wigner-Ville distribution. You should pre-allocate memory for this parameter. The size of **result** should be `floor` (**len** / **timeInterval**) × **numOfFreqBins**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealChoiWilliamsDistribution

```
long SptRealChoiWilliamsDistribution(double x[ ], long len, long
numOfFreqBins, long timeInterval, double alpha, long
analyticalSignal, double * result);
```

Purpose

Computes the Cohen's distribution with the exponential kernel function for real data samples. It reduces crossterm interference, although it increases computation time.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
numOfFreqBins	long	Determines the number of columns in the spectrogram. It must be a power of two.
timeInterval	long	Determines the time interval of the resulting 2D spectrogram. The number of rows of the resulting 2D spectrogram is equal to **len** / **timeInterval**. The smaller the **timeInterval** is, the larger the 2D spectrogram output array, and the greater the computation time. **timeInterval** is limited to a power of two.
alpha	double	Controls resolution and crossterm interference. Decreasing **alpha** suppresses the crossterm interference in the resulting spectrogram and reduces the resolution. **alpha** must be greater than zero.
analyticalSignal	long	Determines whether to convert the input data samples to the corresponding analytical signal. Converting to the analytical signal can reduce the crossterm interference, although it can introduce distortion, particularly at low frequencies. It can be one of the two values: True: one; False: zero.

Output

Name	Type	Description
result	double *	The 2D array for the resulting Choi-Williams distribution. You should pre-allocate memory for this parameter. The size of **result** should be `floor` (**len** / **timeInterval**) × (**numOfFreqBins** / 2).

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxChoiWilliamsDistribution

```
long SptCxChoiWilliamsDistribution(ComplexNum CxData[ ], long len,
long numOfFreqBins, long timeInterval, double alpha, double * result);
```

Purpose

Computes the Cohen's distribution with the exponential kernel function for complex data samples. It reduces crossterm interference, although it increases computation time.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
numOfFreqBins	long	Determines the number of columns in the spectrogram. It must be a power of two.
timeInterval	long	Determines the time interval of the resulting 2D spectrogram. The number of rows of the resulting 2D spectrogram is equal to **len** / **timeInterval**. The smaller the **timeInterval** is, the larger the 2D spectrogram output array, and the greater the computation time. **timeInterval** is limited to a power of two.
alpha	double	Controls the resolution and crossterm interference. Decreasing **alpha** suppresses the crossterm interference in the resulting spectrogram and reduces the resolution. **alpha** must be greater than zero.

Output

Name	Type	Description
result	double *	The 2D array for the resulting Choi-Williams distribution. You should pre-allocate memory for this parameter. The size of **result** should be `floor` (**len** / **timeInterval**) × **numOfFreqBins**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealConeShapedDistribution

```
long SptRealConeShapedDistribution(double x[ ], long len, long
numOfFreqBins, long timeInterval, double alpha, long
analyticalSignal, double * result);
```

Purpose

Computes the Cohen's distribution with the Cone-shaped kernel function for real data samples. It reduces crossterm interference, although it increases computation time.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
numOfFreqBins	long	Determines the number of columns in the spectrogram. It must be a power of two.
timeInterval	long	Determines the time interval of the resulting 2D spectrogram. The number of rows of the resulting 2D spectrogram is equal to **len** / **timeInterval**. The smaller the **timeInterval** is, the larger the 2D spectrogram output array, and the greater the computation time. **timeInterval** is limited to a power of two.
alpha	double	Controls the resolution and crossterm interference. Decreasing **alpha** suppresses the crossterm interference in the resulting spectrogram and reduces the resolution. **alpha** must be greater than zero.
analyticalSignal	long	Determines whether to convert the input data samples to the corresponding analytical signal. Converting to the analytical signal can reduce the crossterm interference, although it can introduce distortion, particularly at low frequencies. It can be one of the two values: True: one; False: zero.

Output

Name	Type	Description
result	double *	A 2D array for the resulting Cone-shaped distribution. You should pre-allocate memory for this parameter. The size of **result** should be `floor` (**len** / **timeInterval**) × (**numOfFreqBins** / 2).

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxConeShapedDistribution

```
long SptCxConeShapedDistribution(ComplexNum CxData[ ], long len, long
numOfFreqBins, long timeInterval, double alpha, double * result);
```

Purpose

Computes the Cohen's distribution with the Cone-shaped kernel function for complex data samples. It reduces crossterm interference, although it increases computation time.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
numOfFreqBins	long	Determines the number of columns in the spectrogram. It must be a power of two.
timeInterval	long	Determines the time interval of the resulting 2D spectrogram. The number of rows of the resulting 2D spectrogram is equal to **len** / **timeInterval**. The smaller the **timeInterval** is, the larger the 2D spectrogram output array, and the greater the computation time. **timeInterval** is limited to a power of two.
alpha	double	Controls the resolution and crossterm interference. Decreasing **alpha** suppresses the crossterm interference in the resulting spectrogram and reduces the resolution. **alpha** must be greater than zero.

Output

Name	Type	Description
result	double *	A 2D array for the resulting Cone-shaped distribution. You should pre-allocate memory for this parameter. The size of **result** should be `floor` (**len** / **timeInterval**) × **numOfFreqBins**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptFastDualFunction

```
long SptFastDualFunction(SptWindowInfo * info, long equal, double
gamma[ ]);
```

Purpose

Computes the dual functions of the given function, **info.win**, which is used for Gabor transform and Gabor expansion.

Parameters

Input

Name	Type	Description
info	SptWindowInfo *	The analysis or synthesis window information, which includes the window function **win**, window length **w_len**, Gabor time interval **dM,** and the number of frequency bins **N**. **w_len** must be evenly divisible by **dM** and **N**. **N** must be a power of two. The oversampling rate, ratio of **N** to **dM**, must be greater than or equal to one.
equal	long	Determines whether the lengths of the window function and the signal are equal. When the window length is the same as the length of signal, the chance of obtaining the corresponding dual function is higher. It can be one of the two values: True: one; False: zero.

Output

Name	Type	Description
gamma	double[ ]	Dual function that has the same length as the given function **info. win**. **gamma** and **win** constitute a pair of analysis and synthesis functions for Gabor transform and Gabor expansion. The Gabor time sampling interval, **dM**, and the number of frequency bins, **N**, of the resulting Gabor expansion are the same as those of the input. You should pre-allocate memory for this parameter. The size of **gamma** should be **info. w_len**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptOptimalGaussianDual

```
long SptOptimalGaussianDual(long gaussianWindowLen, double var, long
oversampling, long equal, SptWindowInfo * dualFunctionInfo,
SptWindowInfo * synthesisWindowInfo)
```

Purpose

This function computes the dual function for an optimal Gaussian window.

Parameters

Input

Name	Type	Description
gaussianWindowLen	long	Determines the length of the Gaussian Window.
var	double	Determines the variance of the Gaussian Window.
oversampling	long	The ratio of N to dM, where N is the number of frequency bins and dM is the Gabor time interval. For stable reconstruction, **oversampling** must be greater than or equal to one.
equal	long	Determines whether the lengths of the window function and the signal are equal. When the window length is the same as the length of the signal, the chance of obtaining the corresponding dual function is higher. One represents TRUE while zero represents FALSE.

Output

Name	Type	Description
dualFunctionInfo	SptWindowInfo *	Information about the dual function: **win[ ]** is the dual function of the given Gaussian Window. **w_len** is the actual length of the dual function, gamma[ ]. **dM** is the Gabor time sampling interval. The length of **win[ ]** is evenly divisible by **dM**. **N** is the number of frequency bins of the resulting Gabor Transform. You should pre-allocate memory for **win[ ]**. The size of **win[ ]** should be `floor` (**gaussianWindowLen** / Q) $\times$ Q where $Q = \text{oversampling} \times 2^{\lfloor \log_2(\sqrt{\pi \text{var}/2}) + 0.5 \rfloor}$
synthesisWindowInfo	SptWindowInfo *	Information about the synthesis function: **win[ ]** contains the samples of the Gaussian Window. **w_len** is the actual length of the Gaussian window, **win[ ]**. **dM** is the Gabor time sampling interval. The length of the **win[ ]** is evenly divisible by **dM**. **N** is the number of frequency bins of the resulting Gabor transform. You should pre-allocate memory for **win[ ]**. The size of the synthesis function **win[ ]** should be the same as that of **dualFunctionInfo.win[ ]**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptMeanInstFreq

```
long SptMeanInstFreq(double * spectra, long rows, long cols, double
freq[ ], double power[ ], double * IF);
```

Purpose

Computes the mean instantaneous frequency based on the given spectrogram.

Parameters

Input

Name	Type	Description
spectra	double *	A 2D spectrogram.
rows	long	Determines the number of rows in **spectra**.
cols	long	Determines the number of columns in **spectra**.

Output

Name	Type	Description
freq	double[ ]	A 1D array of mean frequency at each time instant. You should pre-allocate memory for this parameter. The size of **freq** should be **rows**.
power	double[ ]	The instantaneous power, which is a 1D array of the power of the signal at each time instant. You should pre-allocate memory for this parameter. The size of **power** should be **rows**.
IF	double *	The normalized mean instantaneous frequency. You should pre-allocate memory for this parameter. The size of **IF** should be **rows** × **cols**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

15 Super-Resolution Spectral Analysis for LabWindows/CVI

This chapter describes functions used to perform super-resolution and parameter estimation.

SptRealCovariance

```
long SptRealCovariance (double x[ ], long len, long numOfComplexSin,
double coefficient[ ], ComplexNum CxRoots[ ], double * noiseEst);
```

Purpose

Computes auto-regressive (AR) model coefficients for a real waveform, **x**, using the covariance method. For this method, the order of the AR model is equal to the number of complex sinusoids in **x**.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.

Name	Type	Description
numOfComplexSin	long	Determines the number of complex sinusoids in **x**. Since each real sinusoid counts for two complex sinusoids, **numOfComplexSin** must be an odd number if DC is presented. In all other situations, **numOfComplexSin** is an even number. You can use the FFT to estimate **numOfComplexSin**. The number of complex sinusoids counted from FFT is the lower bound of the true number of sinusoids. You also can use the `SptRealMinDescriptLen` function in this toolset to estimate **numOfComplexSin**, but this method is time consuming. **numOfComplexSin** cannot be greater than 2 × **len** / 3.

Output

Name	Type	Description
coefficient	double[ ]	The resulting AR model coefficients. You should pre-allocate memory for this parameter. The size of **coefficient** should be: **numOfComplexSin** + 1.
CxRoots	ComplexNum[ ]	The information about normalized resonant frequencies. You should pre-allocate memory for this parameter. The size of the **CxRoots** should be **numOfComplexSin**.
noiseEst	double *	Indicates the variance of Gaussian white noise.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxCovariance

```
long SptCxCovariance (ComplexNum CxData[ ], long len, long
numOfComplexSin, ComplexNum CxCoefficient[ ], ComplexNum CxRoots[ ],
double * noiseEst);
```

Purpose

Computes AR model coefficients for a complex waveform, **CxData**, using the covariance method. For this method, the order of the AR model is equal to the number of complex sinusoids in **CxData**.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
numOfComplexSin	long	Determines the number of complex sinusoids in **CxData**. You can use the FFT to estimate **numOfComplexSin**. The number of complex sinusoids counted from FFT is the lower bound of the true number of sinusoids. You also can use the `SptCxMinDescriptLen` function in this toolset to estimate **numOfComplexSin**, but this method is time consuming. **numOfComplexSin** cannot be greater than 2 × **len** / 3.

Output

Name	Type	Description
CxCoefficient	ComplexNum[ ]	The resulting AR model coefficients. You should pre-allocate memory for this parameter. The size of the **CxCoefficient** should be **numOfComplexSin** + 1.
CxRoots	ComplexNum[ ]	The information about normalized resonant frequencies. You should pre-allocate memory for this parameter. The size of the **CxRoots** should be **numOfComplexSin**.
noiseEst	double *	Indicates the variance of Gaussian white noise.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealCovPowerSpectrum

```
long SptRealCovPowerSpectrum(double x[ ], long len, long
numOfComplexSin, long numOfBins, double sampleFreq,
SptPowerSpectrumPlot * powerSpectrumPlot, double * noiseEst);
```

Purpose

Computes the covariance method-based power spectrum for a real waveform, **x**. For this method, the order of the AR model is equal to the number of complex sinusoids in **x**.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
numOfComplexSin	long	Determines the number of complex sinusoids in **x**. Since each real sinusoid counts for two complex sinusoids, **numOfComplexSin** must be an odd number if DC is presented. In all other situations, **numOfComplexSin** is an even number. You can use the FFT to estimate **numOfComplexSin**. The number of complex sinusoids counted from FFT is the lower bound of the true number of sinusoids. You also can use the `SptRealMinDescriptLen` function in this toolset to estimate **numOfComplexSin**, but this method is time consuming. **numOfComplexSin** cannot be greater than 2 × **len** / 3.
numOfBins	long	Determines the number of frequency bins in the resulting spectrum.
sampleFreq	double	The sampling frequency in Hz. The default value is 1 Hz.

