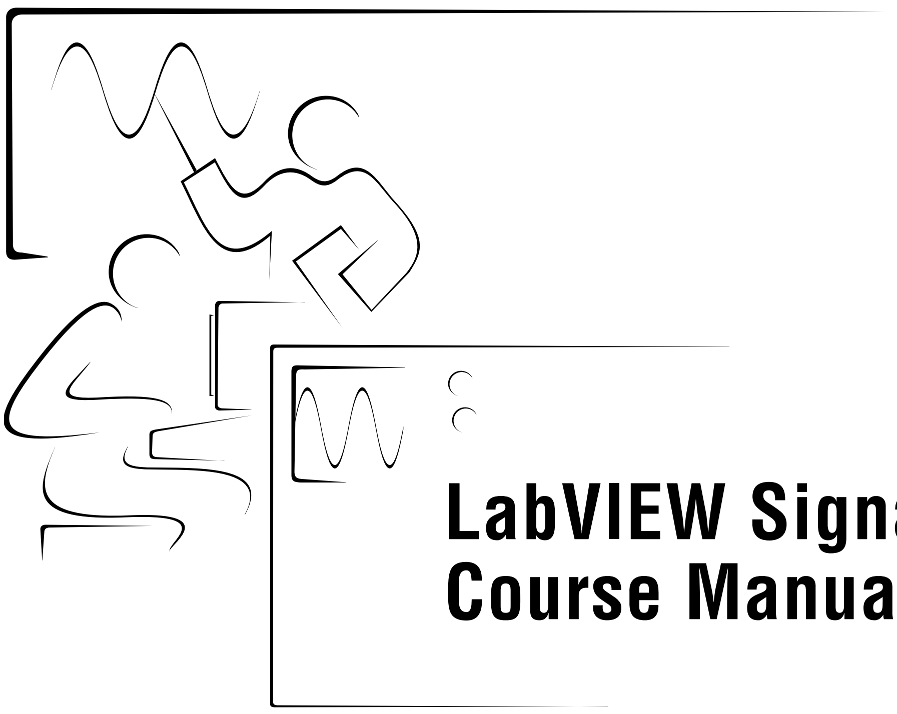


LabVIEW Signal Processing Course Manual

Course Software Version 1.0

September 1997 Edition
Part Number 321569A-01

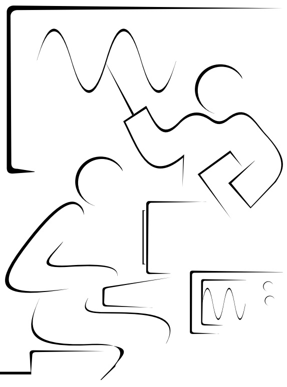
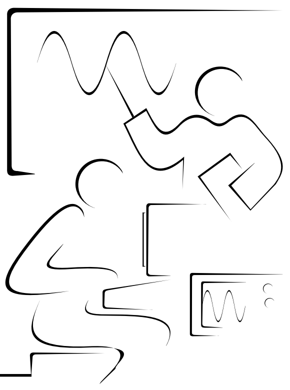


Table of Contents



Student Guide

Introduction

Welcome to the LabVIEW Signal Processing Course. This course is intended for practicing engineers and scientists who want to learn how to use LabVIEW or BridgeVIEW to process and analyze digital signals in real-world practical applications. The course focuses on the advanced analysis library and the various signal processing toolkits that are specifically devoted to designing digital filters, solving mathematics problems, and analyzing nonstationary signals. In addition to teaching you how to use the analysis VIs and toolkits, the course also covers the basic fundamentals necessary for understanding and interpreting the analysis results.

This student guide describes the course contents and suggests ways in which you can most effectively use the course materials. The guide discusses the following topics:

A. Self-Paced Use

B. Course Description

C. Prerequisites

D. Course Goals

E. Course Non-Goals

F. Course Map

G. Course Conventions

A. Self-Paced Use

Thank you for purchasing the LabVIEW Signal Processing Course kit. You should be able to begin developing your application soon after you have worked through this manual. This course manual and accompanying software are used in the two-day, hands-on LabVIEW Signal Processing Course. Several exercises in this manual use the following National Instruments hardware products:

- AT-MIO-16E-2 data acquisition board
- DAQ Signal Accessory

To get started, read the information on the next page regarding the accompanying disks and then follow the instructions on the subsequent pages for the computer platform you are using. If you have comments, suggestions for improving this course, or are not satisfied with the material, please contact:

LabVIEW Signal Processing Technical Support
6504 Bridge Point Parkway
Austin, TX 78730-5039
(512) 795-8248
`support@natinst.com`

Attending the Course

You can apply the full purchase price of this course kit toward the corresponding course registration fee if you register within 90 days of purchasing the kit. To register for a course or for course information, please contact National Instruments.

North America

Telephone: (512) 794-0100

E-mail: `custedu.info@natinst.com` (information requests only)

24-hour automated retrieval of course outlines/latest course schedule

Fax on Demand: (800) 329-7177 or (512) 418-1111

World Wide Web: `http://www.natinst.com/custed`

Other Countries

Please contact your local National Instruments branch office (the phone numbers are on the back cover).

Course Disk

The following table lists the contents of the LabVIEW Signal Processing Course disk. The course disk contains a zip file containing two VI libraries.

Filename	Description
`Lvspc.zip`	A compressed file containing the VIs used in the course exercises (`Lvspcex.llb`) as well as the VIs containing the solutions (`Lvspcsol.llb`).
`unzip.exe`	A utility to decompress `lvspc.zip`.

Note: ***The solution VIs have the word "Solution" at the end of the VI name.***

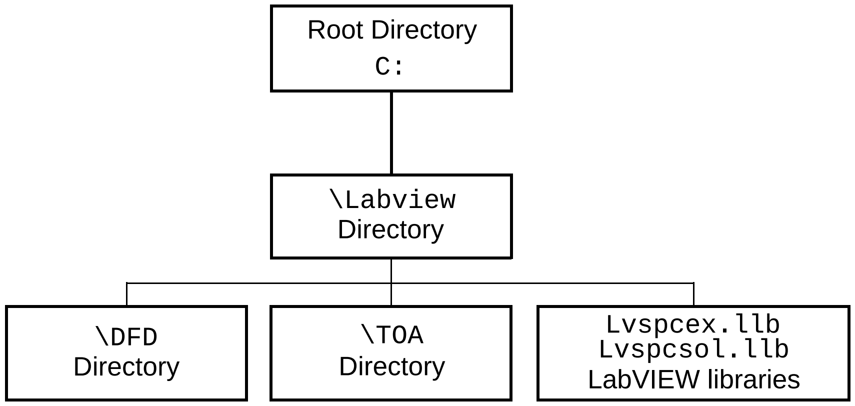
You Will Need the Following Equipment:

- IBM PC AT or compatible.
- LabVIEW or BridgeVIEW for Windows Full Development System, ver 4.0 or later.
- AT-MIO-16E-2 data acquisition board.
- DAQ Signal Accessory.
- Optional—A word processing application such as Write or Wordpad.

Installing the Course Software

1. Copy the files `Lvspc.zip` and `unzip.exe` from the PC disk accompanying this manual to the `Labview` directory on your hard disk.
2. Type in the following at the DOS prompt: `unzip -d Lvspc.zip` <enter>. This extracts the VI libraries that contain the class VIs (`Lvspcex.llb`) and the solution VIs (`Lvspcsol.llb`). In addition, it also extracts a DFD folder and a TOA folder to be used for the lessons on the Digital Filter Design Toolkit and the Third-Octave Analyzer Toolkit, respectively.

The course assumes the following directory structure:



B. Course Description

The LabVIEW Signal Processing course teaches you how to implement and use the VIs in the advanced analysis library and the signal processing toolkits, and also how to interpret and understand the results of your analysis. The course is divided into lessons, each covering a topic or a set of topics. Each lesson consists of:

- An introduction that describes the lesson’s purpose and what you will learn.
- A discussion of the topics.
- A set of exercises to reinforce the topics presented in the discussion.
- A set of additional exercises to be done if time permits.
- A summary that outlines important concepts and skills taught in the lesson.
- Review questions to check for understanding.

Some of the topics have been simplified considerably to make the basic concepts and ideas easier to understand. The simplification has been done at the expense of mathematical detail while striving to provide clear and precise concepts. For a more detailed presentation of such topics, see the list of references at the end of this manual.

C. Prerequisites

- Familiarity with the Windows operating system.
- Familiarity with basic LabVIEW programming techniques.
- Experience writing algorithms in the form of flowcharts or block diagrams.
- Previous exposure to digital signal processing, through either an introductory course or work experience.

D. Course Goals

This course teaches you to:

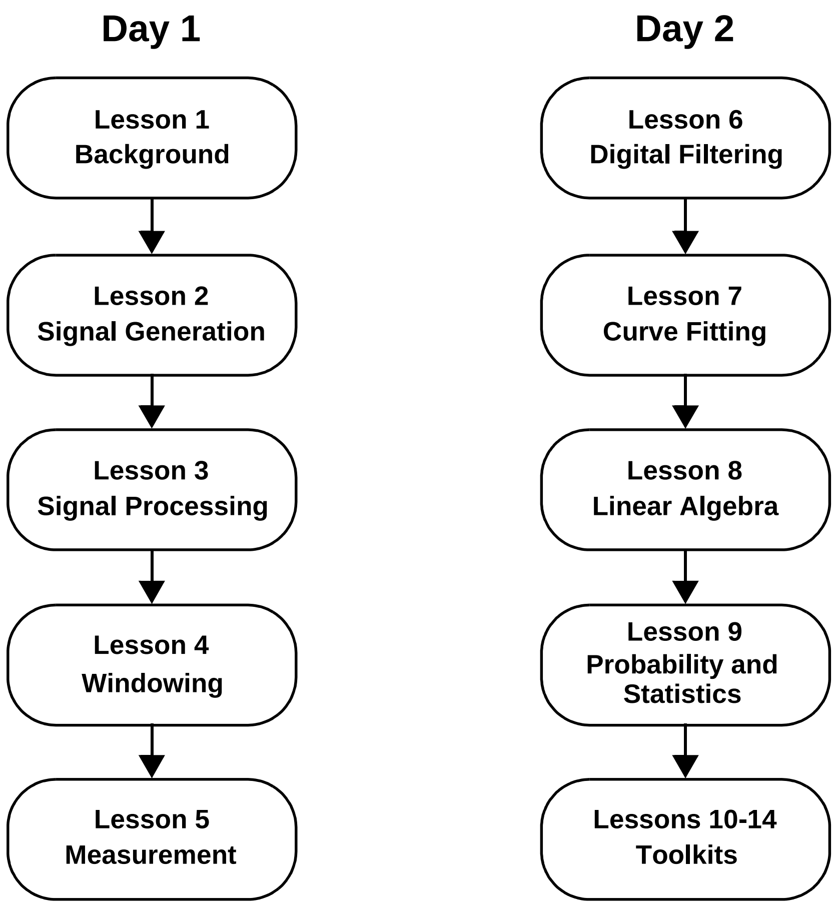
- Become familiar with the analysis capabilities of LabVIEW and BridgeVIEW.
- Understand the basics of digital signal processing and analysis.
- Choose intelligently between several options/methods that are available for performing similar tasks (for example, choosing between different types of windows, filter design methods, or algorithms for curve fitting).
- Implement the VIs from the analysis library in practical applications for solving real-world problems.
- Learn about the various specialized toolkits such as those available for solving mathematical problems, analyzing nonstationary signals, or designing digital filters.

E. Course Non-Goals

It is not the purpose of this course to do any of the following:

- Teach LabVIEW or BridgeVIEW basics.
- Teach programming theory.
- Discuss every built-in LabVIEW Analysis VI.
- Discuss each and every analysis algorithm.
- Develop a complete application for any student in the class.

F. Course Map

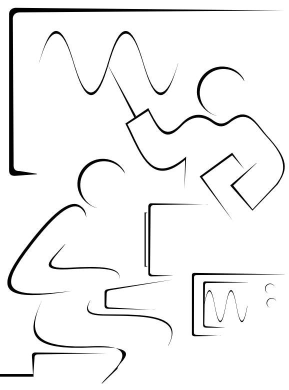

Day 1
Lesson 1 Background
Lesson 2 Signal Generation
Lesson 3 Signal Processing
Lesson 4 Windowing
Lesson 5 Measurement
Day 2
Lesson 6 Digital Filtering
Lesson 7 Curve Fitting
Lesson 8 Linear Algebra
Lesson 9 Probability and Statistics
Lessons 10-14 Toolkits

G. Course Conventions

The following conventions are used in this course manual:

bold	Text in bold refers to LabVIEW menus, menu items, palettes, subpalettes, functions, and VIs. For example, **File**.
italic	Text in italics is for emphasis, a cross-reference, or an introduction to a key concept.
`Courier`	Text in this font indicates drive names, libraries, directories, pathnames, filenames, and sections of programming code. Courier also indicates information you must type. For example, type `Digital Indicator` at the prompt.
`Courier italic`	Text in this font denotes that you must supply the appropriate words or values in the place of these items.
`Courier bold`	Text in this font denotes a computer prompt.
<>	Angle brackets enclose the name of a key. For example, <Enter>.
-	A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys. For example, <Control-Alt-Delete>.

Notes



Lesson 1
Background

Introduction

A *digital* signal is one that can assume only a finite set of values in both the *dependent* and *independent* variables. The independent variable is usually time or space, and the dependent variable is usually amplitude.

Digital signals are everywhere in the world around us. Telephone companies use digital signals to represent the human voice. Radio, TV, and hi-fi sound systems are all gradually converting to the digital domain because of its superior fidelity, noise reduction, and its signal processing flexibility. Data is transmitted from satellites to earth ground stations in digital form. NASA's pictures of distant planets and outer space are often processed digitally to remove noise and to extract useful information. Economic data, census results, and stock market prices are all available in digital form. Because of the many advantages of digital signal processing, analog signals are also converted to digital form before they are processed with a computer. This lesson provides a background in basic digital signal processing and an introduction to the LabVIEW/BridgeVIEW Analysis Library, which consists of hundreds of VIs for signal processing and analysis.

You Will Learn:

A. About the digital (sampled) representation of an analog signal.

B. About aliasing and how to prevent it.

C. About the need for antialiasing filters.

D. About why we use the decibel scale to display amplitudes.

E. About the contents of the LabVIEW/BridgeVIEW Analysis Library.

A. Sampling Signals

To use digital signal processing techniques, you must first convert an analog signal into its digital representation. In practice, this is implemented by using an analog-to-digital (A/D) converter. Consider an analog signal $x(t)$ that is sampled every Δt seconds. The time interval Δt is known as the *sampling interval* or s*ampling period*. Its reciprocal, $1/\Delta t$, is known as the *sampling frequency,* with units of samples/second. Each of the discrete values of x(t) at t = 0, Δt, $2\Delta t$, $3\Delta t$, etc., is known as a *sample.* Thus, x(0), x(Δt), x($2\Delta t$), ..., are all samples. The signal $x(t)$ can thus be represented by the discrete set of samples

$$\{x(0), x(\Delta t), x(2\Delta t), x(3\Delta t), \ldots, x(k\Delta t), \ldots \}.$$

Figure 1-1 below shows an analog signal and its corresponding sampled version. The sampling interval is Δt. Observe that the samples are defined at discrete points in time.

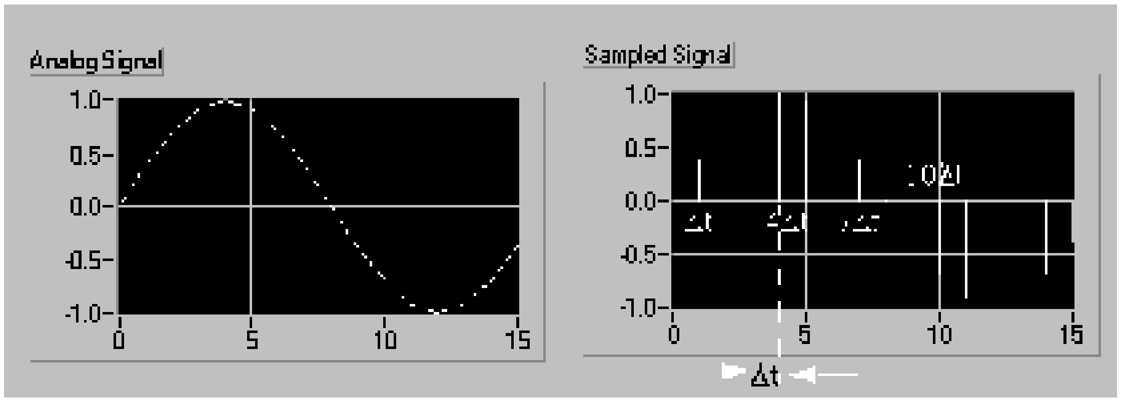


Figure 1-1. Analog Signal and Corresponding Sampled Version

In this course, the following notation represents the individual samples:

$$x[i] = x(i\Delta t), \qquad \text{for } i = 0, 1, 2, \ldots$$

If N samples are obtained from the signal $x(t)$, then $x(t)$ can be represented by the sequence

$$X = \{x[0], x[1], x[2], x[3], \ldots, x[N-1] \}$$

This is known as the *digital representation* or the *sampled version* of $x(t)$. Note that the sequence $X = \{x[i]\}$ is indexed on the integer variable i, and does not contain any information about the sampling rate. So by knowing just the values of the samples contained in X, you will have no idea of what the sample rate is.

B. Sampling Considerations

A/D converters (ADCs) are an integral part of data acquisition (DAQ) boards. One of the most important parameters of an analog input system is the rate at which the DAQ board samples an incoming signal. The sampling rate determines how often an analog-to-digital (A/D) conversion takes place. A fast sampling rate acquires more points in a given time and can form a better representation of the original signal than a slow sampling rate. Sampling too slowly may result in a poor representation of your analog signal. Figure 1-2 shows an adequately sampled signal, as well as the effects of undersampling. The effect of undersampling is that the signal appears as if it has a different frequency than it truly does. This misrepresentation of a signal is called an *alias*.

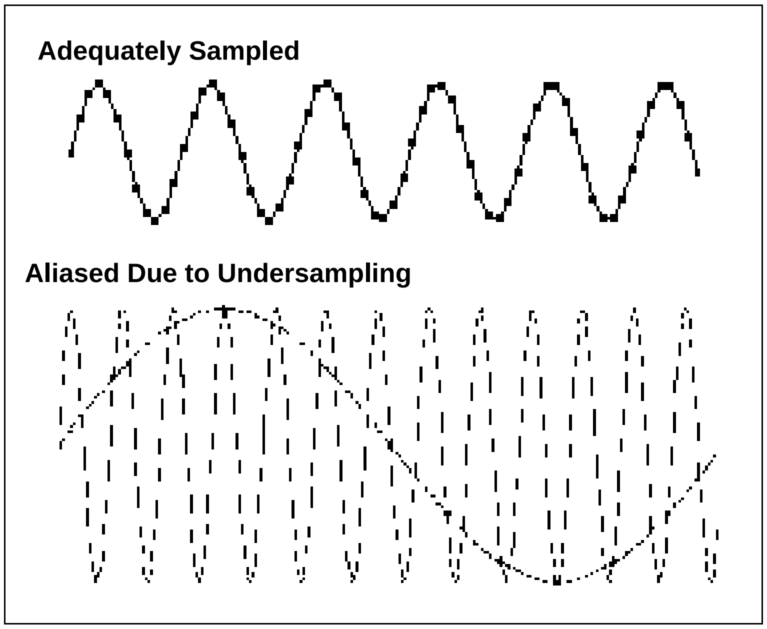


Figure 1-2. Aliasing Effects of an Improper Sampling Rate

According to the *Nyquist theorem*, to avoid aliasing you must sample at a rate greater than twice the maximum frequency component in the signal you are acquiring. For a given sampling rate, the maximum frequency that can be represented accurately, without aliasing, is known as the *Nyquist frequency*. The Nyquist frequency is one half the sampling frequency. Signals with frequency components above the Nyquist frequency will appear aliased between DC and the Nyquist frequency. The alias frequency is the absolute value of the difference between the frequency of the input signal and the closest integer multiple of the sampling rate. Figures 1-3 and 1-4 illustrate this phenomenon. For example, assume *f*s, the sampling frequency, is 100 Hz. Also, assume the input signal contains the following frequencies—25 Hz, 70 Hz, 160 Hz, and 510 Hz. These frequencies are shown in the following figure.

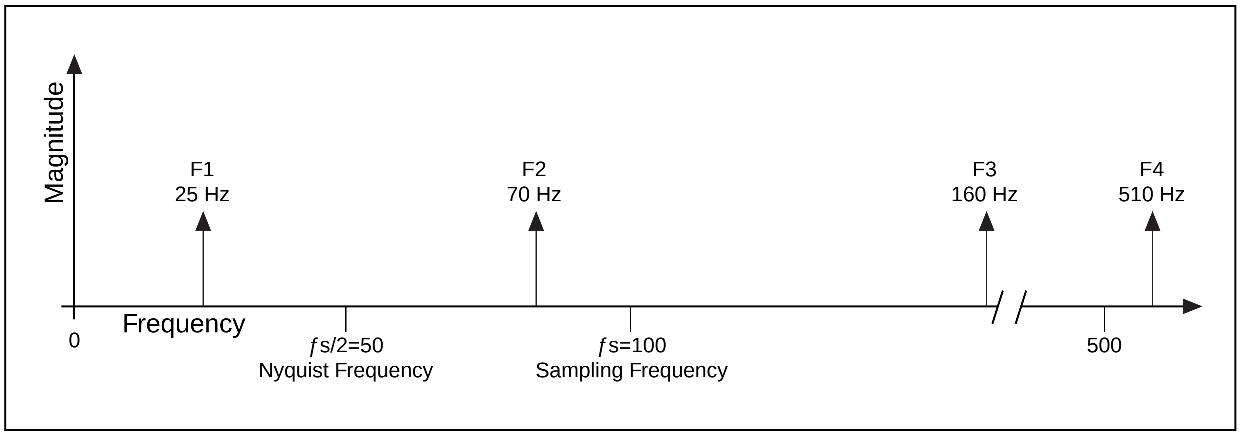


Figure 1-3. Actual Signal Frequency Components

In Figure 1-4, frequencies below the Nyquist frequency (*f*s/2=50 Hz) are sampled correctly. Frequencies above the Nyquist frequency appear as aliases. For example, F1 (25 Hz) appears at the correct frequency, but F2 (70 Hz), F3 (160 Hz), and F4 (510 Hz) have aliases at 30 Hz, 40 Hz, and 10 Hz, respectively. To calculate the alias frequency, use the following equation:

Alias Freq. = ABS (Closest Integer Multiple of Sampling Freq. - Input Freq.)

where ABS means "the absolute value." For example,

Alias F2 = |100 - 70| = 30 Hz

Alias F3 = |(2)100 - 160| = 40 Hz

Alias F4 = |(5)100 - 510| = 10 Hz

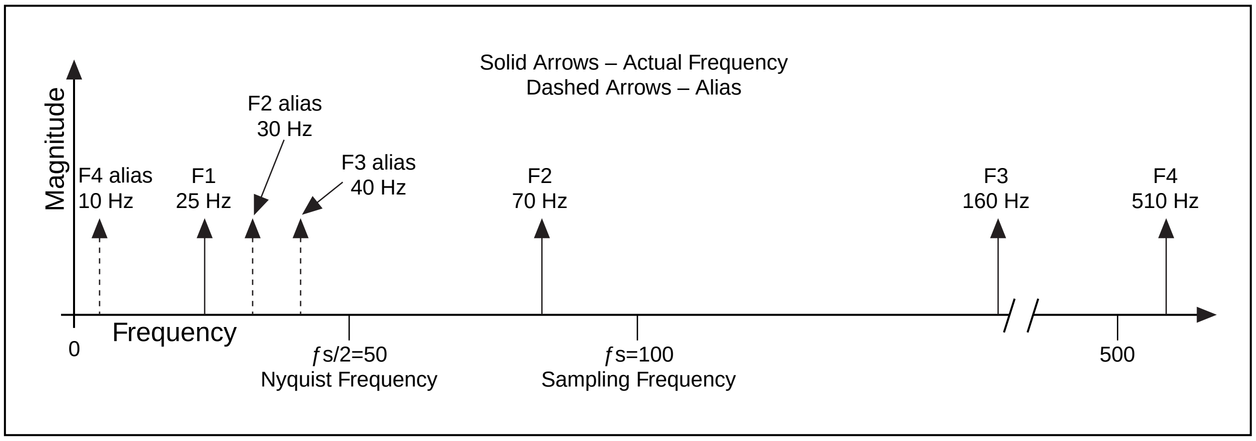


Figure 1-4. Signal Frequency Components and Aliases

A question often asked is, “How fast should I sample?” Your first thought may be to sample at the maximum rate available on your DAQ board. However, if you sample very fast over long periods of time, you may not have enough memory or hard disk space to hold the data. Figure 1-5 shows the effects of various sampling rates. In case A, the sine wave of frequency *f* is sampled at the same frequency *f*. The reconstructed waveform appears as an alias at DC. However, if you increase the sampling rate to 2*f*, the digitized waveform has the correct frequency (same number of cycles), but appears as a triangle waveform. By increasing the sampling rate to well above *f*, for example 5*f*, you can more accurately reproduce the waveform. In case C, the sampling rate is at 4*f*/3. Because in this case the Nyquist frequency is below *f* (4*f*/3 * 1/2 = 2*f*/3), this sampling rate reproduces an alias waveform of incorrect frequency and shape.

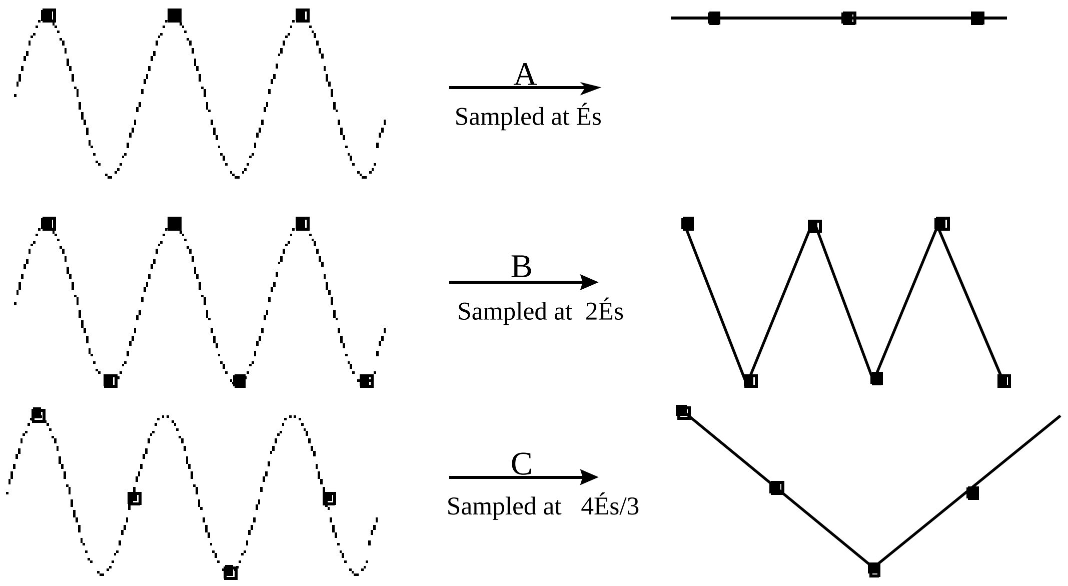


Figure 1-5. Effects of Sampling at Different Rates

The Nyquist theorem gives you a starting point for the adequate sampling rate—greater than two times the highest frequency component in the signal. Unfortunately, this rate is often inadequate for practical purposes. Real-world signals often contain frequency components that lie above the Nyquist frequency. These frequencies are erroneously aliased and added to the components of the signal that are sampled accurately, producing distorted sampled data. Therefore, for practical purposes, sampling is usually done at several times the maximum frequency—five to 10 times is typical in industry.

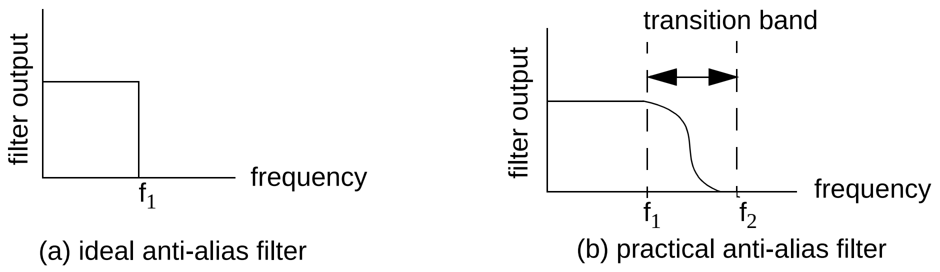
Note: ***Sampling should be done at least at the Nyquist frequency, but usually much higher.***

C. Why Do You Need Antialiasing Filters?

You have seen that the sampling rate should be at least twice the maximum frequency of the signal that you are sampling. In other words, the maximum frequency of the input signal should be less than or equal to half of the sampling rate. But how do you ensure that this is definitely the case in practice? Even if you are sure that the signal being measured has an upper limit on its frequency, pickup from stray signals (such as the powerline frequency or from local radio stations) could contain frequencies higher than the Nyquist frequency. These frequencies may then alias into the desired frequency range and thus give us erroneous results.

To be completely sure that the frequency content of the input signal is limited, a lowpass filter (a filter that passes low frequencies but attenuates the high frequencies) is added before the sampler and the ADC. This filter is called an *antialias* filter because by attenuating the higher frequencies (greater than Nyquist), it prevents the aliasing components. Because at this stage (before the sampler and the ADC) you are still in the analog world, the antialiasing filter is an analog filter.

An ideal antialias filter is as shown in figure (a) below.



An ideal anti-aliasing filter passes all the desired input frequencies (below f_1) and cuts off all the undesired frequencies (above f_1). However, such a filter is not physically realizable. In practice, filters look as shown in figure (b) above. They pass all frequencies $< f_1$, and cut-off all frequencies $> f_2$. The region between f_1 and f_2 is known as the *transition band,* which contains a gradual attenuation of the input frequencies. Although you want to pass only signals with frequencies $< f_1$, those signals in the transition band could still cause aliasing. Therefore, in practice, the sampling frequency should be greater than two times the highest frequency in the transition band. So, this turns out to be more than two times the maximum input frequency (f_1). That is one reason why you may see that the sampling rate is more than twice the maximum input frequency. We will see in a later lesson how the transition band of the filter depends on the filter type being designed.

D. Why Use Decibels?

On some instruments, you will see the option of displaying the amplitude in a linear or decibel (dB) scale. The linear scale shows the amplitudes as they are, whereas the decibel scale is a transformation of the linear scale into a logarithmic scale. You will now see why this transformation is necessary.

Suppose you want to display a signal with very large as well as very small amplitudes. Assume you have a display of height 10 cm and will use the entire height of the display for the largest amplitude. So, if the largest amplitude in the signal is 100 V, a height of 1 cm of the display corresponds to 10 V. If the smallest amplitude of the signal is 0.1 V, this corresponds to a height of only 0.1 mm. This will barely be visible on the display!

To see all the amplitudes, from the largest to the smallest, you need to change the amplitude scale. Alexander Graham Bell invented a unit, the Bell, which is logarithmic, compressing large amplitudes and expanding the small amplitudes. However, the Bell was too large of a unit, so commonly the decibel (1/10th of a Bell) is used. The decibel (dB) is defined as

$$\text{one dB} = 10 \log_{10} (\text{Power Ratio}) = 20 \log_{10} (\text{Voltage Ratio})$$

The following table shows the relationship between the decibel and the power and voltage ratios.

dB	Power Ratio	Voltage Ratio
+40	10000	100
+20	100	10
+6	4	2
+3	2	1.4
0	1	1
-3	1/2	1/1.4
-6	1/4	1/2
-20	1/100	1/10
-40	1/10000	1/100

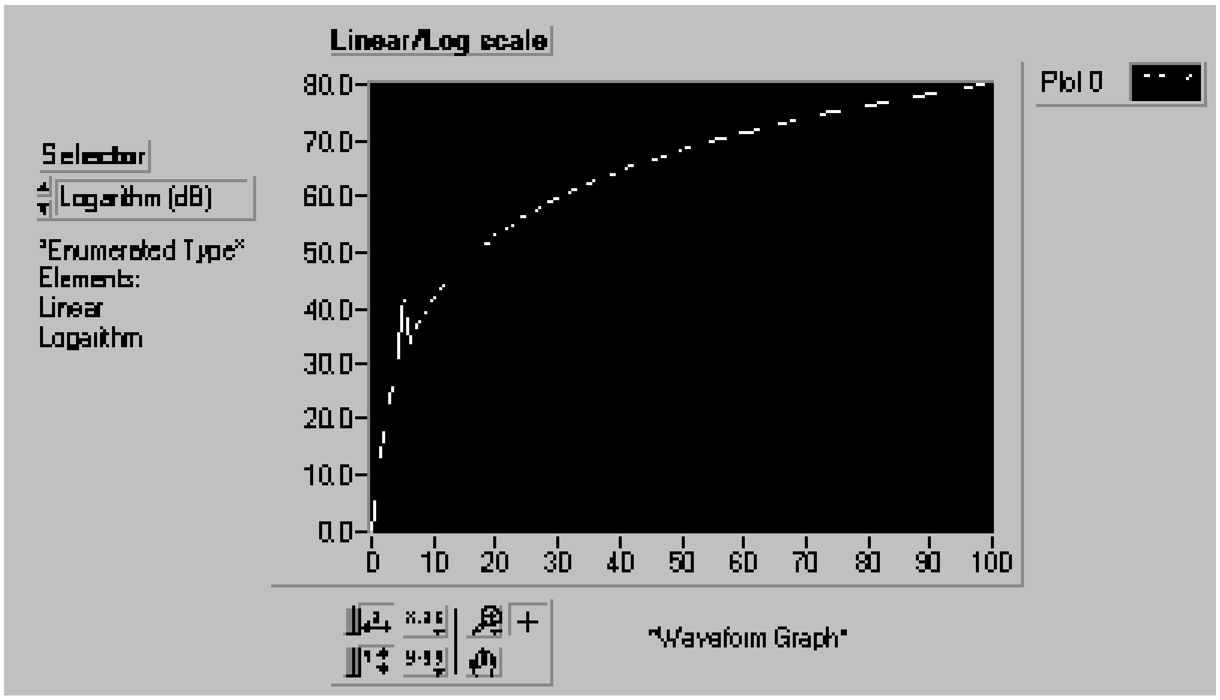
Thus, you see that the dB scale is useful in compressing a wide range of amplitudes into a small set of numbers. The decibel scale is often used in sound and vibration measurements and in displaying frequency domain information. You will now do an exercise that shows a signal in linear and logarithmic scales.

Exercise 1-1

Objective: To build a VI that displays the signal amplitude in both linear and dB scales.

This VI will display the square of 100 data points on a waveform graph. The fifth data point will create a spike. You will observe that the spike is visible on the dB scale.

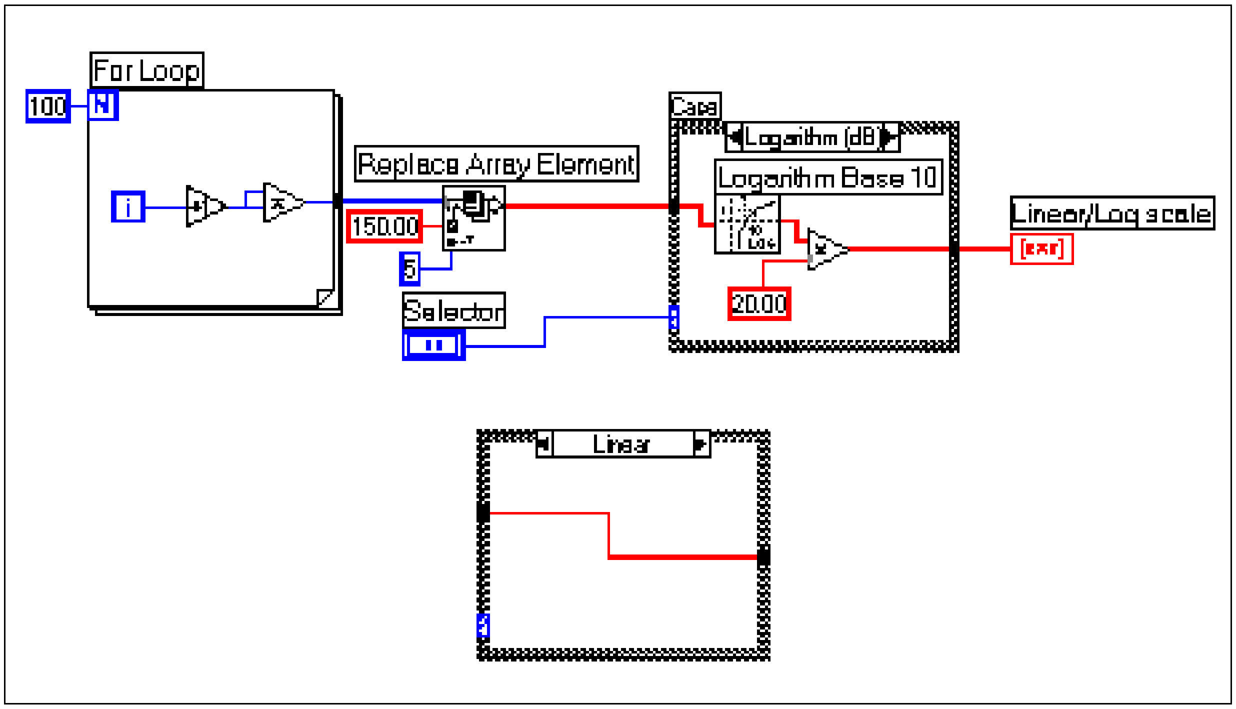
Front Panel



1. Build a VI with the front panel shown above.

   The **Selector** control (**Controls » List and Ring » Enumerated Type**) has two options, *Linear* scale and *Logarithm* (*dB*) scale.

Block Diagram



2. Build the block diagram as shown above.

The **For Loop** (**Functions » Structures** subpalette) generates the square of 100 data points to be displayed on the **Waveform Graph** (**Controls » Graph** palette). (You add +1 to the loop count to avoid taking the logarithm of zero, which results in a value of -×.) Thus, the values of the data points range from 1^2 to 100^2, giving a total range of 1 to 10,000. This corresponds to a ratio of 10,000 between the largest (10,000) and the smallest (1) squared value.

The **Replace Array Element** function (**Functions » Array** subpalette) replaces the 5th data point, which has a value of $5^2 = 25$, by 150, to create a spike at the fifth element. You will see how the spike is barely noticeable on the linear scale, but is easily distinguishable on the dB scale.

Depending on the selector control, the **Case** structure (**Functions » Structures** subpalette) either passes the data directly (*Linear* scale) to the **Waveform Graph** or calculates *20* times the *logarithm to the base 10* (*Logarithm* (*dB*) scale) of the data points and sends the result to the **Waveform Graph**.

The **Logarithm Base 10** function is found in the **Functions » Numeric » Logarithmic** subpalette.

3. Select the *Linear* option from the **Selector** control, and run the VI. Note that the spike at element 5 is barely visible.
4. Select the *Logarithm (dB)* option from the **Selector** control and run the VI. Note that the spike at element 5 is very easily noticeable.

☞ **Note:** ***Observe the change in the y-axis scale as you switch between the "Linear" and "Logarithm (dB)" options.***

5. After you have finished, save the VI as **dB_linear.vi** in the `Lvspcex.llb` library**.**

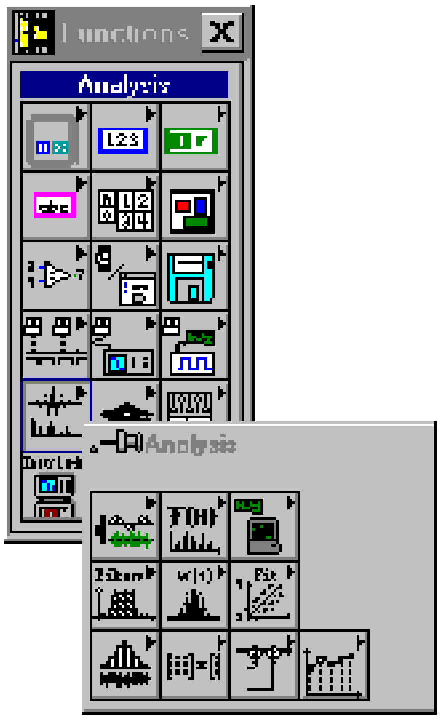
End of Exercise 1-1

E. Overview of the Advanced Analysis Library

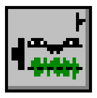
Once the analog signal has been converted to digital form by the ADC and is available in your computer as a digital signal (a set of samples), you will usually want to process these samples in some way. The processing could be to determine the characteristics of the system from which the samples were obtained, to measure certain features of the signal, or to convert them into a form suitable for human understanding, to name a few.