Output

Name	Type	Description
powerSpectrumPlot	SptPowerSpectrumPlot *	The information about the power spectrum plot, which includes: **lowBoundFreq** indicates the lower bound of the frequency range, which is equal to –**sampleFreq** / 2. **freqInc** indicates the frequency increment. **powerSpectrum** contains the estimated power spectrum in the log scale. The dimension of **powerSpectrum** is power of two. **powerSpectrum** is greater than or equal to the input parameter **numOfBins** and less than two times the value of **numOfBins**. You should pre-allocate memory for this parameter. Its size should be: $2^{\lceil \log_2(\mathrm{numOfBins}) \rceil}$
noiseEst	double *	Indicates the variance of Gaussian white noise.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxCovPowerSpectrum

```
long SptCxCovPowerSpectrum(ComplexNum CxData[ ], long len, long
numOfComplexSin, long numOfBins, double sampleFreq,
SptPowerSpectrumPlot * powerSpectrumPlot, double * noiseEst);
```

Purpose

Computes the covariance method-based power spectrum for a complex waveform, **CxData**. For this method, the order of the AR model is equal to the number of complex sinusoids in **CxData**.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
numOfComplexSin	long	Determines the number of complex sinusoids in **CxData**. You can use the FFT to estimate **numOfComplexSin**. The number of complex sinusoids counted from FFT is the lower bound of the true number of sinusoids. You also can use the `SptCxMinDescriptLen` function in this toolset to estimate **numOfComplexSin**, but this method is time consuming. **numOfComplexSin** cannot be greater than 2 × **len** / 3.
numOfBins	long	Determines the number of frequency bins in the resulting spectrum.
sampleFreq	double	The sampling frequency in Hz. The default value is 1 Hz.

Output

Name	Type	Description
powerSpectrumPlot	SptPowerSpectrumPlot *	The information about the power spectrum plot, which includes: **lowBoundFreq** indicates the lower bound of the frequency range, which is equal to –**sampleFreq** / 2. **freqInc** indicates the frequency increment. **powerSpectrum** contains the estimated power spectrum in the log scale. The dimension of **powerSpectrum** is power of two. **powerSpectrum** is greater than or equal to the input parameter **numOfBins** and less than two times the value of **numOfBins**. You should pre-allocate memory for this parameter. Its size should be: $2^{\lceil \log_2(\mathrm{numOfBins}) \rceil}$
noiseEst	double *	Indicates the variance of Gaussian white noise.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealPrincipalComAutogressive

```
long SptRealPrincipalComAutogressive(double x[ ], long len, long order,
long numOfComplexSin, double coefficient[ ], ComplexNum CxRoots[ ]);
```

Purpose

Computes AR model coefficients for a real waveform, **x**, by the principal component autoregressive (PCAR) method. The PCAR method is less sensitive to noise but needs more computing time than the covariance method.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
order	long	Determines the estimated order of the AR model. **order** must be greater than or equal to the **numOfComplexSin** and less than two thirds of the number of data samples, **len**.
numOfComplexSin	long	Determines the number of complex sinusoids in **x**. Since each real sinusoid counts for two complex sinusoids, **numOfComplexSin** must be an odd number if DC is presented. In all other situations, **numOfComplexSin** is an even number. You can use the FFT to estimate **numOfComplexSin**. The number of complex sinusoids counted from FFT is the lower bound of the true number of sinusoids. You also can use the `SptRealMinDescriptLen` function in this toolset to estimate **numOfComplexSin**, but this method is time consuming. **numOfComplexSin** cannot be greater than 2 × **len** / 3.

Output

Name	Type	Description
coefficient	double[ ]	The resulting AR model coefficients. You should pre-allocate memory for this parameter. Its size should be **order** + 1.
CxRoots	ComplexNum[ ]	The information about normalized resonant frequencies. You should pre-allocate memory for this parameter. Its size should be **order**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxPrincipalComAutogressive

```
long SptCxPrincipalComAutogressive(ComplexNum CxData[ ], long len,
long order, long numOfComplexSin, ComplexNum CxCoefficient[ ],
ComplexNum CxRoots[ ]);
```

Purpose

Computes AR model coefficients for a complex waveform, **CxData**, by the principal component autoregressive (PCAR) method. The PCAR method is less sensitive to noise but needs more computing time than the covariance method.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
order	long	The estimated order of the AR model. **order** must be greater than or equal to the **numOfComplexSin** and less than two-thirds of the number of data samples, **len**.
numOfComplexSin	long	Determines the number of complex sinusoids in **CxData**. You can use the FFT to estimate **numOfComplexSin**. The number of complex sinusoids counted from FFT is the lower bound of the true number of sinusoids. You also can use the `SptCxMinDescriptLen` function in this toolset to estimate **numOfComplexSin**, but this method is time consuming. **numOfComplexSin** cannot be greater than 2 × **len** / 3.

Output

Name	Type	Description
Coefficient	ComplexNum[ ]	The resulting AR model coefficients. You should pre-allocate memory for this parameter. Its size should be **order** + 1.
CxRoots	ComplexNum[ ]	The information about normalized resonant frequencies. You should pre-allocate memory for this parameter. Its size should be **order**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealPCARPowerSpectrum

```
long SptRealPCARPowerSpectrum(double x[ ], long len, long order, long
numOfComplexSin, long numOfBins, double sampleFreq, long
positiveOnly, double amplitude[ ], double phase[ ], double
frequency[ ], double * noiseEst, SptPowerSpectrumPlot *
powerSpectrumPlot, long * outputSize);
```

Purpose

Computes PCAR-based power spectrum for a real waveform, **x**. This algorithm is less sensitive to noise but needs more computing time than `SptRealCovPowerSpectrum`.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
order	long	Determines the estimated order of the AR model. **order** must be greater than or equal to the **numOfComplexSin** and less than two-thirds of the number of data samples, **len**.
numOfComplexSin	long	Determines the number of complex sinusoids in **x**. Since each real sinusoid counts for two complex sinusoids, **numOfComplexSin** must be an odd number if DC is presented. In all other situations, **numOfComplexSin** is an even number. You can use the FFT to estimate **numOfComplexSin**. The number of complex sinusoids counted from FFT is the lower bound of the true number of sinusoids. You also can use the `SptRealMinDescriptLen` function in this toolset to estimate **numOfComplexSin**, but this method is time consuming. **numOfComplexSin** cannot be greater than 2 × **len** / 3.

Name	Type	Description
numOfBins	long	The bin number of the power spectrum.
sampleFreq	double	The sampling frequency in Hz. The default value is 1 Hz.
positiveOnly	long	Determines whether or not to compute only the positive frequency components. Can be one of the following two values: True: one; False: zero.

Output

Name	Type	Description
amplitude	double[ ]	The amplitude of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
phase	double[ ]	The phase of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
frequency	double[ ]	The frequency of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
noiseEst	double *	Indicates the variance of Gaussian white noise.

Name	Type	Description
powerSpectrumPlot	SptPowerSpectrumPlot *	The information about the power spectrum plot, which includes: **lowBoundFreq** indicates the lower bound of the frequency range, which is equal to **–sampleFreq** / 2. **freqInc** indicates the frequency increment. **powerSpectrum** contains the estimated power spectrum in the log scale. The dimension of **powerSpectrum** is power of two. **powerSpectrum** is greater than or equal to the input parameter **numOfBins** and less than two times the value of **numOfBins**. You should pre-allocate memory for this parameter. Its size should be: $2^{\lceil \log_2(\mathrm{numOfBins}) \rceil}$
OutputSize	long *	The actual array size of **amplitude**, **phase,** and **frequency**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxPCARPowerSpectrum

```
long SptCxPCARPowerSpectrum(ComplexNum CxData[ ], long len, long
order, long numOfComplexSin, long numOfBins, double sampleFreq, double
amplitude[ ], double phase[ ], double frequency[ ], double * noiseEst,
SptPowerSpectrumPlot * powerSpectrumPlot, long * outputSize);
```

Purpose

Computes PCAR-based power spectrum for a complex waveform, **CxData**. This algorithm is less sensitive to noise but needs more computing time than `SptCxCovPowerSpectrum`.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
order	long	Determines the estimated order of the AR model. **order** must be greater than or equal to the **numOfComplexSin** and less than two-thirds of the number of data samples, **len**.
numOfComplexSin	long	Determines the number of complex sinusoids in **CxData**. You can use the FFT to estimate **numOfComplexSin**. The number of complex sinusoids counted from FFT is the lower bound of the true number of sinusoids. You also can use the `SptCxMinDescriptLen` function in this toolset to estimate **numOfComplexSin**, but this method is time consuming. **numOfComplexSin** cannot be greater than 2 × **len** / 3.
numOfBins	long	The bin number of the power spectrum.
sampleFreq	double	The sampling frequency in Hz. The default value is 1 Hz.

Output

Name	Type	Description
amplitude	double[ ]	The amplitude of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
phase	double[ ]	The phase of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
frequency	double[ ]	The frequency of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
noiseEst	double *	Indicates the variance of the Gaussian white noise.
powerSpectrumPlot	SptPowerSpectrumPlot*	The information about the power spectrum plot, which includes: **lowBoundFreq** indicates the lower bound of the frequency range, which is equal to –**sampleFreq** / 2. **freqInc** indicates the frequency increment. **powerSpectrum** contains the estimated power spectrum in the log scale. The dimension of **powerSpectrum** is power of two. **powerSpectrum** is greater than or equal to the input parameter **numOfBins** and less than two times the value of **numOfBins**. You should pre-allocate memory for this parameter. Its size should be: $2^{\lceil \log_2(\mathrm{numOfBins}) \rceil}$
outputSize	long *	The actual array size of **amplitude**, **phase,** and **frequency**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealProny

```
long SptRealProny(double x[ ], long len, long numOfComplexSin, double
sampleFreq, long positiveOnly, double amplitude[ ], double phase[ ],
double dampFactor[ ], double frequency[ ], long * outputSize);
```

Purpose

Uses Prony's method to estimate the parameters of exponentially damped sinusoids. The parameters include amplitudes, phases, damping factors, and frequencies. For this method, the order of the AR model is equal to the number of complex sinusoids in **x**.

Parameter

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
numOfComplexSin	long	Determines the number of complex sinusoids in **x**. Since each real sinusoid counts for two complex sinusoids, **numOfComplexSin** must be an odd number if DC is presented. In all other situations, **numOfComplexSin** is an even number. You can use the FFT to estimate **numOfComplexSin**. The number of complex sinusoids counted from FFT is the lower bound of the true number of sinusoids. You also can use the `SptRealMinDescriptLen` function in this toolset to estimate **numOfComplexSin**, but this method is time consuming. **numOfComplexSin** cannot be greater than 2 × **len** / 3.
sampleFreq	double	The sampling frequency in Hz. The default value is 1 Hz.
positiveOnly	long	Determines whether to compute only the positive frequency components. Can be one of the following two values: True: one; False: zero.

Output

Name	Type	Description
amplitude	double[ ]	The amplitude of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
phase	double[ ]	The phase of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
dampFactor	double[ ]	The damping factor, alpha, of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
frequency	double[ ]	The frequency of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
outputSize	long *	The actual array size of **amplitude**, **phase**, **dampFactor**, and **frequency**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxProny

```
long SptCxProny(ComplexNum CxData[ ], long len, long numOfComplexSin,
double sampleFreq, double amplitude[ ], double phase[ ], double
dampFactor[ ], double frequency[ ], long * outputSize);
```

Purpose

Uses Prony's method to estimate the parameters of exponentially damped sinusoids. The parameters include amplitudes, phases, damping factors, and frequencies. For this method, the order of the AR model is equal to the number of complex sinusoids in **CxData**.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
numOfComplexSin	long	Determines the number of complex sinusoids in **CxData**. You can use the FFT to estimate **numOfComplexSin**. The number of complex sinusoids counted from FFT is the lower bound of the true number of sinusoids. You also can use the `SptCxMinDescriptLen` function in this toolset to estimate **numOfComplexSin**, but this method is time consuming. **numOfComplexSin** cannot be greater than 2 × **len** / 3.
sampleFreq	double	The sampling frequency in Hz. The default value is 1 Hz.