The LabVIEW/BridgeVIEW **Analysis** library contains VIs to perform extensive numerical analysis, signal generation and signal processing, curve fitting, measurement, and other analysis functions. The Analysis Library, included in the LabVIEW/BridgeVIEW full development system, is a key component in building a virtual instrumentation system. Besides containing the analysis functionality found in many math packages, it also features many unique signal processing and measurement functions that are designed exclusively for the instrumentation industry.

The analysis VIs are available in the **Analysis** subpalette of the **Functions** palette in LabVIEW or BridgeVIEW.



There are 10 analysis VI libraries. The main categories are:



Signal Generation: VIs that generate digital patterns and waveforms.

Digital Signal Processing: VIs that perform frequency domain transformations, frequency domain analysis, time domain analysis, and other transforms such as the Hartley and Hilbert transforms.

Measurement: VIs that perform measurement-oriented functions such as single-sided spectrums, scaled windowing, and peak power and frequency estimation.

Filters: VIs that perform IIR, FIR, and nonlinear digital filtering functions.

Windows: VIs that perform data windowing.

Curve Fitting: VIs that perform curve fitting functions and interpolations.

Probability and Statistics: VIs that perform descriptive statistics functions, such as identifying the mean or the standard deviation of a set of data, as well as inferential statistics functions for probability and analysis of variance (ANOVA).

Linear Algebra: VIs that perform algebraic functions for real and complex vectors and matrices.

Array Operations: VIs that perform common, one- and two-dimensional numerical array operations, such as linear evaluation and scaling.

Additional Numerical Methods: VIs that use numerical methods to perform root-finding, numerical integration, and peak detection.

In this course, you will learn how to design and use the VIs from the analysis library to build a function generator and a simple, yet practical, spectrum analyzer. You will also learn how to design and use digital filters, the purpose of windowing, and the advantages of different types of windows, how to perform simple curve-fitting tasks, and much more. The exercises in this course require the LabVIEW/BridgeVIEW full development system. For the more adventurous, an extensive set of examples that demonstrate how to use the analysis VIs can be found in the **labview » examples » analysis** folder.

In addition to the Analysis library, National Instruments also offers many analysis add-ons that make LabVIEW or BridgeVIEW one of the most powerful analysis software packages available. These add-ons include the *Joint Time-Frequency Analysis Toolkit*, which includes the National Instruments award-winning Gabor Spectrogram algorithm that analyzes time-frequency features not easily obtained by conventional Fourier analysis; the *G Math Toolkit*, which offers extended math functionality like a formula parser, routines for optimization and solving differential equations, numerous types of 2D and 3D plots, and more; the *Digital Filter Design Toolkit*; and many others. These specialized add-ons will also be discussed later in this course.

Summary

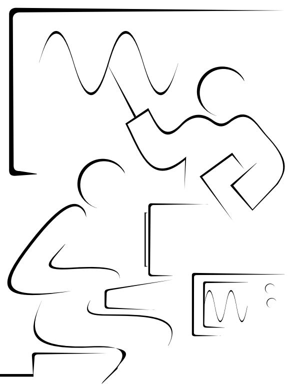
- This lesson introduced the digital (sampled) representation of a signal.
- To convert an analog signal into a digital signal, the sampling frequency (*fs*) should be at least twice the highest frequency contained in the signal. If this is not the case, the frequencies in the signal that are greater than the *Nyquist frequency* (*fs/2*) appear as undesirable aliases.
- You can use a lowpass filter before sampling the analog signal to limit its frequency content to less than *fs/2*. Such a filter used to prevent the effect of aliasing is known as an *antialias filter*.
- You saw how to use a logarithmic scale (the decibel) to display a large range of values. It does this by compressing large values and expanding small ones.
- This lesson also gave an overview of the LabVIEW/BridgeVIEW Analysis Library and its contents.

Review Questions

1. Give some examples of digital signals in everyday life.
2. Given a set of sample values x = {x[i]} where i is an integer variable, what is the sampling rate?
3. What is aliasing? How can it be avoided?
4. Given that the sampling frequency is 100 Hz, what is the alias frequency (if any) for the following: 13 Hz, 25 Hz, 40 Hz, 75 Hz, 99 Hz, 101 Hz, 200 Hz, and 350 Hz?
5. Why do we use the decibel scale? In what applications is it normally used?
6. Which of the following is possible using the analysis VIs?
   a. Finding the mean or standard deviation of census data.
   b. Designing a filter to remove noise from an electrocardiogram.
   c. Detecting peaks in a blood pressure waveform to measure the heart rate.
   d. Interpolating between data points to plot the trajectory of an object (for example, a comet or a cannonball).

Notes

Notes



Lesson 2
Signal Generation

Introduction

In this lesson, you will learn how to use the VIs in the analysis library to generate many different types of signals. Some of the applications for signal generation are:

- Simulating signals to test your algorithm when real-world signals are not available (for example, when you do not have a DAQ board for obtaining real-world signals).
- Generating signals to apply to a D/A converter (for example, in control applications such as opening or closing a valve).

You Will Learn:

A. About the concept of *normalized* frequency.

B. About the difference between *Wave* and *Pattern* VIs (for example, the **Sine Wave** VI and the **Sine Pattern** VI).

C. About how to build a simple function generator using the VIs in the **Signal Generation** subpalette.

A. Normalized Frequency

In the analog world, a signal frequency is measured in Hz or cycles per second. But the digital system often uses a digital frequency, which is the ratio between the analog frequency and the sampling frequency:

digital frequency = analog frequency / sampling frequency

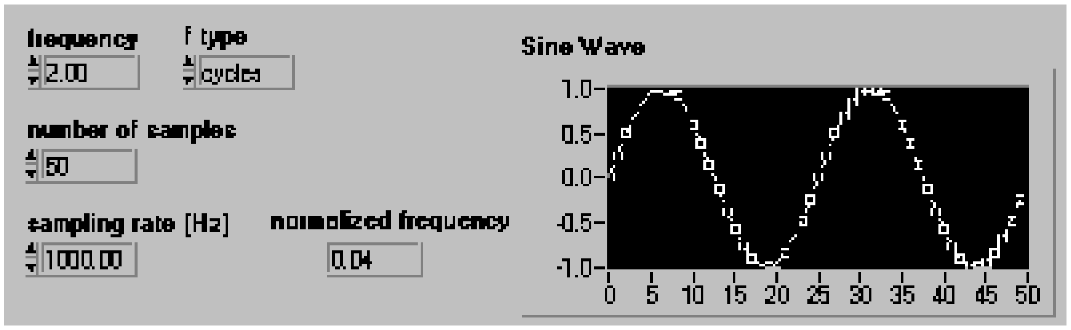
This digital frequency is known as the *normalized* frequency. Its units are cycles/sample.

Some of the Signal Generation VIs use an input frequency control, *f*, that is assumed to use *normalized frequency* units of *cycles per sample*. This frequency ranges from 0.0 to 1.0, which corresponds to a real frequency range of 0 to the sampling frequency *fs*. This frequency also wraps around 1.0, so that a normalized frequency of 1.1 is equivalent to 0.1. As an example, a signal that is sampled at the Nyquist rate (*fs/2*) means that it is sampled twice per cycle (that is, two samples/cycle). This will correspond to a normalized frequency of 1/2 cycles/sample = 0.5 cycles/sample. The reciprocal of the normalized frequency, *1/f*, gives you the number of times that the signal is sampled in one cycle.

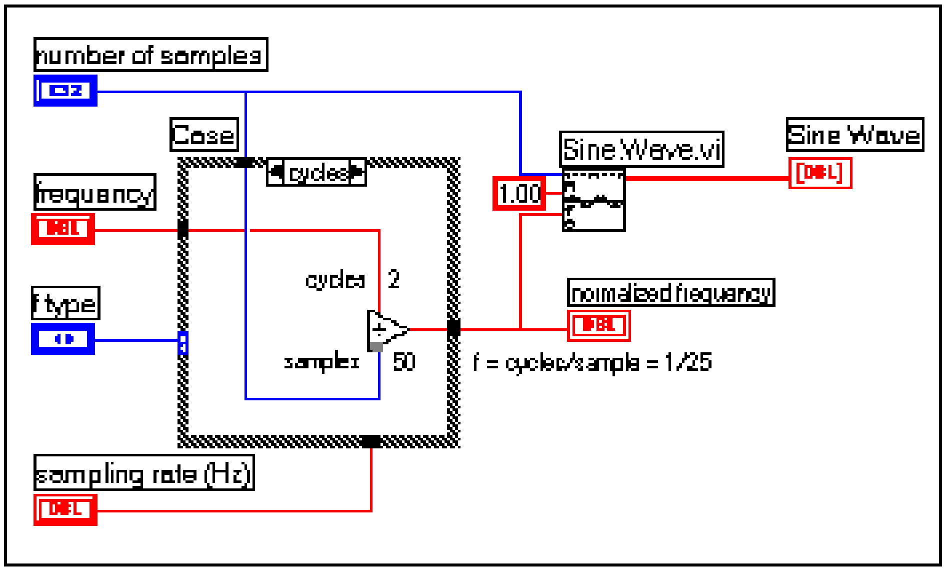
When you use a VI that requires the normalized frequency as an input, you must convert your frequency units to the normalized units of cycles/sample. You must use these normalized units with the following VIs.

- Sine Wave
- Square Wave
- Sawtooth Wave
- Triangle Wave
- Arbitrary Wave
- Chirp Pattern

If you are used to working in frequency units of cycles, you can convert cycles to cycles/sample by dividing cycles by the number of samples generated. The following illustration shows the **Sine Wave** VI, which is being used to generate two cycles of a sine wave.



The following illustration shows the block diagram for converting cycles to cycles/sample.

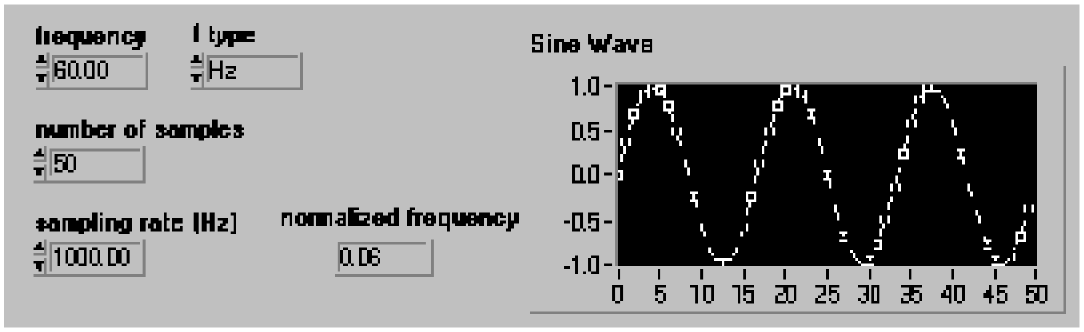


You need only divide the frequency (in cycles) by the number of samples. In the above example, the frequency of 2 cycles is divided by 50 samples, resulting in a normalized frequency of *f* = 1/25 cycles/sample. This means that it takes 25 (the reciprocal of *f*) samples to generate one cycle of the sine wave.

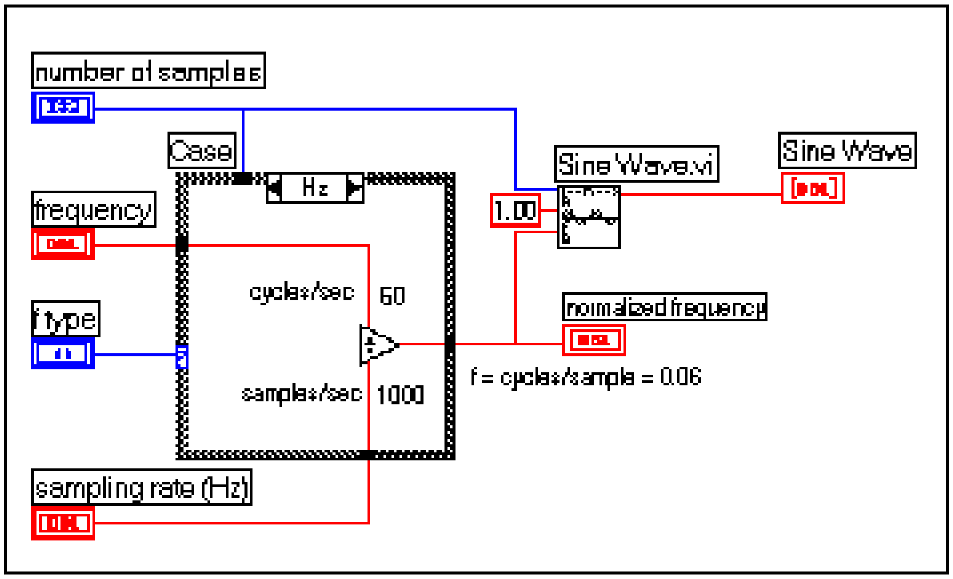
However, you may need to use frequency units of Hz (cycles/second). If you need to convert from Hertz (or cycles/second) to cycles/sample, divide your frequency in cycles/second by the sampling rate given in samples/second.

$$\frac{\text{cycles/second}}{\text{samples/second}} = \frac{\text{cycles}}{\text{sample}}$$

The following illustration shows the **Sine Wave** VI used to generate a 60 Hz sine signal.



Below is a block diagram for generating a Hertz sine signal. You divide the frequency of 60 Hz by the sampling rate of 1000 Hz to get the *normalized* frequency of *f* = 0.06 cycles/sample. Therefore, it takes almost 17 (1/0.06) samples to generate one cycle of the sine wave.



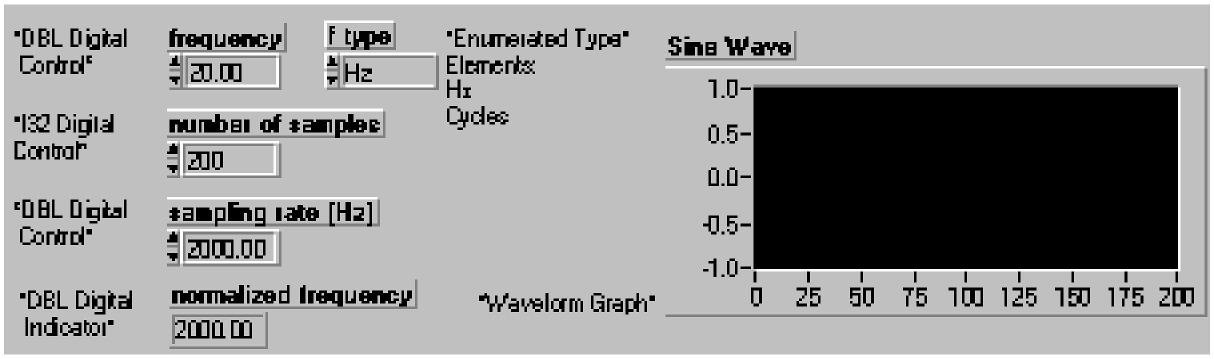
The signal generation VIs create many common signals required for network analysis and simulation. You can also use the signal generation VIs in conjunction with National Instruments hardware to generate analog output signals.

Exercise 2-1

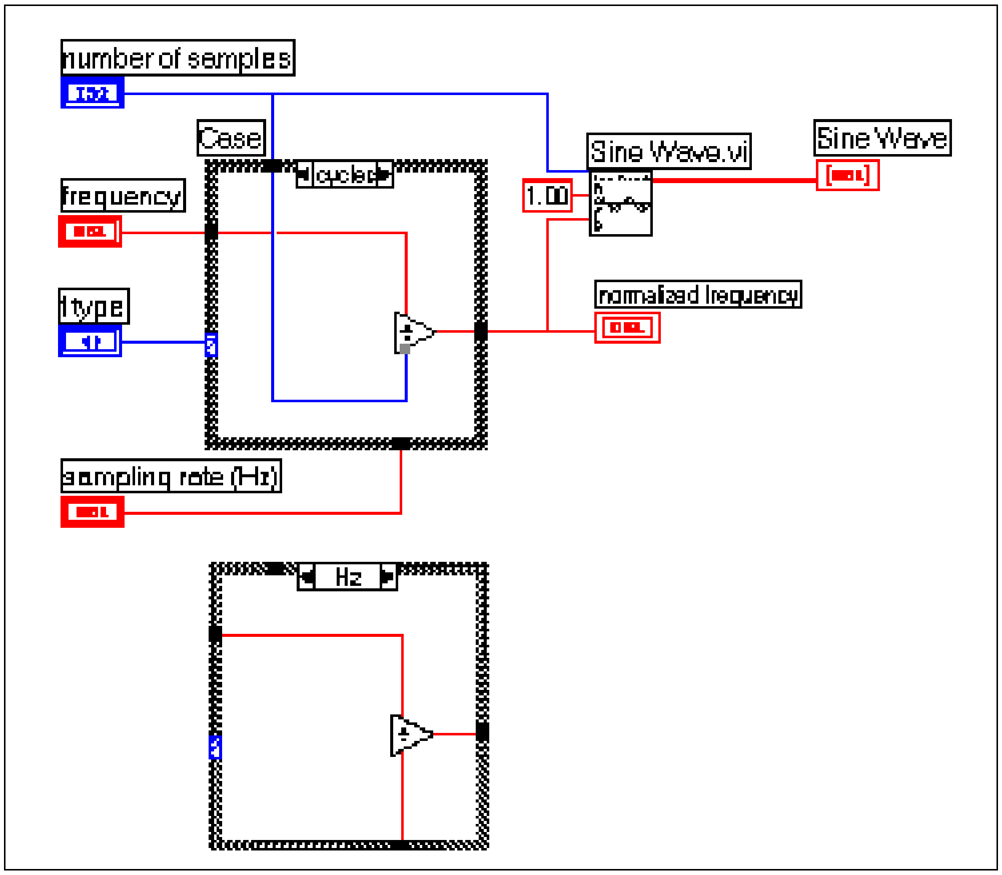
Objective: To understand the concept of normalized frequency.

1. Build the VI front panel and block diagram shown below.

Front Panel



Block Diagram



Sine Wave VI (**Analysis » Signal Generation** subpalette).

2. Select a frequency of 2 cycles (**frequency** = 2 and **f type** = cycles) and **number of samples** = 100. Run the VI. Note that the plot will show 2 cycles. (The normalized frequency indicator tells you the normalized frequency.)
3. Increase the **number of samples** to 150, 200, and 250. How many cycles do you see?
4. Now keep the **number of samples** = 100. Increase the number of cycles to 3, 4, and 5. How many cycles do you see?

Thus, when you choose the frequency in terms of cycles, you will see that many cycles of the input waveform on the plot. Note that the sampling rate is irrelevant in this case.

5. Change **f type** to Hz and **sampling rate (Hz)** to 1000.
6. Keeping the **number of samples** fixed at 100, change the **frequency** to 10, 20, 30, and 40. How many cycles of the waveform do you see on the plot for each case? Explain your observations.
7. Repeat the above step by keeping the **frequency** fixed at 10 and change the **number of samples** to 100, 200, 300, and 400. How many cycles of the waveform do you see on the plot for each case? Explain your observations.
8. Keep the **frequency** fixed at 20 and the **number of samples** fixed at 200. Change the **sampling rate (Hz)** to 500, 1000, and 2000. Make sure you understand the results.
9. Save the VI as **Normalized Frequency.vi** in the library `Lvspcex.llb`.

End of Exercise 2-1

B. Wave and Pattern VIs

You will notice that the names of most of the signal generation VIs have the word *wave* or *pattern* in them. There is a basic difference in the operation of the two different types of VIs. It has to do with whether or not the VI can keep track of the phase of the signal that it generates each time it is called.

Phase Control

The *wave* VIs have a *phase in* control where you can specify the initial phase (in degrees) of the first sample of the generated waveform. They also have a *phase out* indicator that specifies what the phase of the next sample of the generated waveform is going to be. In addition, a *reset phase* control decides whether or not the phase of the first sample generated when the *wave* VI is called is the phase specified at the *phase in* control, or whether it is the phase available at the *phase out* control when the VI last executed. A TRUE value of *reset phase* sets the initial phase to *phase in,* whereas a FALSE value sets it to the value of *phase out* when the VI last executed.

The *wave* VIs are all reentrant (can keep track of phase internally) and accept frequency in normalized units (cycles/sample). The only *pattern* VI that presently uses normalized units is the **Chirp Pattern** VI. Setting the *reset phase* Boolean to FALSE allows for continuous sampling simulation.

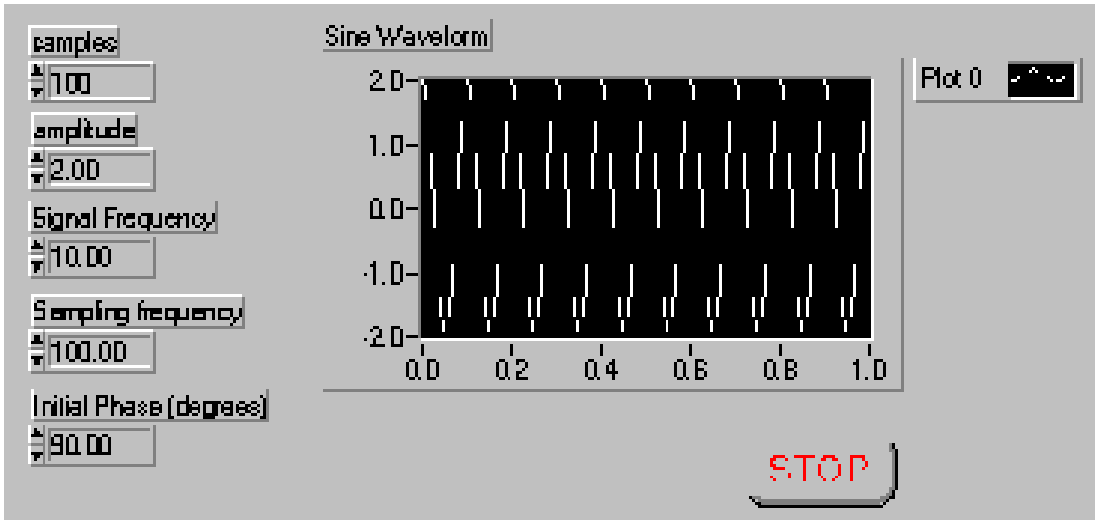
Note: ***Wave VIs are reentrant and accept the frequency input in terms of normalized units.***

In the next exercise, you will generate a sine wave using both the **Sine Wave** VI and the **Sine Pattern** VI. You will see how in the **Sine Wave** VI you have more control over the initial phase than in the **Sine Pattern** VI.

Exercise 2-2

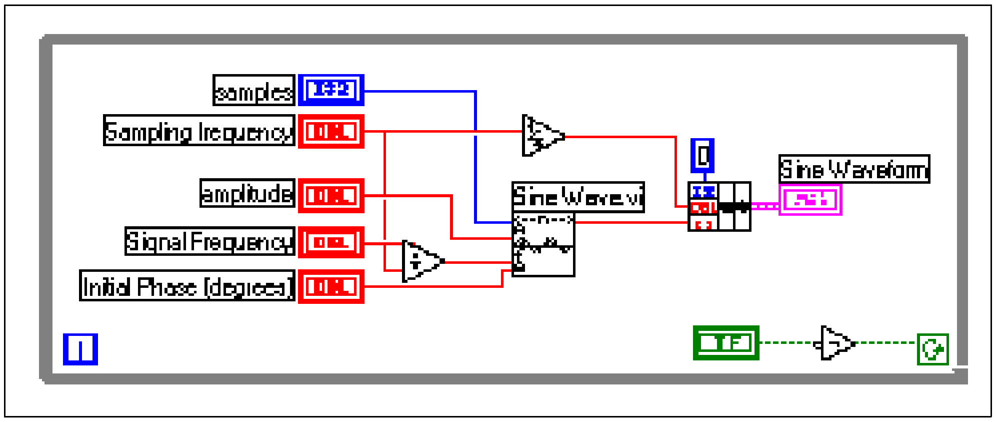
OBJECTIVE: To generate a sine wave of a particular frequency and see the effect of aliasing.

Front Panel

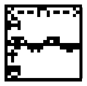


1. Open the **Generate Sine** VI from the library `Lvspcex.llb`.
2. The front panel contains controls for the number of sample points to be generated, the amplitude, analog frequency, and initial phase (in degrees) of the sine wave to be generated, and the frequency at which this waveform is sampled.
3. Do not change the front panel default values. Switch to the block diagram.

Block Diagram



4. Examine the block diagram.



Sine Wave VI (**Analysis » Signal Generation** subpalette). In this exercise, this VI generates 100 points of a 10 Hz sine wave sampled at 100 Hz.

5. Notice in the block diagram that the signal frequency is divided by the sampling frequency *before* it is connected to the **Sine Wave** VI. This is because the **Sine Wave** VI requires the digital (normalized) frequency of the signal.
6. Run the VI. With the default front panel values, a 10 Hz sine wave should appear on the graph.

Sampling and Aliasing

7. Change the signal frequency on the front panel to 90 Hz and observe the waveform. The resulting signal looks just like the 10 Hz waveform.

   As you saw in the previous lesson, this phenomenon is called *aliasin*g, which occurs only in the digital domain. The famous Nyquist Sampling Theorem dictates that the highest representable useful frequency is at most half of the sampling frequency. In our case, the sampling frequency is 100 Hz, so the maximum representable frequency is 50 Hz. If the input frequency is over 50 Hz, as in our case of 90 Hz, it will be aliased back to ((n*50) - 90) Hz > 0, which is (100-90) Hz, or 10 Hz. In other words, this digital system with a sampling frequency of 100 Hz cannot discriminate 10 Hz from 90 Hz, 20 Hz from 80 Hz, 51 Hz from 49 Hz, and so on.

   ### The Importance of an Analog Antialiasing Filter

   Therefore, in designing a digital system, you must make sure that any frequencies over half of the sampling frequency do not enter the system. *Once they are in, there is no way to remove them!* To prevent aliasing, you typically use an analog antialiasing lowpass filter. So, in this example, you can use an analog antialiasing filter to remove any frequencies over 50 Hz. After the signal is filtered, you are assured that whenever you see a 10 Hz signal with a 100 Hz sampling frequency, it is 10 Hz and not 90 Hz.
8. When you are done, stop the VI by clicking on the STOP button. Close the VI. Do not save any changes.

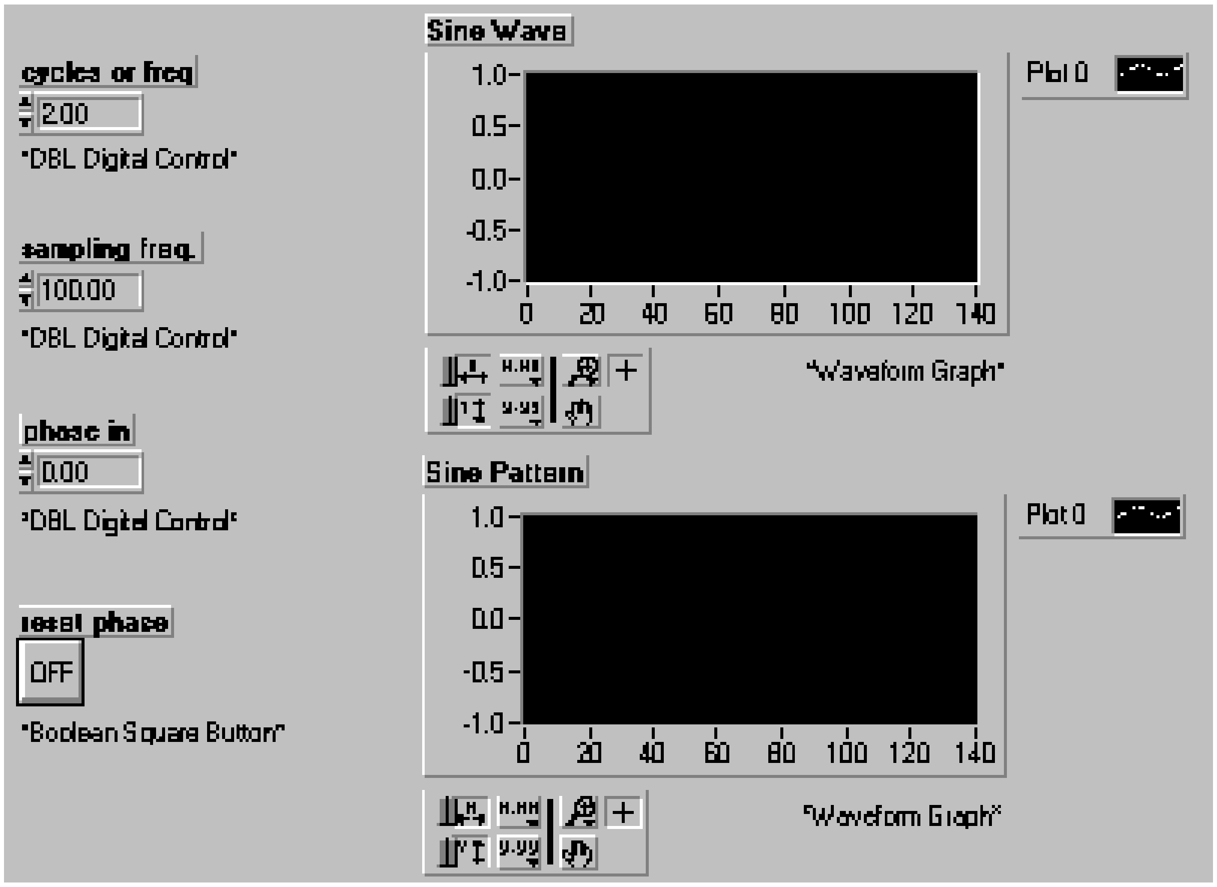
End of Exercise 2-2

Exercise 2-3

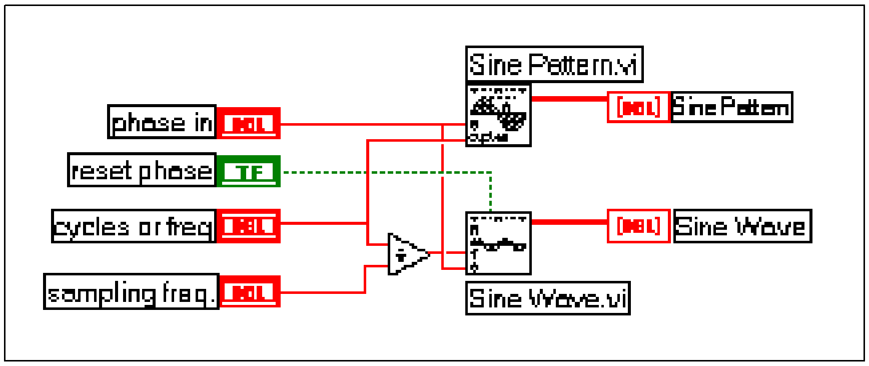
Objective: To generate a sinusoidal waveform using both the Sine Wave VI and the Sine Pattern VI and to understand the differences.

1. Build the VI front panel and block diagram shown below.

Front Panel



Block Diagram



Sine Pattern VI (**Analysis » Signal Generation** palette).

Sine Wave VI (**Analysis » Signal Generation** palette).

2. Set the controls to the following values:

cycles or freq:	2.00
sampling freq:	100
phase in:	0.00
reset phase:	OFF

   Run the VI several times.

   Observe that the **Sine Wave** plot changes each time you run the VI. Because **reset phase** is set to OFF, the phase of the sine wave changes with each call to the VI, being equal to the value of **phase out** during the previous call. However, the Sine Pattern plot always remains the same, showing 2 cycles of the sinusoidal waveform. The initial phase of the Sine Pattern plot is equal to the value set in the **phase in** control.

Note: ***"Phase in" and "phase out" are specified in degrees.***

3. Change **phase in** to 90 and run the VI several times. Just as before, the Sine Wave plot changes each time you run the VI. However, the Sine Pattern plot does not change, but the initial phase of the sinusoidal pattern is 90 degrees—the same as that specified in the **phase in** control.
4. With **phase in** still at 90, set **reset phase** to ON and run the VI several times. The sinusoidal waveforms shown in both the Sine Wave and

Sine Pattern plots start at 90 degrees, but do not change with successive calls to the VI.

5. Keeping **reset phase** as ON, run the VI several times for each of the following values of **phase in**: 45, 180, 270, and 360. Note the initial phase of the generated waveform each time that the VI is run.
6. When you have finished, save the VI as **Wave and Pattern.vi** in the library `Lvspcex.llb`**.**

End of Exercise 2-3

Exercise 2-4 (Optional)

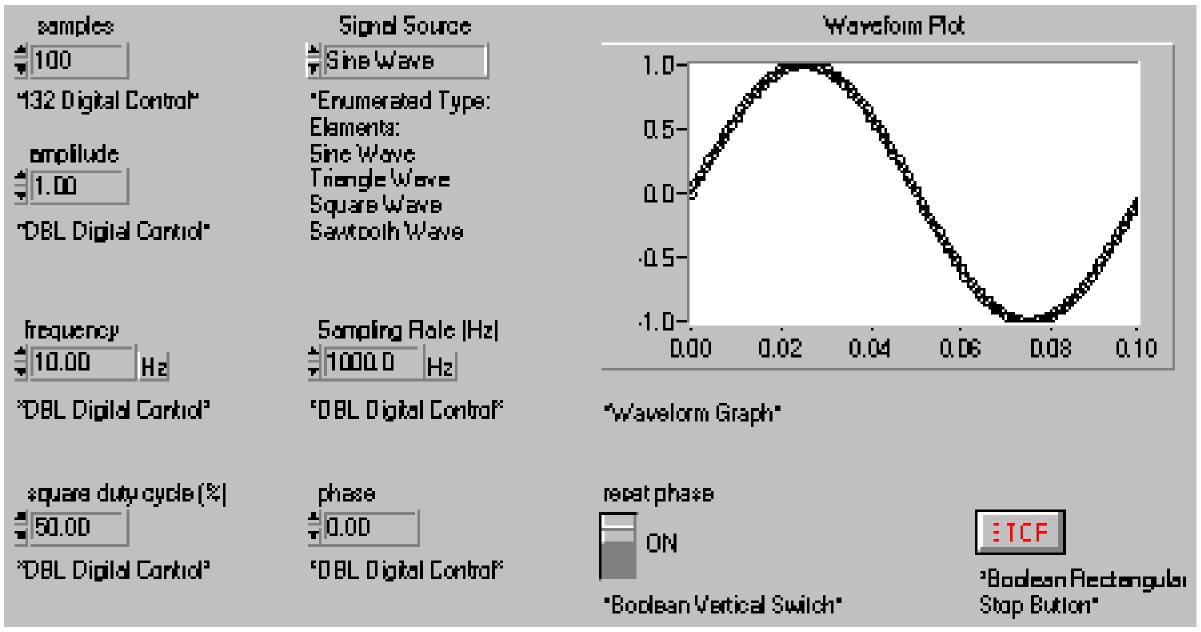
Objective: To build a simple function generator.

In this exercise, you will build a very simple function generator than can generate the following waveforms:

- Sine Wave
- Square Wave
- Triangle Wave
- Sawtooth Wave

1. Build the VI front panel and block diagram shown below.

Front Panel



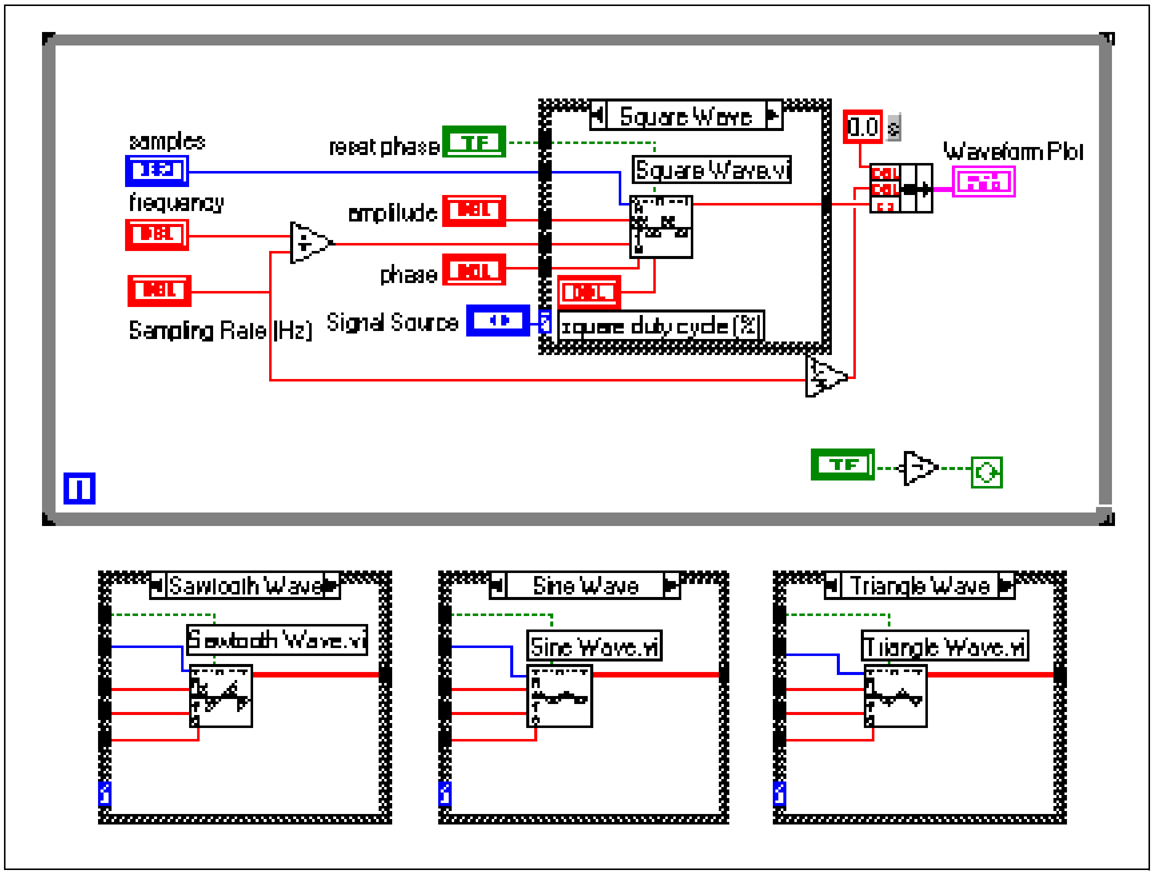
The **Signal Source** control selects the type of waveform that you want to generate.

The **square duty cycle** control is used only for setting the duty cycle of the square wave.

The **samples** control determines the number of samples in the plot.

Note that these are all wave VIs, and therefore they require the frequency input to be the normalized frequency. So, you divide **frequency** by the **sampling rate** and the result is the normalized frequency wired to the *f* input of the VIs.

Block Diagram



Sine Wave VI (**Analysis » Signal Generation** subpalette) generates a sine wave of normalized frequency *f*.

Triangle Wave VI (**Analysis » Signal Generation** subpalette) generates a triangular wave of normalized frequency *f*.

Square Wave VI (**Analysis » Signal Generation** subpalette) generates a square wave of normalized frequency *f* with specified duty cycle.

Sawtooth Wave VI (**Analysis » Signal Generation** subpalette) generates a sawtooth wave of normalized frequency *f*.

2. Select a **sampling rate** of 1000 Hz, **amplitude** = 1, **samples** = 100, **frequency** = 10, **reset phase** = ON, and **signal source** = sine wave. Note that because **sampling rate** = 1000 and **frequency** = 10 Hz, every 100 samples corresponds to one cycle.
3. Run the VI and observe the resulting plot.

4. Change **samples** to 200, 300, and 400. How many cycles of the waveform do you see? Explain why.
5. With **samples** set to 100, change **reset phase** to OFF. Do you notice any difference in the plot?
6. Change **frequency** to 10.01 Hz. What happens? Why?
7. Change **reset phase** to ON. Now what happens? Explain why.
8. Repeat steps 4 – 7 for different waveforms selected in the **Signal Source** control.
9. When you finish, save the VI as **Function Generator.vi** in the `Lvspcex.llb` library.

End of Exercise 2-4

Summary

In this lesson, you learned:

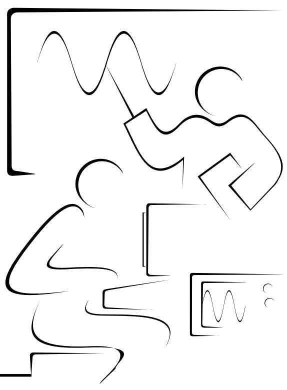
- About the normalized frequency (*f*) that has units of cycles/sample.
- How to generate a sine wave of a particular frequency.
- That the wave VIs can keep track of the phase of the generated waveform.
- How to build a simple function generator that can generate a sine, square, triangular, and sawtooth wave.

Review Questions

1. Name two practical applications in which you would want to generate signals.
2. What is the normalized signal frequency for the following?
   a. sampling frequency = 100 Hz
      number of samples = 200
      signal frequency = 15 Hz
   b. sampling frequency = 100 Hz
      number of samples = 200
      signal frequency = 15 cycles
3. What are two main differences between the Wave and Pattern VIs?
4. Which of the following VIs require a normalized frequency input?
   a. Sine wave
   b. Sine pattern
   c. Chirp pattern
   d. Square wave

Notes

Notes



Lesson 3
Signal Processing

Introduction

In this lesson, you will learn the basics of transforming a signal from the time domain into the frequency domain.

You Will Learn:

A. About the discrete Fourier transform (DFT) and the fast Fourier transform (FFT).

B. How to determine the frequency spacing between the samples of the FFT (that is, the relationship between the sampling frequency *fs*, number of samples *N*, and the frequency spacing Δf).

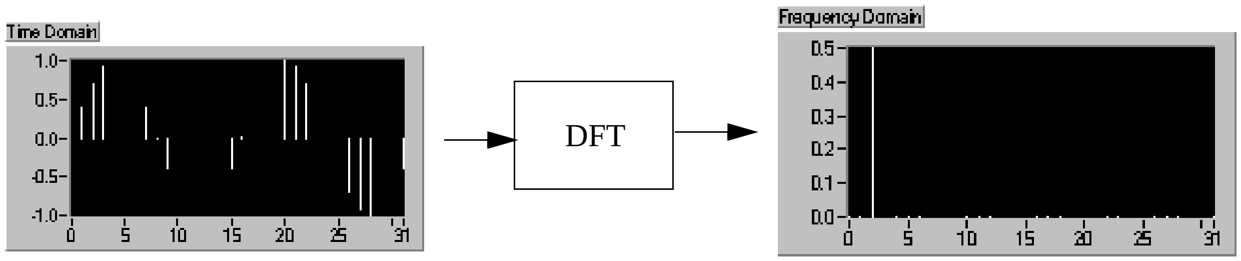
C. About the power spectrum and how it differs from both the DFT and the FFT.

D. About how to interpret the information in the frequency domain for the DFT/FFT and the power spectrum, for both even and odd *N*.

A. The Discrete Fourier Transform (DFT) and the Fast Fourier Transform (FFT)

The samples of a signal obtained from a DAQ board constitute the *time domain* representation of the signal. This representation gives the amplitudes of the signal at the instants of *time* during which it had been sampled. However, in many cases you want to know the frequency content of a signal rather than the amplitudes of the individual samples. The representation of a signal in terms of its individual frequency components is known as the *frequency domain* representation of the signal. The frequency domain representation could give more insight about the signal and the system from which it was generated.

The algorithm used to transform samples of the data from the time domain into the frequency domain is known as the *discrete Fourier transform* or DFT. The DFT establishes the relationship between the samples of a signal in the time domain and their representation in the frequency domain. The DFT is widely used in the fields of spectral analysis, applied mechanics, acoustics, medical imaging, numerical analysis, instrumentation, and telecommunications.



time domain representation of x[n]

frequency domain representation

Suppose you have obtained *N* samples of a signal from a DAQ board. If you apply the DFT to *N* samples of this time domain representation of the signal, the result is also of length *N* samples, but the information it contains is of the frequency domain representation. The relationship between the *N* samples in the time domain and the *N* samples in the frequency domain is explained below.

If the signal is sampled at a sampling rate of f_S Hz, then the time interval between the samples (that is, the sampling interval) is Δt, where

$$\Delta t = \frac{1}{f_s}$$

The sample signals are denoted by x[i], 0 ð i ð *N*-1 (that is, you have a total of *N* samples). When the discrete Fourier transform, given by

$$X_k = \sum_{i=0}^{N-1} x_i e^{-j2\pi ik/N} \quad \text{for } k = 0, 1, 2, \ldots, N\text{-1} \qquad (1)$$

is applied to these *N* samples, the resulting output (*X*[k], 0 ð k ð N-1) is the frequency domain representation of x[i]. Note that both the time domain *x* and the frequency domain *X* have a total of *N* samples. Analogous to the *time spacing* of Δt between the samples of *x* in the time domain, you have a frequency spacing of

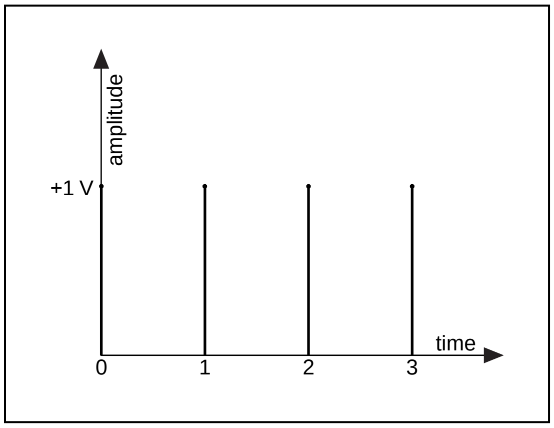
$$\Delta f = \frac{f_s}{N} = \frac{1}{N\Delta t}$$

between the components of *X* in the frequency domain. Δf is also known as the *frequency resolution*. To increase the frequency resolution (smaller Δf) you must either increase the number of samples *N* (with *fs* constant) or decrease the sampling frequency *fs* (with *N* constant).

In the following example, you will go through the mathematics of equation (1) to calculate the DFT for a DC signal.

DFT Calculation Example

In the next section, you will see the exact frequencies to which the *N* samples of the DFT correspond. For the present discussion, assume that *X*[0] corresponds to DC, or the average value, of the signal. To see the result of calculating the DFT of a waveform with the use of equation (1), consider a DC signal having a constant amplitude of +1 V. Four samples of this signal are taken, as shown in the figure below.



Each of the samples has a value +1, giving the time sequence

x[0] = x[1] = x[2] = x[3] = 1

Using equation (1) to calculate the DFT of this sequence and making use of Euler's identity,

exp (–jθ) = cos(θ) - jsin(θ)

you get:

$$X[0] = \sum_{i=0}^{N-1} x_i e^{-j2\pi i0/N} = x[0] + x[1] + x[2] + x[3] = 4$$

$$X[1] = x[0] + x[1]\left(\cos\left(\frac{\pi}{2}\right) - j\sin\left(\frac{\pi}{2}\right)\right) + x[2](\cos(\pi) - j\sin(\pi)) +$$
$$x[3]\left(\cos\left(\frac{3\pi}{2}\right) - j\sin\left(\frac{3\pi}{2}\right)\right) = (1 - j - 1 + j) = 0$$

$$X[2] = x[0] + x[1](\cos(\pi) - j\sin(\pi)) + x[2](\cos(2\pi) - j\sin(2\pi)) +$$
$$x[3](\cos(3\pi) - j\sin(3\pi)) = (1 - 1 + 1 - 1) = 0$$

$$X[3] = x[0] + x[1]\left(\cos\left(\frac{3\pi}{2}\right) - j\sin\left(\frac{3\pi}{2}\right)\right) + x[2](\cos(3\pi) - j\sin(3\pi)) +$$
$$x[3]\left(\cos\left(\frac{9\pi}{2}\right) - j\sin\left(\frac{9\pi}{2}\right)\right) = (1 - j - 1 - j) = 0$$

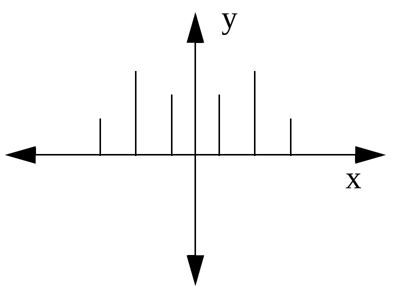
Therefore, except for the DC component, *X*[0], all the other values are zero, which is as expected. However, the calculated value of *X*[0] depends on the value of *N* (the number of samples). Because you had *N* = 4, X[0] = 4. If *N* = 10, then you would have calculated *X*[0] = 10. This dependency of *X*[.] on *N* also occurs for the other frequency components. Thus, you usually divide the DFT output by *N*, so as to obtain the correct magnitude of the frequency component.

Magnitude and Phase Information

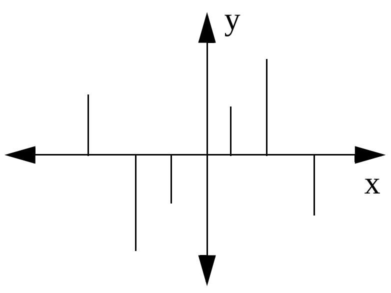
You have seen that *N* samples of the input signal result in *N* samples of the DFT. That is, the number of samples in both the time and frequency representations is the same. From equation (1), you see that regardless of whether the input signal *x*[i] is real or complex, *X*[k] is always complex (although the imaginary part may be zero). Thus, because the DFT is complex, it contains two pieces of information—the amplitude and the phase. It turns out that for real signals (*x*[i] real) such as those obtained from the output of one channel of a DAQ board, the DFT is symmetric about the index N/2 with the following properties:

| *X*[k] | = | *X*[N-k] | and phase (*X*[k]) = - phase(*X*[N-k])

The terms used to describe this symmetry are that the magnitude of X[k] is *even symmetric*, and phase(X[k]) is *odd symmetric*. An even symmetric signal is one that is symmetric about the y-axis, whereas an odd symmetric signal is symmetric about the origin. This is shown in the following figures.



even symmetry



odd symmetry

The net effect of this symmetry is that there is repetition of information contained in the *N* samples of the DFT. Because of this repetition of information, only half of the samples of the DFT actually need to be computed or displayed, as the other half can be obtained from this repetition.

Note: ***If the input signal is complex, the DFT will be nonsymmetric and you cannot use this trick.***

B. Frequency Spacing and Symmetry of the DFT/FFT

Because the sampling interval is Δt seconds, and if the first ($k = 0$) data sample is assumed to be at 0 seconds, the k*th* ($k > 0$, k integer) data sample is at $k\Delta t$ seconds. Similarly, the frequency resolution being Δf ($\Delta f = \frac{f_s}{N}$) means that the k*th* sample of the DFT occurs at a frequency of $k\Delta f$ Hz. (Actually, as you will soon see, this is valid for only up to about half the number of samples. The other half represent negative frequency components.) Depending on whether the number of samples, N, is even or odd, you can have a different interpretation of the frequency corresponding to the *k*th sample of the DFT.

Even Number of Samples

For example, suppose N is even and let $p = \frac{N}{2}$. The following table shows the frequency to which each element of the complex output sequence X corresponds.

Array Element	Corresponding Frequency
X[0]	DC component
X[1]	Δf
X[2]	$2\Delta f$
X[3]	$3\Delta f$
. . .	. . .
X[p-2]	$(p\text{-}2)\Delta f$
X[p-1]	$(p\text{-}1)\Delta f$
X[p]	$p\Delta f$ (Nyquist frequency)
X[p+1]	$- (p\text{-}1)\Delta f$
X[p+2]	$- (p\text{-}2)\Delta f$
. . .	. . .
X[N-3]	$- 3\Delta f$
X[N-2]	$- 2\Delta f$
X[N-1]	$- 1\Delta f$

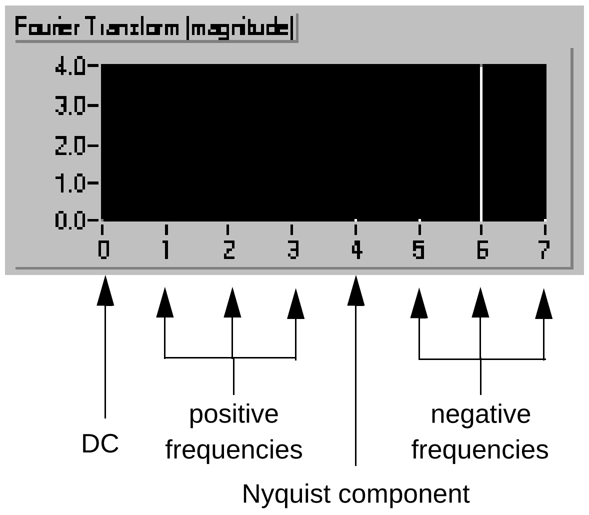
Note that the p^{th} element, X[p], corresponds to the Nyquist frequency. The negative entries in the second column beyond the Nyquist frequency represent *negative* frequencies.

For example, if $N = 8$, $p = N/2 = 4$, then

X[0]	DC
X[1]	Δf
X[2]	$2\Delta f$
X[3]	$3\Delta f$
X[4]	$4\Delta f$ (Nyquist freq)
X[5]	$-3\Delta f$
X[6]	$-2\Delta f$
X[7]	$-\Delta f$

Here, X[1] and X[7] will have the same magnitude, X[2] and X[6] will have the same magnitude, and X[3] and X[5] will have the same magnitude. The difference is that whereas X[1], X[2], and X[3] correspond to positive frequency components, X[5], X[6], and X[7] correspond to negative frequency components. Note that X[4] is at the Nyquist frequency.

The following illustration represents this complex sequence for $N = 8$.



Such a representation, where you see both the positive and negative frequencies, is known as the *two-sided* transform.

Odd Number of Samples

Now suppose that *N* is odd. Let $p = \frac{N-1}{2}$. The following table shows the frequency to which each element of the complex output sequence *X* corresponds.

Array Element	**Corresponding Frequency**
X[0]	DC component
X[1]	Δf
X[2]	$2\Delta f$
X[3]	$3\Delta f$
. . .	. . .
X[p-1]	$(p\text{-}1)\Delta f$
X[p]	$\mathrm{p}\Delta f$
X[p+1]	$-p\Delta f$
X[p+2]	$-(p\text{-}1)\Delta f$
. . .	. . .
X[N-3]	$-3\Delta f$
X[N-2]	$-2\Delta f$
X[N-1]	$-\Delta f$

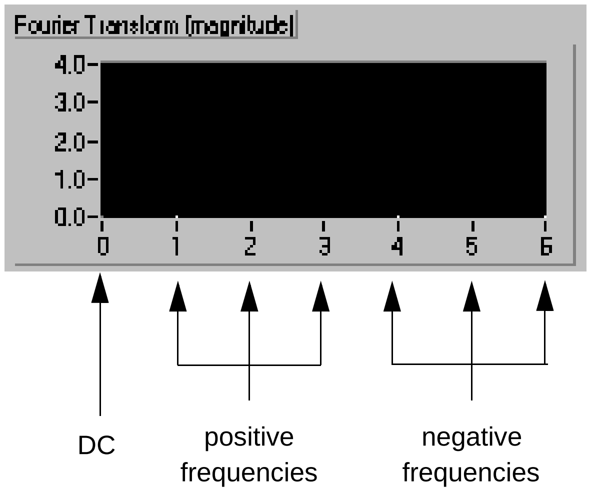
Note that when *N* is odd, *N*/2 is not an integer, and thus there is no component at the Nyquist frequency.

For example, if N = 7, p = (N-1)/2 = (7-1)/2 = 3, and you have

X[0]	DC
X[1]	Δf
X[2]	$2\Delta f$
X[3]	$3\Delta f$
X[4]	$-3\Delta f$
X[5]	$-2\Delta f$
X[6]	$-\Delta f$

Now X[1] and X[6] have the same magnitude, X[2] and X[5] have the same magnitude, and X[3] and X[4] have the same magnitude. However, whereas X[1], X[2], and X[3] correspond to positive frequencies, X[4], X[5], and X[6] correspond to negative frequencies. Because N is odd, there is no component at the Nyquist frequency.

The following illustration represents the preceding table for $N = 7$.



This is also a two-sided transform, because you have both the positive and negative frequencies.

Fast Fourier Transforms

Direct implementation of the DFT (equation (1) on page 3-3) on N data samples requires approximately N^2 complex operations and is a time-consuming process. However, when the size of the sequence is a power of 2,

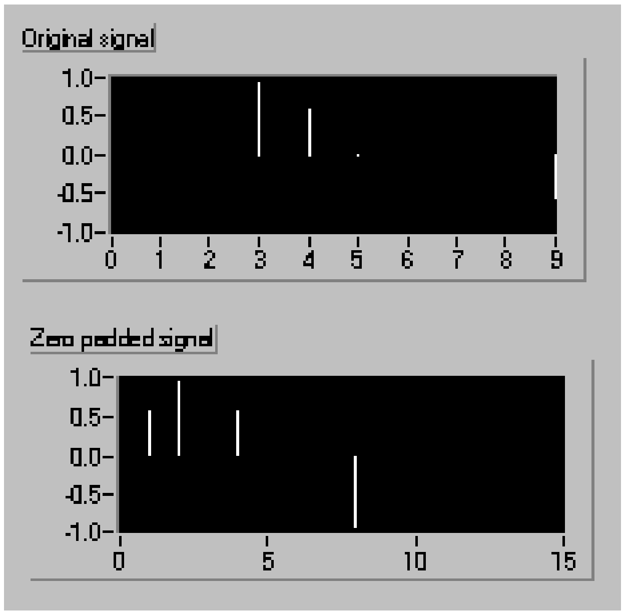
$$N = 2^m \quad \text{for } m = 1, 2, 3, \ldots$$

you can implement the computation of the DFT with approximately $N \log_2(N)$ operations. This makes the calculation of the DFT much faster, and DSP literature refers to these algorithms as fast Fourier transforms (FFTs). The FFT is nothing but a fast algorithm for calculating the DFT when the number of samples (N) is a power of 2.

The advantages of the FFT include speed and memory efficiency, because the VI performs the transform in place. The size of the input sequence, however, must be a power of 2. The DFT can efficiently process any size sequence, but the DFT is slower than the FFT and uses more memory, because it must store intermediate results during processing.

Zero Padding

A technique employed to make the input sequence size equal to a power of 2 is to add zeros to the end of the sequence so that the total number of samples is equal to the next higher power of 2. For example, if you have 10 samples of a signal, you can add six zeros to make the total number of samples equal to 16 (= 2^4—a power of 2). This is shown below:



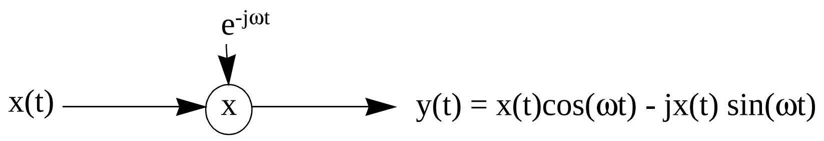
In addition to making the total number of samples a power of two so that faster computation is made possible by using the FFT, zero padding also helps in increasing the frequency resolution (recall that Δf = fs/N) by increasing the number of samples, *N*.

FFT VIs in the Analysis Library

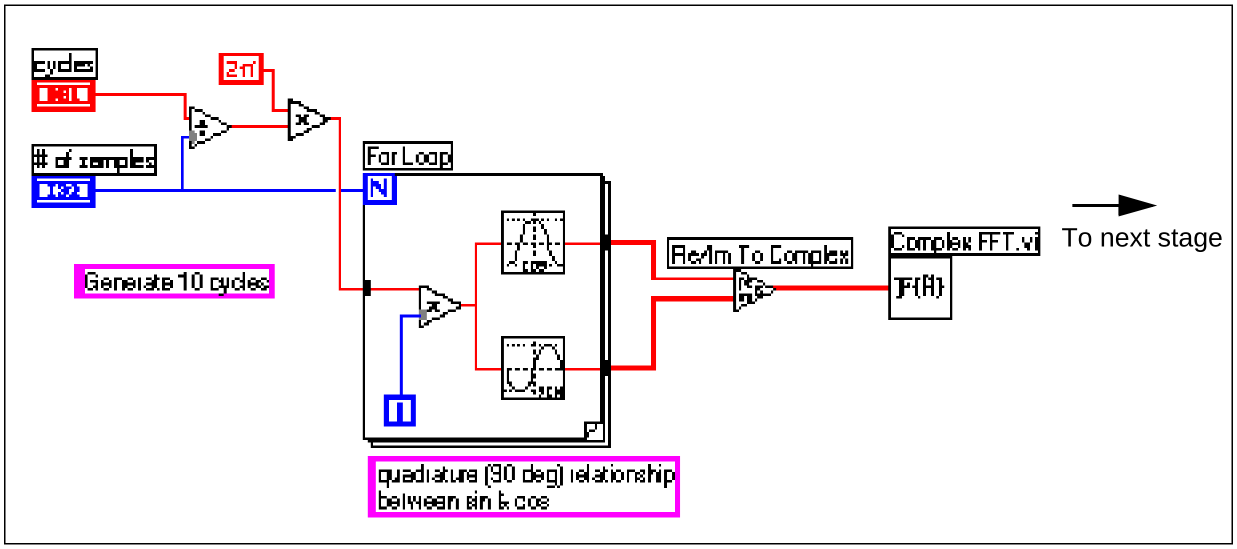
The analysis library contains two VIs that compute the FFT of a signal. They are the **Real FFT** and **Complex FFT**.

The difference between the two VIs is that the **Real FFT** computes the FFT of a real-valued signal, whereas the **Complex FFT** computes the FFT of a complex-valued signal. However, keep in mind that the outputs of both VIs are complex.

Most real-world signals are real valued, and hence you can use the **Real FFT** for most applications. Of course, you could also use the **Complex FFT** by setting the imaginary part of the signal to zero. An example of an application where you could use the **Complex FFT** is when the signal consists of both a real and imaginary component. Such a type of signal occurs frequently in the field of telecommunications, where you modulate a waveform by a complex exponential. The process of modulation by a complex exponential results in a complex signal, as shown below:



The block diagram below shows a simplified version of how you can generate 10 cycles of a complex signal:

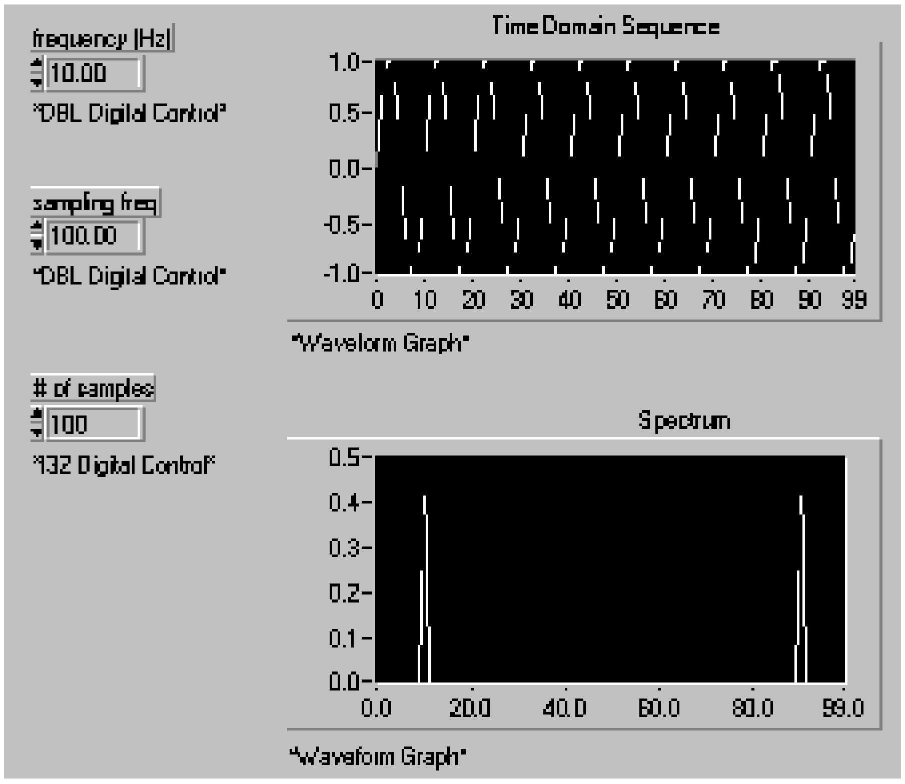


Exercise 3-1

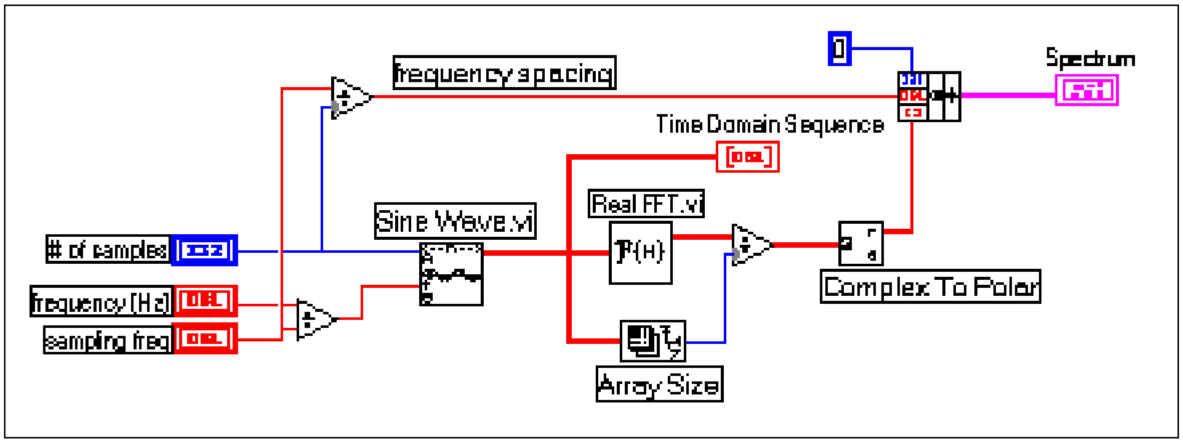
Objective: To display the two-sided and the one-sided Fourier transform of a signal using the Real FFT VI, and to observe the effect of aliasing in the frequency spectrum.

1. Build the VI front panel and block diagram as shown below.

Front Panel



Block Diagram



Array Size function (**Functions » Array** subpalette) scales the output of the FFT by the number of samples so as to obtain the correct amplitude of the frequency components.

Sine Wave function (**Functions » Analysis » Signal Generation** subpalette) generates a time domain sinusoidal waveform.

Real FFT function (**Functions » Analysis » Digital Signal Processing** subpalette) computes the FFT of the input data samples. The output of the **Real FFT** function is divided by the FFT size (number of data points) to obtain the correct sample values.

Complex to Polar function (**Functions » Numeric » Complex** subpalette) separates the complex output of the FFT into its magnitude and phase parts. The phase information is in units of radians. Here you are displaying only the magnitude of the FFT.

The frequency spacing, Δf, is given by dividing the **sampling freq** by the **# of samples**.

2. Select **frequency (Hz)** = 10, **sampling freq** = 100, and **# of samples** = 100. Run the VI.

   Notice the plots of the time waveform and the frequency spectrum. Because **sampling freq** = **# of samples** = 100, you are in effect sampling for 1 second. Thus, the number of cycles of the sine wave you see in the time waveform is equal to the **frequency(Hz)** you select. In this case, you will see 10 cycles. (If you change the **frequency (Hz)** to 5, you will see five cycles.)

Two-Sided FFT

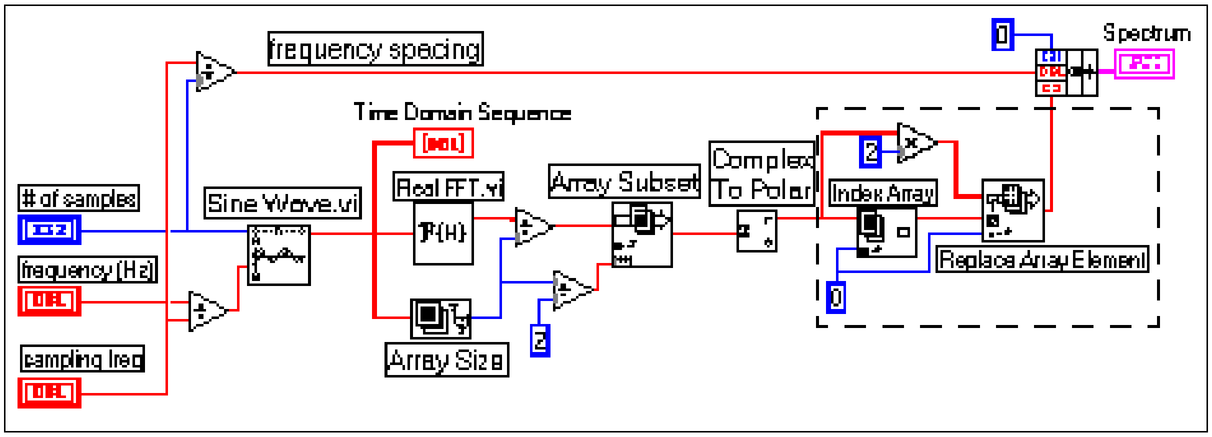
3. Examine the frequency spectrum (the Fourier transform). You will notice two peaks, one at 10 Hz and the other at 90 Hz. The peak at 90 Hz is actually the negative frequency of 10 Hz. The plot you see is known as the *2-sided FFT* because it shows both the positive and the negative frequencies.
4. Run the VI with **frequency (Hz)** = 10 and then with **frequency (Hz)** = 20. For each case, note the shift in both peaks of the spectrum.

Note: ***Also observe the time domain plot for frequency (Hz) = 10 and 20. Which one gives a better representation of the sine wave? Why?***

5. Because fs = 100 Hz, you can accurately sample only signals having a frequency < 50 Hz (Nyquist frequency = fs/2). Change **frequency (Hz)** to 48 Hz. You should see the peaks at +- 48 Hz on the spectrum plot.
6. Now change **frequency (Hz)** to 52 Hz. Is there any difference between the result of step 5 and what you see on the plots now? Because 52 > Nyquist, the frequency of 52 is aliased to |100 - 52| = 48 Hz.
7. Change **frequency (Hz)** to 30 Hz and 70 Hz and run the VI. Is there any difference between the two cases? Explain why.
8. Save this VI as **FFT_2sided.vi** in the library `Lvspcex.llb`.

One-Sided FFT

9. Modify the block diagram of the VI as shown in the following diagram. You have seen that the FFT had repetition of information because it contained information about both the positive and the negative frequencies. This modification now shows only half the FFT points (only the positive frequency components). This representation is known as the *1-sided FFT*. The 1-sided FFT shows only the positive frequency components. Note that you need to multiply the positive frequency components by two to obtain the correct amplitude. The DC component, however, is left untouched.



Equal To O? function (**Functions » Comparison** subpalette) tests to see if the array index is equal zero. If so, it corresponds to the D.C. component and should not be multiplied by two.

10. Run the VI with the following values: **frequency (Hz)** = 30, **sampling freq** = 100, **# of samples** = 100.
11. Change the value of **frequency (Hz)** to 70 and run the VI. Do you notice any difference between this and the result of step 9?
12. Save the VI as **FFT_1sided.vi** in the library `Lvspcex.llb`.

End of Exercise 3-1

C. The Power Spectrum

You have seen that the DFT (or FFT) of a real signal is a complex number, having a real and an imaginary part. The *power* in each frequency component represented by the DFT/FFT can be obtained by squaring the magnitude of that frequency component. Thus, the power in the k^{th} frequency component (the k^{th} element of the DFT/FFT) is given by $|X[k]|^2$. The plot showing the power in each of the frequency components is known as the *power spectrum*. Because the DFT/FFT of a real signal is symmetric, the power at a positive frequency of $k\Delta f$ is the same as the power at the corresponding negative frequency of $-k\Delta f$ (DC and Nyquist components not included). The total power in the DC and Nyquist components are $|X[0]|^2$ and $\left|X\left[\frac{N}{2}\right]\right|^2$, respectively.

Loss of Phase Information

Because the power is obtained by squaring the magnitude of the DFT/FFT, the power spectrum is always real and all the phase information is lost. If you want phase information, you must use the DFT/FFT, which gives you a complex output.

You can use the power spectrum in applications where phase information is not necessary (for example, to calculate the harmonic power in a signal). You can apply a sinusoidal input to a nonlinear system and see the power in the harmonics at the system output.

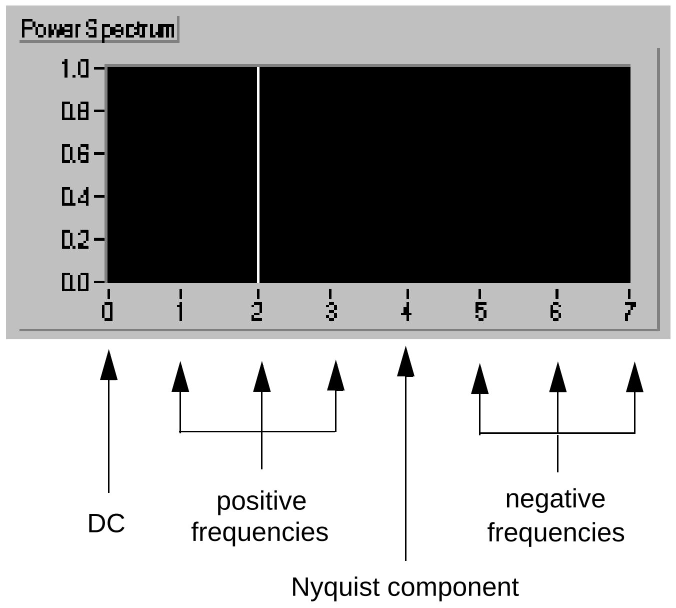
Frequency Spacing Between Samples

You can use the **Power Spectrum** VI in the **Analysis » Digital Signal Processing** subpalette to calculate the power spectrum of the time domain data samples. Just like the DFT/FFT, the number of samples from the **Power Spectrum** VI output is the same as the number of data samples applied at the input. Also, the frequency spacing between the output samples is Δf = fs/N.

In the following table, the power spectrum of a signal x[n] is represented by *Sxx*. If *N* is even, let $p = \frac{N}{2}$. The following table shows the format of the output sequence *Sxx* corresponding to the power spectrum.

Array Element	Interpretation
Sxx[0]	Power in DC component
Sxx[1] = *Sxx*[N-1]	Power at frequency Δf
Sxx[2] = *Sxx*[N-2]	Power at frequency $2\Delta f$
Sxx[3] = *Sxx*[N-3]	Power at frequency $3\Delta f$
. . .	. . .
Sxx[p-2]= *Sxx*[N-(p-2)]	Power at frequency $(p\text{-}2)\Delta f$
Sxx[p-1] = *Sxx*[N-(p-1)]	Power at frequency $(p\text{-}1)\Delta f$
Sxx[p]	Power at Nyquist frequency

The following illustration represents the information in the preceding table for a sine wave with amplitude = 2 V_{peak} (V_{pk}), and N = 8.

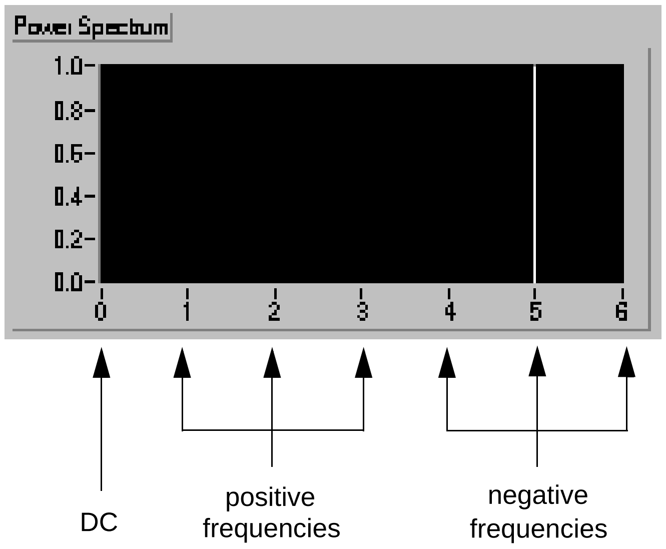


The output units of the **Power Spectrum** VI are in Volts rms squared (V^2_{rms}). So, if the peak amplitude (V_{pk}) of the input signal is 2 V_{pk}, its rms value is $V_{rms} = \frac{2}{\sqrt{2}} = \sqrt{2}$, so $V^2_{rms} = 2$. This value is divided equally between the positive and negative frequency components, resulting in the plot shown above.

If N is odd, let $p = \frac{N-1}{2}$. The following table shows the format of the output sequence *Sxx* corresponding to the power spectrum.

Array Element	**Interpretation**
Sxx[0]	Power in DC component
Sxx[1] = *Sxx*[N-1]	Power at frequency Δf
Sxx[2]= *Sxx*[N-2]	Power at frequency $2\Delta f$
Sxx[3] = *Sxx*[N-3]	Power at frequency $3\Delta f$
. . .	. . .
Sxx[p-2] = *Sxx*[N-(p-2)]	Power at frequency $(p\text{-}2)\Delta f$
Sxx[p-1] = *Sxx*[N-(p-1)]	Power at frequency $(p\text{-}1)\Delta f$
Sxx[p] = *Sxx*[p]	Power at frequency $p\Delta f$

The following illustration represents the information in the preceding table for N = 7.

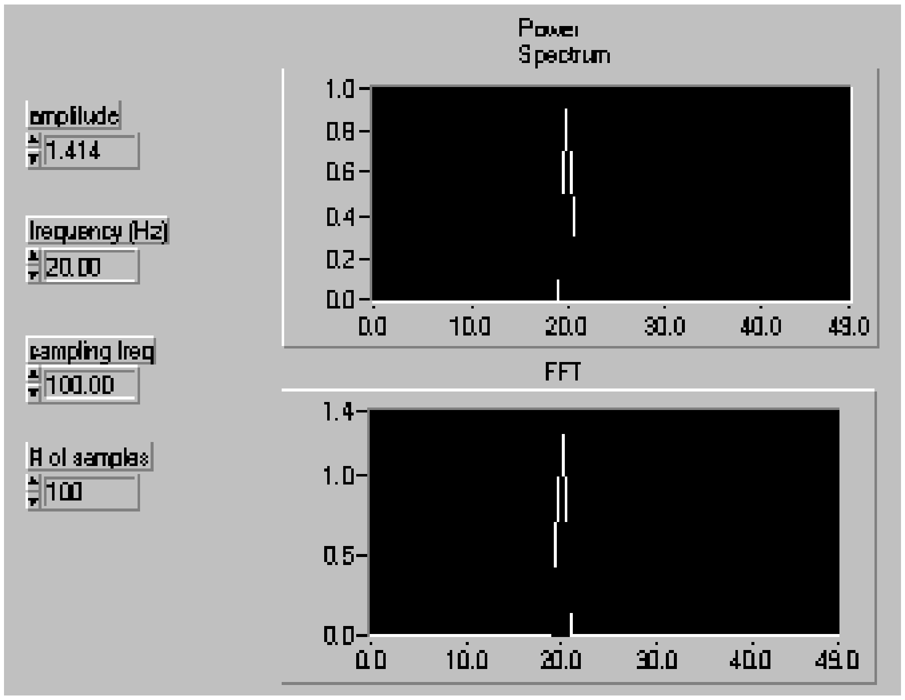


Exercise 3-2

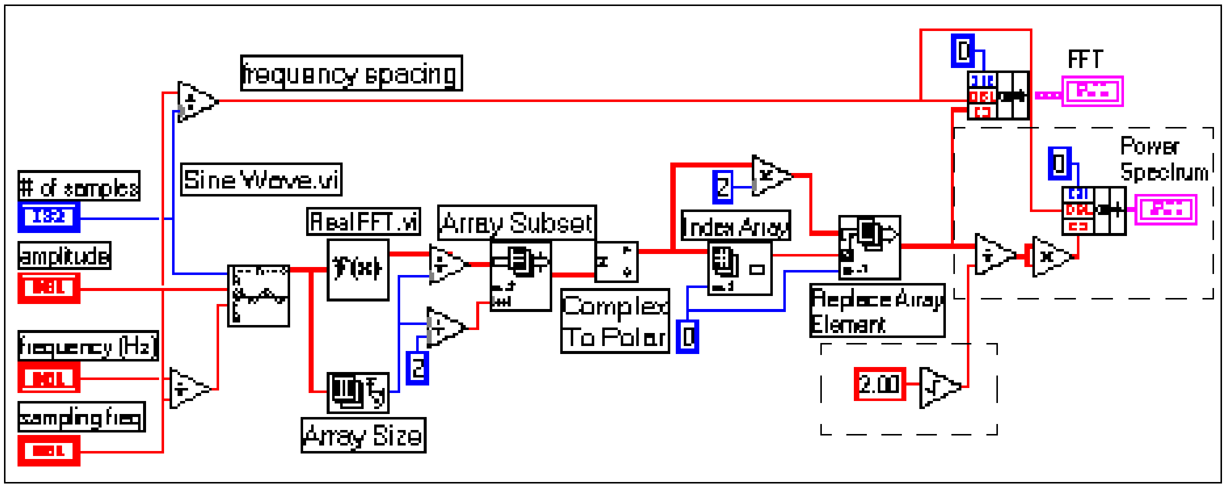
Objective: To observe the difference between the FFT and the power spectrum representations.