Output

Name	Type	Description
amplitude	double[ ]	The amplitude of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
phase	double[ ]	The phase of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
dampFactor	double[ ]	The damping factor, alpha, of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
frequency	double[ ]	The frequency of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
outputSize	long *	The actual array size of **amplitude**, **phase**, **dampFactor**, and **frequency**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealMatrixPencil

```
long SptRealMatrixPencil(double x[ ], long len, long order, long
numOfComplexSin, double sampleFreq, long positiveOnly, double
amplitude[ ], double phase[ ], double dampFactor[ ], double
frequency[ ], double * noiseEst, long * outputSize);
```

Purpose

Uses the matrix pencil method to estimate the parameters of exponentially damped sinusoids. The parameters include amplitudes, phases, damping factors, and frequencies. This method is faster and less sensitive to noise than the Prony's method.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
order	long	Determines the estimated order of the AR model. **order** must be greater than or equal to the **numOfComplexSin** and less than two-thirds of the number of data samples, **len**.
numOfComplexSin	long	Determines the number of complex sinusoids in **x**. Since each real sinusoid counts for two complex sinusoids, **numOfComplexSin** must be an odd number if DC is presented. In all other situations, **numOfComplexSin** is an even number. You can use the FFT to estimate **numOfComplexSin**. The number of complex sinusoids counted from FFT is the lower bound of the true number of sinusoids. You also can use the `SptRealMinDescriptLen` function in this toolset to estimate **numOfComplexSin**, but this method is time consuming. **numOfComplexSin** cannot be greater than 2 × **len** / 3.

Name	Type	Description
sampleFreq	double	The sampling frequency in Hz. The default value is 1 Hz.
positiveOnly	long	Determines whether or not to compute only the positive frequency components. Can be one of the following two values: True: one; False: zero.

Output

Name	Type	Description
amplitude	double[ ]	The amplitude of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
phase	double[ ]	The phase of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
dampFactor	double[ ]	The damping factor, alpha, of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
frequency	double[ ]	The frequency of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
noiseEst	double *	Indicates the Gaussian white noise covariance.
outputSize	long *	The actual array size of **amplitude**, **phase**, **dampFactor**, and **frequency**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxMatrixPencil

```
long SptCxMatrixPencil(ComplexNum CxData[ ], long len, long order,
long numOfComplexSin, double sampleFreq, double amplitude[ ], double
phase[ ], double dampFactor[ ], double frequency[ ], double *
noiseEst, long * outputSize);
```

Purpose

Uses the matrix pencil method to estimate the parameters of exponentially damped sinusoids. The parameters include amplitudes, phases, damping factors, and frequencies. This method is faster and less sensitive to noise than the Prony's method.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
order	long	Determines the estimated order of the AR model. **order** must be greater than or equal to the **numOfComplexSin** and less than two thirds of the number of data samples, **len**.
numOfComplexSin	long	Determines the number of complex sinusoids in **CxData**. You can use the FFT to estimate **numOfComplexSin**. The number of complex sinusoids counted from FFT is the lower bound of the true number of sinusoids. You also can use the `SptCxMinDescriptLen` function in this toolset to estimate **numOfComplexSin**, but this method is time consuming. **numOfComplexSin** cannot be greater than 2 × **len** / 3.
sampleFreq	double	The sampling frequency in Hz. The default value is 1 Hz.

Output

Name	Type	Description
amplitude	double[ ]	The amplitude of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
phase	double[ ]	The phase of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
dampFactor	double[ ]	The damping factor, alpha, of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
frequency	double[ ]	The frequency of each component. You should pre-allocate memory for this parameter. Its size should be **numOfComplexSin**.
noiseEst	double *	Indicates the variance of Gaussian white noise.
outputSize	long *	The actual array size of **amplitude**, **phase**, **dampFactor**, and **frequency**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptRealMinDescriptLen

```
long SptRealMinDescriptLen(double x[ ], long len, long largest, long
* optimalAROrder);
```

Purpose

Estimates the order of the AR model for a real waveform, **x**. The output of this function also can be used to estimate the number of sinusoids.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
largest	long	Determines the upper bound of the AR order. **largest** cannot be greater than two-thirds the value of **len**.

Output

Name	Type	Description
optimalAROrder	long *	The estimated AR order of the waveform, **x**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptCxMinDescriptLen

```
long SptCxMinDescriptLen(ComplexNum CxData[ ], long len, long largest,
long * optimalAROrder);
```

Purpose

Estimates the order of the AR model for a complex waveform, **CxData**. The output of this function also can be used to estimate the number of sinusoids.

Parameters

Input

Name	Type	Description
CxData	ComplexNum[ ]	A 1D array of complex data samples.
len	long	The array size of **CxData**.
largest	long	Determines the upper bound of the AR order. **largest** cannot be greater than two-thirds the value of **len**.

Output

Name	Type	Description
optimalAROrder	long *	The estimated AR order of the waveform, **CxData**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

16

Wavelet Analysis for LabWindows/CVI

This chapter describes functions used to perform wavelet analysis.

SptInterpolationFilter

```
long SptInterpolationFilter(double x[ ], long len, double
filterCoef[ ], long lenFilterCoef, long interpfactor, long shift,
double output[ ]);
```

Purpose

Performs interpolation filtering as described by the following equation:

$$\mathrm{y}[i] = \mathrm{SUM}\{x[i - Mn] \times \mathrm{h}[n]\}, \ 0 \leq n < N$$

M denotes the interpolation factor. *N* is the length of interpolation filter, h[*n*].

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
filterCoef	double[ ]	The filter coefficients.
lenFilterCoef	long	The array size of **filterCoef**.

Name	Type	Description
interpfactor	long	Determines the interpolation factor. Before the filtering operation, this function inserts a specific number of zeros, equal to **interpfactor** – 1, between every two samples of **x**.
shift	long	Determines the number of zeros to be padded at the beginning of **x**, before the filtering operation.

Output

Name	Type	Description
output	double[ ]	The output array. You should pre-allocate memory for this parameter. The size of **output** should be (**len** × **interpfactor** – **lenFilterCoef** + 1+ **shift**).

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptDecimationFilter

```
long SptDecimationFilter(double x[ ], long len, double coef[ ], long
lenCoef, SptExtension extension, double initialCondition[ ], double
finalCondition[ ], long decFact, long shift, double output[ ]);
```

Purpose

Performs decimation filtering as described by the following equation:

$$y[i] = \mathrm{SUM}\{x[Mi + \mathrm{shift} - n] \times h[n]\},\ 0 \le n < N$$

M denotes the decimation factor.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
coef	double[ ]	The filter coefficients.
lenCoef	long	The array size of **coef**.
extension	SptExtension	Selects the extension method that decreases the artificial jumps at the beginning and end of data samples. Select from the following extension methods: • zero padding: 0 • symmetrical: 1 • 1st derivative (smooth padding of order 1): 2 • 2nd derivative (smooth padding of order 2): 3 • user defined: 4

Name	Type	Description
initialCondition	double[ ]	The samples that decrease the artificial jumps at the beginning of the spectrogram. If you select **user defined** for **extension**, **initialCondition** is padded before the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the initial condition. The size of **initialCondition** must be **lenCoef** – 1.
finalCondition	double[ ]	The samples that decrease the artificial jumps at the end of the spectrogram. If you select **user defined** for **extension**, **finalCondition** is padded after the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the final condition. The size of **finalCondition** should be the same as the size of **initialCondition**.
decFact	long	Indicates the data reduction rate in the output array. Only every *M*th point of the output from the filter is kept in the output array, where *M* is **decFact**.
shift	long	Determines the index to begin decimation. **shift** must be less than **decFact**.

Output

Name	Type	Description
output	double[ ]	The output array. You should pre-allocate memory for this parameter. The size of **output** should be `ceil` ((double)(**len** + **lenCoef** – **shift** – 1) / **decFact**).

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptAnalysisFilterBank

```
long SptAnalysisFilterBank(double x[ ], long len, SptFilter *
analysisFilter, SptExtension extension, double initialCondition[ ],
double finalCondition[ ], long shift, double lowPass[ ], double
highPass[ ]);
```

Purpose

Computes the output parameters of an analysis filter bank. It performs two-channel decimation filtering as described in the following equations:

$$\mathrm{y0}[i] = \mathrm{SUM}\{x[2i + shift - n]^*\mathrm{h0}[n]\},\ 0 \le n < N$$

$$\mathrm{y1}[i] = \mathrm{SUM}\{x[2i + shift - n]^*\mathrm{h1}[n]\},\ 0 \le n < N$$

y0[*i*] and y1[*i*] denote the output parameters for the lowpass and highpass channels, respectively.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
analysisFilter	SptFilter *	Sets the following analysis filter bank coefficients: **lowPass** represents the lowpass analysis filter coefficients. **lowLen** is the array size of **lowPass**. **highPass** represents the highpass analysis filter coefficients. **highLen** is the array size of **highPass**.

Name	Type	Description
extension	SptExtension	Selects the extension method that decreases the artificial jumps at the beginning and end of data samples. Select from the following extension methods: • zero padding: 0 • symmetrical: 1 • 1st derivative (smooth padding of order 1): 2 • 2nd derivative (smooth padding of order 2): 3 • user defined: 4
initialCondition	double[ ]	The samples that decrease the artificial jumps at the beginning of the spectrogram. If you select **user defined** for **extension**, **initialCondition** is padded before the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the initial condition. If *L_lo* denotes **analysisFilter.lowLen** and *L_hi* denotes **analysisFilter.highLen**, the size of **initialCondition** must be 2*`floor`{[Maximum(*L_lo*, *L_hi*)+1] / 2}– 1.
finalCondition	double[ ]	The samples that decrease the artificial jumps at the end of the spectrogram. If you select **user defined** for **extension**, **finalCondition** is padded after the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the final condition. The size of **finalCondition** should be same as the size of **initialCondition**.
shift	long	Determines the indexes of decimation. For this function, **shift** can be zero or one. Zero denotes even indexes, while one represents odd indexes.

Output

Name	Type	Description
lowPass	double[ ]	The output of the analysis lowpass filter. You should pre-allocate memory for this parameter. If *L_lo* denotes **analysisFilter**.**lowLen** and *L_hi* denotes **analysisFilter**.**highLen**, the size of **lowPass** should be `ceil`{{**len** + 2 × `floor`{[Maximum(*L_lo*, *L_hi*) + 1] / 2} – 1 – **shift**} / 2.0}.
highPass	double[ ]	The output of the analysis highpass filter. You should pre-allocate memory for this parameter. The size of **highPass** should be same as that of **lowPass**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptSynthesisFilterBank

```
long SptSynthesisFilterBank(double lowPass[ ], double highPass[ ],
long len, SptFilter * synthesisFilter, long shift, double output[ ]);
```

Purpose

Computes the output parameters of a synthesis filter bank. It performs two-channel interpolation filtering as described by the following equation:

$$x[i] = SUM\{y0[i-2n]*h0[n]\} + SUM\{y1[i-2n]*h1[n]\}$$

y0[i] and y1[i] denote the lowpass and highpass samples, h0[n] denotes the lowpass synthesis filter coefficients, and h1[n] denotes the highpass synthesis filter coefficients.

Parameters

Input

Name	Type	Description
lowPass	double[ ]	A 1D data array for the synthesis lowpass filter. It is often the output from the analysis lowpass filter.
highPass	double[ ]	A 1D array for the synthesis highpass filter. It is often the output from the analysis highpass filter.
len	long	Determines the array size of **lowPass** and **highPass**.
synthesisFilter	SptFilter *	Sets the following synthesis filter coefficients: **lowPass** contains the lowpass synthesis filter coefficients. **lowLen** is the array size of **lowPass**. **highPass** contains the highpass synthesis filter coefficients. **highLen** is the array size of **highPass**.
shift	long	Determines the number of zeros to be padded at the beginning of **lowPass** and **highPass** before the filtering operation. In this function, **shift** can be zero or one.