1. Open the **FFT_1sided** VI (from the library `Lvspcex.llb`) that you built in the previous exercise. Modify the block diagram and front panel as shown below.

Front Panel



Block Diagram



Sine Wave function (**Functions » Analysis » Signal Generation** palette) generates a time domain sinusoidal waveform.

Real FFT function (**Functions » Analysis » Digital Signal Processing** subpalette) computes the FFT of the input data samples.

Array Subset (**Functions » Array** subpalette) returns a portion of the array. Here you are selecting half the array.

Complex to Polar function (**Functions » Numeric » Complex** subpalette) separates the complex output of the FFT into its magnitude and phase parts. The phase information is in radians. Here, you are displaying only the magnitude of the FFT.

The power spectrum is obtained by squaring the magnitude of the FFT. The division by $\sqrt{2}$ (1.414) makes the conversion from V_{pk} to V_{rms}.

Note: ***You could also have wired the output of the Sine Wave VI directly to the input of the Power Spectrum VI (Analysis » Digital Signal Processing subpalette). The output of the Power Spectrum VI would directly be the power spectrum of the signal. However, in that case, the phase information would be lost.***

2. Enter the following values in the controls: **amplitude** = 1.414, **frequency** = 20 Hz, **sampling freq** = 100, and **# of samples** = 100, and run the VI. Do you notice any difference in the FFT and power spectrum representations?
3. Change the amplitude to 1.00 and run the VI. What difference do you notice in the FFT and power spectrum representations?
4. Save the VI as **FFT and Power Spectrum.vi** in the library `Lvspcex.llb`.

End of Exercise 3-2

Summary

- The time domain representation (sample values) of a signal can be converted into the frequency domain representation using the discrete Fourier transform (DFT).
- Fast calculation of the DFT is possible by using an algorithm known as the fast Fourier transform (FFT). You can use this algorithm when the number of signal samples is a power of two.
- The output of the conventional DFT/FFT is two-sided because it contains information about both the positive and the negative frequencies. This output can be converted into a one-sided DFT/FFT by using only half the number of output points.
- The frequency spacing between the samples of the DFT/FFT is Δf = fs/N.
- The power spectrum can be calculated from the DFT/FFT by squaring the magnitude of the individual frequency components. The **Power Spectrum** VI in the advanced analysis library does this automatically for you. The **Power Spectrum** VI units of the output are V^2_{rms}. However, the power spectrum does not provide any phase information.
- The DFT, FFT, and power spectrum are useful for measuring the frequency content of stationary or transient signals. The FFT provides the average frequency content of the signal over the entire time that the signal was acquired. For this reason, you use the FFT mostly for stationary signal analysis (when the signal is not significantly changing in frequency content over the time that the signal is acquired), or when you want only the average energy at each frequency line.
- For measuring frequency information that changes during the acquisition, you should use the joint time-frequency analysis (JTFA) toolkit or the wavelet and filter banks designer (WFBD) toolkit. These toolkits are covered in later lessons.

Review Questions

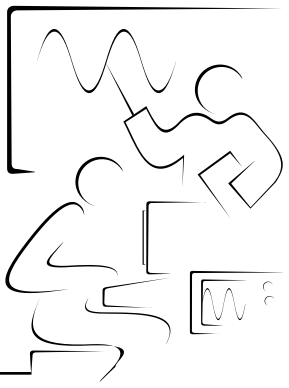
1. Which of the following provides you with both the magnitude and phase information?
   a. FFT
   b. Power spectrum
   c. DFT
   d. Time domain waveform
2. Which of the following are true?
   a. The magnitude spectrum is always even symmetric.
   b. The DFT is a fast algorithm for computing the FFT.
   c. The frequency spacing is given by

      $$\Delta f = \frac{fs}{\text{number of samples}}$$

      where *fs* is the sampling frequency.
   d. An even number of samples always results in a two-sided transform.
3. If you have 1024 samples, how many times faster is the FFT as compared to the DFT in calculating the Fourier transform?

Notes

Notes



Lesson 4 Windowing

Introduction

In this lesson, you will learn about windows and how they affect the spectral characteristics of a signal.

You Will Learn:

A. About spectral leakage and smoothing windows.

B. About the difference (both time and frequency domains) between a windowed and a nonwindowed signal.

C. About the differences between the various types of windows in the Analysis library and their applications.

D. How to separate two sine waves of large amplitude difference but with frequencies very close to each other.

A. About Spectral Leakage and Smoothing Windows

In practical applications, you can obtain only a finite number of samples of the signal. When you use the DFT/FFT to find the frequency content of a signal, it is inherently assumed that the data that you have is a single period of a periodically repeating waveform. This is shown below in Figure 4-1. The first period shown is the one sampled. The waveform corresponding to this period is then repeated in time to produce the periodic waveform.

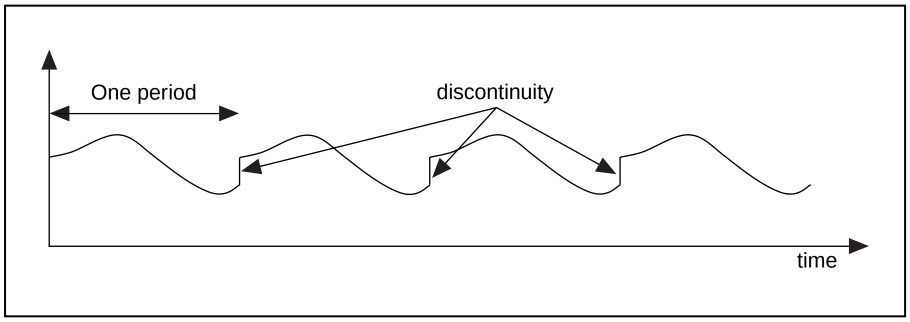


Figure 4-1. Periodic Waveform Created from Sampled Period

As seen in the previous figure, because of the assumption of periodicity of the waveform, discontinuities between successive periods will occur. This happens when you sample a noninteger number of cycles. These "artificial" discontinuities turn up as very high frequencies in the spectrum of the signal, frequencies that were not present in the original signal. These frequencies could be much higher than the Nyquist frequency, and as you have seen before, will be aliased somewhere between 0 and fs/2. The spectrum you get by using the DFT/FFT therefore will not be the actual spectrum of the original signal, but will be a smeared version. It appears as if the energy at one frequency has "leaked out" into all the other frequencies. This phenomenon is known as *spectral leakage*.

Figure 4-2 shows a sine wave and its corresponding Fourier transform. The sampled time domain waveform is shown in Graph 1. Because the Fourier transform assumes periodicity, you repeat this waveform in time, and the periodic time waveform of the sine wave of Graph 1 is shown in Graph 2. The corresponding spectral representation is shown in Graph 3. Because the time record in Graph 2 is periodic, with no discontinuities, its spectrum is a single line showing the frequency of the sine wave. The reason that the waveform in Graph 2 does not have any discontinuities is because you have sampled an integer number of cycles (in this case, 1) of the time waveform.

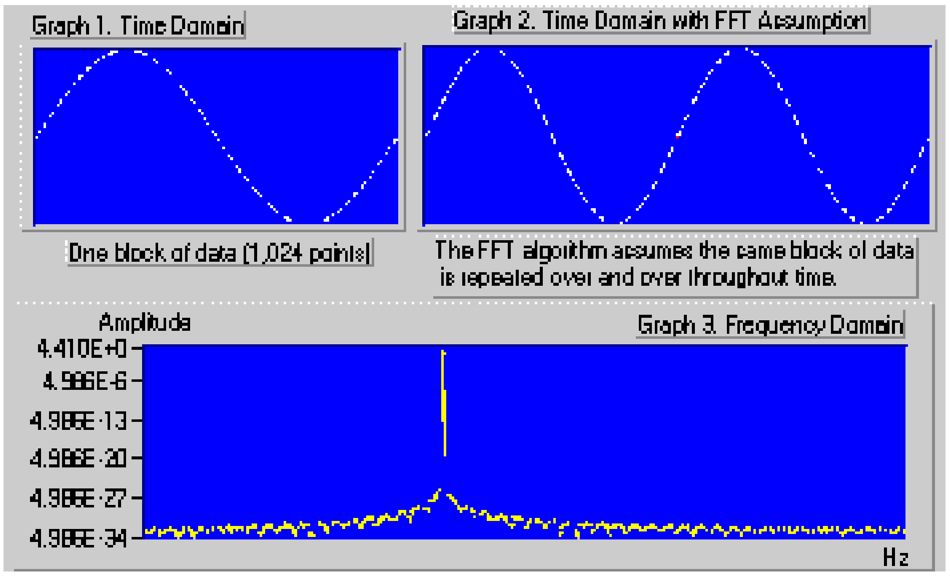

Graph 1. Time Domain
Graph 2. Time Domain with FFT Assumption
One block of data (1,024 points)
The FFT algorithm assumes the same block of data is repeated over and over throughout time.
Amplitude
Graph 3. Frequency Domain
4.410E+0
4.986E-6
4.986E-13
4.986E-20
4.986E-27
4.986E-34
Hz

Figure 4-2. Sine Wave and Corresponding Fourier Transform

In Figure 4-3, you see the spectral representation when you sample a noninteger number of cycles of the time waveform (namely 1.25). Graph1 now consists of 1.25 cycles of the sine wave. When you repeat this periodically, the resulting waveform, as shown in Graph 2, consists of discontinuities. The corresponding spectrum is shown in Graph 3. Notice how the energy is now spread over a wide range of frequencies. This smearing of the energy is *spectral leakage*. The energy has leaked out of one of the FFT lines and smeared itself into all the other lines.

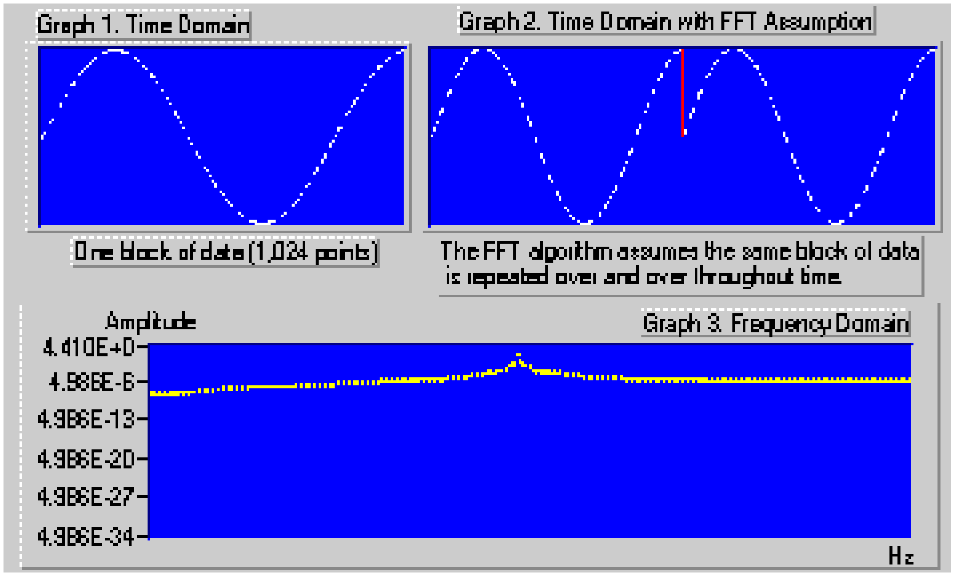

Graph 1. Time Domain
Graph 2. Time Domain with FFT Assumption
One block of data (1,024 points)
The FFT algorithm assumes the same block of data is repeated over and over throughout time.
Amplitude
Graph 3. Frequency Domain
4.410E+0
4.986E-6
4.986E-13
4.986E-20
4.986E-27
4.986E-34
Hz

Figure 4-3. Spectral Representation When Sampling a Nonintegral Number of Samples

Leakage exists because of the finite time record of the input signal. To overcome leakage, one solution is to take an infinite time record, from -× to +×. Then the FFT would calculate one single line at the correct frequency. Waiting for infinite time is, however, not possible in practice. So, because you are limited to having a finite time record, another technique, known as *windowing*, is used to reduce the spectral leakage.

The amount of spectral leakage depends on the amplitude of the discontinuity. The larger the discontinuity, the more the leakage, and vice versa. You can use windowing to reduce the amplitude of the discontinuities at the boundaries of each period. It consists of multiplying the time record by a finite length window whose amplitude varies smoothly and gradually towards zero at the edges. This is shown in Figure 4-4, where the original time signal is windowed using a *Hamming* window. Notice that the time waveform of the windowed signal gradually tapers to zero at the ends. Therefore, when performing Fourier or spectral analysis on finite-length data, you can use windows to minimize the transition edges of your sampled waveform. A smoothing window function applied to the data before it is transformed into the frequency domain minimizes spectral leakage.

Note that if the time record contains an integral number of cycles, as shown in Figure 4-2, the assumption of periodicity does not result in any discontinuities, and thus there is no spectral leakage. The problem arises only when you have a nonintegral number of cycles.

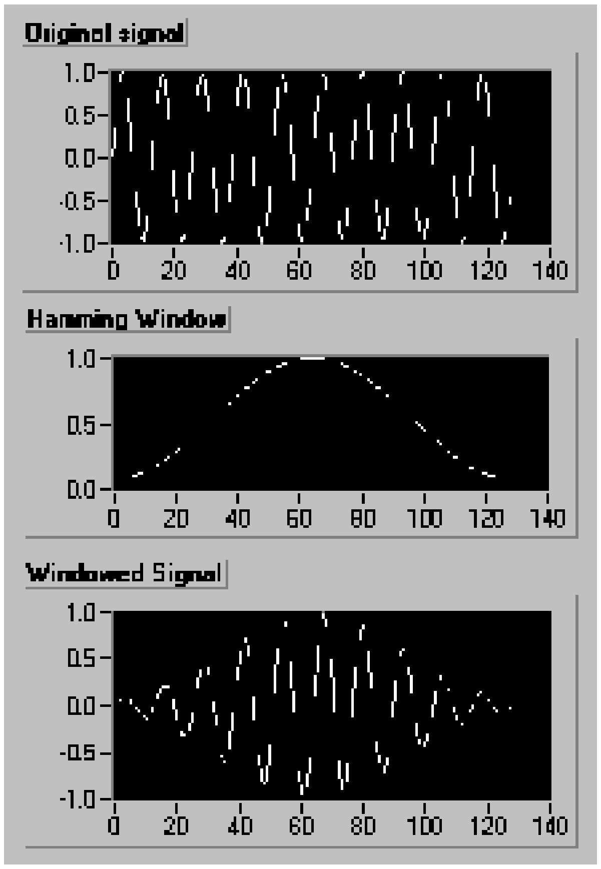


Figure 4-4. Time Signal Windowed Using a Hamming Window

B. Windowing Applications

There are several reasons to use windowing. Some of these are:

- To define the duration of the observation.
- Reduction of spectral leakage.
- Separation of a small amplitude signal from a larger amplitude signal with frequencies very close to each other.

C. Characteristics of Different Types of Window Functions

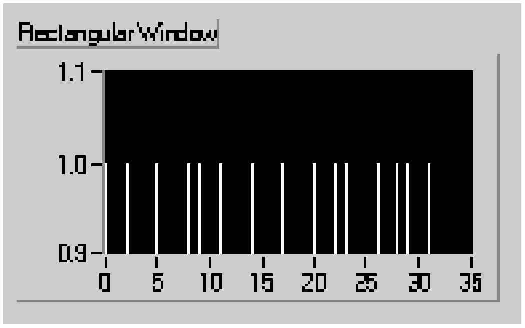
Applying a window to (windowing) a signal in the time domain is equivalent to multiplying the signal by the window function. Because multiplication in the time domain is equivalent to convolution in the frequency domain, the spectrum of the windowed signal is a convolution of the spectrum of the original signal with the spectrum of the window. Thus, windowing changes the shape of the signal in the time domain, as well as affecting the spectrum that you see.

Many different types of windows are available in the LabVIEW/ BridgeVIEW analysis library. Depending on your application, one may be more useful than the others. Some of these windows are:

1. *Rectangular (None):* The rectangular window has a value of one over its time interval. Mathematically, it can be written as:

   $$w[n] = 1.0 \quad \text{for } n = 0, 1, 2........N-1$$

   where *N* is the length of the window. Applying a rectangular window is equivalent to not using any window. This is because the rectangular function just truncates the signal to within a finite time interval. The rectangular window has the highest amount of spectral leakage. The rectangular window for *N* = 32 is shown below:

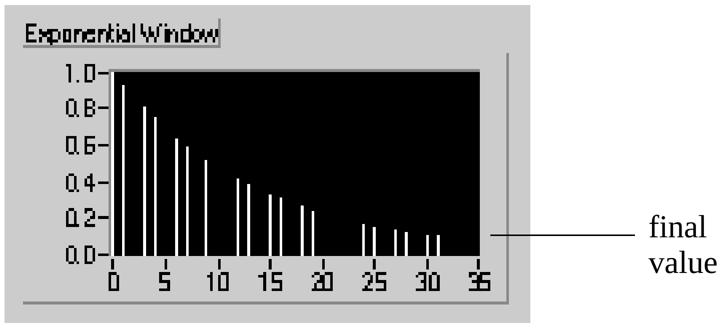
   

   The rectangular window is useful for analyzing transients that have a duration shorter than that of the window. It is also used in *order tracking,* where the sampling frequency is adjusted depending on the speed of the shaft of a machine. In this application, it detects the main mode of vibration of the machine and its harmonics.

2. *Exponential:* The shape of this window is that of a decaying exponential. It can be mathematically expressed as:

   $$w[n]= \left(\frac{n}{N-1} \times \ln(f)\right) \quad \text{for } n = 0, 1, 2.......N - 1$$

   where *f* is the final value. The initial value of the window is one, and it gradually decays towards zero. The final value of the exponential can be adjusted to between 0 and 1. The exponential window for N = 32, with the final value specified as 0.1, is shown below:

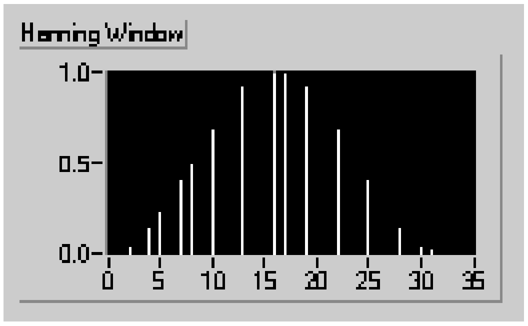


This window is useful in analyzing transients (signals that exist only for a short time duration) whose duration is longer than the length of the window. This window can be applied to signals that decay exponentially, such as the response of structures with light damping that are excited by an impact (for example, a hammer).

3. *Hanning:* This window has a shape similar to that of half a cycle of a cosine wave. Its defining equation is

   $w[n] = 0.5 - 0.5\cos(2\pi n/N)$ for $n = 0, 1, 2, ...N-1$

   A Hanning window with N = 32 is shown below:

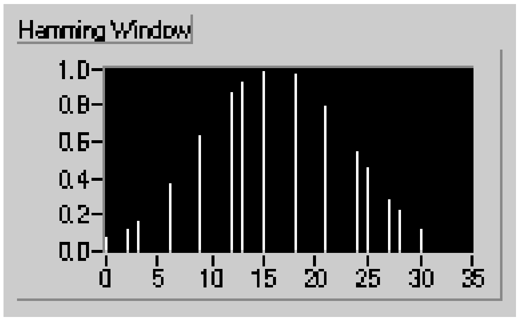


   The Hanning window is useful for analyzing transients longer than the time duration of the window, and also for general-purpose applications.

4. *Hamming:* This window is a modified version of the Hanning window. Its shape is also similar to that of a cosine wave. It can be defined as

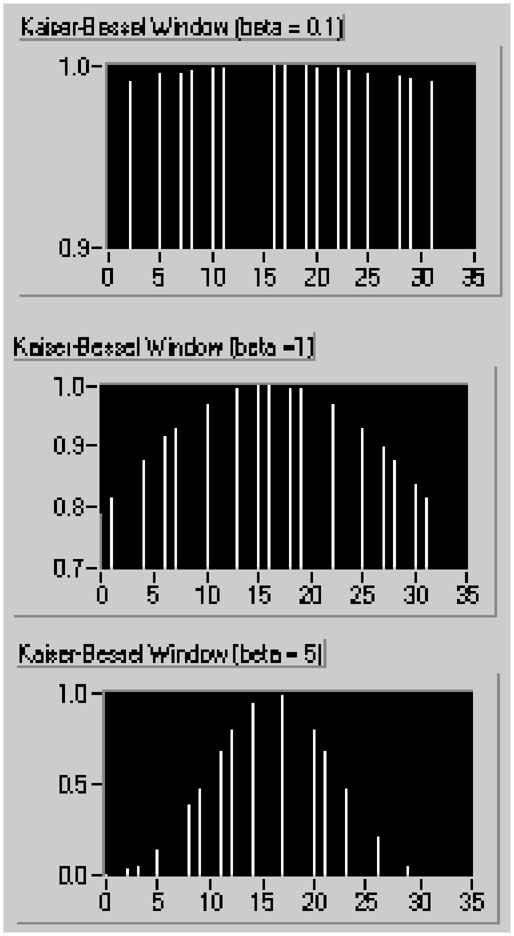
   $w[n] = 0.54 - 0.46\cos(2\pi n/N)$ for $n = 0, 1, 2, ...N-1$

A Hamming window with N = 32 is shown below:



You see that the Hanning and Hamming windows are somewhat similar. However, note that in the time domain, the Hamming window does not get as close to zero near the edges as does the Hanning window.

5. *Kaiser-Bessel:* This window is a "flexible" window whose shape the user can modify by adjusting the parameter *beta*. Thus, depending on your application, you can change the shape of the window to control the amount of spectral leakage. The Kaiser-Bessel window for different values of *beta* are shown below:



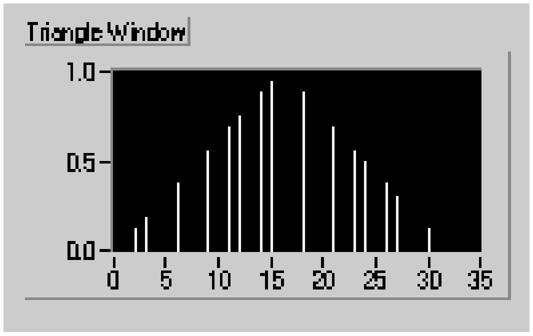
Note that for small values of beta, the shape is close to that of a rectangular window. Actually, for beta = 0 .0, you do get a rectangular window. As you increase beta, the window tapers off more to the sides.

This window is good for detecting two signals of almost the same frequency, but significantly different amplitudes.

6. *Triangle:* The shape of this window is that of a triangle. It is given by

   w[n] = 1 - | (2n-N) / N | for n = 0, 1, 2, ..., n–1

   A triangle window for N = 32 is shown below:



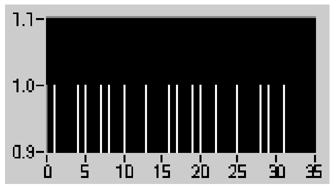
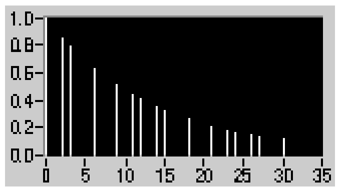
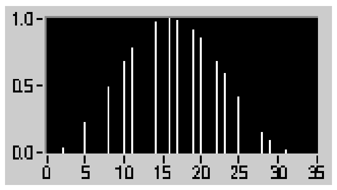
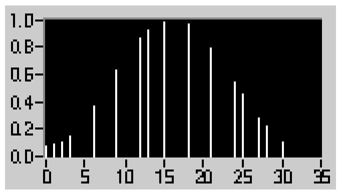
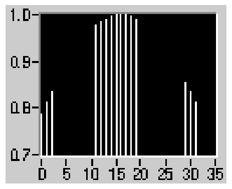
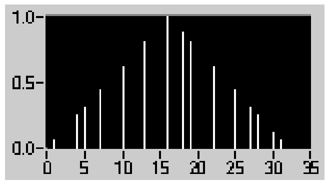
What Type of Window Do I Use?

Now that you have seen several of the many different types of windows that are available, you may ask, “What type of window should I use?” The answer depends on the type of signal you have and what you are looking for. Choosing the correct window requires some prior knowledge of the signal that you are analyzing. In summary, the following table shows the different types of signals and the appropriate windows that you can use with them.

Type of signal	Window
Transients whose duration is shorter than the length of the window	Rectangular
Transients whose duration is longer than the length of the window	Exponential, Hanning
General-purpose applications	Hanning
Order tracking	Rectangular
System analysis (frequency response measurements)	Hanning (for random excitation), rectangular (for pseudorandom excitation)
Separation of two tones with frequencies very close to each other, but with widely differing amplitudes	Kaiser-Bessel
Separation of two tones with frequencies very close to each other, but with almost equal amplitudes	Rectangular

In many cases, you may not have sufficient prior knowledge of the signal, so you need to experiment with different windows to find the best one.

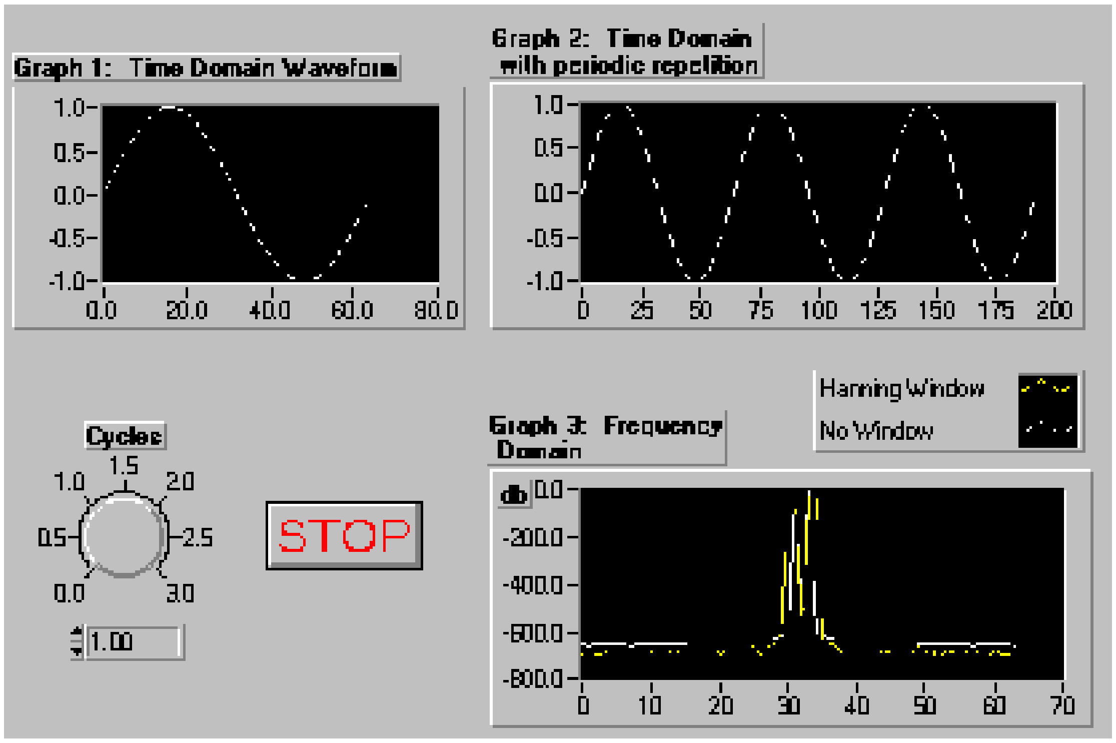
The following table summarizes the different windows.

Window	Equation	Shape	Applications
Rectangular (None)	$w[n] = 1.0$		Detecting transients whose duration is shorter than the length of the window; order tracking; separating two tones with frequencies and amplitudes very close to each other; system response
Exponential	$w[n] = \frac{n}{N-1}\ln(f)$ where f = final value		Transients whose duration is longer than the length of the window
Hanning	$w[n] = 0.5 - 0.5\cos\left(\frac{2\pi n}{N}\right)$		General-purpose applications; system analysis; transients whose duration is longer than the length of the window
Hamming	$w[n] = 0.54 - 0.46\cos\left(\frac{2\pi n}{N}\right)$		
Kaiser-Bessel	$w[n] = \frac{I_0(\beta\sqrt{1-a^2})}{I_0(\beta)}$		Separation of two tones with frequencies very close to each other, but with almost equal amplitudes
Triangle	$w[n] = 1 - \left\lvert\frac{2n-N}{N}\right\rvert$		

Exercise 4-1

Objective: To see the effect of windowing on spectral leakage.

1. Open the **Spectral Leakage** VI from the library `Lvspcex.llb`. The VI is running when it opens. Using this VI, you can see the effect of windowing on spectral leakage.


Graph 1: Time Domain Waveform
Graph 2: Time Domain with periodic repetition
Graph 3: Frequency Domain
Hanning Window
No Window
Cycles
1.00
STOP
db

Note: ***The Spectral Leakage VI searches for the Nyquist Shift VI. The Nyquist Shift VI is in LabVIEW » Examples » Analysis » dspxmpl.llb.***

You can see three plots on the front panel:

Graph 1 shows the time record of the signal that has been sampled.

Graph 2 shows the repeated time record (assuming periodicity).

Graph 3 shows the frequency spectrum (in dB). The white line shows the spectrum without windowing and the yellow line shows the spectrum by windowing using a Hanning window.

You can use the **cycles** dial to control the number of time domain waveform cycles that have been sampled. The display below the **Cycles** dial tells you the exact number of cycles (to two decimal places). You can also type a specific value in this display.

2. First, you will see the effect of windowing when you sample an integral number of cycles.

   Set the **Cycles** dial to 1.0. (or type 1.0 in the display beneath it.)

As you can see in **Graph 1**, you have exactly one cycle of the time waveform. **Graph 2** shows the repeated time record. Notice the absence of any discontinuities, and the peak corresponding to the sine wave in the frequency domain (**Graph 3**). You see two peaks because you have the two-sided spectrum.

Notice the spreading around the frequency components. This spreading is the effect of using a window function. In this case, you used the Hanning window. Different windows have different amounts of spreading.

3. Now see what happens when you sample a nonintegral number of cycles. Set the **Cycles** dial to 1.3 and observe the difference in the plots in **Graph 3**. Experiment by changing the cycles dial and observing the waveforms in graphs 2 and 3.

   In the white plot corresponding to **No Window**, the energy in the frequency of interest spreads out across the spectrum. Hence the frequency of interest is sometimes not clearly distinguishable. In the yellow plot corresponding to **Hanning Window**, the spectral leakage across the spectrum is reduced and the energy is more concentrated around the frequency of interest.
4. Stop the VI by pressing the STOP button.
5. Close the VI. Do not save any changes.

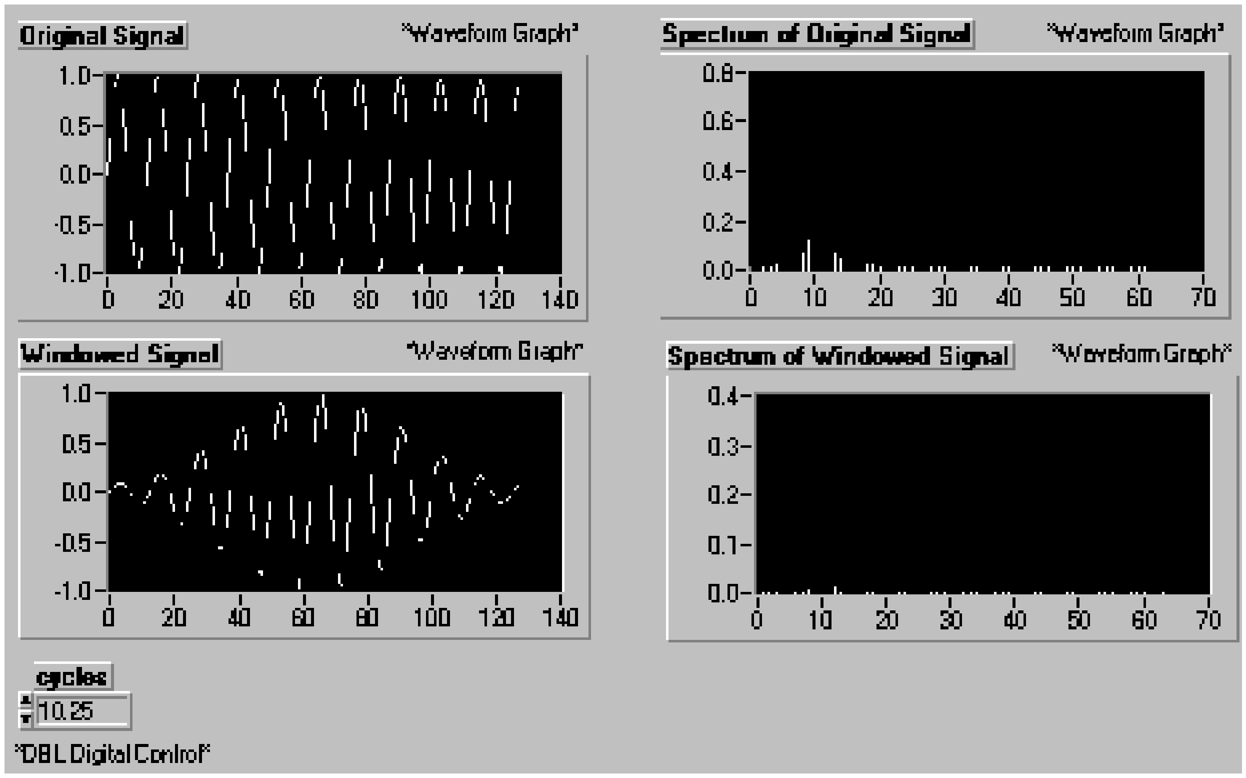
End of Exercise 4-1

Exercise 4-2 (Optional)

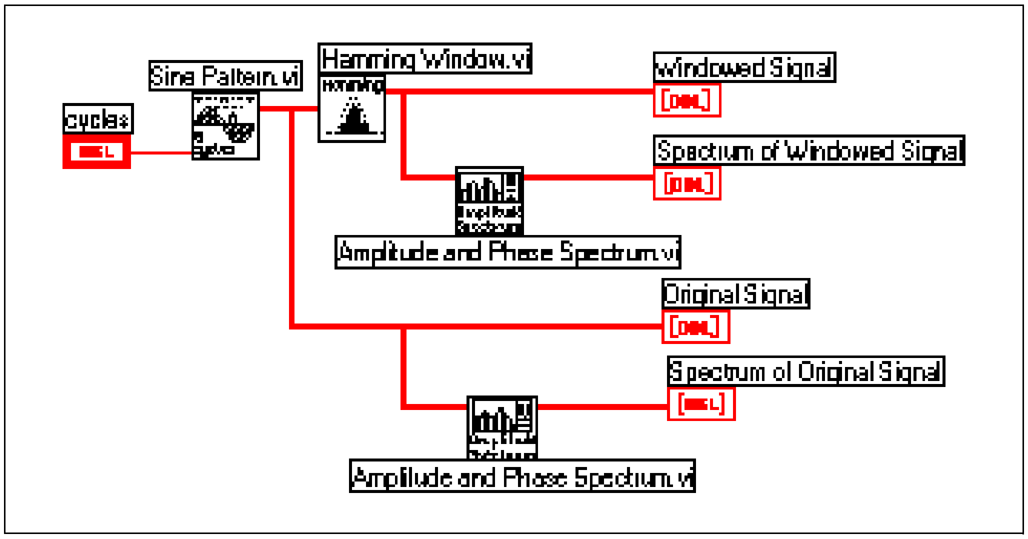
Objective: To see the difference (both time and frequency domains) between a windowed and nonwindowed signal.

1. Build the VI front panel and block diagram as shown below.

Front Panel



Block Diagram


cycles
Sine Pattern.vi
Hamming Window.vi
Windowed Signal
Spectrum of Windowed Signal
Amplitude and Phase Spectrum.vi
Original Signal
Spectrum of Original Signal
Amplitude and Phase Spectrum.vi

The **Sine Pattern** VI (**Functions » Analysis » Signal Generation** subpalette) generates a sine wave with the number of cycles specified in the **cycles** control.

The time waveform of the sine wave is windowed using the **Hamming Window** VI (**Functions » Analysis » Windows** subpalette), and both the windowed and nonwindowed time waveforms are displayed on the left two plots on the front panel.

The **Amplitude and Phase Spectrum** VI (**Functions » Analysis » Measurement** subpalette) obtains the amplitude spectrum of the windowed and nonwindowed time waveforms. These waveforms are displayed on the two plots on the right side of the front panel.

2. Set **cycles** to 10 (an integral number) and run the VI. Note that the spectrum of the windowed signal is broader (wider) than the spectrum of the nonwindowed signal. But both the spectra are concentrated near 10 on the x-axis.
3. Change **cycles** to 10.25 (a nonintegral number) and run the VI. Note that the spectrum of the nonwindowed signal is now more spread out than it was before. This is because now you have a noninteger number of cycles, and when you repeat the waveform to make it periodic, you get discontinuities. The spectrum of the windowed signal is still concentrated, but that of the nonwindowed signal has now smeared all over the frequency domain. (This is spectral leakage.)

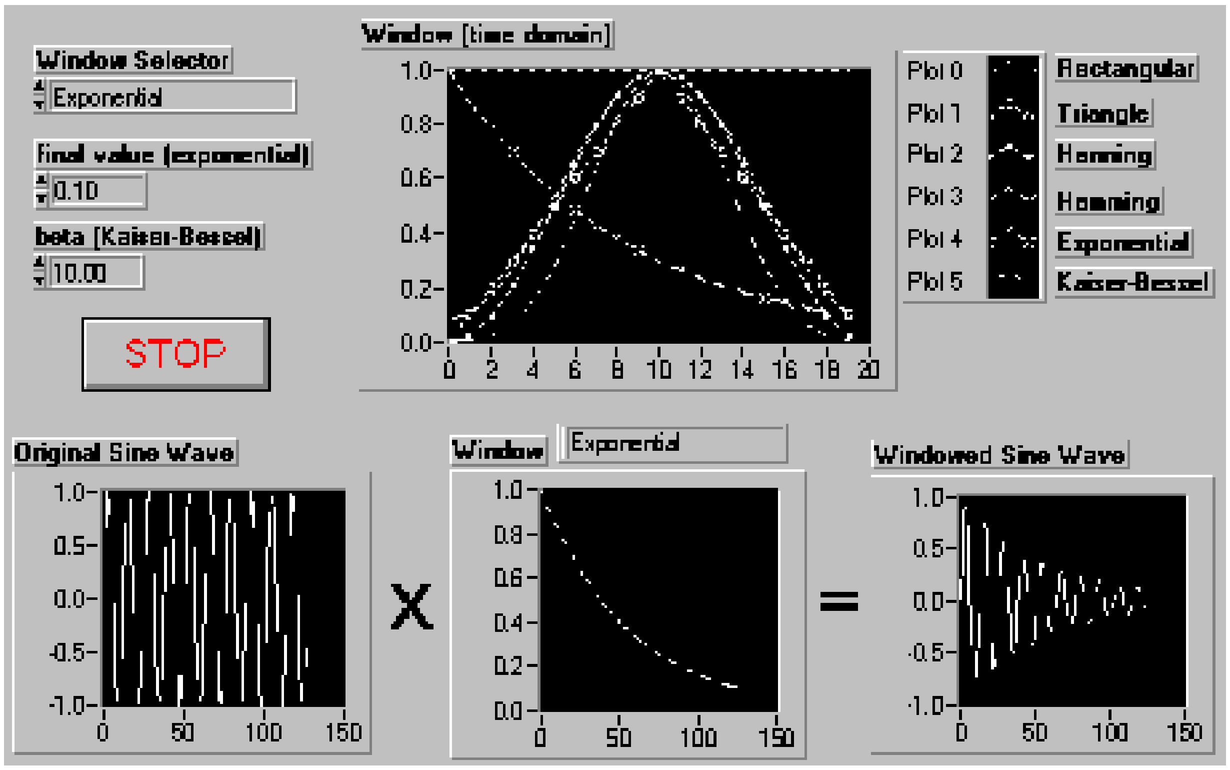
4. Change **cycles** to 10.5 and observe the frequency domain plots. Spectral leakage of the original signal is clearly apparent.
5. When you finish, save the VI as **Windowed and Unwindowed Signal.vi** in the library `Lvspcex.llb`.
6. Close the VI.