Output

Name	Type	Description
output	double[ ]	The output synthesized signal from the synthesis filter bank. You should pre-allocate memory for this parameter. If *L_lo* denotes **synthesisFilter**.**lowLen** and *L_hi* denotes **synthesisFilter**.**highLen**, the size of **output** should be 2 × **len** – 2 × `floor`{[Maximum(*L_lo*, *L_hi*) + 1] / 2} + 1 + **shift**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptDiscreteWaveletTransform

```
long SptDiscreteWaveletTransform(double x[ ], long len, SptFilter *
filter, SptExtension extension, double initialCondition[ ], double
finalCondition[ ], long scales, long shift, double dwt[ ], long *
outputSize, long lengths[ ]);
```

Purpose

Computes discrete wavelet transform.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
filter	SptFilter *	Sets the following analysis filter bank coefficients: **lowPass** contains the lowpass analysis filter coefficients. **lowLen** is the array size of **lowPass**. **highPass** contains the highpass analysis filter coefficients. **highLen** is the array size of **highPass**.
extension	`SptExtension`	Selects the extension method that decreases the artificial jumps at the beginning and end of data samples. Select from the following extension methods: • zero padding: 0 • symmetrical: 1 • 1st derivative (smooth padding of order 1): 2 • 2nd derivative (smooth padding of order 2): 3 • user defined: 4

Name	Type	Description
initialCondition	double[ ]	The samples that decrease the artificial jumps at the beginning of the spectrogram. If you select **user defined** for **extension**, **initialCondition** is padded before the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the initial condition. If *L_lo* denotes **filter.lowLen** and *L_hi* denotes **filter.highLen**, the size of **initialCondition** must be 2 × `floor`{[Maximum(*L_lo*, *L_hi*)+1] / 2}–1.
finalCondition	double[ ]	The samples that decrease the artificial jumps at the end of the spectrogram. If you select **user defined** for **extension**, **finalCondition** is padded after the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the final condition. The size of **finalCondition** should be same as that of **initialCondition**.
scales	long	Determines the number of scales of the wavelet transform, which must be greater than zero and less then or equal to log2(**len**).
shift	long	Determines the indexes of decimation. For this function, **shift** can be zero or one. Zero denotes even indexes, while one represents odd indexes.

Output

Name	Type	Description
dwt	double[ ]	Stores the resulting 2D wavelet transform as a 1D array. For N scale wavelet transform, the resulting wavelet transform is stored as follows: 0th scale: Nth scale: N-1th scale: .. : 2nd scale: 1st scale. If *L_lo* denotes **filter**.**lowLen** and *L_hi* denotes **filter**.**highLen**, you should pre-allocate `ceil`{**len** + **scales** × [Maximum(*L_lo*, *L_hi*) + 1]} size of memory for **dwt**. The actual size of **dwt** is stored in **outputSize**.
outputSize	long *	The actual size of **dwt**.
lengths	long[ ]	Stores the length information for each scale as a 1D array. The last element of **lengths** is always equal to the length of the original data samples. You should pre-allocate memory for this parameter. The size of **lengths** should be **scales** + 2.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptInvDiscreteWaveletTransform

```
long SptInvDiscreteWaveletTransform(double dwt[ ], long dwtLen, long
lengths[ ], long lengthLen, SptFilter * filter, long shift, double
x[ ]);
```

Purpose

Computes the inverse discrete wavelet transform.

Parameters

Input

Name	Type	Description
dwt	double[ ]	Stores the resulting 2D wavelet transform as a 1D array. For N scale wavelet transform, the resulting wavelet transform is stored as follows: 0th scale: Nth scale: N-1th scale: .. : 2nd scale: 1st scale.
dwtLen	long	The array size of **dwt**.
lengths	long[ ]	Stores the length information for each scale as a 1D array. The last element of **lengths** is equal to the length of the original data samples.
lengthLen	long	The array size of **lengths**.
filter	SptFilter *	Sets the following synthesis filters coefficients: **lowPass** contains the lowpass synthesis filter coefficients. **lowLen** is the array size of **lowPass**. **highPass** contains the highpass filter coefficients. **highLen** is the array size of **highPass**.
shift	long	Determines the number of zeros to be padded at the beginning of Discrete Wavelet Transform results, before the filtering operation. In this function, **shift** can be zero or one.

Output

Name	Type	Description
x	double[ ]	Stores the 1D reconstructed real time data samples. You should pre-allocate memory for this parameter. The size of **x** should be **lengths**[**lengthLen** – 1].

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptTruncatedAnalysisFilterBank

```
long SptTruncatedAnalysisFilterBank(double x[ ], long len, SptFilter
* analysisFilter, SptExtension extension, double initialCondition[ ],
double finalCondition[ ], long shift, double lowPass[ ], double
highPass[ ]);
```

Purpose

Performs lowpass and highpass filtering. The length of the outputs is equal to half of the input data samples. You cannot reconstruct the original data samples from the outputs of this function, as with the `SptAnalysisFilterBank`. Only use this function when you do not need to reconstruct the original data.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
analysisFilter	SptFilter *	Sets the following analysis filter bank coefficients: **lowPass** represents the lowpass analysis filter coefficients. **lowLen** is the array size of **lowPass**. **highPass** represents the highpass analysis filter coefficients. **highLen** is the array size of **highPass**.

Name	Type	Description
extension	SptExtension	Selects the extension method that decreases the artificial jumps at the beginning and end of data samples. Select from the following extension methods: • zero padding: 0 • symmetrical: 1 • 1st derivative (smooth padding of order 1): 2 • 2nd derivative (smooth padding of order 2): 3 • user defined: 4
initialCondition	double[ ]	The samples that decrease the artificial jumps at the beginning of the spectrogram. If you select **user defined** for **extension**, **initialCondition** is padded before the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the initial condition. If *L_lo* denotes **analysisFilter.lowLen** and *L_hi* denotes **analysisFilter.highLen**, the size of **initialCondition** must be 2 × `floor`{[Maximum(*L_lo*, *L_hi*)+1] / 2}-1.
finalCondition	double[ ]	The samples that decrease the artificial jumps at the end of the spectrogram. If you select **user defined** for **extension**, **finalCondition** is padded after the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the final condition. The size of **finalCondition** should be same as that of **initialCondition**.
shift	`long`	Determines the indexes of decimation. Shift can be either zero or one, where zero denotes even indexes and one represents odd indexes.

Output

Name	Type	Description
lowPass	double[ ]	The lowpass filter array. You should pre-allocate memory for this parameter. The size of **lowPass** should be half the number of the input samples.
highPass	double[ ]	The highpass filter array. You should pre-allocate memory for this parameter. The size of **highPass** should be the same as the size of **lowPass**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptArbitraryPath

```
long SptArbitraryPath(double x[ ], long len, char * path, long
pathLen, SptFilter * filter, SptExtension extension, double
initialCondition[ ], double finalCondition[ ], long shift, double
waveletTransform[ ]);
```

Purpose

Uses a group of cascaded lowpass or highpass filters to perform band-pass filtering.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
path	char *	Determines how the filters are connected. Zero indicates lowpass, while one represents highpass. For example, the string 0101 means lowpass – highpass – lowpass – highpass.
pathLen	long	The array size of **path**.
filter	SptFilter *	Sets the following analysis filter bank coefficients: **lowPass** contains the lowpass analysis filter coefficients. **lowLen** is the array size of **lowPass**. **highPass** contains the highpass filter coefficients. **highLen** is the array size of **highPass**.

Name	Type	Description
extension	SptExtension	Selects the extension method that decreases the artificial jumps at the beginning and end of data samples. Select from the following extension methods: • zero padding: 0 • symmetrical: 1 • 1st derivative (smooth padding of order 1): 2 • 2nd derivative (smooth padding of order 2): 3 • user defined: 4
initialCondition	double[ ]	The samples that decrease the artificial jumps at the beginning of the spectrogram. If you select **user defined** for **extension**, **initialCondition** is padded before the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the initial condition. If *L_lo* denotes **filter.lowLen** and *L_hi* denotes **filter.highLen**, the size of **initialCondition** must be 2 × `floor`{[Maximum(*L_lo*, *L_hi*)+1] / 2}-1.
finalCondition	double[ ]	The samples that decrease the artificial jumps at the end of the spectrogram. If you select **user defined** for **extension**, **finalCondition** is padded after the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the final condition. The size of **finalCondition** should be same as that of **initialCondition**.
shift	long	Determines the indexes of decimation. For this function, **shift** can be zero or one. Zero denotes even indexes, while one represents odd indexes.

Output

Name	Type	Description
waveletTransform	double[ ]	The 1D output data samples. You should pre-allocate memory for this parameter. The size of **waveletTransform** should be `floor`**[len / (2^pathLen)**].

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptAnalysisFilterBank2D

```
long SptAnalysisFilterBank2D(double * x, long rows, long cols,
SptFilter * analysisFilter, SptExtension extension, long shift, double
* lowLow, double * lowHigh, double * highLow, double * highHigh);
```

Purpose

Computes the output parameters of a 2D image that passes through an analysis filter bank.

Parameters

Input

Name	Type	Description
x	double *	The 2D input image data.
rows	long	Determines the number of rows in **x.**
cols	long	Determines the number of columns in **x**.
analysisFilter	SptFilter *	Sets the following analysis filter bank coefficients: **lowPass** contains the lowpass filter coefficients. **lowLen** is the array size of **lowPass**. **highPass** contains the highpass filter coefficients. **highLen** is the array size of **highPass**.
extension	SptExtension	The extension method that decreases the artificial jumps at the beginning and end of data samples. Select from the following extension methods: • zero padding: 0 • symmetrical: 1
shift	long	Determines the indexes of decimation. For this function, **shift** can be zero or one. Zero denotes even indexes, while one represents odd indexes.

Output

Name	Type	Description
lowLow	double *	The output of the first subimage from the analysis filter bank. You should pre-allocate memory for this parameter. If *L_lo* denotes **analysisFilter**.**lowLen** and *L_hi* denotes **analysisFilter**.**highLen**, the size of **lowLow** should be *size0* × *size1*, where the *size0* equals `ceil`{{**rows** + 2 × `floor`{[Maximum(*L_lo*, *L_hi*) + 1] / 2}– 1– **shift**} / 2.0} and *size1* equals `ceil`{{**cols** + 2 × `floor`{[Maximum(*L_lo*, *L_hi*) + 1] / 2}– 1– **shift**} / 2.0}.
lowHigh	double *	The output of the second subimage from the analysis filter bank. You should pre-allocate memory for this parameter. The size of **lowHigh** should be the same as the size of **lowLow**.
highLow	double *	The output of the third subimage from the analysis filter bank. You should pre-allocate memory for this parameter. The size of **highLow** should be the same as the size of **lowLow**.
highHigh	double *	The output of the fourth subimage from the analysis filter bank. You should pre-allocate memory for this parameter. The size of **highHigh** should be the same as the size of **lowLow**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptSynthesisFilterBank2D

```
long SptSynthesisFilterBank2D(double * lowLow, double * lowHigh,
double * highLow, double * highHigh, long rows, long cols, SptFilter
* synthesisFilter, long rowsOutput, long colsOutput, long shift,
double * output);
```

Purpose

Computes the 2D output of a synthesis filter bank. If the four images are output parameters from the same 2D Analysis Filter Bank function, `SptSynthesisFilterBank2D` reconstructs the four sub-images into the original image

Parameters

Input

Name	Type	Description
lowLow	double *	The first subimage from the analysis filter bank.
lowHigh	double *	The second subimage from the analysis filter bank.
highLow	double *	The third subimage from the analysis filter bank.
highHigh	double *	The fourth subimage from the analysis filter bank.
rows	long	Determines the number of rows in **lowLow, lowHigh, highLow, highHigh**.
cols	long	Determines the number of columns in **lowLow**, **lowHigh, highLow. highHigh**.
synthesisFilter	SptFilter *	Sets the following synthesis filters coefficients: **lowPass** contains the lowpass synthesis filter coefficients. **lowLen** is the array size of **lowPass**. **highPass** contains the highpass synthesis filter coefficients. **highLen** is the array size of **highPass**.

Name	Type	Description
rowsOutput	long	Determines the number of rows in the reconstructed 2D signal.
colsOutput	long	Determines the number of columns in the reconstructed 2D signal.
shift	long	Determines the number of rows and columns to be padded at the beginning of the subimages, before the filtering operation. In this function, **shift** can be zero or one.