End of Exercise 4-2

Exercise 4-3

Objective: To learn about the different windows in the Analysis library.

1. Open the **Window Plots** VI from the library `Lvxpcex.llb`. It is running when it opens.



The topmost plot shows you the shapes (in the time domain) of six different types of windows in the Analysis library. They are all shown on the same plot for comparison purposes.

The bottom three plots show the effect of multiplying a time domain signal (a sine wave) by the window. The left plot shows the original time signal, the middle plot shows the shape of the window being applied, and the right plot shows the resulting signal.

In the **Window Selector** control, you can select one of the six different types of windows.

The **final value (exponential)** control specifies the value to which the exponential window should decay. This value is normally between 0 and 1.

The value in the **beta (Kaiser-Bessel)** control can be adjusted to change the shape of the Kaiser-Bessel window. The higher the value of *beta*, the lower the spectral leakage, and vice versa.

2. Select different windows in the **Window Selector** control and observe their shapes.

In particular, select the **Rectangular (None)** window. What difference do you see in the first and the third plots shown on the bottom? Explain.

Select the **Exponential** window. Observe the shape when you change the value in the **final value (exponential)** control. What happens as the final value increases? Decreases? Is equal to 1.0?

Select the **Kaiser-Bessel** window. Observe the shapes when you change the value of beta between -10 and +10. What happens when *beta* = 0.0?

3. When you finish, stop the VI by pressing the STOP button.
4. Close the VI. Do not save any changes.

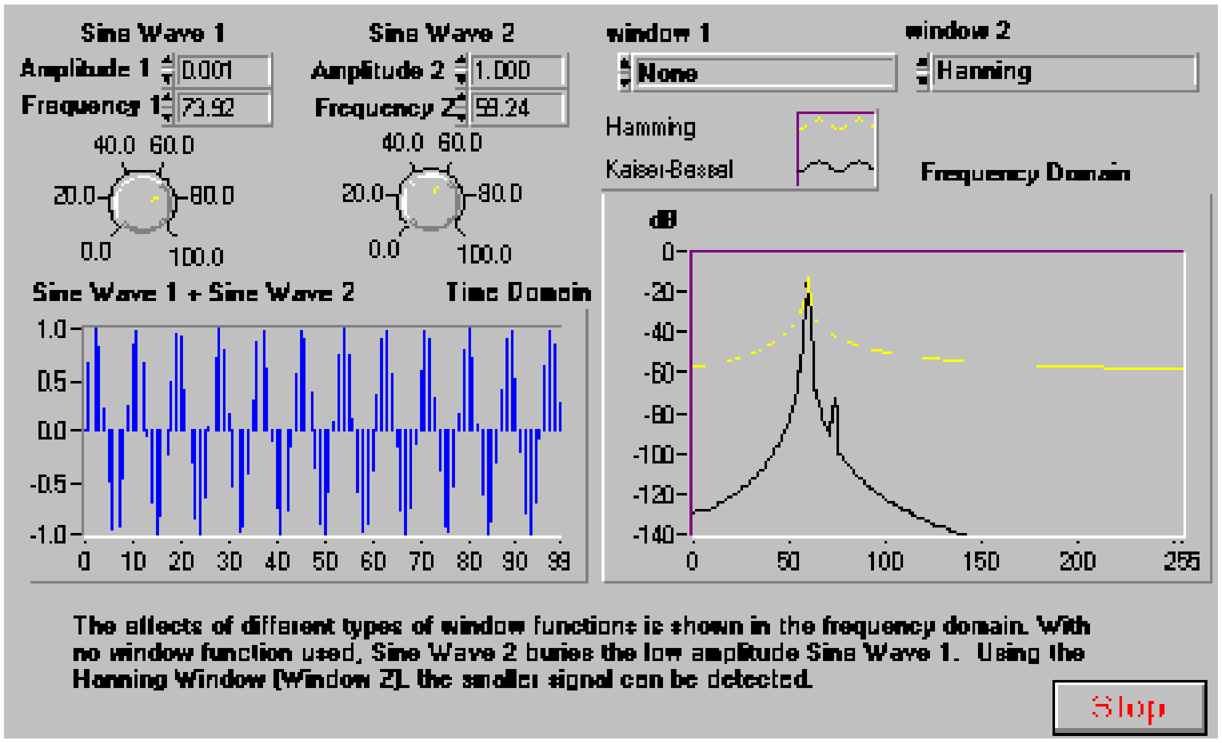
End of Exercise 4-3

Exercise 4-4

Objective: To use windows to separate two sine waves of almost the same frequency, but widely differing amplitudes.

In this exercise, two sine waves of different amplitudes are summed together and then transformed into the frequency domain. Sine wave 1 has a much smaller amplitude than sine wave 2. Without windowing, it is not possible to distinguish between the two sine waves in the frequency domain. With an appropriate choice of a window, you can clearly separate the peaks in the frequency domain corresponding to the two sine waves. The frequency domain plot shows the results so that you can compare the effect of different window functions.

1. Open and run the **Window Comparison** VI from the library `Lvspcex.llb`.


Sine Wave 1
Amplitude 1 0.001
Frequency 1 73.92

Sine Wave 2
Amplitude 2 1.000
Frequency 2 59.24

window 1 None

window 2 Hanning

The effects of different types of window functions is shown in the frequency domain. With no window function used, Sine Wave 2 buries the low amplitude Sine Wave 1. Using the Hanning Window (Window 2), the smaller signal can be detected.

- The frequency of each sine wave is adjustable with either the knob or digital controls.
- The amplitude of each sine wave is adjustable with the digital controls.
- You can select a different window function from the **window 1** and **window 2** controls.

2. Using the digital controls, set the amplitude of Sine Wave 1 as 0.001, and that of Sine Wave 2 as 1.000. With the knob controls, set the

frequency of Sine Wave 1 to near 70, and that of Sine Wave 2 to near 60. In effect, you are adjusting the frequency of Sine Wave 2 using the knob control, so that the smaller amplitude is nearer the larger amplitude in the frequency domain plot.

3. Notice in the graph that when the frequency of the smaller amplitude signal (Sine Wave 1) is closer to that of the larger amplitude signal (Sine Wave 2), the peak corresponding to the smaller signal is not detected. Applying a window function is the only way to detect the smaller signal. The discontinuity is what causes the spectrum to spread out. Signals at smaller amplitudes are lost in the sidelobes of the larger amplitude signal.
4. Compare different window functions by choosing another window from the **window 1** and **window 2** controls. Which one(s) can distinguish between the two frequency components?
5. When you are done, stop the VI by clicking on the STOP button. Close the VI. Do not save any changes.

End of Exercise 4-4

Summary

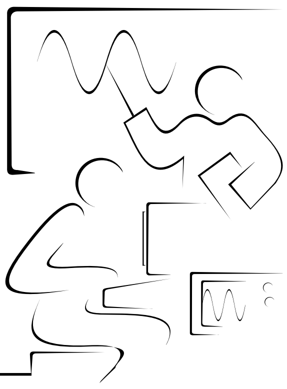
- The DFT/FFT assumes that the finite time waveform that you obtain is one period of a periodic signal that exists for all time. This assumption of periodicity could result in discontinuities in the periodic signal and gives rise to a phenomenon known as spectral leakage, whereby the energy at a particular frequency leaks throughout the spectrum.
- To reduce the spectral leakage, the finite time waveform is multiplied by a “window” function.
- Windows can be used to separate two sine waves that have widely different amplitudes, but are very close in frequency.

Review Questions

1. Why does spectral leakage occur?
2. Name four applications of using windows.
3. Which window(s) would you use for the following?
   a. Separation of two tones with frequencies very close to each other.
   b. System analysis.
   c. Detecting the modes of vibration of a machine.

Notes

Notes



Lesson 5 Measurement

Introduction

You will now learn about some of the VIs that are already available in the analysis library to perform various signal processing tasks. These VIs are collectively referred to as measurement VIs.

You Will Learn:

A. About the measurement VIs and how they can perform various signal processing operations.

B. How to calculate the frequency (amplitude and phase) spectrum of a time domain signal, with the appropriate units.

C. About the coherent gain (CG) and equivalent noise bandwidth (ENBW) window constants.

D. How to determine the total harmonic distortion present in a signal.

A. The Measurement VIs

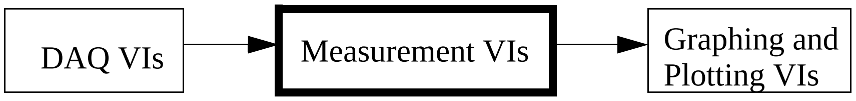
The measurement VIs perform specific measurement tasks such as:

- Calculating the total harmonic distortion present in a signal.
- Determining the impulse response, or transfer function, of a system.
- Estimating pulse parameters such as the rise time, overshoot, and so on.
- Computing the amplitude and phase spectrum of a signal.
- Calculating the AC and DC components of a signal.

In the past, these computations have traditionally been performed by benchtop instruments. The measurement VIs make these measurements possible in the G programming language on your desktop computer. They are built on top of the digital signal processing VIs and have the following characteristics:

- The input time-domain signal is assumed to be real valued.
- Outputs are in magnitude and phase, scaled, and in the appropriate units, ready for immediate graphing.
- The spectrums calculated are single-sided and range from DC to Nyquist (sampling frequency/2).
- Wherever appropriate, corrections are automatically applied for the windows being used. The windows are scaled so that each window gives the same peak spectrum amplitude result within its amplitude accuracy constraints.

In general, you can directly connect the inputs of the measurement VIs to the output of data acquisition VIs. The outputs of the measurement VIs can be connected to graphs for an appropriate visual display.



Several measurement VIs perform commonly used time domain-to-frequency domain transformations such as calculation of the amplitude and phase spectrum, the power spectrum, the network transfer function, and so on. Other measurement VIs interact with VIs that perform such functions as scaled time domain windowing and power and frequency estimation.

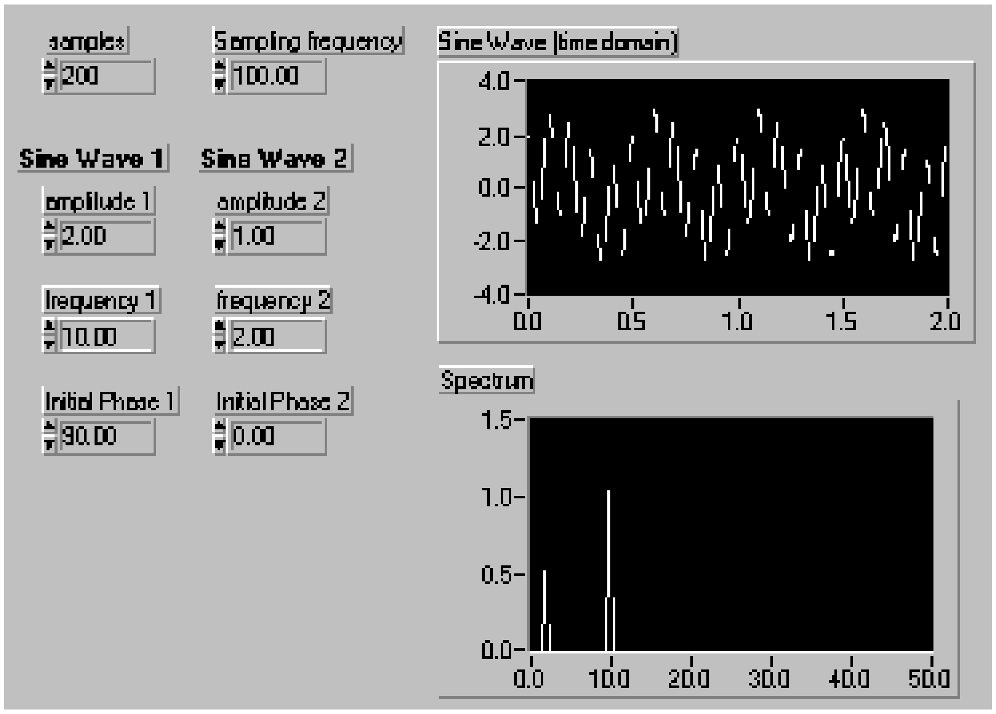
B. Calculating the Frequency Spectrum of a Signal

In many applications, knowing the frequency content of a signal provides insight into the system that generated the signal. The information thus obtained can be used in the design of bridges, for calibration purposes, for estimating the amount of noise and vibration generated by parts of machines, and so on. The next exercise demonstrates how to use the **Amplitude and Phase Spectrum** VI to identify two frequency components.

Exercise 5-1

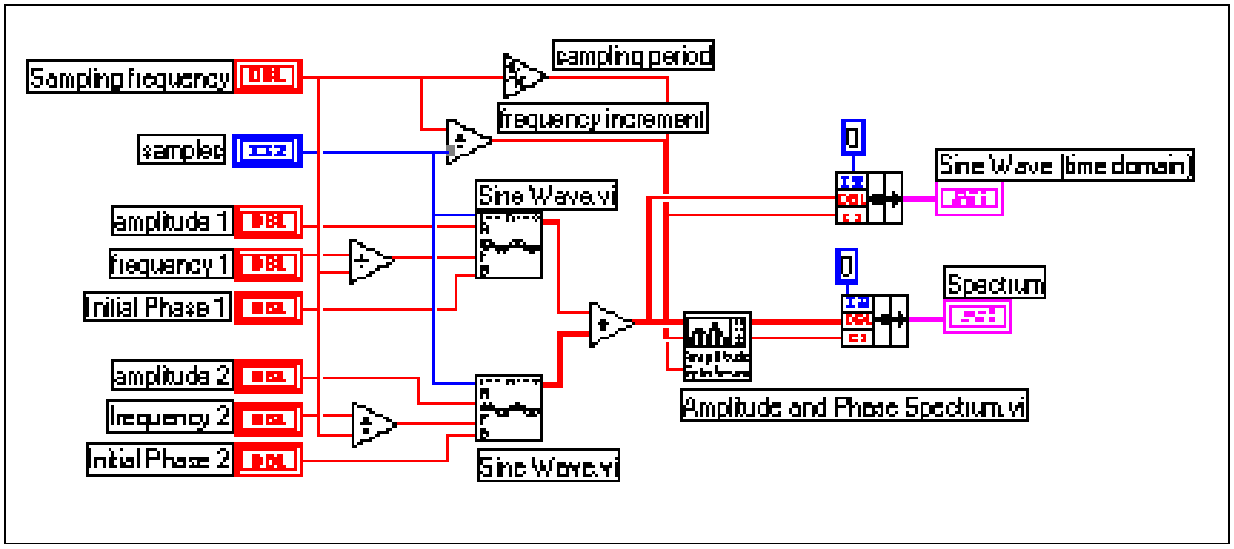
OBJECTIVE: To compute the frequency spectrum of a signal.

Front Panel



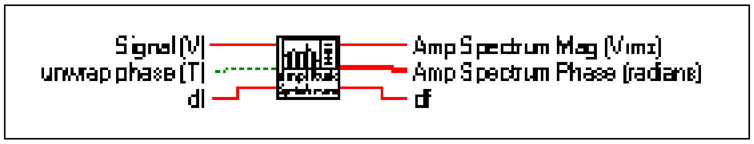
1. Open the **Compute Frequency Spectrum** VI found in the `Lvspcex.llb` library. Two sine waves of frequencies 2 Hz and 10 Hz are superimposed. The 10 Hz sine wave has an amplitude of 2 V, and the 2 Hz sine wave has an amplitude of 1 V. The sampling frequency is 100 Hz and 200 points of data are generated.
2. Switch to the block diagram.

Block Diagram



3. Examine the block diagram.

The **Amplitude and Phase Spectrum** VI (**Analysis » Measurement** subpalette) calculates the amplitude spectrum and the phase spectrum of a time domain signal. The connections to this VI are shown below.



The input time domain signal is applied at the **Signal (V)** control. The magnitude and phase of the input signal spectrum are available at the **Amp Spectrum Mag (Vrms)** and **Amp Spectrum Phase (radians)** outputs, respectively.

Note: ***The initial phase input to the Sine Wave VI is specified in degrees.***

Note: ***If the units of the input time domain signal are in volts peak (V_p), the units of the magnitude of the amplitude spectrum is in volts rms (V_{rms}). The relationship between the units is $V_{rms} = V_p/\sqrt{2} = 0.707*V_p$.***

4. Run the VI.

The graph should display two peaks, one at 2 Hz and the other at 10 Hz. The amplitude of the 2 Hz sine wave is 0.717 V, and that of the 10 Hz waveform is 1.414 V, which are the rms values for sine waveforms of amplitudes 1 and 2 V, respectively. (The RMS of a sine waveform = 0.707*peak amplitude.)

5. Change the phase of the sine waves by adjusting the **Initial Phase 1** and **Initial Phase 2** controls, and run the VI.

   Do you notice any change in the time waveform? The spectrum?

6. Make the parameters of both the sine waves equal. That is, set **amplitude 1** = **amplitude 2** = 2, **frequency 1** = **frequency 2** = 10, **Initial Phase 1** = **Initial Phase 2** = 0, and **sampling frequency** = 200, and run the VI.

   Is the amplitude of the peak in the power spectrum the value that you would expect?

7. When you are done, close the VI. Do not save any changes.

End of Exercise 5-1

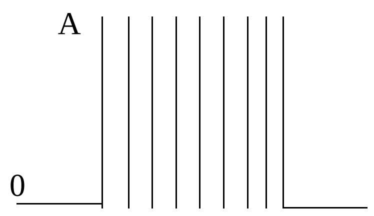
C. Coherent Gain (CG) and Equivalent Noise Bandwidth (ENBW)

You saw in Lesson 4 that many different types of windows can be applied to a signal. These windows have different shapes, and they affect the signal in different ways. It is important to have some method to compare the effect that different windows have on the signal. Two parameters that are useful in comparing various types of windows are the *coherent gain* (CG) and the *equivalent noise bandwidth* (ENBW).

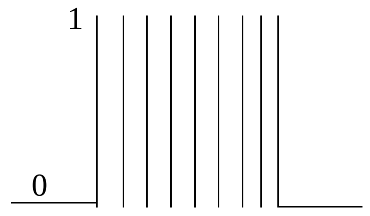
Coherent Gain

The coherent gain of a window is the *zero frequency* gain (or the *dc gain*) of the window. It is calculated by normalizing the maximum amplitude of the window to one, and then summing the values of the window amplitudes over the duration of the window. The result is then divided by the length of the window (that is, the number of samples).

For example, consider the rectangular window shown below with amplitude equal to A and nine samples:



You first normalize (divide) all the heights by A to get the maximum height equal to one:



Then you add all the heights to get nine (nine lines each with a height equal to one). This sum is then divided by the number of samples (nine) to get a value of one. Thus, the CG of the rectangular window is equal to one. Mathematically, the CG is given by

$$CG = \frac{1}{N}\sum_{n=0}^{N-1} w[n]$$

where N is the total number of samples over the duration of the window and $w[n]$ are the normalized amplitudes of the samples.

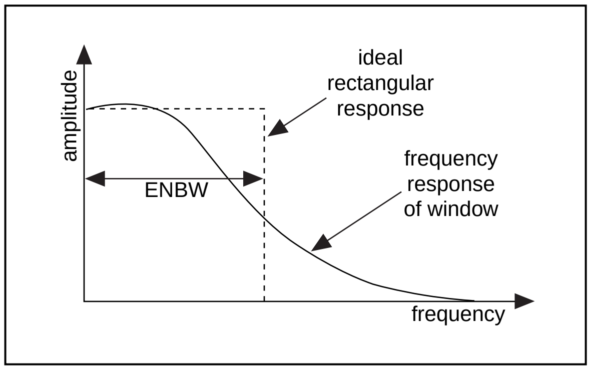
For comparison purposes, the table below shows the CG of several commonly used windows. Note that the rectangular (uniform) window has the highest CG, whereas the CG of other windows is lower than that of the rectangular window. Can you explain why this is so?

Window	CG
Uniform (None)	1.00
Hamming	0.54
Hanning	0.5
Triangle	0.5
Exact Blackman	0.46
Blackman	0.42
Blackman-Harris	0.42
4 Term B-Harris	0.36
Flat Top	0.28
7 Term B-Harris	0.27

Equivalent Noise Bandwidth (ENBW)

You can use the ENBW to compare frequency responses of different shapes. An ideal frequency response is supposed to be rectangular in shape (see the lesson on digital filters). However, in practice, the frequency response differs from the ideal. Because different windows have different shapes of their frequency response, they will pass different amounts of noise power. The ENBW for a particular window is equal to the width of a frequency response having an ideal rectangular shape that will pass the same amount of noise power as the frequency response of that window.

Suppose that the solid line in the figure below shows the frequency response of a window, and the dashed line shows the ideal rectangular response. The responses are first adjusted to have a gain of unity at zero frequency (DC).



To calculate the ENBW, the width of the rectangular response is adjusted so that it has the same area as that of the nonideal response of the window. This width is then equal to the ENBW for that window. It is found that the ENBW is the smallest for the uniform window, and larger for the other windows. The table below shows the ENBW for different windows, relative to the ENBW for the uniform window.

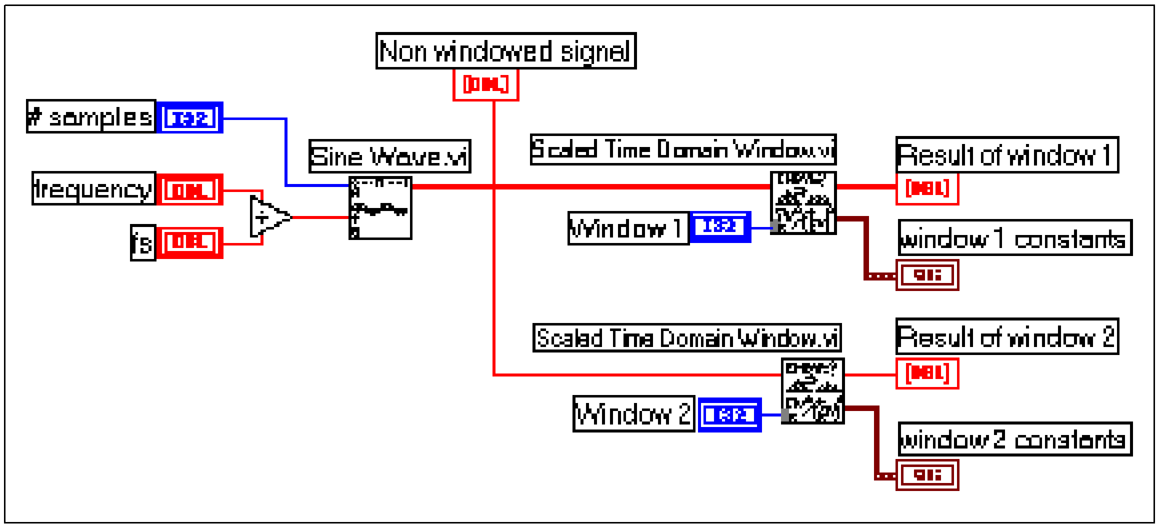
Window	**ENBW**
Uniform (None)	1.00
Hamming	1.36
Hanning	1.5
Triangle	1.33
Exact Blackman	1.57
Blackman	1.73
Blackman-Harris	1.71
4 Term B-Harris	2.00
Flat Top	2.97
7 Term B-Harris	2.63

Exercise 5-2

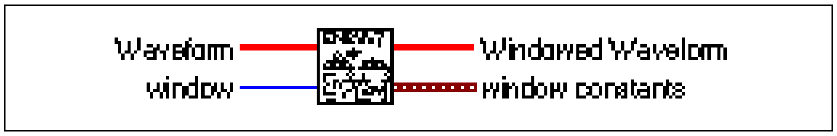
Objective: To learn about the Coherent Gain and Equivalent Noise Bandwidth properties of windows.

This exercise will familiarize you with different window shapes and their CG and ENBW. The values of the CG and ENBW are used in other measurement VIs such as the **Power and Frequency Estimate** VI and **Spectrum Unit Conversion** VI.

1. Open the **Time Domain Windows** VI found in the library `Lvspcex.llb`.
2. Open and examine the block diagram.



The VI that windows the input signal (a sine wave generated by the **Sine Wave** VI) and gives as the output the resulting windowed waveform is the **Scaled Time Domain Window** VI. The connections to this VI are shown below.

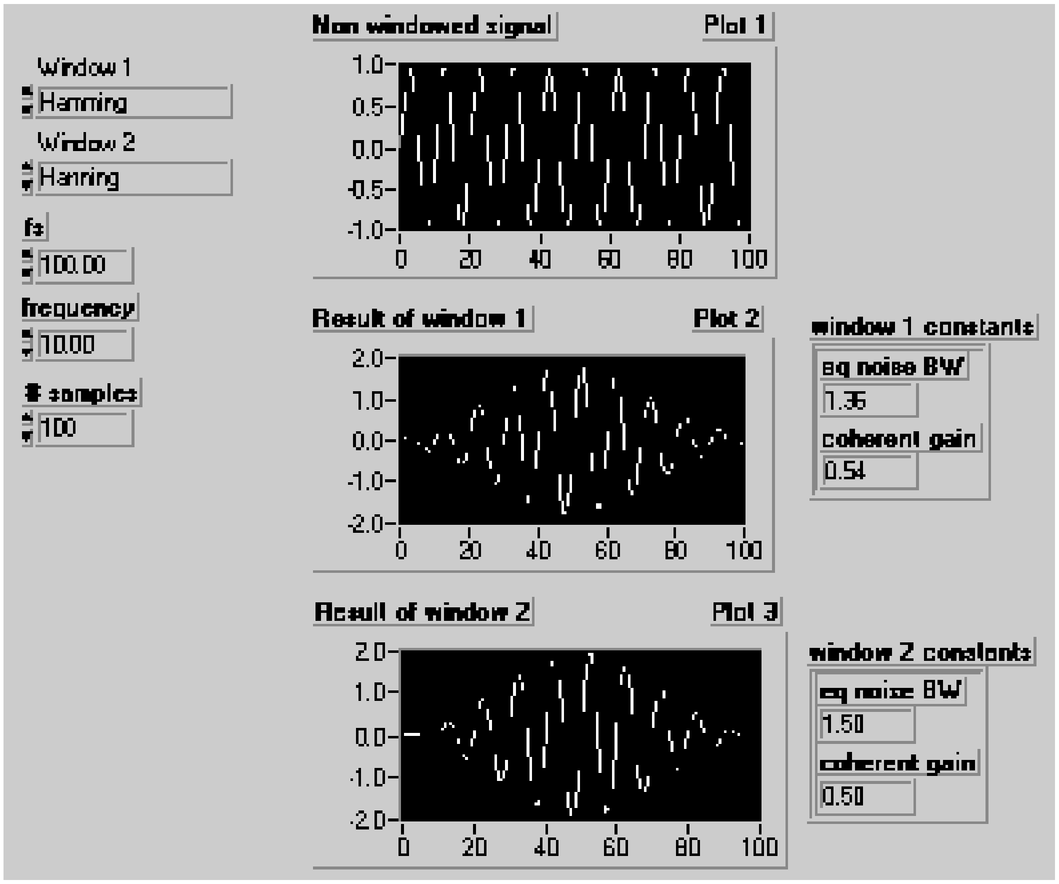


The input time waveform is applied at the **Waveform** control, and the window selection is done by the **window** control. The VI also outputs the CG and the ENBW of the selected window at the **window constants** terminal.

The resulting output waveform is automatically scaled so that when you compute the amplitude or power spectrum of the windowed waveform, all the windows will give the same value.

You will pass the sine wave through two of these VIs and compare the resulting windowed waveforms.

3. Switch to the front panel.



Window 1: Hamming
Window 2: Hanning
fs: 100.00
frequency: 10.00
# samples: 100

window 1 constants
eq noise BW: 1.36
coherent gain: 0.54

window 2 constants
eq noise BW: 1.50
coherent gain: 0.50

The **Window 1** and **Window 2** controls select the two types of windows that you want to apply to your signal (the sine wave).

The number of samples and the frequency of the sine wave are controlled by the **# samples** and **frequency** controls, respectively. The sampling frequency is adjusted by the **fs** control.

The topmost plot shows you the original time domain waveform (without windowing). The lower two plots show you the signal after application of the two windows specified in the **Window 1** and **Window 2** controls.

4. Select **Window 1** as *None (Uniform)* and **Window 2** as *Hanning* and run the VI. Leave all the controls at their default values.

   Compare the waveforms in **Plot 1** and **Plot 2** and observe that using the Uniform window is equivalent to not using any window.

Observe from **Plot 2** and **Plot 3** the difference in the windowed time domain waveform due to the application of the *uniform* and *Hanning* windows, respectively.

Note the differences in the CG and ENBW of the two windows.

5. The shapes of the *Hamming* and *Hanning* windows are very close to each other. Choose these windows in the **Window 1** and **Window 2** controls. Run the VI and compare the waveforms in **Plot 2** and **Plot 3**.

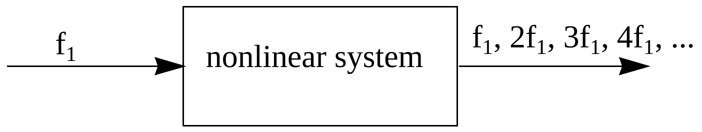
   Can you notice any difference? Which one is wider? In particular, compare the values of the ENBW and the CG.

6. As mentioned before, the CG was the same as the DC gain. Choose different windows and run the VI. Observe that multiplying the maximum amplitude of the windowed signal by the CG of the window gives unity.
7. When you finish, close the VI. Do not save any changes.

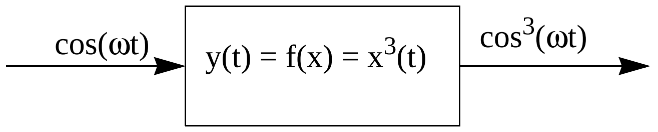
End of Exercise 5-2

D. Harmonic Distortion

When a signal, *x*(*t*), of a particular frequency (for example, f_1) is passed through a nonlinear system, the output of the system consists of not only the input frequency (f_1), but also its harmonics ($f_2 = 2*f_1$, $f_3 = 3*f_1$, $f_4 = 4*f_1$, and so on). The number of harmonics, and their corresponding amplitudes, that are generated depends on the degree of nonlinearity of the system. In general, the more the nonlinearity, the higher the harmonics, and vice versa.



An example of a nonlinear system is a system where the output *y(t)* is the cube of the input signal *x(t)*.



So, if the input is

$$x(t) = \cos(\omega t),$$

the output is

$$x^3(t) = 0.5*\cos(\omega t) + 0.25*[\cos(\omega t) + \cos(3\omega t)]$$

Therefore, the output contains not only the input fundamental frequency of ω, but also the third harmonic of 3ω.

Total Harmonic Distortion

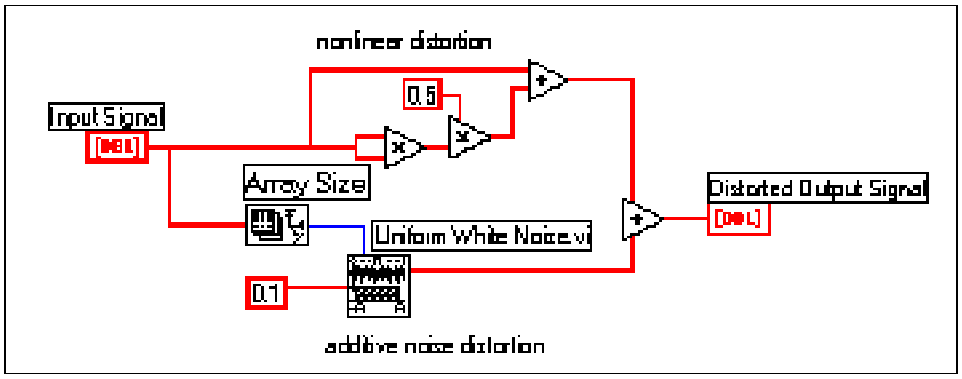
To determine the amount of nonlinear distortion that a system introduces, you need to measure the amplitudes of the harmonics that were introduced by the system relative to the amplitude of the fundamental. Harmonic distortion is a relative measure of the amplitudes of the harmonics as compared to the amplitude of the fundamental. If the amplitude of the fundamental is A_1, and the amplitudes of the harmonics are A_2 (second harmonic), A_3 (third harmonic), A_4 (fourth harmonic), ...A_N (Nth harmonic), the total harmonic distortion (THD) is given by

$$THD = \frac{\sqrt{A_1^2 + A_2^2 + A_3^2 + ...A_N^2}}{A_1}$$

and the percentage total harmonic distortion (% THD) is

$$\%THD = \frac{100 \times \sqrt{A_1^2 + A_2^2 + A_3^2 + ...A_N^2}}{A_1}$$

In the next exercise, you will generate a sine wave and pass it through a nonlinear system. The block diagram of the nonlinear system is shown below:



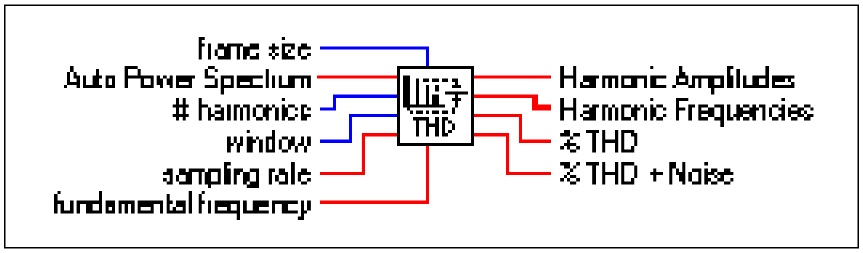
Verify from the block diagram that if the input is x(t) = cos(ωt), the output is

$$
\begin{aligned}
y(t) &= \cos(\omega t) + 0.5\cos^2(\omega t) + 0.1n(t) \\
&= \cos(\omega t) + [1 + \cos(2\omega t)]/4 + 0.1n(t) \\
&= 0.25 + \cos(\omega t) + 0.25\cos(2\omega t) + 0.1n(t)
\end{aligned}
$$

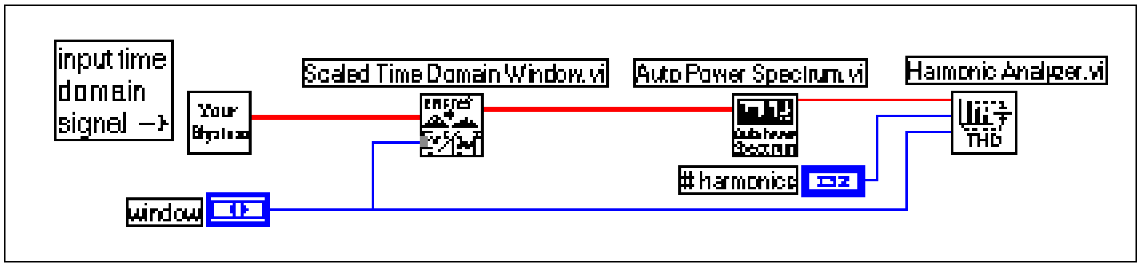
Therefore, this nonlinear system generates an additional DC component as well as the second harmonic of the fundamental.

Using the Harmonic Analyzer VI

You can use the **Harmonic Analyzer** VI to calculate the %THD present in the signal at the output of the nonlinear system. It finds the fundamental and harmonic components (their amplitudes and corresponding frequencies) present in the power spectrum applied at its input, and calculates the percentage of total harmonic distortion (%THD) and the percentage of total harmonic distortion plus noise (%THD + Noise). The connections to the **Harmonic Analyzer** VI are shown below:



To use this VI, you need to give it the power spectrum of the signal whose THD you want it to calculate. Thus, in this example, you need to make the following connections:



The **Scaled Time Domain Window** VI applies a window to the output y(t) of the nonlinear system (**Your System**). This is then passed on to the **Auto Power Spectrum** VI, which sends the power spectrum of y(t) to the **Harmonic Analyzer** VI, which then calculates the amplitudes and frequencies of the harmonics, the THD, and the %THD.

You can specify the number of harmonics you want the VI to find in the # harmonics control. Their amplitudes and corresponding frequencies are returned in the **Harmonic Amplitudes** and **Harmonic Frequencies** array indicators.

Note: ***The number specified in the # harmonics control includes the fundamental. So, if you enter a value of 2 in the # harmonics control, it means to find the fundamental (say, of freq f_1) and the second harmonic (of frequency $f_2 = 2*f_1$). If you enter a value of N, the VI will find the fundamental and the corresponding (N-1) harmonics.***

The following are explanations of some of the other controls:

fundamental frequency is an estimate of the frequency of the fundamental component. If left as zero (the default), the VI uses the frequency of the non-DC component with the highest amplitude as the fundamental frequency.

window is the type of window you applied to your original time signal. It is the window that you select in the **Scaled Time Domain Window** VI. For an accurate estimation of the THD, it is recommended that you select a window function. The default is the uniform window.

sampling rate is the input sampling frequency in Hz.

The **% THD + Noise** output requires some further explanation. The calculations for **% THD + Noise** are almost similar to that for **% THD**, except that the noise power is also added to that of the harmonics. It is given by

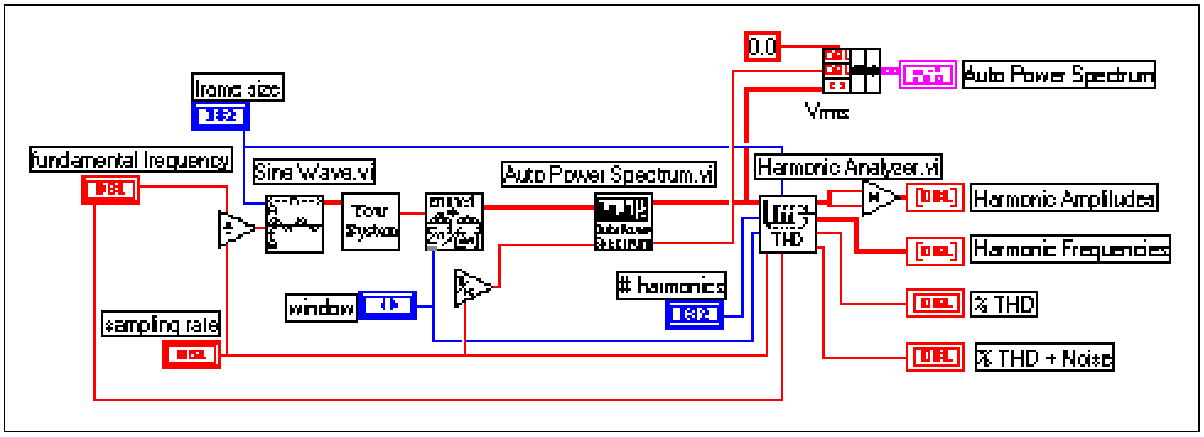
$$\%THD + Noise = 100 \times \frac{\sqrt{sum(APS)}}{A_1}$$

where *sum(APS)* is the sum of the Auto Power Spectrum elements minus the elements near DC and near the index of the fundamental frequency.

Exercise 5-3

Objective: To use the Harmonic Analyzer VI for harmonic distortion calculations.