Output

Name	Type	Description
output	double *	The reconstructed image of the signal. You should pre-allocate memory for this parameter. The size of **output** should be **rowsOutput** × **colsOutput**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptMotherWavScaling

```
long SptMotherWavScaling(SptFilter * filter, long refine, double
scaleFunc[ ], long * scaleFuncSize, double motherWav[ ], long *
waveSize, double * dt);
```

Purpose

Computes the mother wavelet and scaling function of a filter bank.

Parameters

Input

Name	Type	Description
filter	`SptFilter *`	Sets the following filter bank coefficients: **lowPass** contains the lowpass filter coefficients. **lowLen** is the array size of **lowPass**. **highPass** contains the highpass filter coefficients. **highLen** is the array size of **highPass**.
refine	long	Indicates how many levels of lowpass filters the function uses to calculate the Mother Wavelet and Scaling Function.

Output

Name	Type	Description
scaleFunc	double[ ]	The scaling function array. You should pre-allocate memory for this parameter. The size of **scaleFunc** should be [Maximum(**filters.lowLen, filters.highLen**) – 1] × 2^**refine**. The actual size of the array is returned in **scaleFuncSize**.
scaleFuncSize	long *	The array size of **scaleFunc**.

Name	Type	Description
motherWav	double[ ]	The mother wavelet array. You should pre-allocate memory for this parameter. The size of **motherWav** should be [Maximum(**filters.lowLen, filters.highLen**) – 1] × 2^**refine**. The actual size of the array is returned in **waveSize**.
waveSize	long *	The array size of **motherWav**.
dt	double *	Indicates the time duration between two points in the Mother Wavelet and Scaling Function output parameters.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptConWaveletTransform

```
long SptConWaveletTransform(double x[ ], long len, SptWaveletType
type, SptFilter * filter, long scales, double * conWaveletTransform,
SptWaveletPlot * plot);
```

Purpose

Computes continuous wavelet transform.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
type	SptWaveletType	Performs continuous wavelet transform. Choose from the following wavelet types: • **Mexican Hat**: 0 • **Meyr**: 1 • **Morlet**: 2 • **From analysis filter**: 3 If you select 3, this function refines **filter** to derive the wavelet.
filter	SptFilter *	Sets the following analysis filter bank coefficients: **lowPass** contains the lowpass analysis filter coefficients. **lowLen** is the array size of **lowPass**. **highPass** contains the highpass analysis filter coefficients. **highLen** is the array size of **highPass**.
scales	long	Determines the number of scales of wavelet transform. **scales** must be greater than zero.

Output

Name	Type	Description
conWaveletTransform	double *	Stores 2D real-valued wavelet transform. You should pre-allocate memory for this parameter. The size of **conWaveletTransform** should be **scales** × **len**.
plot	SptWaveletPlot*	Plot of the wavelet. It includes four parts: **Y**, **sizeY**, **X**, and **sizeX**. You should pre-allocate memory for **X** and **Y**. The sizes of **X** and **Y** are Maximum(size, 1024), where size equals [Maximum (**filters.lowLen**, **filters.highLen**) -1]*64.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptDeNoise1D

```
long SptDeNoise1D(double x[ ], long len, long scales,
SptThreshholdRule thresholdRule, long hardThreshold, SptScale
rescaling, SptExtension extension, SptFilter * analysisFilter,
SptFilter * synthesisFilter, long shift, double deNoised[ ]);
```

Purpose

Performs an automatic denoising process on a 1D signal using wavelets at a given level.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
scales	long	Determines the number of scales of the wavelet transform, which must be greater than zero.
thresholdRule	SptThreshholdRule	Denoises according to one of the following four threshold criteria: • rigrsure uses principle of Stein's Unbiased Rick. • heursure is a heuristic variant of the first option. • sqtwolog uses sqrt(2 $\times$ log (**len**)) as thresholding. • minmaxi uses the minimax method.
hardThreshold	long	Selects whether to use soft or hard thresholding. One represents hard thresholding while zero represents soft thresholding.

Name	Type	Description
rescaling	`SptScale`	Defines multiplicative threshold rescaling: • one for no rescaling. • sln for rescaling using a single estimation of level noise based on first level coefficients. • mln for rescaling using level dependent estimation of level noise.
extension	SptExtension	Selects the extension method that decreases the artificial jumps at the beginning and end of data samples. Select from the following extension methods: • zero padding: 0 • symmetrical: 1 • 1st derivative (smooth padding of order 1): 2 • 2nd derivative (smooth padding of order 2): 3
analysisFilter	SptFilter *	Sets the following analysis filter bank coefficients: **lowPass** contains the lowpass analysis filter coefficients. **lowLen** is the array size of **lowPass**. **highPass** contains the highpass analysis filter coefficients. **highLen** is the array size of **highPass**.

Name	Type	Description
synthesisFilter	SptFilter *	Sets the following synthesis filter bank coefficients: **lowPass** contains the lowpass synthesis filter coefficients. **lowLen** is the array size of lowPass. **highPass** contains the highpass synthesis filter coefficients. **highLen** is the array size of **highPass**.
shift	long	Determines the indexes of decimation. For this function, **shift** can be zero or one. Zero denotes even indexes, while one represents odd indexes.

Output

Name	Type	Description
deNoised	double[ ]	The denoised version of the input signal, **x,** obtained by thresholding the wavelet coefficients. You should pre-allocate memory for this parameter. The size of **deNoised** should be **len**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptDetrend

```
long SptDetrend(double x[ ], long len, double trendLevel, SptExtension
extension, double initialCondition[ ], double finalCondition[ ],
SptFilter * analysisFilter, SptFilter * synthesisFilter, long shift,
double detrend[ ], double trend[ ]);
```

Purpose

Removes trend from the given data samples.

Parameters

Input

Name	Type	Description
x	double[ ]	A 1D array of real data samples.
len	long	The array size of **x**.
trendLevel	double	Determines the similarity between **trend** and **x**. The range of **trendLevel** is from zero to one. As **trendLevel** increases, **trend** becomes more similar to **x**. When **trendLevel** is zero, the **trend** is equal to zero and consequently the **detrend** is identical to **x**. When **trendLevel** is one, the **trend** is the same as **x** and consequently the **detrend** becomes zero
extension	SptExtension	Selects the extension method that decreases the artificial jumps at the beginning and end of data samples. Select from the following extension methods: • zero padding: 0 • symmetrical: 1 • 1st derivative (smooth padding of order 1): 2 • 2nd derivative (smooth padding of order 2): 3 • user defined: 4

Name	Type	Description
initialCondition	double[ ]	The samples that decrease the artificial jumps at the beginning of the spectrogram. If you select **user defined** for **extension, initialCondition** is padded before the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the initial condition. If *L_lo* denotes **analysisFilter.lowLen** and *L_hi* denotes **analysisFilter.highLen**, the size of **initialCondition** must be 2 × `floor`{[Maximum(*L_lo*, *L_hi*)+1] / 2}-1.
finalCondition	double[ ]	The samples that decrease the artificial jumps at the end of the spectrogram. If you select **user defined** for **extension**, **finalCondition** is padded after the signal, **x**. Otherwise, the function ignores this parameter and automatically sets the final condition. The size of **finalCondition** should be same as the size of **initialCondition**.
analysisFilter	SptFilter *	Sets the following analysis filter bank coefficients: **lowPass** contains the lowpass analysis filter coefficients. **lowLen** is the array size of **lowPass**. **highPass** contains the highpass analysis filter coefficients. **highLen** is the array size of **highPass**.

Name	Type	Description
synthesisFilter	SptFilter *	Sets the following synthesis filter bank coefficients: **lowPass** contains the lowpass synthesis filter coefficients. **lowLen** is the array size of **lowPass**. **highPass** contains the highpass synthesis filter coefficients. **highLen** is the array size of **highPass**.
shift	long	Determines the indexes of decimation. For this function, **shift** can be zero or one. Zero denotes even indexes, while one represents odd indexes.

Output

Name	Type	Description
detrend	double[ ]	Stores the difference between **x** and **trend**. You should pre-allocate memory for this parameter. The size of **detrend** should be **len**.
trend	double[ ]	Stores computed trend of **x**. You should pre-allocate memory for this parameter. The size of **trend** should be **len**.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptAllocCoeffWFBD

```
void * filterCoeff = SptAllocCoeffWFBD(void);
```

Purpose

Allocates the WFBD filter bank coefficients structure. You must call this function once to properly allocate the WFBD filter coefficients structure. Call the `SptFreeCoeffWFBD` function to deallocate the structure.

SptReadCoeffWFBD

```
long SptReadCoeffWFBD(char coeffPath[ ], SptFilter * analysisFilter,
SptFilter * synthesisFilter);
```

Purpose

Reads the analysis and synthesis filter bank coefficients from a text file. You must call Allocated Coefficient WFBD first to allocate filter bank structures for both analysis filter banks and synthesis filter banks.

Parameters

Input

Name	Type	Description
coeffPath	char[ ]	Denotes the path of the text file to be read.

Output

Name	Type	Description
analysisFilter	SptFilter *	Holds the analysis filter bank coefficients. If this pointer is set to **NULL**, the function does not read the analysis filter bank coefficients.
synthesisFilter	SptFilter *	Holds the synthesis filter bank coefficients. If this pointer is set to **NULL**, the function does not read the synthesis filter bank coefficients.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

SptFreeCoeffWFBD

```
long SptFreeCoeffWFBD(void * fptr);
```

Purpose

This function frees the WFBD filter bank coefficients structure and all of its coefficient arrays.

Parameters

Input

Name	Type	Description
fptr	void *	Points to allocated filter bank structure.

Return Value

Name	Type	Description
status	long	Refer to Appendix B, *SPT for LabWindows/CVI Error Codes*, for error codes.

17

Using Your Coefficient Designs with DFD Utilities

This chapter describes the DFD utilities you use for filtering applications.

LabWindows/CVI Utilities

This section contains descriptions of the DFD utilities you can use within your LabWindows/CVI applications. With these utilities, you can read DFD filter coefficient files and use the coefficients to filter your data.

The DFD Instrument Driver

The DFD toolkit provides a LabWindows/CVI instrument driver file named DFDUTILS.FP. You can find this file in the CVI Support\instr subdirectory of your Digital Filter Design installation directory.

The DFD utility functions contained in the instrument driver DFDUTILS.FP use a filter coefficient structure to hold the filter coefficients. The header file DFDUTILS.H contains this filter structure and the following four DFD utility function prototypes:

```
#define intnum long
#define floatnum double
typedef struct {
   intnum type;  /* type of filter (1p,hp,bp,bs) */
   intnum order; /* order of filter */
   intnum reset; /* 0 - don't reset, 1- reset */
   intnum a;     /* number of a coefficients */
   floatnum *a;  /* pointer to a coefficients */
   intnum nb;    /* number of b coefficients */
   floatnum *b;  /* pointer to b coefficients */
   intnum ns;    /* number of internal states */
   floatnum *s;  /* pointer to internal state array */
} FilterStruct, *FilterPtr;
FilterPtr AllocCoeffDFD (void);
long ReadCoeffDFD (char coeffPath[], FilterPtr filterCoefficients,
   double *samplingrate);
```

```
long FilterDFD (double inputArray[], long n,
   FilterPtr filterCoefficients, double outputArray[]);
long FreeCoeffDFD (FilterPtr filterCoefficients);
```

Using the DFD Instrument Driver

Add the `DFDUTILS.FP` to your project and `DFDUTILS.H` to your source code. Now you can call the DFD utility functions in your C code. An example called `DFDXMPL.PRJ`, in the `CVI Support\example` subdirectory, shows you how to call the DFD utility functions.

AllocCoeffDFD

```
FilterPtr fptr = AllocCoeffDFD (void);
```

Purpose

Allocates and clears the DFD filter coefficient structure. You must call this function once to allocate the DFD filter coefficient structure properly.

Return Value

Name	Type	Description
fptr	FilterPtr	Pointer allocated to filter structure.

ReadCoeffDFD

```
long err = ReadCoeffDFD (char coeffPath[], FilterPtr
filterCoefficients, double * samplingRate);
```

Purpose

Reads the DFD filter coefficient file. You must call `AllocCoeffDFD` once before calling this function.

Parameters

Input

Name	Type	Description
coeffPath	char[ ]	Pathname of DFD coefficient file.