1. Open the **THD Example** VI from the `Lvspcex.llb` library.
2. Switch to the block diagram.



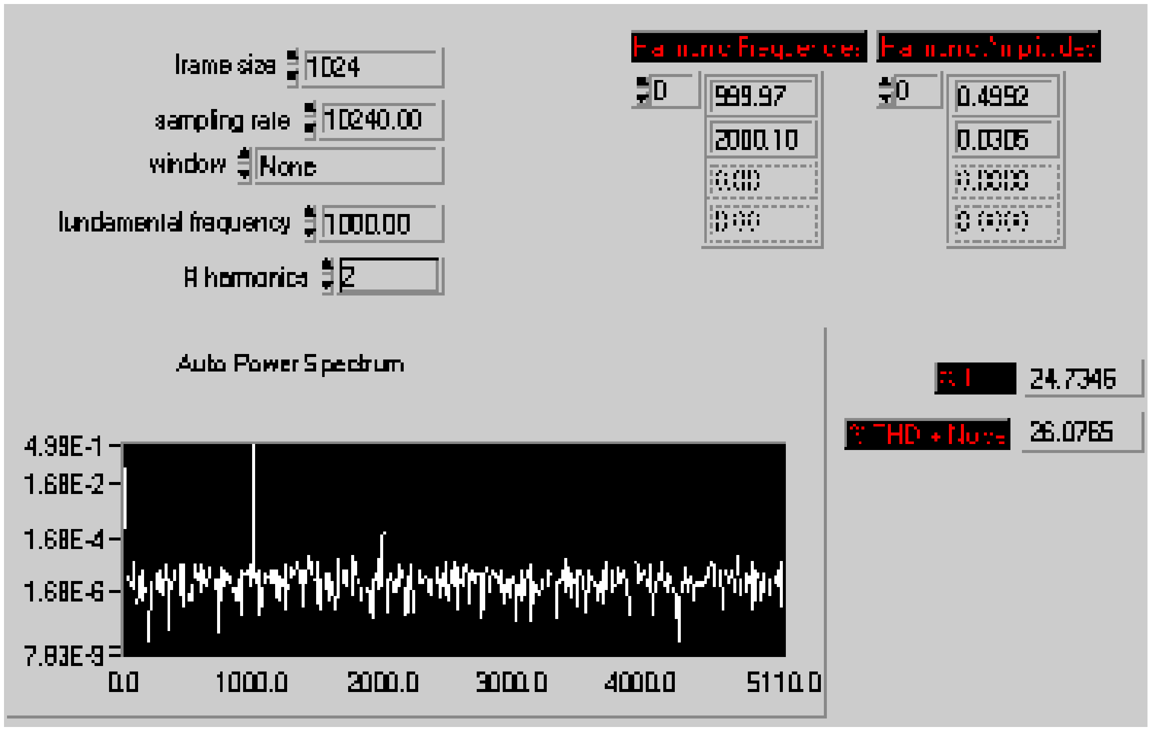
Some of this will already be familiar to you. **Your System** is the nonlinear system that you saw previously. Its output is windowed, and the power spectrum calculated and given to the **Harmonic Analyzer** VI.

The **Sine Wave** VI generates a fundamental of frequency specified in the **fundamental frequency** control.

The output of the **Harmonic Analyzer** VI is in V_{rms} (if the input from the Auto Power Spectrum is in V^2_{rms}). This output is then squared to convert it to V^2_{rms}.

3. Switch to the front panel.

   At the bottom, you see a plot of the power spectrum of the output of the nonlinear system. On the top right side are the array indicators for the frequencies and amplitudes of the fundamental and its harmonics. The size of the array depends on the value entered in the **# harmonics** control.



frame size 1024

sampling rate 10240.00

window None

fundamental frequency 1000.00

harmonics 2

Harmonic Frequencies 0: 999.97, 2000.10, 0.000, 0.000

Harmonic Amplitudes 0: 0.4992, 0.0305, 0.0000, 0.0000

% THD 24.7346

% THD + Noise 26.0765

4. Change the **fundamental frequency** to 1000, **# harmonics** to 2, and run the VI several times. Each time, note the values in the output indicators (**Harmonic Frequencies**, **Harmonic Amplitudes**, **% THD**, and **% THD + Noise**).

   Why do you get different values each time you run the VI?

   Which of the values, % THD or % THD + Noise, is larger? Can you explain why?

5. Run the VI with different selections of the window control and observe the peaks in the power spectrum.

   Which window gives the narrowest peaks? The widest? Can you explain why?

   Hint: See the values of the ENBW for each window in the table on page 9.

6. Change the fundamental frequency to 3000 and run the VI.

   Why do you get an error?

   Hint: Consider the relationship between the Nyquist frequency and the frequency of the harmonic(s).

7. When you finish, close the VI.

End of Exercise 5-3

Summary

- The ready-made VIs to perform common measurements are available in the **Analysis » Measurements** subpalette.
- Some of these measurement tasks include calculating the amplitude and phase spectrum of a signal and the amount of harmonic distortion. Other VIs calculate properties of a system such as its transfer function, its impulse response, the cross power spectrum between the input and output signals, and so on.
- Because a real-world signal is time-limited by a window function, the study of certain properties of these window functions is an important consideration in interpreting the results of your measurements.
- The CG is a measure of the DC gain of the window, whereas the ENBW is a measure of the amount of noise power that a window introduces into a measurement.

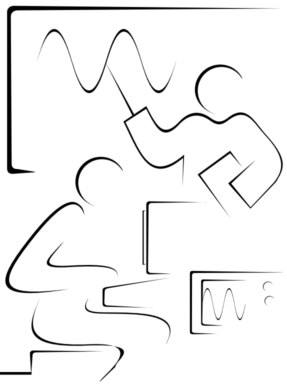
Review Questions

1. Why is it important to know the coherent gain of a window?
2. Which Measurement VIs calculate both the amplitude and phase spectrum of the input waveform?
3. Name some applications where you would use the Measurement VIs.
4. When a Measurement VI calculates the spectrum of a signal, is it a one-sided or a two-sided spectrum?
5. What is the peak value corresponding to $2V_{rms}$?
6. Which window has the highest coherent gain? Why?
7. What is the difference between %THD and %THD + Noise? Which is larger?

Notes

Notes

Lesson 6
Digital Filtering



Introduction

In this lesson, you will learn about the characteristics of different types of digital filters and how to use them in practical filtering applications.

You Will Learn:

A. What is filtering and why it is needed (applications).

B. About the frequency response characteristics of different types of ideal filters—lowpass, highpass, bandpass, bandstop.

C. About the differences between practical (nonideal) filters and ideal filters.

D. About the advantages of digital filters over analog filters.

E. About the differences between IIR and FIR filters.

F. About the characteristics of different types of IIR filters.

G. About the transient response of IIR filters.

H. About the characteristics of FIR filters.

A. What Is Filtering?

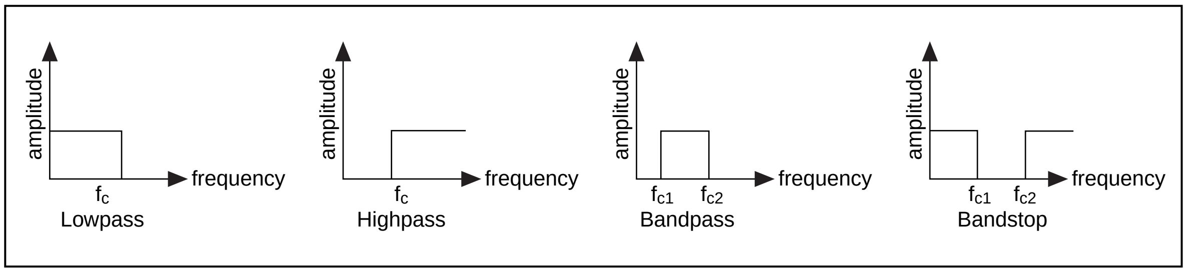
Filtering is the process by which the frequency content of a signal is altered. It is one of the most commonly used signal processing techniques. Common everyday examples of filtering are the bass and treble controls on your stereo system. The bass control alters the low-frequency content of a signal, and the treble control alters the high-frequency content. By varying these controls, you are actually filtering the audio signal. Some other applications where filtering is useful are removing noise and performing decimation (lowpass filtering the signal and reducing the sample rate).

B. Ideal Filters

Filters alter or remove unwanted frequencies. Depending on the frequency range that they either pass or attenuate, they can be classified into the following types:

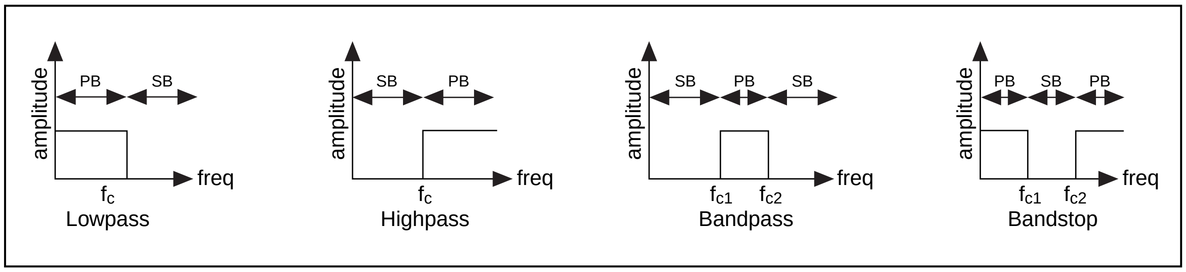
- *A lowpass filter* passes low frequencies, but attenuates high frequencies.
- A *highpass filter* passes high frequencies, but attenuates low frequencies.
- A *bandpass filter* passes a certain band of frequencies.
- A *bandstop filter* attenuates a certain band of frequencies.

The ideal frequency response of these filters is shown below:



You see that the lowpass filter passes all frequencies below f_c, whereas the highpass filter passes all frequencies above f_c. The bandpass filter passes all frequencies between f_{c1} and f_{c2}, whereas the bandstop filter attenuates all frequencies between f_{c1} and f_{c2}. The frequency points f_c, f_{c1} and f_{c2} are known as the cutoff frequencies of the filter. When designing filters, you need to specify these cut-off frequencies.

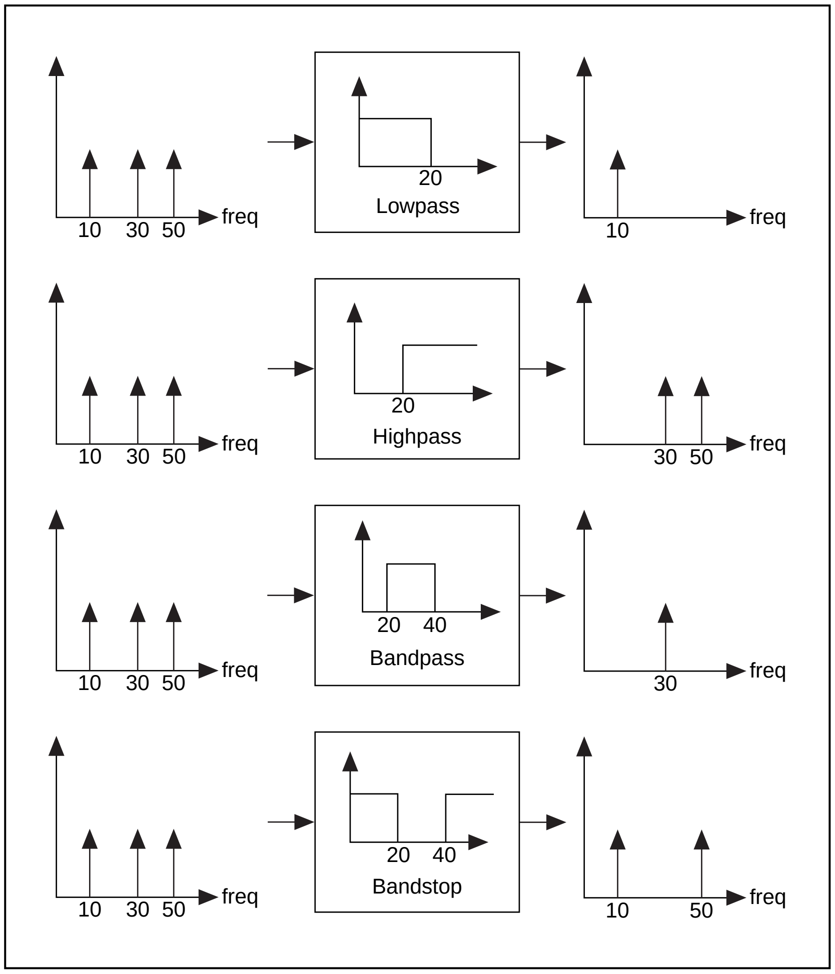
The frequency range that is passed through the filter is known as the *passband* (PB) of the filter. An ideal filter has a gain of one (0 dB) in the passband so that the amplitude of the signal neither increases nor decreases. The *stopband* (SB) corresponds to that range of frequencies that do not pass through the filter at all and are rejected (attenuated). The passband and the stopband for the different types of filters are shown below:



Note that whereas the lowpass and highpass filters have one passband and one stopband, the bandpass filter has one passband, but two stopbands, and the bandstop filter has two passbands, but one stopband.

How Filters Affect Signal Frequency Content

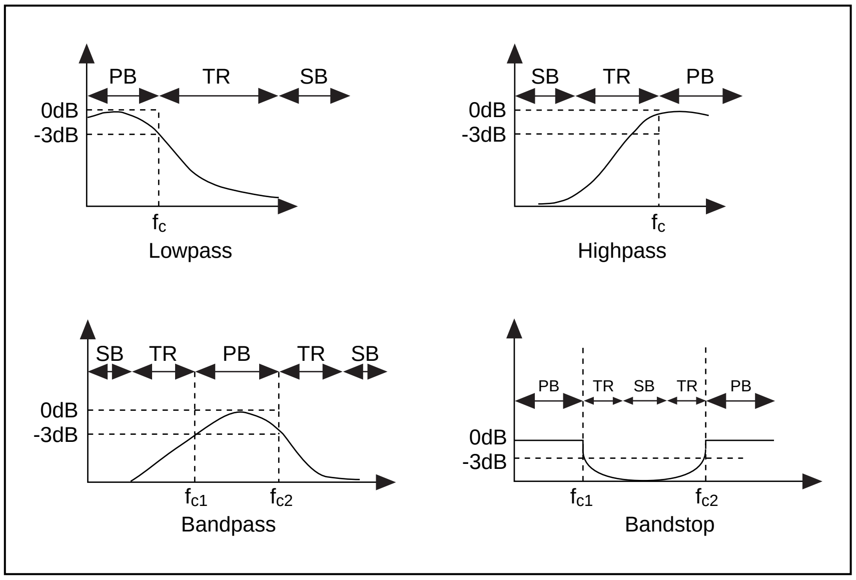
Suppose you have a signal containing frequencies of 10 Hz, 30 Hz, and 50 Hz. This signal is passed through a lowpass, highpass, bandpass, and bandstop filter. The lowpass and highpass filters have a cutoff frequency of 20 Hz, and the bandpass and bandstop filters have cutoff frequencies of 20 Hz and 40 Hz. The output of the filter in each case is shown below:



C. Practical (Nonideal) Filters

The Transition Band

Ideally, a filter should have a unit gain (0 dB) in the passband, and a gain of zero (-× dB) in the stopband. However, in a real implementation, not all of these criteria can be fulfilled. In practice, there is always a finite transition region between the passband and the stopband. In this region, the gain of the filter changes gradually from 1 (0 dB) in the passband to 0 (-×) in the stopband. The following diagrams show the passband, the stopband, and the transition region (TR) for the different types of nonideal filters. Note that the passband is now the frequency range within which the gain of the filter varies from 0 dB to -3 dB. Although the -3 dB range is most commonly used, depending on the application, other values (-0.5 dB, -1 dB, etc.) may also be considered.



Passband Ripple and Stopband Attenuation

In many applications, it is okay to allow the gain in the passband to vary slightly from unity. This variation in the passband is called the *passband ripple* and is the difference between the actual gain and the desired gain of unity. The *stopband attenuation*, in practice, cannot be infinite, and you must specify a value with which you are satisfied. Both the passband ripple and the stopband attenuation are measured in decibels or dB, defined by:

$$dB = 20*\log_{10}(A_o(f)/A_i(f))$$

where $\log_{10}$ denotes the logarithm to the base 10, and $A_i(f)$ and $A_o(f)$ are the amplitudes of a particular frequency *f* before and after the filtering, respectively.

For example, for -0.02 dB passband ripple, the formula gives:

$-0.02 = 20*\log_{10}(A_o(f)/A_i(f))$

$A_o(f)/A_i(f) = 10^{-0.001} = 0.9977$

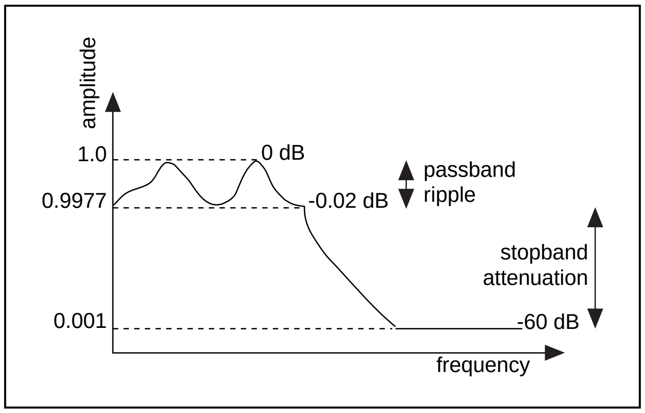
which shows that the ratio of input and output amplitudes is close to unity.

If you have -60 dB attenuation in the stopband, you have

$-60 = 20*\log_{10}(A_o(f)/A_i(f))$

$A_o(f)/A_i(f) = 10^{-3} = 0.001$

which means the output amplitude is 1/1000 of the input amplitude. The following figure, though not drawn to scale, illustrates this concept.



Note: ***Attenuation is usually expressed in decibels without the word "minus," but a negative dB value is normally assumed.***

D. Advantages of Digital Filters over Analog Filters

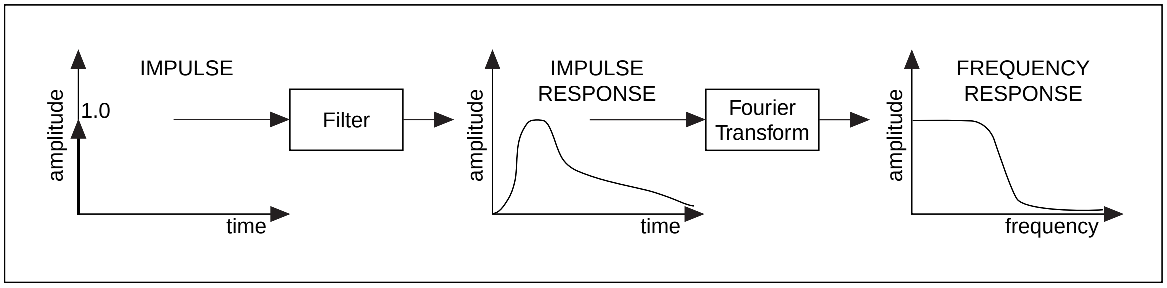
An analog filter has an analog signal at both its input and its output. Both the input, *x(t)*, and output, *y(t)*, are functions of a continuous variable *t* and can take on an infinite number of values. Analog filter design is about 50 years older than digital filter design. Thus, analog filter design books featuring simple, well- tested filter designs exist and can be found extensively in the literature. However, this type of filter design is often reserved for specialists because it requires advanced mathematical knowledge and understanding of the processes involved in the system affecting the filter.

Modern sampling and digital signal processing tools have made it possible to replace analog filters with digital filters in applications that require flexibility and programmability. These applications include audio, telecommunications, geophysics, and medical monitoring. The advantages of digital filters over analog filters are:

- They are software programmable, and so are easy to “build” and test.
- They require only the arithmetic operations of multiplication and addition/subtraction and so are easier to implement.
- They are stable (do not change with time nor temperature) and predictable.
- They do not drift with temperature or humidity or require precision components.
- They have a superior performance-to-cost ratio.
- They do not suffer from manufacturing variations or aging.

E. IIR and FIR Filters

Another method of classification of filters is based on their impulse response. But what is an impulse response? The response of a filter to an input that is an impulse (*x*[0] = 1 and *x*[i] = 0 for all *i* ¦ 0) is called the *impulse response* of the filter (see figure below). The Fourier transform of the impulse response is known as the *frequency response* of the filter. The frequency response of a filter tells you what the output of the filter is going to be at different frequencies. In other words, it tells you the gain of the filter at different frequencies. For an ideal filter, the gain should be 1 in the passband and 0 in the stopband. So, all frequencies in the passband are passed "as is" to the output, but there is no output for frequencies in the stopband.



If the impulse response of the filter falls to zero after a finite amount of time, it is known as a *finite impulse response (FIR)* filter. However, if the impulse response exists indefinitely, it is known as an *infinite impulse response (IIR)* filter. Whether the impulse response is finite or not (that is, whether the filter is FIR or IIR) depends on how the output is calculated.

The basic difference between FIR and IIR filters is that for FIR filters, the output depends only on the current and past input values, whereas for IIR filters, the output depends not only on the current and past input values, but also on the past output values.

As an example, consider a cash register at a supermarket. Let *x*[*k*] be the cost of the k^{th} item that a customer buys, where 1 ð *k* ð *N*, and *N* is the total number of items. The cash register adds the cost of each item to produce a "running" total. This "running" total *y*[k] , up to the k^{th} item, is given by

$$y[k] = x[k] + x[k-1] + x[k-2] + x[k-3] + ...+ x[1] \qquad (1a)$$

Thus, the total for *N* items is *y[N]*. Because *y[k]* is the total up to the k^{th} item, and *y[k-1]* is the total up to the $(k-1)^{st}$ item, you can rewrite equation (1a) as

$$y[k] = y[k-1] + x[k] \qquad (1b)$$

If you add a sales tax of 8.25%, equations (1a) and (1b) can be rewritten as

$$y[k] = 1.0825x[k] + 1.0825x[k-1] + 1.0825\,x[k-2] + 1.0825x[k-3] + \ldots + 1.0825x[1] \qquad (2a)$$

$$y[k] = y[k-1] + 1.0825x[k] \qquad (2b)$$

Note that both equations (2a) and (2b) are identical in describing the behavior of the cash register. The difference is that whereas (2a) is implemented only in terms of the inputs, (2b) is implemented in terms of both the input and the output. Equation (2a) is known as the *nonrecursive*, or FIR, implementation. Equation (2b) is known as the *recursive*, or IIR, implementation.

Filter Coefficients

In equation (2a), the multiplying constant for each term is 1.0825. In equation (2b), the multiplying constants are 1 (for $y[k-1]$) and 1.0825 (for $x[k]$). These multiplying constants are known as the *coefficients* of the filter. For an IIR filter, the coefficients multiplying the inputs are known as the *forward coefficients*, and those multiplying the outputs are known as the *reverse coefficients*.

Equations of the form 1a, 1b, 2a, or 2b that describe the operation of the filter are known as *difference* equations.

Advantages and Disadvantages of FIR and IIR Filters

Comparing IIR and FIR filters, 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. Thus, 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 may be appropriate. You should use FIR filters for those applications requiring linear phase responses. The recursive nature of IIR filters makes them more difficult to design and implement.

F. Infinite Impulse Response Filters

You have seen that infinite impulse response filters (IIR) are digital filters whose output is calculated by adding a weighted sum of past output values with a weighted sum of past and current input values. Denoting the input values by $x[.]$ and the output values by $y[.]$, the general difference equation characterizing IIR filters is

$$a_0 y[i] + a_1 y[i-1] + a_2 y[i-2] + \ldots + a_{N_y - 1} y[i - (N_y - 1)] =$$
$$b_0 x[i] + b_1 x[i-1] + b_2 x[i-2] + \ldots + b_{N_x - 1} x[i - (N_x - 1)]$$

$$a_0 y[i] = -a_1 y[i-1] - a_2 y[i-2] + -\ldots - a_{N_y - 1} y[i - (N_y - 1)] +$$
$$b_0 x[i] + b_1 x[i-1] + b_2 x[i-2] + \ldots + b_{N_x - 1} x[i - (N_x - 1)]$$

$$y[i] = \frac{1}{a_0}\left(-\sum_{j=1}^{N_y - 1} a[j] y[i-j] + \sum_{k=0}^{N_x - 1} b[k] x[i-k] \right) \qquad (3)$$

where N_x is the number of *forward* coefficients ($b[k]$) and N_y is the number of *reverse* coefficients ($a[j]$). The output sample at the present sample index i is the sum of scaled present and past inputs (x[i] and x[i-k] when j ¦ 0) and scaled past outputs ($y[i\text{-}j]$). Usually, N_x is equal to N_y and this value is known as the *order* of the filter.

Note: ***In all of the IIR filters implemented in LabVIEW/BridgeVIEW, the coefficient a_0 is 1.***

Practical IIR Filters

A lower order reduces arithmetic operations and therefore reduces computation error. A problem with higher order filtering is that you quickly run into precision errors with orders much greater than 20-30. This is the main reason for the “cascade” implementations over the “direct” form. Refer to the *Analysis VI Reference Manual* for more details on cascade form implementations. It is recommended that the orders of 1-20 are reasonable, with 30 being an upper limit. A higher order also means more filter coefficients and hence longer processing time.

The impulse response of the filter described by equation (3) is of 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.

In practice, the frequency response of filters differs from that of ideal filters. Depending on the shape of the frequency response, the IIR filters can be further classified into

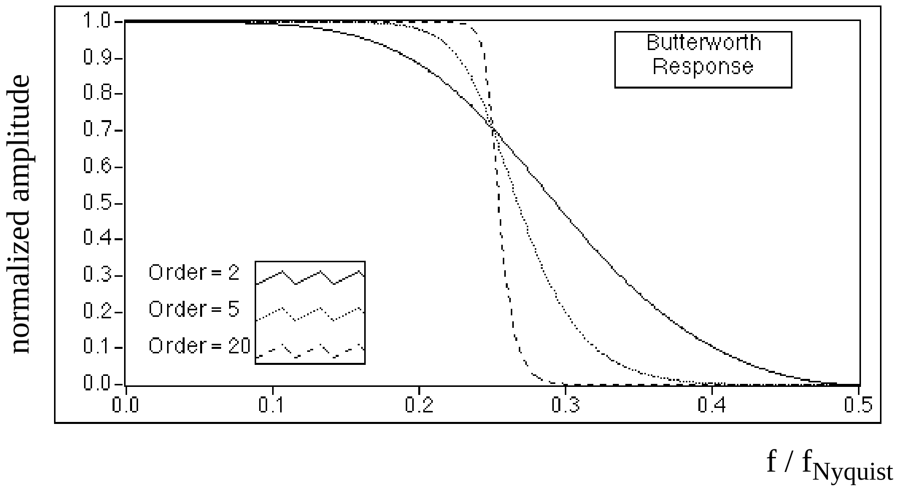
- Butterworth filters
- Chebyshev filters

- Chebyshev II or inverse Chebyshev filters
- Elliptic or Cauer filters

The characteristics of each filter type are described below.

Butterworth Filters

A Butterworth filter has no ripples in either the passband or the stopband. Due to the lack of ripples, it is also known as the *maximally flat filter*. Its frequency response is characterized by a smooth response at all frequencies. The following illustration shows the response of a lowpass Butterworth filter of different orders—the x-axis scaling is in terms of f / f_{Nyquist}, whereas the y-axis is scaled so that the gain in the passband is unity.



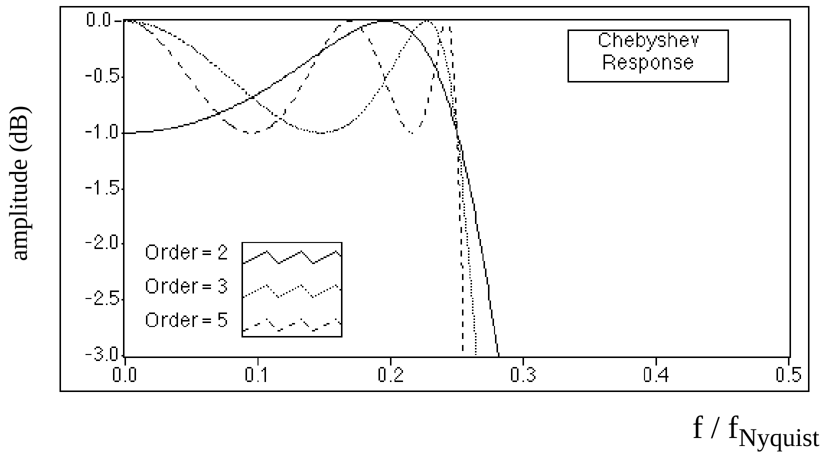
The region where the output of the filter is equal to 1 (or very close to 1) is the passband of the filter. The region where the output is 0 (or very close to 0) is the stopband. The region in between the passband and the stopband where the output gradually changes from 1 to 0 is the transition region.

The advantage of Butterworth filters is a smooth, monotonically decreasing frequency response in the transition region. As seen from the figure, the higher the filter order, the steeper the transition region.

Chebyshev Filters

The frequency response of Butterworth filters is not always a good approximation of the ideal filter response because of the slow rolloff between the passband (the portion of interest in the spectrum) and the stopband (the unwanted portion of the spectrum). On the other hand, Chebyshev filters have a smaller transition region than a Butterworth filter of the same order. However, this is achieved at the expense of ripples in the passband. Using LabVIEW or BridgeVIEW, you can specify the maximum amount of ripple (in dB) in the passband for a Chebyshev filter. The frequency response characteristics of Chebyshev filters have an equiripple (ripples all have the same magnitude) magnitude response in the passband, monotonically decreasing magnitude response in the stopband, and a sharper rolloff in the transition region as compared to Butterworth filters of the same order.

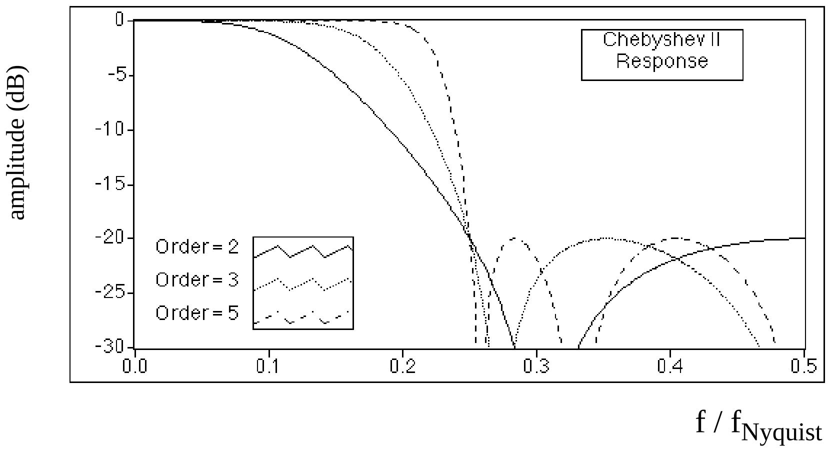
The following graph shows the response of a lowpass Chebyshev filter of different orders. In this case, the y-axis scaling is in decibels. Once again, note that the steepness of the transition region increases with increasing order. Also, the number of ripples in the passband increases with increasing order.



The advantage of Chebyshev filters over Butterworth filters is the sharper transition between the passband and the stopband with a lower-order filter. As mentioned before, this produces smaller absolute errors and higher execution speeds.

Chebyshev II or Inverse Chebyshev Filters

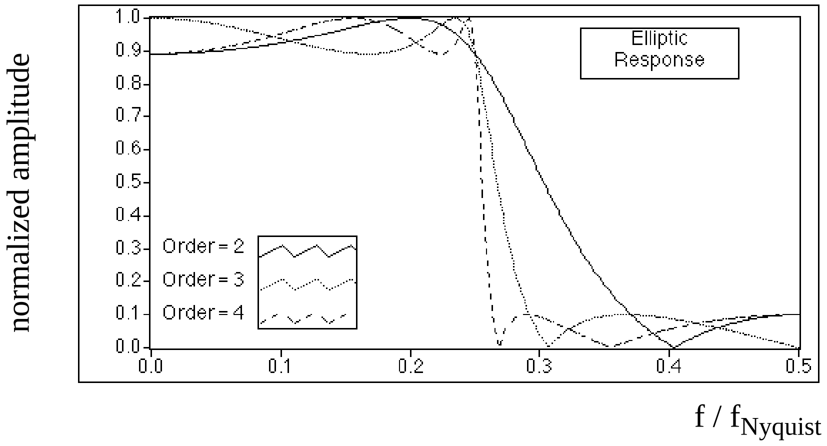
Chebyshev II, also known as inverse Chebyshev or Type II Chebyshev filters, are similar to Chebyshev filters, except that Chebyshev II filters have ripples in the stopband (as opposed to the passband), and are maximally flat in the passband (as opposed to the stopband). For Chebyshev II filters, you can specify the amount of attenuation (in dB) in the stopband. The frequency response characteristics of Chebyshev II filters are equiripple magnitude response in the stopband, monotonically decreasing magnitude response in the passband, and a rolloff sharper than Butterworth filters of the same order. The following graph plots the response of a lowpass Chebyshev II filter of different orders.



The advantage of Chebyshev II filters over Butterworth filters is that Chebyshev II filters give a sharper transition between the passband and the stopband with a lower order filter. This difference corresponds to a smaller absolute error and higher execution speed. One advantage of Chebyshev II filters over regular Chebyshev filters is that Chebyshev II filters have the ripples in the stopband instead of the passband.

Elliptic Filters

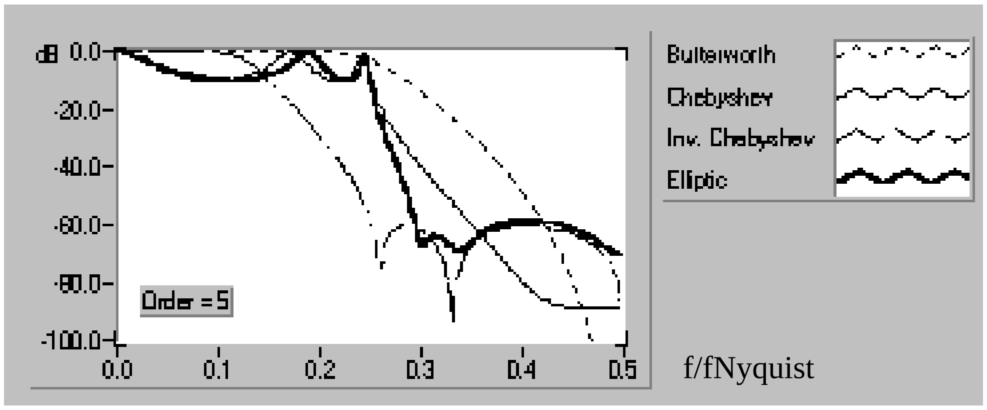
You saw that Chebyshev (type I or II) filters have a sharper transition region than a Butterworth filter of the same order. This is because they allowed ripples in the passband (type I) or the stopband (type II). Elliptic filters distribute the ripples over both the passband as well as the stopband. Equiripples in the passband and the stopband characterize the magnitude response of elliptic filters. Therefore, compared with the same order Butterworth or Chebyshev filters, the elliptic design provides the sharpest transition between the passband and the stopband. For this reason, elliptic filters are quite popular in applications where short transition bands are required and where ripples can be tolerated. The following graph plots the response of a lowpass elliptic filter of different orders. The x-axis scaling is in terms of f / f_{Nyquist}, whereas the y-axis is scaled so that the gain in the passband is unity.



Notice the sharp transition edge for even low-order elliptic filters. For elliptic filters, you can specify the amount of ripple (in dB) in the passband as well as the attenuation (in dB) in the stopband.

IIR Filter Comparison

A comparison of the lowpass frequency responses for the four different IIR filter designs, all having the same order (five), is shown in the figure below. The Elliptic filter has the narrowest transition region, whereas the Butterworth filter has the widest.



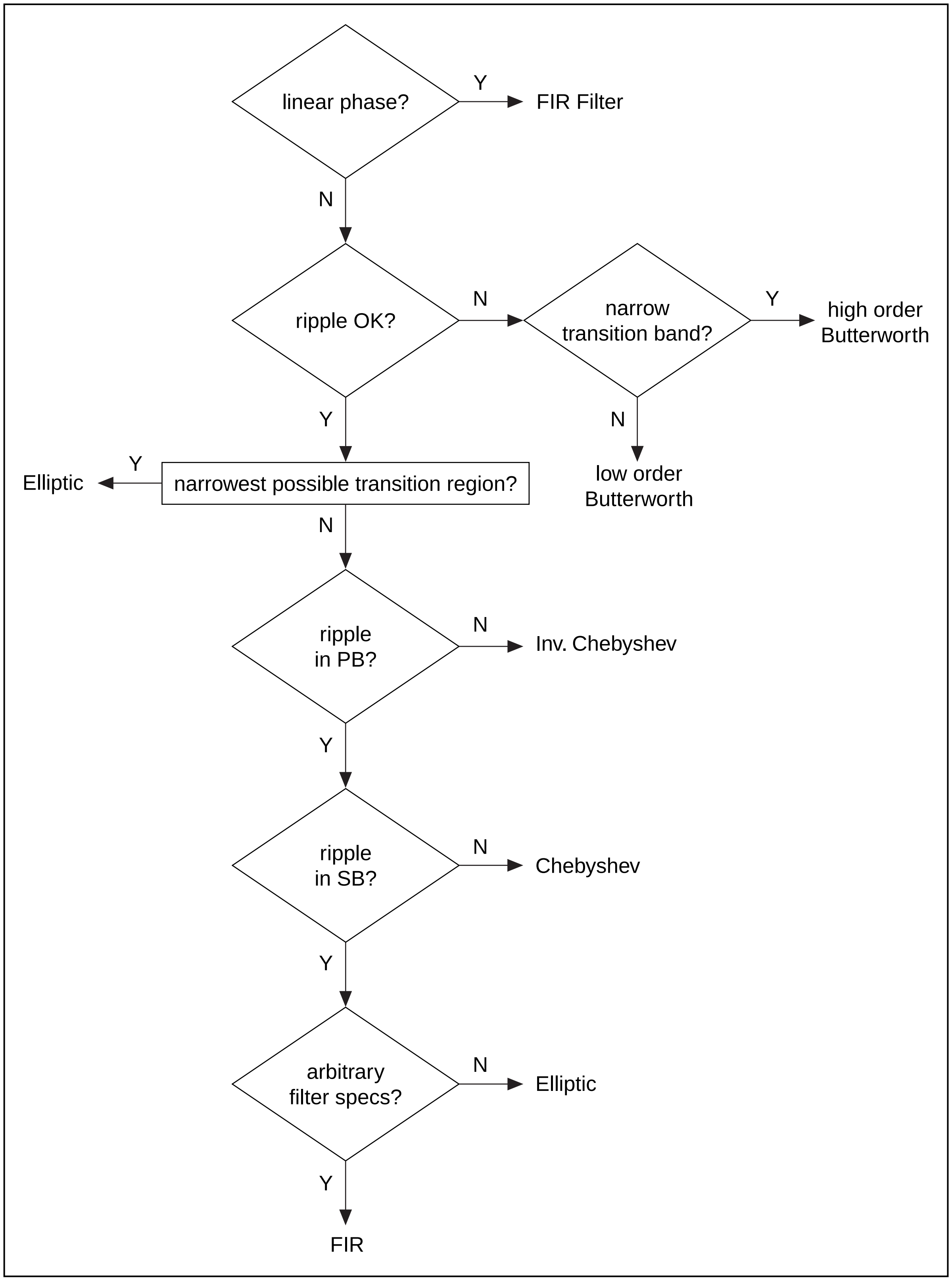
The following table compares the filter types.

IIR Filter Design	Response Characteristics	Width of Transition Region for a Fixed Order	Order Required for Given Filter Specifications
Butterworth	No ripples	Widest	Highest
Chebyshev	Ripples in PB		
Inverse Chebyshev	Ripples in SB		
Elliptic	Ripples in PB and SB	Narrowest	Lowest

The LabVIEW and BridgeVIEW digital filter VIs handle all the design issues, computations, memory management, and actual data filtering internally, and are transparent to the user. You do not need to be an expert in digital filters or digital filter theory to process the data. All you need to do is to specify the control parameters such as the filter order, cutoff frequencies, amount of ripple, and stopband attenuation.

How Do I Decide which Filter to Use?

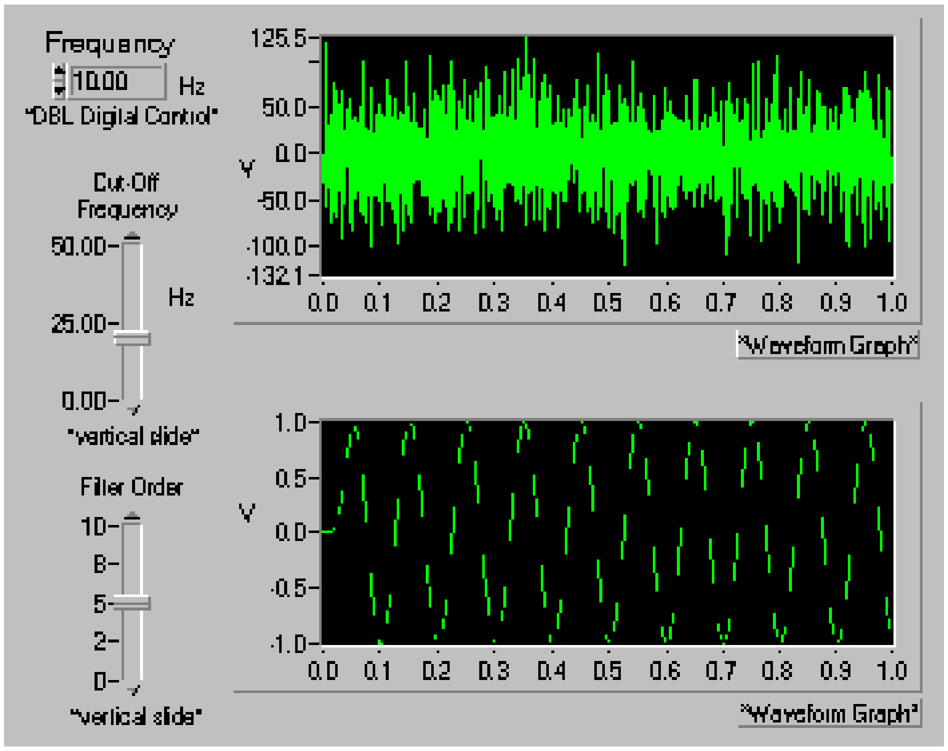
Now that you have seen the different types of filters and their characteristics, the question arises as to which filter design is best suited for your application. In general, some of the factors affecting the choice of a suitable filter are whether you require linear phase, whether you can tolerate ripples, and whether a narrow transition band is required. The following flowchart is expected to serve as a guideline for selecting the correct filter. Keep in mind that in practice, you may need to experiment with several different options before finally finding the best one.


linear phase?
Y
FIR Filter
N
ripple OK?
N
narrow transition band?
Y
high order Butterworth
N
low order Butterworth
Y
narrowest possible transition region?
Y
Elliptic
N
ripple in PB?
N
Inv. Chebyshev
Y
ripple in SB?
N
Chebyshev
Y
arbitrary filter specs?
N
Elliptic
Y
FIR

Exercise 6-1

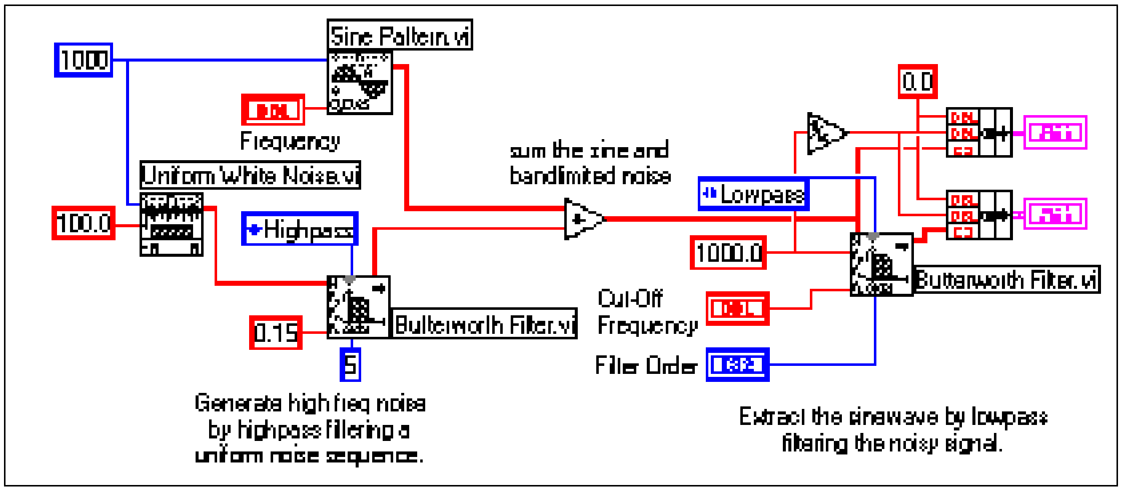
Objective: To filter data samples that consist of both high-frequency noise and a sinusoidal signal.

In this exercise, you combine a sine wave generated by the **Sine Pattern** VI with high-frequency noise. (The high-frequency noise is obtained by highpass filtering uniform white noise with a Butterworth filter.) The combined signal is then lowpass filtered by another Butterworth filter to extract the sine wave.



1. Open a new VI and build the front panel as shown above.
   a. Select a **Digital Control** from the **Numeric** palette and label it *Frequency*.
   b. Select **Vertical Slide** from the **Numeric** palette and label it *Cut-Off Frequency*.
   c. Select another **Vertical Slide** from the **Numeric** palette and label it *Filter Order*.
   d. Select a **Waveform Graph** from the **Graph** palette for displaying the noisy signal, and another **Waveform Graph** for displaying the original signal.

2. Build the block diagram as shown below.


1000
Sine Pattern.vi
Frequency
Uniform White Noise.vi
100.0
Highpass
0.15
5
Butterworth Filter.vi
Generate high freq noise by highpass filtering a uniform noise sequence.
sum the sine and bandlimited noise
Lowpass
1000.0
Cut-Off Frequency
Filter Order
Butterworth Filter.vi
0.0
Extract the sinewave by lowpass filtering the noisy signal.

Sine Pattern VI (**Functions » Analysis » Signal Generation** subpalette) generates a sine wave of the desired frequency.

Uniform White Noise VI (**Functions » Analysis » Signal Generation** subpalette) generates uniform white noise that is added to the sinusoidal signal.

Butterworth Filter VI (**Functions » Analysis » Filters** subpalette) highpass filters the noise.

Note that you are generating 10 cycles of the sine wave, and there are 1000 samples. Also, the sampling frequency to the **Butterworth Filter** VI on the right side is specified as 1000 Hz. Thus, effectively you are generating a 10 Hz signal.

3. Switch back to the front panel. Select a **Frequency** of 10 Hz, a **Cut-Off Frequency** of 25 Hz, and a **Filter Order** of 5. Run the VI.
4. Reduce the **Filter Order** to 4, 3, and 2, and observe the difference in the filtered signal. Explain what happens as you lower the filter order.

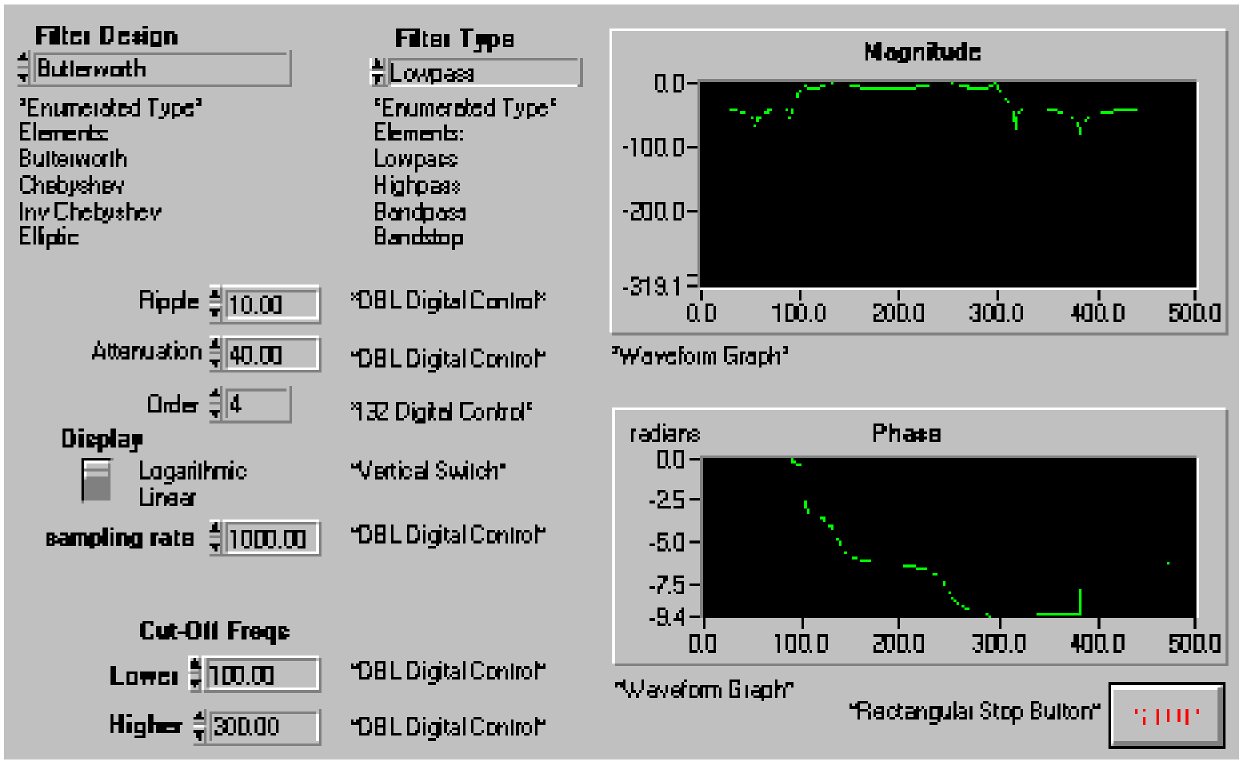
   In particular, observe the filtered waveform. At the beginning, there is a “flat” region. The length of this region depends on the order of the filter. Section G discusses this further.
5. When you finish, save the VI as **Extract the Sine Wave.vi** in the `Lvspcex.llb` library.
6. Close the VI.

End of Exercise 6-1

Exercise 6-2

Objective: To compare the frequency response characteristics of various IIR filters.

1. Open the **IIR Filter Design** VI from the `Lvspcex.llb` library.



Filter Design: Butterworth — "Enumerated Type" Elements: Butterworth, Chebyshev, Inv Chebyshev, Elliptic

Filter Type: Lowpass — "Enumerated Type" Elements: Lowpass, Highpass, Bandpass, Bandstop

Ripple: 10.00 — "DBL Digital Control"

Attenuation: 40.00 — "DBL Digital Control"

Order: 4 — "I32 Digital Control"

Display: Logarithmic / Linear — "Vertical Switch"

sampling rate: 1000.00 — "DBL Digital Control"

Cut-Off Freqs

Lower: 100.00 — "DBL Digital Control"

Higher: 300.00 — "DBL Digital Control"

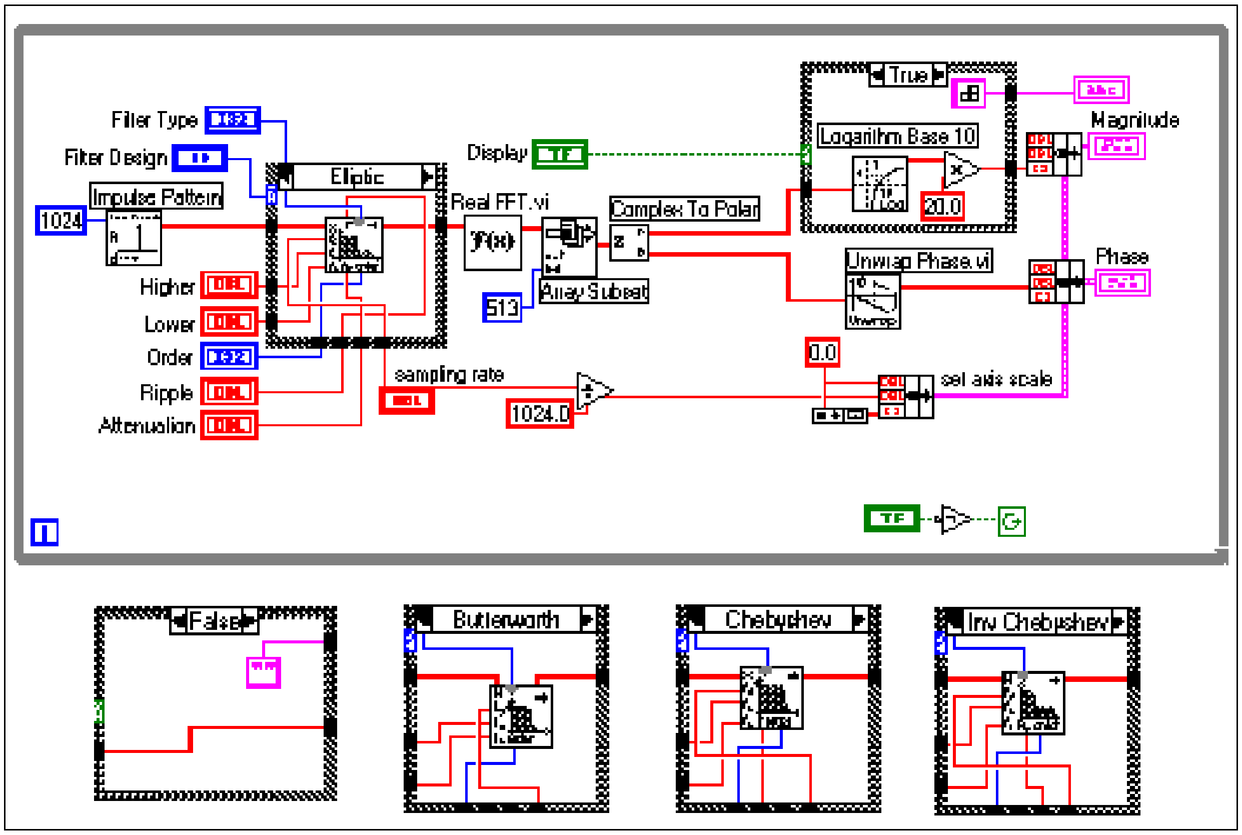
The front panel offers the choice of different types of filters.

The **Filter Design** control selects one of the four different designs of filters: Butterworth, Chebyshev, Chebyshev Type II, or Elliptic.

The **Filter Type** control selects one of four different types of filters: *highpass*, *lowpass*, *bandpass*, or *bandstop*.

The **Display** control selects the display of the magnitude response (magnitude of the frequency response) to be either *linear* or *logarithmic*.

2. The block diagram is shown below.



As seen in the block diagram, note that the frequency response is obtained by applying an impulse at the input of a filter and calculating the Fourier transform of the output.

The **Impulse Pattern** VI (**Functions » Analysis » Signal Generation** subpalette) generates an impulse that is given to the selected filter. The number of sample points is equal to 1024.

The **Real FFT** VI (**Functions » Analysis » Digital Signal Processing** subpalette) computes the Fourier transform of the output of the filter.

The **Array Subset** VI (**Functions » Array** subpalette) selects 513 FFT points out of 1024, so as to generate a one-sided spectrum.

The **Complex to Polar** VI (**Functions » Numeric » Complex** subpalette) converts the complex output of the **Real FFT VI** to its polar (magnitude and phase) representation. The magnitude can then be plotted in either the linear or dB scales.

Normally, the phase shown is limited to between -π and +π. So, even if the phase lies outside the range -π and +π , it is "wrapped" around to lie between these values. The **Unwrap Phase VI** (**Functions » Analysis » Digital Signal Processing** subpalette) is used to "unwrap" the phase to its true value, even if its absolute value exceeds π.

3. Select a **Filter Design** of Butterworth, **Filter Type** = Lowpass, **Ripple** = 10, **Attenuation** = 40, **Order** = 4, **Display** = Logarithmic, **sampling rate** = 1000, **Lower Cut-Off Frequency** = 100, and **Higher Cut-Off Frequency** = 300. Run the VI.
4. Increase the filter **Order** to 5, 10, 15, and 20, and note the difference in the magnitude and phase responses. In particular, what changes do you notice in the transition region?
5. Keep the filter **Order** fixed at 5, and change the **Filter Design** to select different IIR filters. Note the changes in the magnitude and phase plots. For a given filter order, which of the four different filter designs has the smallest transition region?
6. When you finish, stop the VI by clicking on the **STOP** button in the lower right corner.
7. Close the VI. Do not save any changes.

End of Exercise 6-2

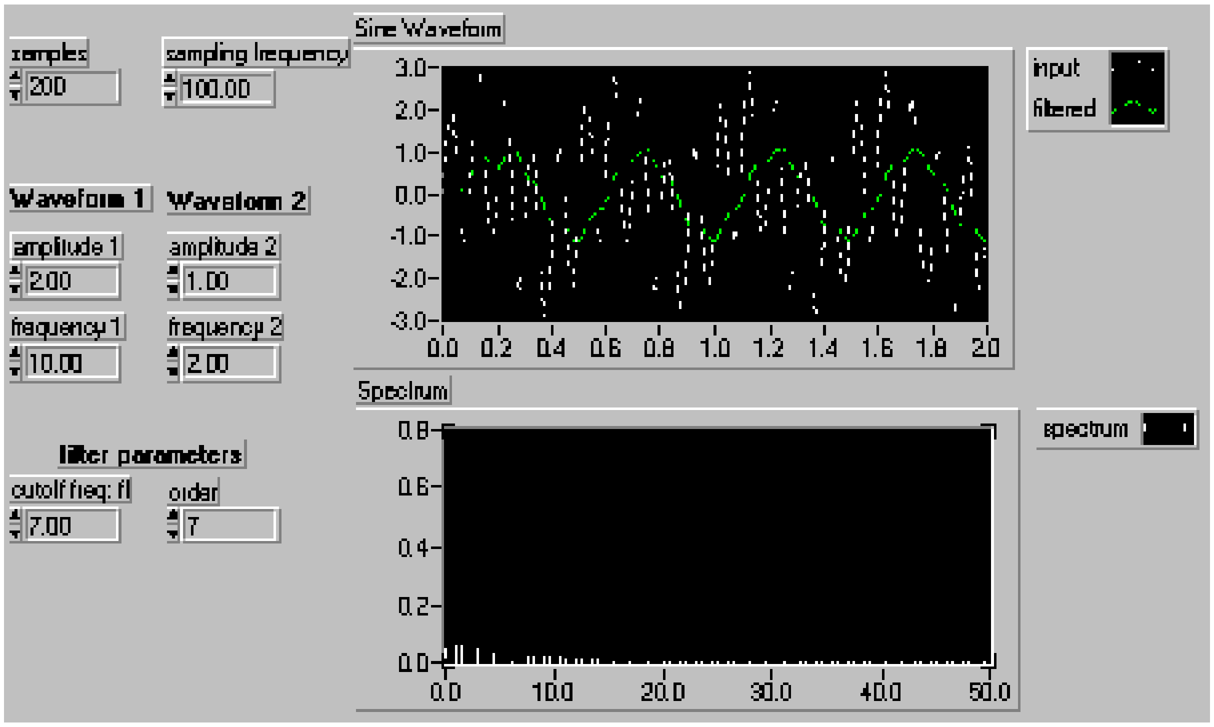
Exercise 6-3 (Optional)

OBJECTIVE: To use a digital filter to remove unwanted frequencies.

In this exercise, you will add two sine waves of different frequencies and then filter the resulting waveform using a Butterworth lowpass filter to obtain only one of the sine waves.

1. Open the **Low Pass Filter** VI from the library `Lvspcex.llb`. This VI shows how to design a lowpass Butterworth filter to remove a 10 Hz signal from a 2 Hz signal.

Front Panel



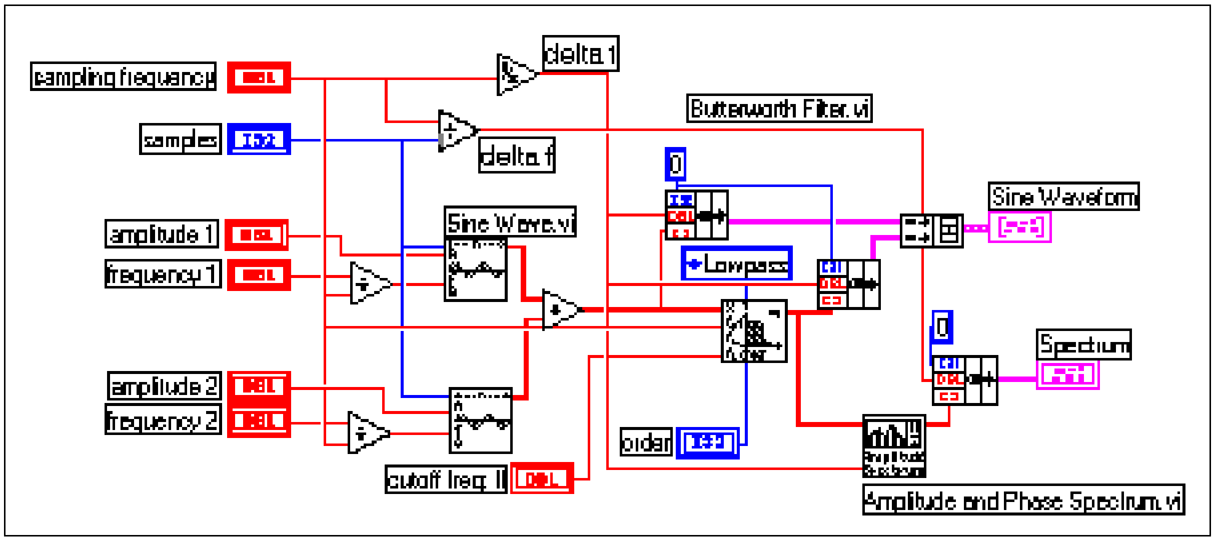
The number of samples to be generated and the sampling frequency is controlled by the **samples** and **sampling frequency** controls, respectively.

The amplitude and frequency of the two sine waves can be controlled by the **amplitude 1**, **amplitude 2**, **frequency 1**, and **frequency 2** controls on the front panel.

The **cutoff freq:fl** control controls the cutoff frequency of the lowpass filter, whose order is adjusted by the **order** control.

2. Examine the block diagram.

Block Diagram



Sine Wave VI generates the two sine waves.

Amplitude and Phase Spectrum VI determines the amplitude and phase spectrum of the output of the filtered signal. In this exercise, you are interested in only the amplitude spectrum.

The reciprocal of the sampling frequency gives the time interval, Δt, between samples.

Remember that the frequency spacing, Δf, is obtained by dividing the sampling frequency by the number of samples.

The two sine waves being combined together have frequencies of 10 Hz and 2 Hz. To separate them, you should set the cut off frequency of the lowpass filter to somewhere between these two values.

3. Keeping the **cutoff freq: fl** control at 7 Hz, run the VI. Observe that the amplitude of the 2 Hz signal is much larger than that of the 10 Hz signal.
4. Reduce the filter order to 5 and run the VI. Repeat with an order of 3. What do you notice about the spectrum amplitudes?
5. Increase the order to 12 and run the VI. Observe the spectrum amplitudes. Explain what happens.
6. When you finish, close the VI. Do not save any changes.

End of Exercise 6-3

G. The Transient Response of IIR Filters

You have seen that the output of a general IIR filter is given by

$$\begin{aligned} y[i] = {} & -a_1 y[i-1] - a_2 y[i-2] - \ldots - a_{N_y-1} y[i-(N_y-1)] + b_0 x[i] \\ & + b_1 x[i-1] + b_2[i-2] + \ldots + b_{N_x-1} x[i-(N_x-1)] \end{aligned} \tag{4}$$

where N_x is the number of forward coefficients, N_y is the number of reverse coefficients, and a_0 is assumed to be equal to 1. Consider a second-order filter where $N_x = N_y = 2$. The corresponding difference equation is:

$$y[i] = -a_1 y[i-1] - a_2 y[i-2] + b_0 x[i] + b_1 x[i-1] + b_2[i-2] \tag{5}$$

To calculate the current output (at the ith instant) of the filter, you need to know the past two outputs (at the (i-1)st and the (i-2)nd time instants) as well as the current input (at the ith time instant) and past two inputs (at the (i-1)st and (i-2)nd time instants).

Now suppose that you have just started the filtering process by taking the first sample of the input data. However, at this time instant you do not yet have the previous inputs (*x*[i-1] and *x*[i-2]) or previous outputs (*y*[i-1] and *y*[i-2]). So, by default, these values are assumed to be zero. When you get the second data sample, you already have the previous input (*x*[i-1]) and the previous output (*y*[i-1]) that you calculated from the first sample, but not yet *x*[i-2] and *y*[i-2]. Again, by default, these are assumed to be zero. It is only after we start processing the third input data sample that all the terms on the right hand side of equation (5) above now have the previously calculated values. Thus, there is a certain amount of delay before which there are calculated values for all the terms on the RHS of the difference equation describing the filter. The output of the filter during this time interval is a transient and is known as the *transient response*. For lowpass and highpass filters implemented in the LabVIEW/BridgeVIEW Analysis library, the duration of the transient response, or *delay*, is equal to the order of the filter. For bandpass and bandstop filters, this delay is 2*order.

IIR filters in the analysis library contain the following properties.

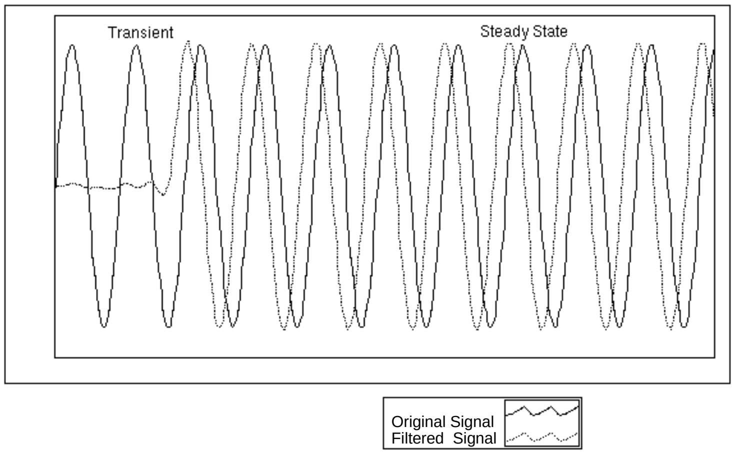
- Negative indices resulting from equation (4) are assumed to be zero the first time you call the VI.
- Because the initial filter state is assumed to be zero (negative indices), a transient proportional to the filter order occurs before the filter reaches a steady state. The duration of the transient response, or delay, for lowpass and highpass filters is equal to the order of the filter:

  delay = order

- The duration of the transient response for bandpass and bandstop filters is twice the filter order:

  delay = 2 * order

So, each time one of the filter VIs is called, this transient appears at the output. You can eliminate this transient response on successive calls by enabling the state memory of the VI. To enable state memory, set the **init/cont** control of the VI to TRUE (continuous filtering). You will see how to do this in a later exercise.

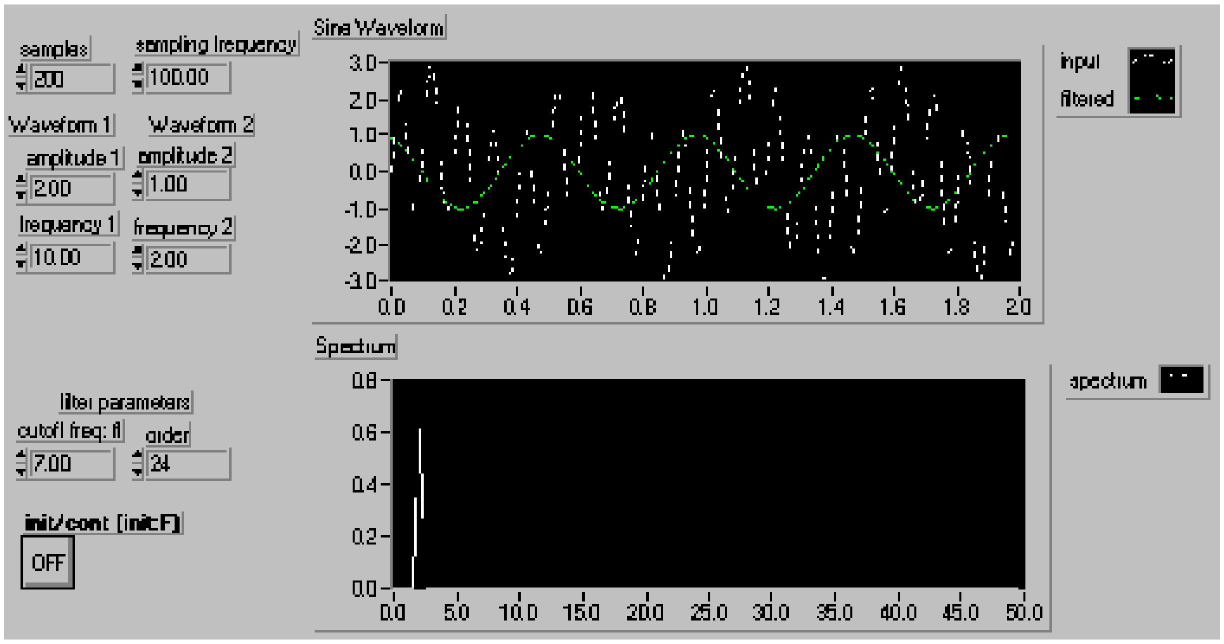


- The number of elements in the filtered sequence equals the number of elements in the input sequence.
- The filter retains the internal filter state values when the filtering completes.

Exercise 6-4

Objective: To see the difference in the filter response with and without enabling state memory.

1. Open the **Low Pass Filter** VI from `Lvspcex.llb`.



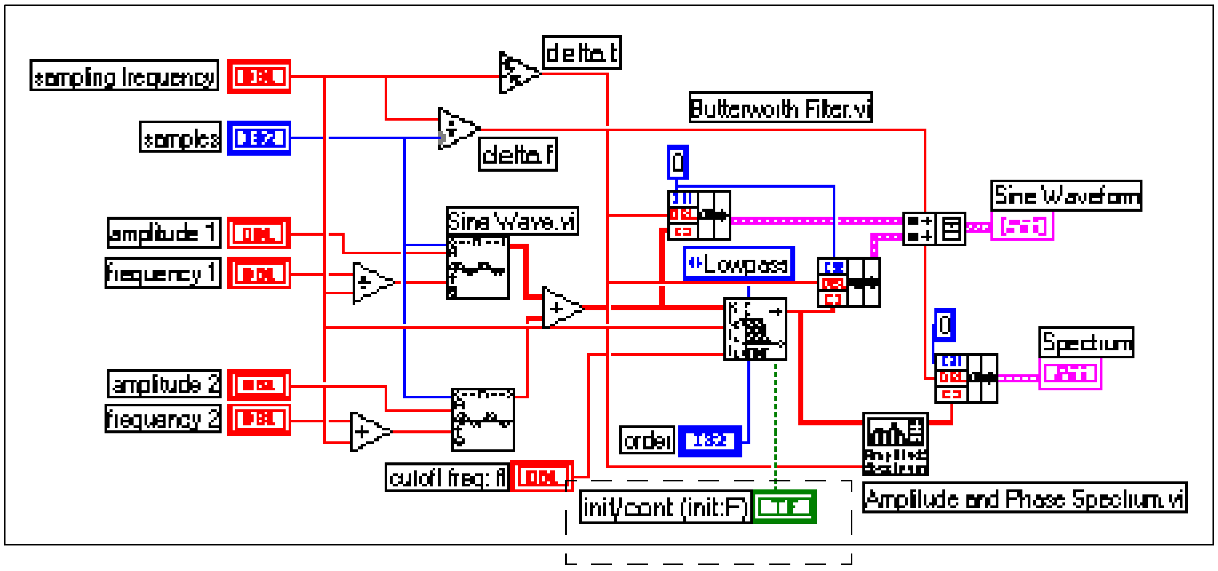
2. Run the VI with the values shown on the front panel above, but with **order** = 7. (Do not worry about the **init/cont (init: F)** control for now.)

   Observe the upper **Sine Waveform** graph, which shows two plots. The white dashed plot is the combined signal, whereas the green full line plot is the filtered signal.

3. Change the filter order to 10 and run the VI. Observe the first few values of the plot corresponding to the filtered signal.

4. Change the filter order to 15, 20, and 25, and run the VI. Each time, observe the transient that occurs at the beginning of the plot corresponding to the filtered signal. This transient can be removed after the first call to the VI by enabling its state memory. This is done by setting the **init/cont** control of the VI to TRUE. Setting this control to TRUE is equivalent to continuous filtering, and except for the first call to the VI, each successive call will not have the transient.

5. Now connect a Boolean control as shown in the diagram to the **init/cont** input of the **Butterworth Filter** VI.



6. With the **init/cont(init:F)** control set to OFF, and **order** = 15, run the VI several times. Note the presence of the transient at the beginning of the filtered signal each time the VI runs.
7. Now set the **init/cont(init:F)** control to ON, and run the VI several times. Observe that the transient is present only the first time that the VI is run. On successive calls to the VI, the transient no longer exists.
8. When you finish, save the VI.

End of Exercise 6-4

H. Finite Impulse Response Filters

You have seen that the output of an IIR filter depends on the previous outputs as well as the current and previous inputs. Because of this dependency on the previous outputs, the IIR filter has infinite memory, resulting in an impulse response of infinite length.

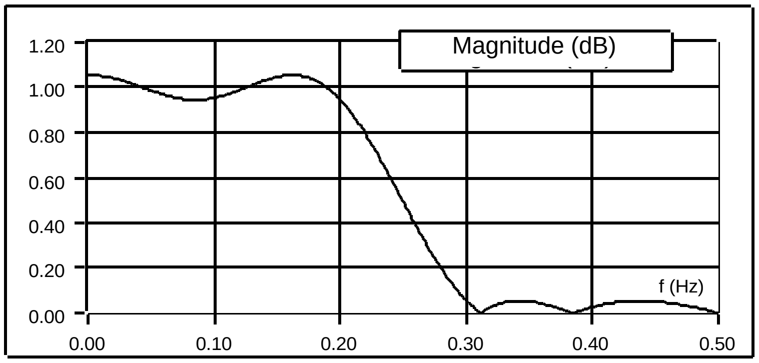
On the other hand, the output of an FIR filter depends only on the current and past inputs. Because it does not depend on the past outputs, its impulse response decays to zero in a finite amount of time. The output of a general FIR filter is given by

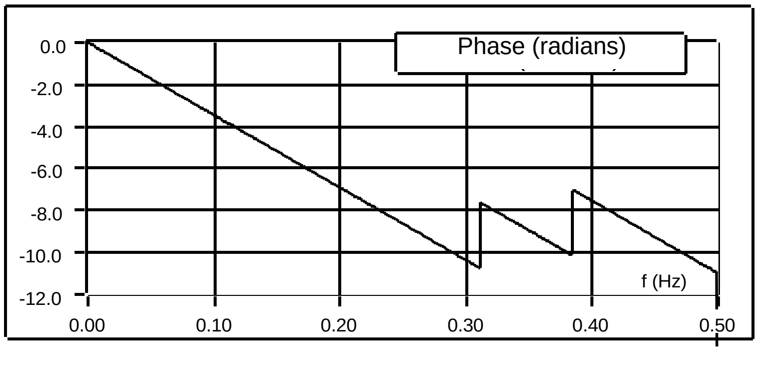
$$y[i] = b_0x[i] + b_1x[i-1] + b_2x[i-2] + ... + b_Nx[i-(M-1)]$$

where *M* is the number of taps of the filter and b_0, b_1, ... b_{M-1}, are its coefficients. FIR filters have some important characteristics:

1. They can achieve linear phase response, and hence they can pass a signal without phase distortion.
2. They are always stable. During filter design or development, you do not need to worry about stability concerns.
3. FIR filters are simpler and easier to implement.

The following graphs plot a typical magnitude and phase response of FIR filters versus normalized frequency. The discontinuities in the phase response arise from the discontinuities introduced when you compute the magnitude response using the absolute value. Notice that the discontinuities in phase are on the order of π. The phase, however, is clearly linear.



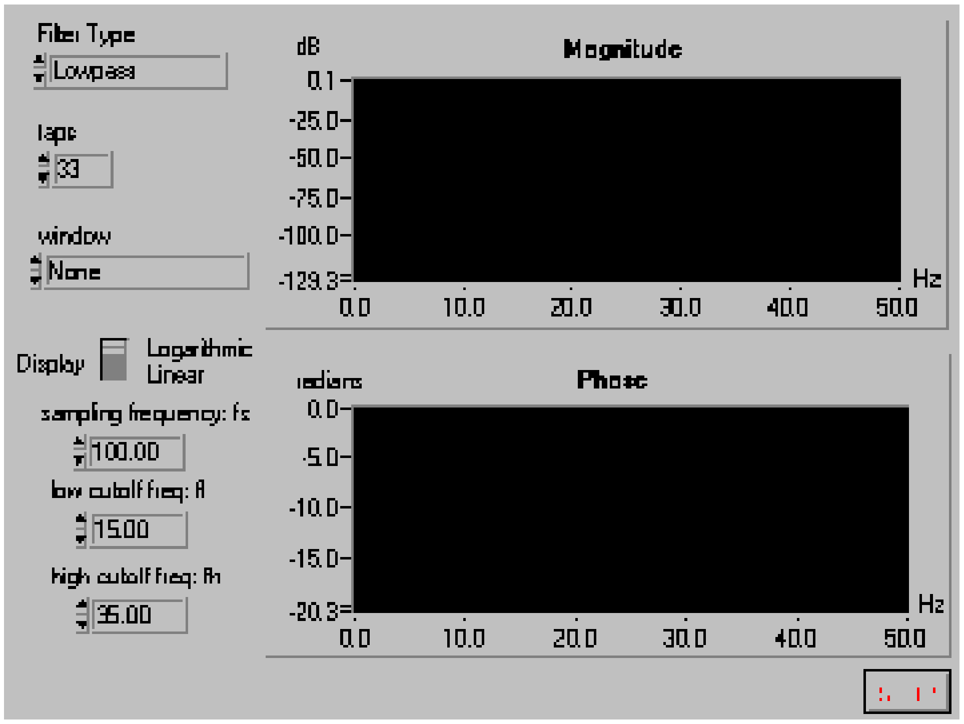


Exercise 6-5

Objective: To see the frequency response characteristics of FIR filters.

In this exercise, you will see the magnitude and phase response characteristics of FIR filters. You will also see the effect of using different windows on the filter response characteristics.

1. Open the **FIR Windowed Filter Design** VI from `Lvspcex.llb`.



The **Filter Type** control specifies the type of FIR filter—lowpass, highpass, bandpass, or bandstop.

The **taps** control specifies the number of filter coefficients. Note that taps must be greater than 0. The higher the number of taps, the steeper the transition between the passband and the stopband.

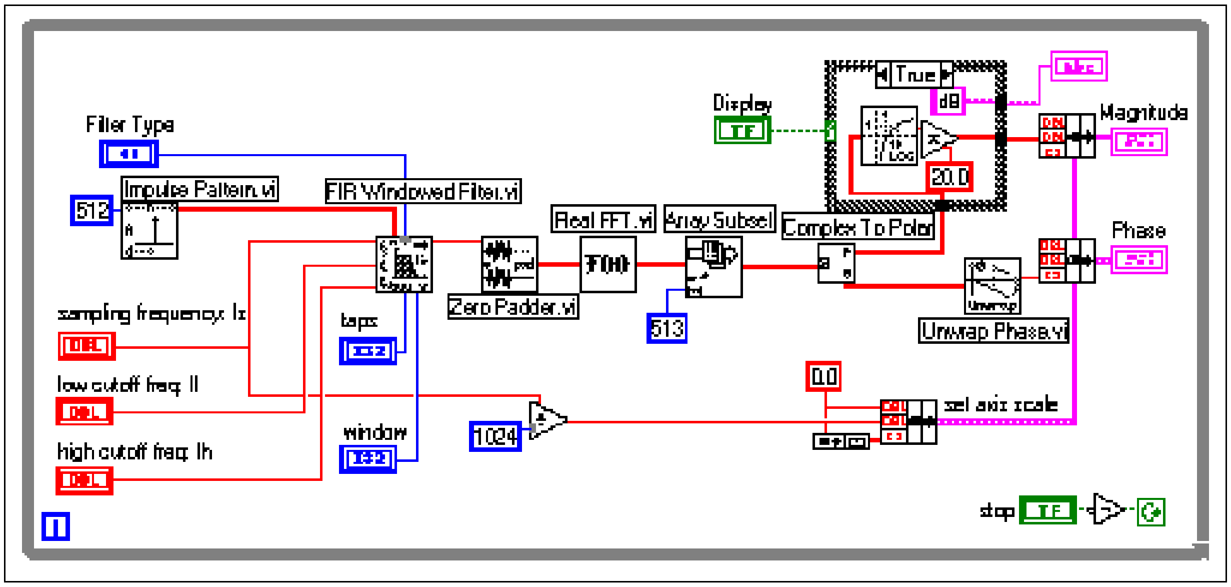
The **window** control selects among nine different types of windows to be applied to the input signal before it is filtered.