Output

Name	Type	Description
filterCoefficients	FilterPtr	Pointer to filter-coefficient structure.
samplingRate	double *	Pointer to sampling rate.

Return Value

Name	Type	Description
err	long integer	Error code.

FreeCoeffDFD

```
long err = FreeCoeffDFD (FilterPtr filterCoefficients);
```

Purpose

Releases the DFD filter coefficient structure and all its coefficient arrays.

Parameters

Input

Name	Type	Description
filterCoefficients	FilterPtr	Pointer to filter-coefficient structure.

Return Value

Name	Type	Description
err	long integer	Contains the error code.

FilterDFD

```
long err = FilterDFD (double inputArray[], long n,
FilterPtr filterCoefficients, double outputArray[]);
```

Purpose

Filters the input samples using the DFD filter coefficients. You must call `AllocCoeffDFD` and `ReadCoeffDFD` once before calling this function.

Use this function to filter blocks of one continuous sequence of input samples. The input state of the filter is maintained using the DFD filter coefficient structure. The number of output samples equals the number of input samples (**n**).

Parameters

Input

Name	Type	Description
inputArray	double[ ]	Input array of unfiltered samples.
n	long	Number of elements in input array.
filterCoefficients	FilterPtr	Pointer to filter-coefficient structure.

Output

Name	Type	Description
outputArray	double[ ]	Output array of filtered samples that must be at least as large as **inputArray**.

Return Value

Name	Type	Description
err	long	Contains the error code.

Windows DLL DFD Utilities

This section contains descriptions of the DFD utilities you can use from within your Windows NT/95 applications. With these utilities, you can read DFD filter coefficient files and filter your data using the coefficients.

When you install the DFD application, the software installs a 32-bit DLL named `DFD32.DLL` for Windows NT/95 users. This DLL is located in the `Libraries` subdirectory of your installation directory, along with the header file `Dfdutils.h`.

The DFD, DLL, and header file have the following function prototypes:

```
FilterPtr AllocCoeffDFD (void);
long ReadCoeffDFD (char coeffPath[], FilterPtr filterCoefficients,
   double *samplingrate);
long FreeCoeffDFD (FilterPtr filterCoefficients);
long FilterDFD (double inputArray[], long n,
   FilterPtr filterCoefficients, double outputArray[]);
```

Refer to the descriptions for each function and its parameters in the previous section, *LabWindows/CVI Utilities*. Call these functions in your code the same way you call other DLL functions.

The DFD toolkit also provides an example for Visual Basic 4.0 that shows you how to call the DFD utility functions. The source code is in the `DFDUTILS\WINSRC\EXAMPLE\VB` subdirectory of your installation directory.

A

Frequently Asked Questions

This chapter addresses some questions users frequently ask.

What is the difference between linear and quadratic JTFA methods?

This package includes both linear and quadratic methods. Linear JTFA transforms include the following methods:

- Gabor expansion, which is the inverse of short-time Fourier transform (STFT)
- STFT, which computes the Gabor coefficients
- adaptive representation, which is the inverse of adaptive transform
- adaptive transform

Quadratic JTFA algorithms include the following methods:

- STFT spectrogram
- Wigner–Ville Distribution (WVD)
- Pseudo Wigner–Ville Distribution (PWVD)
- Cohen’s class
- Choi–Williams Distribution (CWD)
- cone-shaped distribution
- Gabor spectrogram
- adaptive spectrogram

If you consider linear JTFA to be an evolved form of conventional Fourier transform, then quadratic JTFA is the counterpart of the standard power spectrum. The difference between using linear and quadratic JTFA methods is that you can invert linear transform. As with fast-Fourier transform (FFT), you can use the Gabor coefficients to reconstruct the original signal. Linear transform is suitable for signal processing, such as time-varying filtering.

In general, the quadratic form is not reversible. You cannot restore the original time waveform from the time-dependent spectrum. However, quadratic JTFA describes the energy distribution of the signal in the joint time-frequency domain, which is useful for signal analysis.

Which quadratic JTFA algorithms should I use?

Each quadratic JTFA algorithm has advantages and disadvantages. You should select an algorithm that fits the application. Generally speaking, no algorithm is superior to all others in all applications. Table A-1 summarizes the advantages and disadvantages of all quadratic algorithms provided in this package.

Table A-1. Quadratic JTFA Algorithms

Method	Resolution and Crossterm Description	Speed
adaptive spectrogram	extremely high resolution when a signal is made up of Gaussian pulses no crossterms non-negative	slow
CWD	less crossterm than PWVD	very slow
cone-shaped distribution	less crossterm interference than PWVD or CWD	slow
Gabor spectrogram	good resolution robust minor crossterms	moderate
PWVD	extremely high resolution for a few types of signals severe crossterms	fast
STFT spectrogram	poor resolution robust non-negative	fast

If the frequency contents of the analyzed signal do not change rapidly, try the STFT spectrogram first. You can apply a relatively long window function to obtain a good frequency resolution with tolerable time resolution deterioration. Because the STFT spectrogram is fast, it is suitable for online analysis.

The other algorithms generally have better joint time and frequency resolution than the STFT spectrogram, but they require more computation time, which is only suitable for offline analysis. If you need a higher resolution, use the third- or fourth-order Gabor spectrogram to reduce crossterm interference and achieve faster processing speeds.

National Instruments recommends that you use the following procedure when you analyze a signal:

1. Begin with the STFT and determine which analysis window is best—wideband, mediumband, or narrowband.
2. If you are satisfied with the results, use STFT. If not, continue with step 3.
3. If the STFT does not produce satisfactory results, try the Gabor spectrogram. Regardless of the analysis window you use, as the order increases, the Gabor spectrogram converges to the Wigner-Ville Distribution. If the **order** is low, the type of the analysis window influences the Gabor spectrogram, although the effect is not as large as the effect type has on the STFT. Select your Gabor spectrogram analysis window based on the window information obtained in step 1.
4. Increase the **order** until the crossterm interference is evident. For most applications, an **order** of three to five is adequate.
5. Reduce the data **block length** and increase the **freq. zoom** to examine detailed features.

Can I measure a signal’s energy point-to-point in the joint time-frequency domain?

This question addresses a fundamental issue in the joint time-frequency analysis area. Except for a few special cases, the answer is no. At this point, scientists know that no algorithm can meaningfully measure a signal’s energy point-to-point in the joint time-frequency domain.

Roughly speaking, the result, $P(t,w)$, of all quadratic JTFA algorithms indicates a certain type of weighted average energy near the point (t, f). Some algorithms take the average over a larger area, such as the STFT spectrogram. In this case, the time-frequency resolution is poor, but it is always greater than or equal to zero. Some methods cause heavy weights on a small number of points, such as the high-order Gabor spectrogram, which yields better time-frequency resolution. In this case, some points might approach negativity, which is not acceptable for certain applications. In short, every algorithm has advantages and disadvantages.

Figure A-1 shows an STFT spectrogram with a test signal that contains 10 sine cycles at 10 Hz. Although the signal starts at *t* = 1 s and ends at *t* = 2 s, the STFT spectrogram clearly shows something before *t* = 1 s and after *t* = 2 s, as indicated by the arrows. The time-dependent spectrum indicates that the signal not only contains 10 Hz, but that it possesses a certain bandwidth.

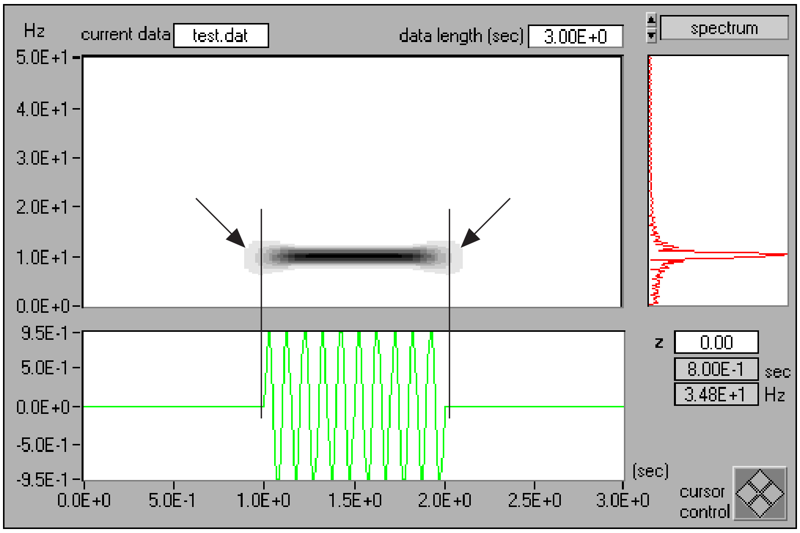


Figure A-1. STFT Spectrogram (Hanning Window)

You can apply other methods to substantially suppress the energy outside 1 s to 2 s and 10 Hz, and achieve a near point-to-point measurement. Figure A-2, shows the Gabor spectrogram. Most of the signal's energy is between 1 s to 2 s and 10 Hz. As the **order** increases, the concentration also increases, and you come closer to achieving a point-to-point measurement. However, a higher order Gabor spectrogram produces negative values, which can cause problems with the classical energy definition. The Gabor spectrogram generally requires more computation time than the STFT spectrogram.

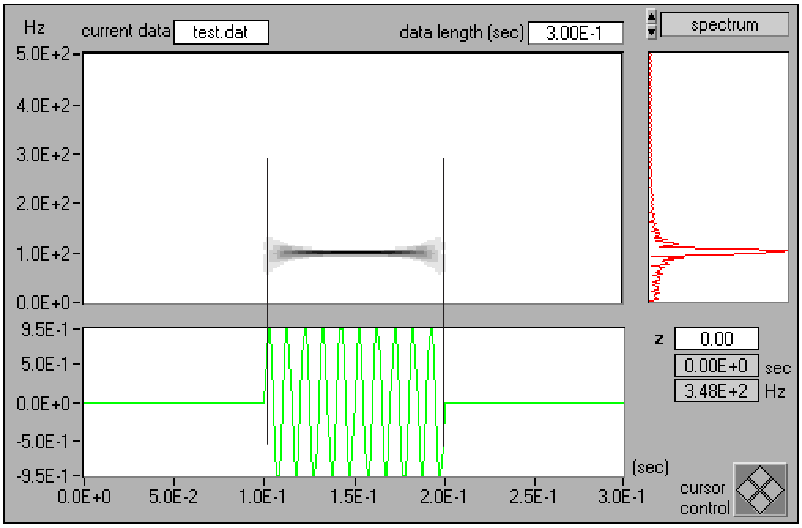


Figure A-2. Gabor Spectrogram (Order Four)

What can I do if the time-dependent spectrum only shows a line at DC?

If the analyzed signal is non-negative, such as an ECG or the stock index, or if it contains a large DC offset, the resulting time-dependent spectrum is dominated by a single line in the vicinity of DC. You might not be able to see more interesting frequency patterns. To suppress the DC component, you have to apply certain types of preprocessing. However, the methods for removing the DC components, or detrending, are application dependent. No general method works in all cases. Common techniques of detrending include lowpass filtering and curve fitting. However, a more promising technique is the wavelet transform. Refer to Part IV, *Wavelet Analysis*, of this manual for information on wavelet-based detrending.

Can I use other software to plot the time-dependent spectrum?

Yes. Save the time-dependent spectrum to a text file. The resulting text file contains only *Z* values and does not retain the time and frequency axis information. The time and frequency axis can be determined as follows.

When **t0** and **f0** are shown on the front panel of **Offline Analyzer**, the time increment Δt is computed by the following equation:

$$\Delta t = \frac{time\ span}{number\ of\ rows}$$

and the frequency increment Δf is determined by the following equation:

$$\Delta f = \frac{sampling\ frequency}{2 \times zoom\ factor \times 128}$$

B

SPT for LabWindows/CVI Error Codes

This chapter contains the error codes that the functions in the Signal Processing Toolset Library return. If an error condition occurs during a call to any of the functions in the LabWindows/CVI Analysis Library, the return value status contains the returned error code. Each error code specifies a different type of error. The following table, SPT library Error Codes Sorted Alphabetically, lists the error codes alphabetically by symbolic name. For your convenience, the SPT library Error Codes Sorted Numerically table lists the error codes in numeric order.

SPT Library Error Codes Sorted Alphabetically

Symbolic Name	Code	Error Message
2/3SmplLenGEAROrderErr	–20098	The AR order must not be greater than two-thirds of the number of samples.
2/3SmpLenGENumOfSinErr	–20097	The number of complex sinusoids must not be greater than two-thirds of the number of samples.
AROrderGENumOfSinErr	–20099	The AR order must be greater than or equal to the number of complex sinusoids.
AROrderGTZeroErr	–20096	The AR order must be greater than zero.
ArraySizeErr	–20008	The input arrays do not contain the correct number of data values for this function.
ConditionsLengthErr	–20094	The size of the initial or final condition array is incorrect.
DfGTZeroErr	–20093	dF must be greater than zero.
DivByZeroErr	–20060	Divide by zero error.
DtGTZeroErr	–20016	dt must be greater than zero.

Symbolic Name	Code	Error Message
FsGTZeroErr	–20504	The sampling frequency must be greater than zero.
GaborCoefErr	–20091	The Gabor coefficients array has the wrong dimensions.
GabordMErr	–20087	The length of the analysis or synthesis window must be evenly divisible by the Gabor time sampling interval.
GabordNErr	–20081	dN or the time interval must be greater than zero.
GaborNErr	–20088	The length of the analysis or synthesis window must be evenly divisible by the number of frequency bins. N must be a power of two.
GaborOrderErr	–20090	The order of the Gabor spectrogram must be greater than zero.
GaborOversamplErr	–20089	The oversampling rate, N/dM, must be greater than one.
GaborTIErr	–20082	The time increment cannot be greater than dM.
IndexLTSamplesErr	–20017	The index must be greater than or equal to zero, but less than the samples.
InvSelectionErr	–20061	The selection is invalid.
JTFAHilbertErr	–20085	The size of the input array and its Hilbert transform must be equal.
JTFATIErr	–20084	The time increment cannot be greater than the window length divided by four.
JTFAWindowLErr	–20083	Window length must be greater than four and a power of two.
MaxIterErr	–20062	The application exceeded the maximum number of iterations.
NoDual FunctionErr	–20092	The dual function does not exist.
NoErr	0	No error.
NumofSinGTZeroErr	–20095	The number of complex sinusoids must be greater than zero.
OrderGTZeroErr	–20021	The order must be greater than zero.
OutOfMemErr	–20001	There is not enough space left to perform the specified routine.

Symbolic Name	Code	Error Message
PolyErr	–20063	The coefficients of the polynomial are invalid.
PowerOfTwoErr	–20009	The size of the input array must be a power of two: size = 2^m, 0 < m < 23.
RankDeficient	20001	The matrix is rank deficient.
SamplesGETwoErr	–20006	The number of samples must be greater than or equal to two.
SamplesGEZeroErr	–20004	The number of samples must be greater than or equal to zero.
SamplesGTZeroErr	–20003	The number of samples must be greater than zero.
SingularMatrixErr	–20041	The system of equations cannot be solved because the input matrix is singular.
SizeGTOrderErr	–20037	The array size must be greater than the order.
SquareMatrixErr	–20040	The input matrix must be a square matrix.
STFTWindowLerr	–20086	The window length must be greater than two and a power of two.

SPT Library Error Codes Sorted Numerically

Code	Symbolic Name	Error Message
–20001	OutOfMemErr	There is not enough space left to perform the specified routine.
–20003	SamplesGTZeroErr	The number of samples must be greater than zero.
–20004	SamplesGEZeroErr	The number of samples must be greater than or equal to zero.
–20006	SamplesGETwoErr	The number of samples must be greater than or equal to two.
–20008	ArraySizeErr	The input arrays do not contain the correct number of data values for this function.
–20009	PowerOfTwoErr	The size of the input array must be a power of two: size = 2^m, 0 < m < 23.

Code	Symbolic Name	Error Message
-20016	DtGTZeroErr	dt must be greater than zero.
–20017	IndexLTSamplesErr	The index must be greater than or equal to zero, but less than the samples.
–20021	OrderGTZeroErr	The order must be greater than zero.
–20037	SizeGTOrderErr	The array size must be greater than the order.
–20040	SquareMatrixErr	The input matrix must be a square matrix.
–20041	SingularMatrixErr	The system of equations cannot be solved because the input matrix is singular.
–20060	DivByZeroErr	Divide by zero error.
–20061	InvSelectionErr	The selection is invalid.
–20062	MaxIterErr	The application exceeded the maximum number of iterations.
–20063	PolyErr	The coefficients of the polynomial are invalid.
–20081	GabordNErr	dN or the time interval must be greater than zero.
–20082	GaborTIErr	The time increment cannot be greater than dM.
–20083	JTFAWindowLErr	Window length must be greater than four and a power of two.
–20084	JTFATIErr	The time increment cannot be greater than the window length divided by four.
–20085	JTFAHilbertErr	The size of the input array and its Hilbert transform must be equal.
–20086	STFTWindowLErr	The window length must be greater than two and a power of two.
–20087	GabordMErr	The length of the analysis or synthesis window must be evenly divisible by the Gabor time sampling interval.
–20088	GaborNErr	The length of the analysis or synthesis window must be evenly divisible by the number of frequency bins. N must be a power of two.
–20089	GaborOversamplErr	The oversampling rate, N/dM, must be greater than one.

Code	Symbolic Name	Error Message
–20090	GaborOrderErr	The order of the Gabor spectrogram must be greater than zero.
–20091	GaborCoefErr	The Gabor coefficients array has the wrong dimensions.
–20092	NoDualFunctionErr	The dual function does not exist.
–20093	DfGTZeroErr	dF must be greater than zero.
–20094	ConditionsLengthErr	The size of the initial or final condition array is incorrect.
–20095	NumofSinGTZeroErr	The number of complex sinusoids must be greater than zero.
–20096	AROrderGTZeroErr	The AR order must be greater than zero.
–20097	2/3SmplLenGENumofSinErr	The number of complex sinusoids must not be greater than two-thirds of the number of samples.
–20098	2/3SmplLenGEAROrderErr	The AR order must not be greater than two-thirds of the number of samples.
–20099	AROrderGENumofSiErr	The AR order must be greater than or equal to the number of complex sinusoids.
–20504	FsGTZeroErr	The sampling frequency must be greater than zero.
0	NoErr	No error.
20001	RankDeficient	The matrix is rank deficient.

C References

This appendix lists reference material that contains more information on the theory and algorithms implemented in the Signal Processing Toolset.

[1] Bastiaans, B., “A sampling theorem for the complex spectrogram, and Gabor’s expansion of a signal in Gaussian elementary signals,” *Optical Engineering*, vol. 20, no. 4, pp. 594-598, July/August 1981.

[2] Choi, H., and W. J. Williams. “Improved time-frequency representation of multicomponent signals using exponential kernels.” *IEEE Trans. Acoustics, Speech, Signal Processing* vol. 37.6 (1989):862–871.

[3] Claasen, T. A. C. M., and W. F. G. Mecklenbräuker, “The Wigner Distribution–A tool for time-frequency signal analysis,” *Phillips J. Res*., vol. 35, pp. 217-250, pp. 276-300, pp. 1067-1072, 1980.

[4] Cohen, L. “Generalized phase-space distribution functions.” *J. Math. Phys.* vol. 7 (1966):781–806.

[5] Cohen, L. “Time-frequency distribution: A review.” *Proc. IEEE* vol. 77.7 (1989):941–981.

[6] Cohen, L. *Time-frequency analysis*. Englewood Cliffs, N.J.: Prentice-Hall, 1995.

[7] Crochiere, R. E., and L. R. Rabiner. *Multirate Digital Signal Processing*. Englewood Cliffs, N.J.: Prentice-Hall, 1983.

[8] Donoho, D. L. “De-noise by soft thresholding.” *IEEE Transactions Information Theory* vol. 3.41 (1995): 613–627.

[9] Haar, A., “Zur Theorie der Orthogonalen Funktionenysystem,” Math. Annal., Vol. 69, pp. 331-371, 1910.

[10] Hua, Y., and T. K. Sarkar. “Matrix Pencil Method for Estimating Parameters of Exponentially Damped/Undamped Sinusoids in Noise.” *IEEE Transaction on Acoustic, Speech, and Signal Processing* vol. 38.5 (1990): 814–824.

[11] Hubbard, B. B., The World According to Wavelets, A K Peters, Second Edition, Wellesley, Massachusetts, 1998.

[12] Gabor, D.,“Theory of communication,” *J. IEE (London),* vol. 93, no. III, pp. 429-457, November 1946.

[13] Jackson, L. B. *Digital Filters and Signal Processing*. Boston: Kluwer, 1986.

[14] Kay, S. M. *Modern Spectral Estimation*. Englewood Cliffs, N.J.: Prentice-Hall, 1987.

[15] Kolmogorov, A. N. "Interpolation and Extrapolation von Stationaren Zufalligen Folgen." *Bull. Acad. Sci. USSR., Ser. Math.* vol. 5 (1941): 3–14.

[16] Mallat, S., and Z. Zhang. "Matching Pursuits with Time-Frequency Dictionaries," *IEEE Trans. Signal Process.* vol. 41.12 (1993): 3397–3415.

[17] Mallat, S., *A Wavelets Tour of Signal Processing*, Academic Press, 1998.

[18] Marple, Jr., S. L. *Digital Spectral Analysis with Applications*. Englewood Cliffs, N.J.: Prentice-Hall, 1987.

[19] Oppenheim, A. V., and R. W. Schafer. *Discrete-Time Signal Processing*. Englewood Cliffs, N.J.: Prentice-Hall, 1989.

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

[21] Parks, T. W., and J. H. McClellan. "A Program for the Design of Linear Phase Finite Impulse Response Filters." *IEEE Trans. Audio Electroacoustics* vol. AU-20.3 (Aug. 1972a): 195–199.

[22] Parks, T. W., and J. H. McClellan. "Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase." *IEEE Trans. Circuit Theory* vol. CT- 19 (March 1972a): 189–194.

[23] Qian, S., and J. M. Morris. "Wigner distribution decomposition and crossterm deleted representation." *Signal Processing* vol. 25.2 (1992):125–144.

[24] Qian, S., and D. Chen. "Discrete Gabor transform." *IEEE Trans. Signal Processing* vol. 41.7 (1993):2429–2439.

[25] Qian, S., and D. Chen. "Signal Representation Using Adaptive Normalized Gaussian Functions." *IEEE Trans. Signal Processing* vol. 36.10 (1994):1–11.

[26] Qian, S., and D. Chen. "Decomposition of the Wigner-Ville distribution and time-frequency distribution series." *IEEE Trans. Signal Processing* vol. 42.10 (1994):2836–2841.

[27] Qian, S., and D. Chen. *Joint Time-Frequency Analysis*. Englewood Cliffs, N.J.: Prentice-Hall, 1996.

[28] Qian, S., and D. Chen, "Joint time-frequency analysis," *IEEE Signal Processing Magazine*, vol. 16, No. 2, pp. 52-67, March 1999.

[29] Shannon C.E., and W. Weaver, *The Mathematical Theory of Communication*, The University of Illinois Press, IL, 1964.

[30] Strang, G., and T. Nguyen. *Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1995.

[31] Vaidyanathan, P.P. *Multirate Systems and Filter Banks*, Englewood Cliffs, N.J.: Prentice-Hall, 1993.

[32] Vaidyanathan, P.P., and T. Nguyen. "A 'Trick' for the Design of FIR Half-Band Filters." *IEEE Transactions on Circuits and Systems* vol. 3 (March 1987): 297–300.

[33] Ville, J.,"Thovrie et applications de la notion de signal analylique," (in French) *Câbles et Transmission*, vol. 2, pp. 61-74, 1948.

[34] Wexler, J., and S. Raz. "Discrete Gabor expansions." *Signal Processing*, vol. 21.3 (1990):207–221.

[35] Whittaker, J., "Interpolaroty Function Theory," *Cambridge Tracts in Math. and Math. Physics*, vol.33, 1935.

[36] Wigner, E. P., "On the Quantum Correction for Thermodynamic Equilibrium," *Phys. Rev.*, vol. 40, pp. 749, 1932.

[37] Williams, A. B., and F. J. Taylor. *Electronic Filter Design Handbook*. New York: McGraw-Hill, 1988.

[38] Wold, H. *Stationary Time Series*. Uppsala, Sweden: Almqvist and Wiksell, 1938, republished 1954.

[39] Xia, X. G., and S. Qian. "Convergence of an iterative time-variant filtering based on discrete Gabor transform." *IEEE Trans. on Signal Processing* vol. 47 (1999).

[40] Zhao, Y., L. E. Atlas, and R. J. Marks. "The use of cone-shaped kernels for generalized time-frequency representations of nonstationary signals." *IEEE Trans. Acoustics, Speech, Signal Processing*, vol. 38.7 (1990):1084–1091.

D

Technical Support Resources

Web Support

National Instruments Web support is your first stop for help in solving installation, configuration, and application problems and questions. Online problem-solving and diagnostic resources include frequently asked questions, knowledge bases, product-specific troubleshooting wizards, manuals, drivers, software updates, and more. Web support is available through the Technical Support section of `ni.com`.

NI Developer Zone

The NI Developer Zone at `ni.com/zone` is the essential resource for building measurement and automation systems. At the NI Developer Zone, you can easily access the latest example programs, system configurators, tutorials, technical news, as well as a community of developers ready to share their own techniques.

Customer Education

National Instruments provides a number of alternatives to satisfy your training needs, from self-paced tutorials, videos, and interactive CDs to instructor-led hands-on courses at locations around the world. Visit the Customer Education section of `ni.com` for online course schedules, syllabi, training centers, and class registration.

System Integration

If you have time constraints, limited in-house technical resources, or other dilemmas, you may prefer to employ consulting or system integration services. You can rely on the expertise available through our worldwide network of Alliance Program members. To find out more about our Alliance system integration solutions, visit the System Integration section of `ni.com`.

Worldwide Support

National Instruments has offices located around the world to help address your support needs. You can access our branch office Web sites from the Worldwide Offices section of `ni.com`. Branch office Web sites provide up-to-date contact information, support phone numbers, e-mail addresses, and current events.

If you have searched the technical support resources on our Web site and still cannot find the answers you need, contact your local office or National Instruments corporate. Phone numbers for our worldwide offices are listed at the front of this manual.

Glossary

Numbers/Symbols

%	Percent.
∞	Infinity.
π	Pi.
1D	One-dimensional.
2D	Two-dimensional.

A

A/D	Analog to digital conversion.
alias term	An image term in the frequency domain.
alternating flip	For a periodic sequence $g[n]$ with a period N, the sequence $(-1)^n g[N-n]$ is considered the alternating flip of $g[n]$.
analysis filter bank	A filter bank that converts a signal from time domain into wavelet domain.
ANSI	American National Standards Institute.
ANSI S1.11-1986	Specifications for octave-band and fractional-octave-band analog and digital filters.
array	Ordered, indexed set of data elements of the same type.
ASCII	American Standard Code for Information Interchange.

B

basis	A core or fundamental function.
biorthogonal filter bank	A filter bank in which analysis and synthesis filter banks are orthogonal to each other.
block diagram	Pictorial description or representation of a program or algorithm. In G, the block diagram is the source code for the VI. It consists of executable icons called nodes as well as wires that carry data between the nodes.
Boolean controls and indicators	Front panel objects used to manipulate and display Boolean (TRUE or FALSE) data. Several styles are available, such as switches, buttons, and LEDs.
Butterworth filter	A special kind of filter in which the low-frequency asymptope is a constant.

C

chart	2D display of one or more plots in which the display retains previous data, up to a maximum that you define. The chart receives the data and updates the display point by point or array by array, retaining a certain number of past points in a buffer for display purposes. *See also* scope chart, strip chart, and sweep chart.
checkbox	Small square box in a dialog box you can select or clear. Checkboxes generally are associated with multiple options that you can set. You can select more than one checkbox.
CIN	*See* Code Interface Node (CIN).
cluster	Set of ordered, unindexed data elements of any data type including numeric, Boolean, string, array, or cluster. The elements must all be controls or all indicators.
Code Interface Node (CIN)	CIN. Special block diagram node through which you can link conventional, text-based code to a VI.
constant Q analysis	Analysis in which the ratio between the center frequency and frequency bandwidth is constant.
continuous run	Execution mode in which a VI is run repeatedly until the operator stops it. You enable it by clicking the **Continuous Run** button.

control	Front panel object for entering data to a VI interactively or to a subVI programmatically, such as a knob, push button, or dial.
CWD	Choi–Williams distribution.

D

DAQ	*See* data acquisition.
data acquisition	DAQ. Process of acquiring data, typically from A/D or digital input plug-in boards.
data type	Format for information. In BridgeVIEW, acceptable data types for tag configuration are analog, discrete, bit array, and string. In LabVIEW, acceptable data types for most functions are numeric, array, string, and cluster.
datalog file	File that stores data as a sequence of records of a single, arbitrary data type that you specify when you create the file. While all the records in a datalog file must be of a single type, that type can be complex; for instance, you can specify that each record is a cluster containing a string, a number, and an array.
Daubechies wavelet and filter bank	Wavelet and filter bank that has a maximum number of zeros at π. The wavelet and filter bank was initially developed by Ingrid Daubechies.
dB	Decibels. A logarithmic unit for measuring ratios of amplitude levels. If the amplitudes are specified in terms of power, then $1\text{ dB} = 10 \times \log 10(P/Pr)$ where *P* is the measured power and *Pr* is the reference power. If the amplitudes are specified in terms of voltage, then $1\text{ dB} = 20 \times \log 10(V/Vr)$ where *V* is the measured voltage and *Vr* is the reference voltage.
decimation filter	The output of the filter does not preserve all points.
denoise	Remove the noise from the original signal.
developer	*See* system developer.
DFD	Digital Filter Design.
DFT	Discrete Fourier transform.
DGT	Discrete Gabor transform.

dialog box	Window that appears when an application needs further information to carry out a command.
distortion term	A term that causes distortion in a filter output.
DLL	Dynamic Link Library.
DSA	Dynamic Signal Acquisition.
DSP	Digital Signal Processing.

E

equiripple filter	A filter with equiripples in the passband and stopband.
error message	Indication of a software or hardware malfunction, or an unacceptable data entry attempt.

F

FFT	Fast Fourier Transform, an efficient and fast method for calculating the discrete Fourier transform. The number of samples must usually be a power of two. The fast Fourier transform (or the discrete Fourier transform) determines the amplitude and phase of the frequency components present in a time domain digital signal.
filter bank	A group of filters.
finite impulse response filter	A filter without feedback that only contains zeros in the *z*-domain.
FIR	Finite impulse response.
Formula Node	Node that executes formulas you enter as text. Formula nodes are especially useful for lengthy formulas that would be cumbersome to build in block diagram form.
frame	Segment of time domain data.

front panel	Interactive user interface of a VI. Modeled after the front panel of physical instruments, it is composed of switches, slides, meters, graphs, charts, gauges, LEDs, or other controls or indicators.
function	Built-in execution element, comparable to an operator, function, or statement in a conventional language.

G

G	Graphical programming language used to develop LabVIEW and BridgeVIEW applications.
General Purpose Interface Bus.	GPIB—Common name for the communications interface system defined in ANSI/IEEE Standard 488.1-1987 and ANSI/IEEE Standard 488.2-1987.
global variable	Non-reentrant subVI with local memory that uses an uninitialized shift register to store data from one execution to the next. The memory of copies of these subVIs is shared and thus can be used to pass global data between them.
GPIB	See General Purpose Interface Bus.

H

halfband filter	A filter with a cut-off frequency at half of the frequency band.
Help	Online instructions that explain how to use a Windows application. The **Help** menu displays specific Help topics. Pressing `<F1>` displays a list of Help topics.
Help window	Special window that displays the names and locations of the terminals for a function or subVI, the description of controls and indicators, the values of universal constants, and descriptions and data types of control attributes.
Hz	Hertz. Cycles per second.

I

I/O	Input/output. Transfer of data to or from a computer system involving communications channels, operator input devices, and/or data acquisition and control interfaces.
icon	Graphical representation of a node on a block diagram.
IEEE	Institute for Electrical and Electronic Engineers.
IIR filters	Infinite impulse response filters.
image compression	Using only part of the data to recover the original image.
indicator	Front panel object that displays output.
inner product	A mathematical operation used to test the difference between two functions.

J

JTFA	Joint time-frequency analysis.

L

LabVIEW	Laboratory Virtual Instrument Engineering Workbench. Program development application based on the programming language G used commonly for test and measurement purposes.
local variable	Variable that enables you to read or write to one of the controls or indicators on the front panel of your VI.

M

matrix	Two-dimensional array.
maximum flat filter	A type I filter that has a maximum number of zeros at π.
MB	Megabytes of memory.
mother wavelet	An elementary wavelet.

multicomponent signal	A signal containing significant energy concentrated around more than one distinct and separate frequency.
multiscale analysis	Analyzing a signal in several different scales.

N

node	Program execution element. Nodes are analogous to statements, operators, functions, and subroutines in conventional programming languages. In a block diagram, nodes include functions, structures, and subVIs.
nonstationary signal	Signal whose frequency content changes within a captured frame.
numeric controls and indicators	Front panel objects used to manipulate and display, or input and output, numeric data.
Nyquist rate	Half the sampling rate.

O

object	Generic term for any item on the front panel or block diagram, including controls, nodes, wires, and imported pictures.
octave	Interval between two frequencies, one of which is twice the other. For example, frequencies of 250 Hz and 500 Hz are one octave apart, as are frequencies of 1 kHz and 2 kHz. *See also* third-octave.
orthogonal filter bank	A filter bank where both the analysis and synthesis filter banks are orthogonal to themselves. It is a special case of biorthogonal filter banks.
oversampling rate	Ratio between the number of Gabor coefficients and the number of test samples.

P

plot	Graphical representation of an array of data shown either in a graph or a chart.
pop up	To call a special menu by right-clicking (Windows) or command-clicking (Macintosh) an object.
pop-up menu	Menu accessed by right-clicking (Windows) or command-clicking (Macintosh) an object. Menu options pertain to that object.

preemphasis	Filtering before processing.
PWVD	Pseudo Wigner–Ville Distribution.

R

reentrant execution	Mode in which calls to multiple instances of a subVI can execute in parallel with distinct and separate data storage.
representation	Subtype of the numeric data type. Representations include signed and unsigned byte, word, and long integers, as well as single-, double-, and extended-precision floating-point numbers, both real and complex.

S

sampling rate	Rate at which a continuous waveform is digitized.
scope chart	Numeric indicator modeled on the operation of an oscilloscope.
signal discontinuity	The point where the first derivative does not exist.
spectral changes	Changes in the frequency content of a signal.
spectral leakage	Phenomenon whereby the measured spectral energy appears to leak from one frequency into other frequencies. It occurs when a sampled waveform does not contain an integral number of cycles over the time period during which it was sampled. To reduce spectral leakage, multiply the time-domain waveform by the window function. *See also* window.
spectrogram	A display of the energy distribution of a signal with one axis being time and the other being frequency.
STFT	Short-time Fourier Transform.
string controls and indicators	Front panel objects used to manipulate and display, or input and output, text.
strip chart	Numeric plotting indicator modeled after a paper strip chart recorder, which scrolls as it plots data.
subVI	VI used in the block diagram of another VI. It is comparable to a subroutine.

sweep chart	Numeric indicator modeled on the operation of an oscilloscope. It is similar to a scope chart, except that a line sweeps across the display to separate old data from new data.
synthesis filter bank	A filter bank that transfers a signal from the wavelet domain into the time domain.
system developer	Creator of the application software to be executed.

T

temporal	Of or relating to time domain.
third-octave	Ratio between two frequencies, equal to $2^{1/3}$. *See also* octave.
two-dimensional	Having two dimensions, such as an array with both rows and columns.
type I filter	The filter coefficients are symmetric around the middle point.

V

VI	*See* virtual instrument.
VI library	Special file that contains a collection of related VIs for a specific use.
virtual instrument	VI. Program in the graphical programming language G; so-called because it models the appearance and function of a physical instrument.
Virtual Instrument Software Architecture	Single interface library for controlling GPIB, VXI, RS-232, and other types of instruments.

W

waveform chart	Indicator that plots data points at a certain rate.
wavelet transform	A transform using wavelet as the elementary functions.
wavelet-based detrend	A method of detrend, which is achieved by wavelet transform.

window	Technique used to reduce spectral leakage by multiplying the time-domain waveform by a window function. The process of windowing reduces the amplitudes of discontinuities at the edges of a waveform, which reduces spectral leakage. If the waveform contains an integral number of cycles, there is no spectral leakage. *See also* spectral leakage.
WVD	Wigner–Ville Distribution.

Index