The **Display** control selects the display units on the **Magnitude** plot to be linear or logarithmic.

The **sampling frequency: fs**, **low cutoff frequency: fl**, and **high cutoff frequency: fh** controls specify the desired response characteristics of the filter. The **high cutoff frequency: fh** is required only when **Filter Type** is chosen as bandpass or bandstop.

The Magnitude and Phase plots show you the magnitude and phase response of the filter. Note that because the filter under consideration is an FIR filter, it is expected to have a linear phase response.

2. Observe the block diagram.



Impulse Pattern VI icon: **Impulse Pattern** VI (**Analysis » Signal Generation** subpalette) generates an impulse to be applied to the input of the FIR filter. The response of the filter to the impulse will give us its impulse response. Finding the FFT of the impulse response gives the magnitude and phase response.

FIR Windowed Filter VI icon: **FIR Windowed Filter** VI (**Analysis » Filters** subpalette) is an FIR filter that filters the input data sequence (in this case an impulse) using the set of windowed FIR filter coefficients specified by the **sampling frequency: fs**, **low cutoff frequency: fl**, **high cutoff frequency: fh**, and **taps** controls. This filter also windows the input signal with the type of window selected in the **window** control.

Zero Padder VI icon: **Zero Padder** VI (**Analysis » Digital Signal Processing** subpalette) resizes the input sequence to the next higher power of two by adding zeros to the end of the sequence. By converting the total number of samples to a power of two, the calculation of the Fourier transform of the impulse response of the FIR filter can be done faster by the **Real FFT** VI.

The default values have been chosen so that you will see the response characteristics of a lowpass FIR filter with cut-off frequency of 15 Hz.

You are not using any window on the input signal (that is, you are using a *rectangular* window) that is sampled at 100 Hz.

3. Run the VI with the default values. Observe the linear phase response in the passband.
4. Select different types of windows using the **window** control and observe both the magnitude and phase responses. Notice how the choice of a window affects both the responses. The phase in all cases will be linear.

The taps control affects the width of the transition region.

5. Observe the Magnitude response plot with the **Filter Type** control set to lowpass filter, the **low cutoff frequency: fl** set to 15 Hz, **sampling frequency: fs** set to 100 Hz, **window** set to Kaiser-Bessel, and **taps** set to 33.
6. Change **taps** to 55 and observe how the transition region becomes narrower.
7. Change **taps** to 10 and observe the increase in the width of the transition region.
8. You can experiment with different values of the other controls. When you are done, stop the VI by clicking on the **STOP** button in the lower right corner.
9. Close the VI without saving any changes.

End of Exercise 6-5

I. Digital Filter Design Toolkit

While it is quite easy and straightforward to design a digital filter in LabVIEW or BridgeVIEW using the analysis VIs, it does require a basic understanding of digital signal theory. For the best tool to learn more about digital filter design, National Instruments offers an add-on toolkit called the Digital Filter Design (DFD) Toolkit. This toolkit was developed using G and has an excellent interactive graphical user interface. By drawing lines in the frequency domain, all types of digital filters can be easily designed. The designed filter coefficients can then be integrated into LabVIEW, BridgeVIEW, LabWindows/CVI, and other programming environments. You will learn more about the DFD Toolkit in Lesson 10.

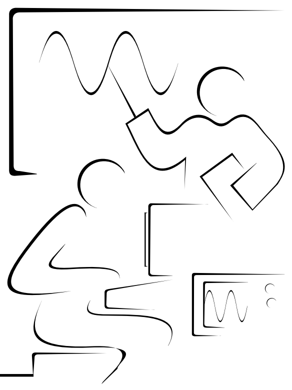
Summary

- You have seen from the frequency response characteristics that practical filters differ from ideal filters.
- For practical filters, the gain in the passband may not always be equal to 1, the attenuation in the stopband may not always be -×, and there exists a transition region of finite width.
- The width of the transition region depends on the filter order, and the width decreases with increasing order.
- The output of FIR filters depends only on the current and past input values, whereas the output of IIR filters depends on the current and past input values as well as the past output values.
- IIR filters can be classified according to the presence of ripples in the passband and/or the stopband.
- Because of the dependence of its output on past outputs, a transient appears at the output of an IIR filter each time the VI is called. This transient can be eliminated after the first call to the VI by setting its init/cont control to a TRUE value.

Review Questions

1. Name three practical examples where filtering is used.
2. If the stopband attenuation of a filter is -80 dB, what is the output of the filter to an input signal of 5 V peak if the frequency of the signal lies in the stopband of the filter?
3. Which filter would you use for the following applications?
   a. When you want the narrowest possible transition region with the smallest order.
   b. When you cannot tolerate any ripples in either the passband or the stopband.
   c. When linear phase is important.
4. Why does a transient initially appear at the output of a filter? How can it be eliminated on successive calls to a VI?

Notes



Lesson 7
Curve Fitting

Introduction

In this lesson, you will learn about the Analysis VIs that are used to fit curves to data points.

You Will Learn:

A. About curve fitting and its applications.

B. About the **General Least Squares Linear Fit** VI.

C. About the **Nonlinear Levenberg-Marquardt Fit** VI.

D. About fitting a curve to Gaussian (Normal) data points.

A. About Curve Fitting

In the digital domain, a data set can be represented by two input sequences, *Y Values* and *X Values*. A sample or point in the data set can be written as

(x[i], y[i])

where x[i] is the *i*th element of the sequence *X Values*, and y[i] is the *i*th element of the sequence *Y Values*. Each y[i] is related to the corresponding x[i]. You are interested in finding the relationship between the y[i] and the x[i] in the digital domain, and expressing it in the form of an equation in the analog domain.

Curve fitting acts as a bridge between the digital and analog worlds. Using curve fitting, digital data can be represented with a continuous model having a certain set of parameters. The basic idea is to extract a set of curve parameters or coefficients from the data set to obtain a functional description of the data set. This functional description consists of the set of parameters a_0, a_1, ..., a_k that best matches the experimental model from which the data samples x[i] and y[i] were obtained. Once you obtain the functional model, you can use it to estimate missing data points, interpolate data, or extrapolate data.

Note: ***Note that y[i] is a function of both the parameters a_k, as well as the data x[i]. The following discussion refers to the terms "linear" and "nonlinear" often. These terms refer to the relationship between y and a, and not y and x.***

The Analysis library offers both linear and nonlinear curve fitting algorithms. The different types of curve fitting in LabVIEW are outlined below:

- *Linear Fit*—fits experimental data to a straight line of the form $y = mx + c$.

  $y[i]=a_0+a_1*x[i]$

- *Exponential Fit*—fits data to an exponential curve of the form y = *aexp(bx)*

  $y[i] = a_0*\exp(a_1*x[i])$

- *General Polynomial Fit*—fits data to a polynomial function of the form $y = a + bx + cx^2 + \ldots$

  $y[i] = a_0+a_1*x[i]+a_2*x[i]^2\ldots$

  but with selectable algorithms for better precision and accuracy.

- *General Linear Fit*—fits data to

  $y[i] = a_0+a_1*f_1(x[i])+a_2*f_2(x[i]) + ...$

  where y[i] is a linear combination of the parameters a_0, a_1, a_2.... The general linear fit also features selectable algorithms for better precision and accuracy. For example, $y = a_0 + a_1*\sin(x)$ is a linear fit because *y* has a linear relationship with parameters a_0 and a_1. Polynomial fits are always linear fits for the same reason. But special algorithms can be designed for the polynomial fit to speed up the fitting processing and improve accuracy.
- *Nonlinear Levenberg-Marquardt Fit*—fits data to

  $y[i] = f(x[i], a_0, a_1, a_2...)$

  where a_0, a_1, a_2... are the parameters. This method is the most general method and does not require *y* to have a linear relationship with a_0, a_1, a_2.... It can be used to fit linear or nonlinear curves, but is almost always used to fit a nonlinear curve, because the general linear fit method is better suited to linear curve fitting. The Levenberg-Marquardt method does not always guarantee a correct result, so it is absolutely necessary to verify the results.

Mean Squared Error

The algorithm used to fit a curve to a particular data set is known as the *Least Squares* method. Let the observed data set be denoted by *y*(*x*), and let *f*(*x*,*a*) be the functional description of the data set where *a* is the set of curve coefficients that best describes the curve. The error *e*(*a*) between the observed values, and its functional description, is defined as

$$e(a) = [f(x,a) - y(x)]$$

For example, let *a* be the vector $a = \{a_0, a_1\}$. The functional description of a line is

$$f(x,a) = a_0 + a_1 x.$$

The least squares algorithm estimates the values for *a* from the values of y[i] and x[i]. After you have the values for *a*, you can obtain an estimate of the observed data set for any value of *x* using the functional description *f*(*x*,*a*). The curve fitting VIs automatically set up and solve the necessary equations and return the set of coefficients that best describes your data set. You can thus concentrate on the functional description of your data and not worry about the methods used for solving for *a*.

For each of the observed data points x[i], the differences between the polynomial value f(x[i],a) and the original data y(x[i]) are called the *residuals* and are given by

$$e_i(a) = f_i(x[i],a) - y_i(x[i])$$

The mean square error (MSE) is a relative measure of these residuals between the expected curve values calculated by the functional description f(x[i],a), and the actual observed values of y[i], and is given by

$$\text{MSE} = \frac{1}{N}\sum_{i=0}^{N-1}(f_i - y_i)^2 = \frac{1}{N}\sum_{i=0}^{N-1}(e_i(a))^2$$

where *f* is the sequence representing the fitted values, *Y* is the sequence representing the observed values, and *N* is the number of sample points observed. The smaller the MSE, the better is the fit between the functional description and the observed y(x).

In general, for each predefined type of curve fit, there are two types of VIs, unless otherwise specified. One type returns only the coefficients. The other type returns the coefficients, the corresponding expected or fitted curve, and the MSE.

Applications of Curve Fitting

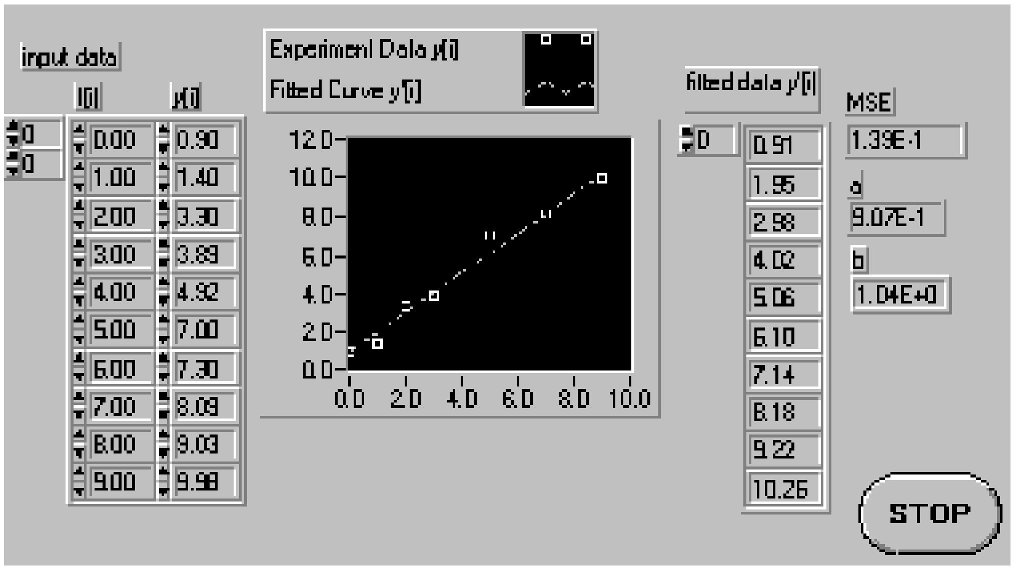
The practical applications of curve fitting are numerous. Some of them are listed below.

- Removal of measurement noise.
- Filling in missing data points (for example, if one or more measurements were missed or improperly recorded).
- Interpolation (estimation of data between data points) (for example, if the time between measurements is not small enough).
- Extrapolation (estimation of data beyond data points) (for example, if you are looking for data values before or after the measurements were taken).
- Differentiation of digital data. (For example, if you need to find the derivative of the data points. The discrete data can be modeled by a polynomial, and the resulting polynomial equation can be differentiated.)
- Integration of digital data (for example, to find the area under a curve when you have only the discrete points of the curve).
- To obtain the trajectory of an object based on discrete measurements of its velocity (first derivative) or acceleration (second derivative).

Exercise 7-1

OBJECTIVE: To perform a linear curve fit on experimental data.

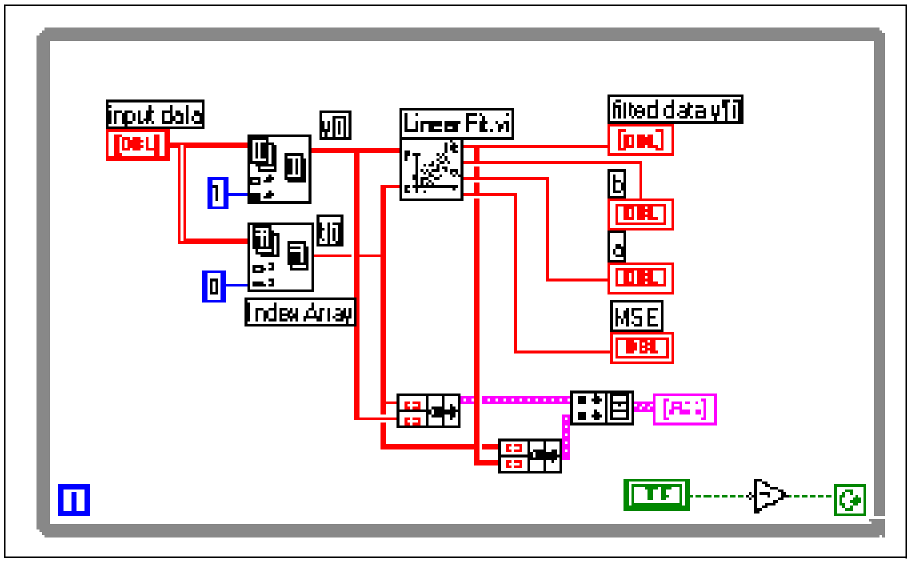
Front Panel



t[i]	y[i]	fitted data y'[i]
0.00	0.90	0.91
1.00	1.40	1.95
2.00	3.30	2.98
3.00	3.89	4.02
4.00	4.92	5.06
5.00	7.00	6.10
6.00	7.30	7.14
7.00	8.09	8.18
8.00	9.03	9.22
9.00	9.98	10.26

1. Open the **Linear Curve Fit** VI from the library `Lvspcex.llb`. This example assumes that you have collected 10 pairs of experimental data *t* and *y*, and have reason to believe there may be a linear relationship between each pair.
2. On the front panel, the input data control on the left shows the values of the data points t[i] and y[i]. After calculating the equation for the linear fit, the calculated values of y[i] for the measured values of t[i] are shown on the fitted data indicator on the right. The VI also gives the values of the parameters a_0 and a_1 (**a** and **b** indicators on the front panel) and the resulting *MSE*.
3. Switch to the block diagram.

Block Diagram



4. Examine the block diagram.

   Linear Fit VI (**Analysis » Curve Fitting** subpalette). In this exercise, this VI fits the data to a line and finds the coefficients a and b such that y[i] = a + b*t[i], as well as the mean squared error between the data and the linear fit.

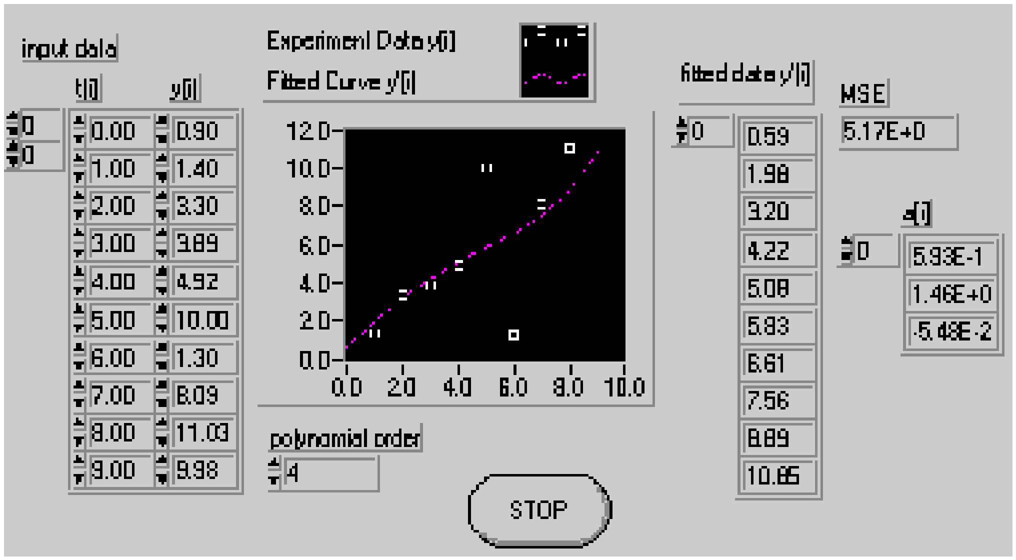
5. The input data is in a two-dimensional array, which is a common format when the data is collected from DAQ hardware. You use the **Index Array** VI to obtain two one-dimensional arrays, y[i] and t[i].
6. The MSE is the mean squared error. A smaller error indicates a better fit.
7. Run the VI. The graph should display the original data, as well as the linear fit.
8. Change the values of y[i] in the input data control. Observe the corresponding plots and the effect on the MSE and slope when there are points that don't fit well.
9. When you finish, stop the VI by clicking on the STOP button.
10. Close the VI. Do not save any changes.

End of Exercise 7-1

Exercise 7-2

OBJECTIVE: To perform a polynomial curve fit on experimental data.

Front Panel



input data

t[i]	y[i]
0.00	0.90
1.00	1.40
2.00	3.30
3.00	3.89
4.00	4.92
5.00	10.00
6.00	1.30
7.00	8.09
8.00	11.03
9.00	9.98

fitted data y'[i]: 0.59, 1.98, 3.20, 4.22, 5.08, 5.83, 6.61, 7.56, 8.89, 10.85

MSE: 5.17E+0

a[i]: 5.93E-1, 1.46E+0, -5.48E-2

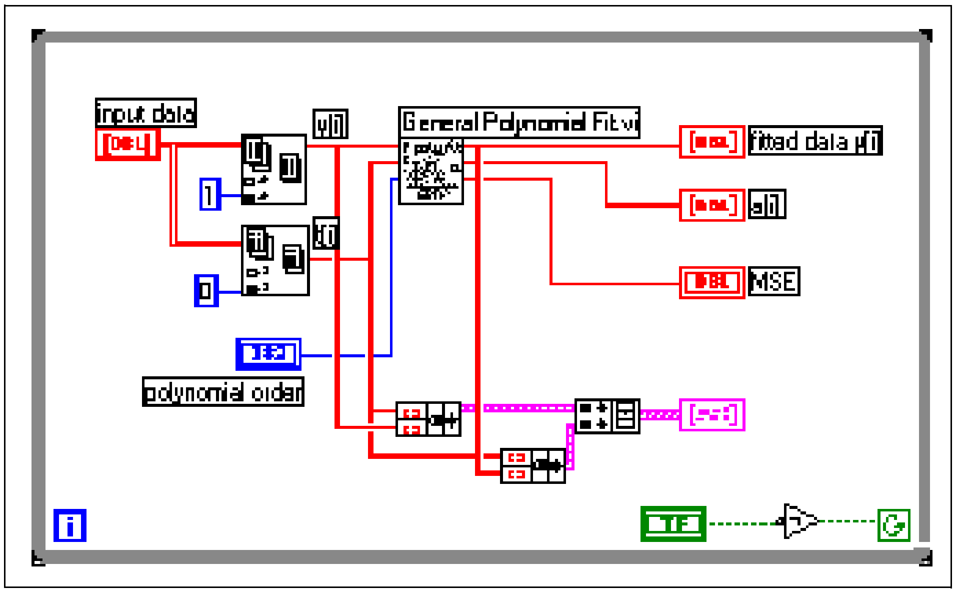
polynomial order: 4

1. Open the **Polynomial Fit** VI from the library `Lvspcex.llb`. This example assumes that the experimental input data has a polynomial relationship where

   $y[i] = a_0 + a_1{*}t[i] + a_2{*}t[i]^2 \ldots$

2. When the polynomial order is 1, there are two coefficients (a_0 and a_1) and the result is a linear fit as in exercise 7-1. However, when the order is 2, it is a second order polynomial fit with three coefficients. The polynomial coefficients are stored in an array a[i]. You can use the **polynomial order** control to choose the order of the polynomial.
3. Switch to the block diagram.

Block Diagram



4. Examine the block diagram.

   General Polynomial Fit VI (**Analysis » Curve Fitting** subpalette). In this exercise, this VI fits the data to a second-order polynomial curve and returns the fit data, the coefficients, and the mean squared error between the data and the polynomial fit.

5. You use the polynomial fit to obtain the fitting coefficients a_0, a_1, a_2, etc. In general, you want to use the lowest order possible to fit the polynomial.
6. Run the VI. The original as well as the fitted data should appear on the graph.
7. Change the values of y[i] in the input data control and the order of the polynomial in the polynomial order control, and observe the changes in the plots and the MSE.
8. When you finish, stop the VI by clicking on the **STOP** button.
9. Close the VI. Do not save any changes.

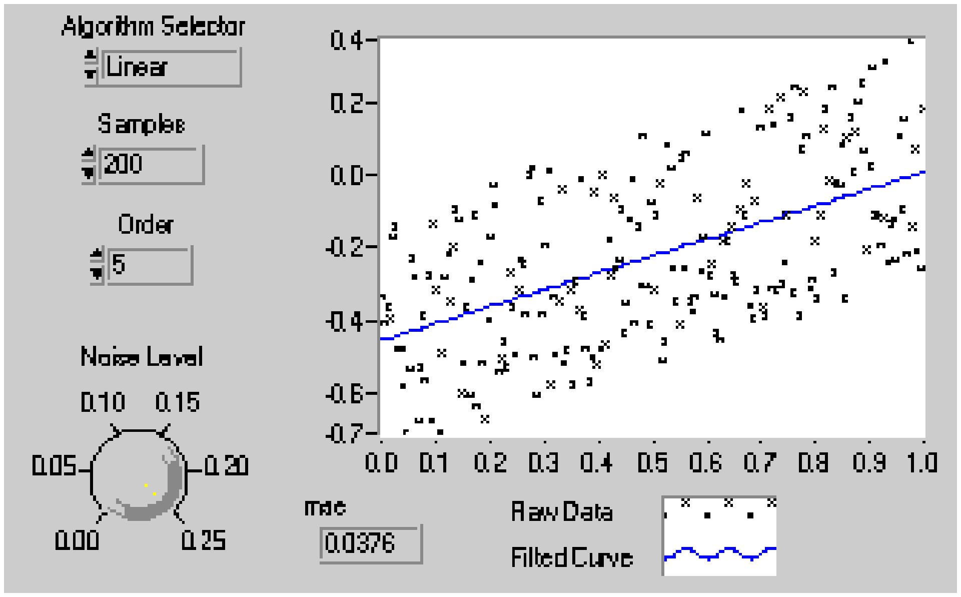
End of Exercise 7-2

Exercise 7-3

OBJECTIVE: To use and compare the Linear, Exponential and Polynomial Curve Fit VIs to obtain the set of least square coefficients that best represent a set of data points.

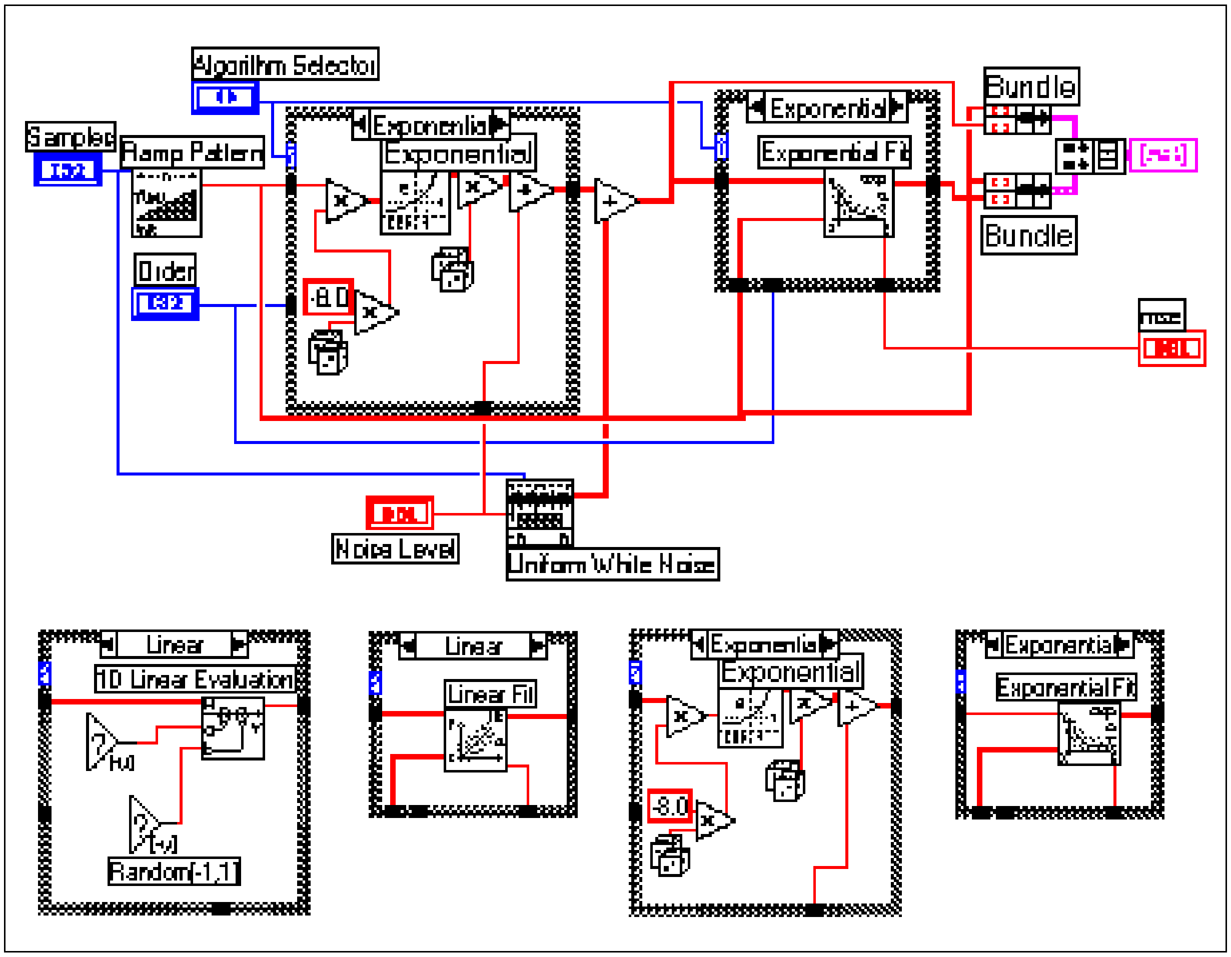
1. Open the **Regressions Demo** VI from the library `Lvspcex.llb`. The front panel and block diagram are already built for you.

Front Panel



This VI generates “noisy” data samples that are approximately linear, exponential, or polynomial. It then uses the corresponding analysis curve fitting VIs to determine the parameters of the curve that best fits those data points. (At this stage, you do not need to worry about how the noisy data samples are generated.) You can control the noise amplitude with the **Noise Level** control on the front panel.

Block Diagram



2. Select *Linear* in the **Algorithm Selector** control, and set the **Noise Level** control to about 0.1. Run the VI. Note the spread of the data points and the fitted curve (straight line).
3. Experiment with different values of **Order** and **Noise Level**. What do you notice? How does the mse change?
4. Change the **Algorithm Selector** to *Exponential* and run the VI. Experiment with different values of **Order** and **Noise Level**. What do you notice?
5. Change the **Algorithm Selector** to *Polynomial* and run the VI. Experiment with different values of **Order** and **Noise Level**. What do you notice?

6. In particular, with the **Algorithm Selector** control set to *Polynomial*, change the **Order** to 0 and run the VI. Then change it to 1 and run the VI. Explain your observations.
7. Depending on your observations in steps 2, 3, 4, and 5, for which of the algorithms (Linear, Exponential, Polynomial) is the **Order** control the most effective? Why?
8. Close the VI. Do not save any changes.

End of Exercise 7-3

B. General LS Linear Fit

The **Linear Fit** VI calculates the coefficients a_0 and a_1 that best fits the experimental data (x[i] and y[i]) to a straight line model given by

$y[i] = a_0 + a_1*x[i]$

Here, y[i] is a linear combination of the coefficients a_0 and a_1. You can extend this concept further so that the multiplier for a1 is some function of *x*. For example:

$y[i] = a_0 + a_1*\sin(\omega x[i])$ or

$y[i] = a_0 + a_1*x[i]^2$ or

$y[i] = a_0 + a_1*\cos(\omega x[i]^2)$

where ω is the angular frequency. In each of these cases, y[i] is a linear combination of the coefficients a_0 and a_1. This is the basic idea behind the **General LS Linear Fit** VI, where the y[i] can be linear combinations of several coefficients, each of which may be multiplied by some function of the x[i]. Therefore, you can use it to calculate coefficients of the functional models that can be represented as linear combinations of the coefficients, such as

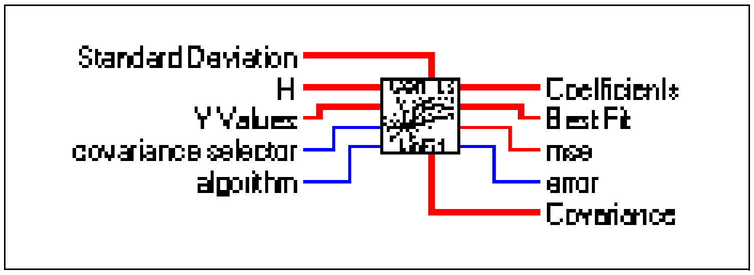
$y = a_0 + a_1*\sin(\omega x)$ or

$y = a_0 + a_1*x^2 + a2*\cos(\omega x^2)$

$y = a_0 + a_1*(3\sin(\omega x)) + a_2*x^3 + a_3 / x + \ldots$

In each case, note that *y* is a *linear* function of the coefficients (although it may be a nonlinear function of *x*).

You will now see how to use the **General LS Linear Fit** VI to find the best linear fit to a set of data points. The inputs and outputs of the **General LS Linear Fit** VI are shown below.



The data that you collect (x[i] and y[i]) is to be given to the inputs **H** and **Y Values**. The **Covariance** output is the matrix of covariances between the coefficients a_k, where c_{ij} is the covariance between a_i and a_j, and c_{kk} is the variance of a_k. At this stage, you need not be concerned about the inputs **Standard Deviation**, **covariance selector,** and **algorithm**. For now, you will just use their default values. You can refer to the *LabVIEW Analysis VI Reference Manual* for more details on these inputs.

The matrix *H* is known as the *observation matrix* and will be explained in more detail later. **Y Values** is the set of observed data points y[i]. For example, suppose you have collected samples (**Y Values**) from a transducer and you want to solve for the coefficients of the model:

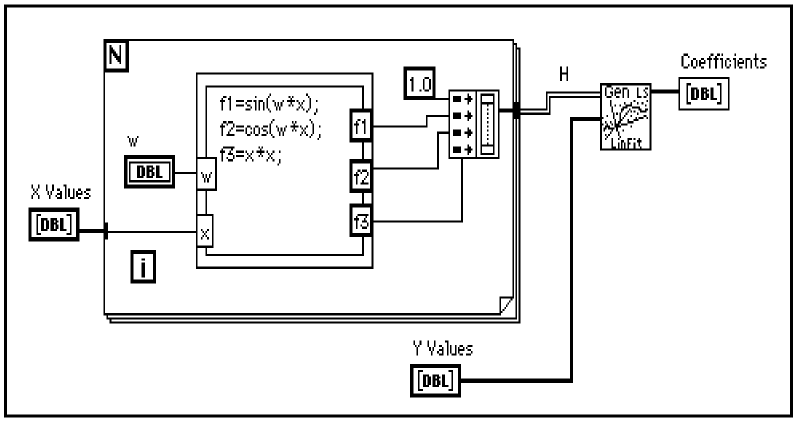
$$y = a_o + a_1 \sin(\omega x) + a_2 \cos(\omega x) + a_3 x^2$$

You see that the multiplier for each a_j (0 ð *j* ð3) is a different function. For example, a_0 is multiplied by 1, a_1 is multiplied by sin(ω*x*), a_2 is multiplied by cos(ω*x*), and so on. To build *H*, you set each column of *H* to the independent functions evaluated at each *x* value, x[i]. Assuming there are 100 "*x*" values, *H* would be:

$$H = \begin{bmatrix} 1 & \sin(\omega x_0) & \cos(\omega x_0) & x_0^2 \\ 1 & \sin(\omega x_1) & \cos(\omega x_1) & x_1^2 \\ 1 & \sin(\omega x_2) & \cos(\omega x_2) & x_2^2 \\ \dots & \dots & \dots & \dots \\ 1 & \sin(\omega x_{99}) & \cos(\omega x_{99}) & x_{99}^2 \end{bmatrix}$$

If you have *N* data points and *k* coefficients (a_0, a_1, ...a_{k-1}) for which to solve, *H* will be an *N-by-k* matrix with *N* rows and *k* columns. Thus, the number of rows of *H* is equal to the number of elements in **Y Values**, whereas the number of columns of *H* is equal to the number of coefficients for which you are trying to solve.

In practice, *H* is not available and must be built. Given that you have the *N* independent **X Values** and observed **Y Values**, the following block diagram demonstrates how to build *H* and use the **General LS Linear Fit** VI.



Exercise 7-4

OBJECTIVE: To learn how to set up the input parameters and use the General LS Linear Fit VI.

This exercise demonstrates how to use the **General LS Linear Fit** VI to obtain the set of least square coefficients *a* and the fitted values, and also how to set up the input parameters to the VI.

The purpose is to find the set of least square coefficients **a** that best represent the set of data points (x[i], y[i]). As an example, suppose you have a physical process that generates data using the relationship

$$y = 2h_0(x) + 3h_1(x) + 4h_2(x) + \text{noise} \qquad (1)$$

where

$$h_0(x) = \sin(x^2),$$

$$h_1(x) = \cos(x),$$

$$h_2(x) = \frac{1}{x+1}, \text{ and}$$

noise is a random value. Also, assume you have some idea of the general form of the relationship between *x* and *y*, but are not quite sure of the coefficient values. So, you may think that the relationship between *x* and *y* is of the form

$$y = a_0f_0(x) + a_1f_1(x) + a_2f_2(x) + a_3f_3(x) + a_4f_4(x) \qquad (2)$$

where

$$f_o(x) = 1.0,$$

$$f_1(x) = \sin(x^2),$$

$$f_2(x) = 3\cos(x),$$

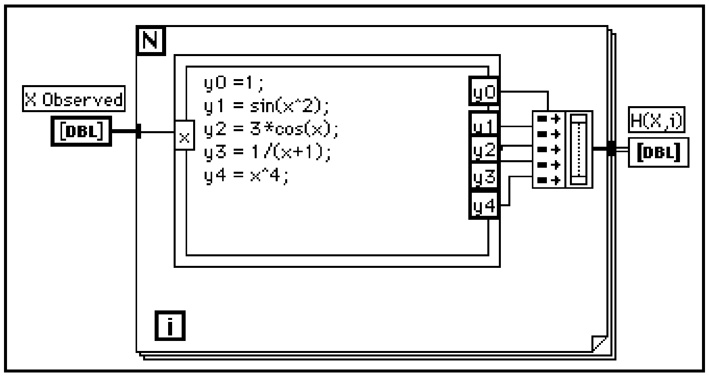
$$f_3(x) = \frac{1}{x+1},$$

$$f_4(x) = x^4 .$$

Equations (1) and (2) respectively correspond to the actual physical process and to your guess of this process. The coefficients you choose in your guess may be close to the actual values, or may be far away from them. Your objective now is to accurately determine the coefficients *a*.

Building the Observation Matrix

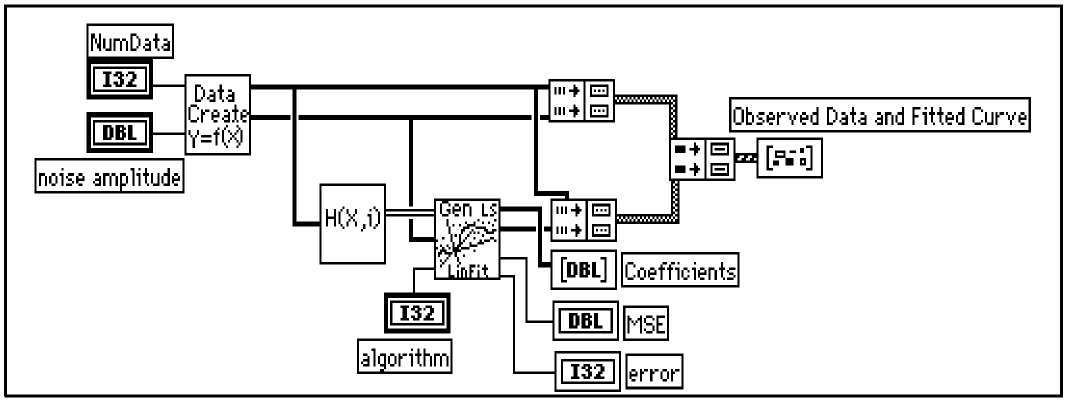
To obtain the coefficients *a*, you must supply the set of (x[i], y[i]) points in the arrays *H* and *Y Values* (where the matrix *H* is a 2D array) to the **General LS Linear Fit** VI. The x[i] and y[i] points are the values observed in your experiment. A simple way to build the matrix *H* is to use the Formula Node as shown in the following block diagram.



You can easily edit the formula node to change, add, or delete functions. At this point, you have all the necessary inputs to use the **General LS Linear Fit** VI to solve for *a*. To obtain equation (1) from equation (2), you need to multiply $f_0(x)$ by 0.0, $f_1(x)$ by 2.0, $f_2(x)$ by 1.0, $f_3(x)$ by 4.0 and $f_4(x)$ by 0.0. Thus, looking at equations (1) and (2), note that the expected set of coefficients are

$$a = \{0.0, 2.0, 1.0, 4.0, 0.0\} \quad .$$

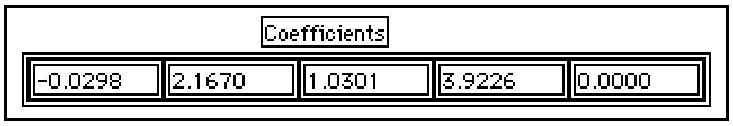
The block diagram below demonstrates how to set up the **General LS Linear Fit** VI to obtain the coefficients and a new set of *y* values.



The subVI labeled **Data Create** generates the **X** and **Y** arrays. You can replace this icon with one that actually collects the data in your experiments. The icon labeled **H(X,i)** generates the 2D matrix *H*.

The last portion of the block diagram overlays the original and the estimated data points and produces a visual record of the General LS

Linear Fit. Executing the **General LS Linear Fit** VI with the values of **X**, **Y**, and **H** returns the following set of coefficients.



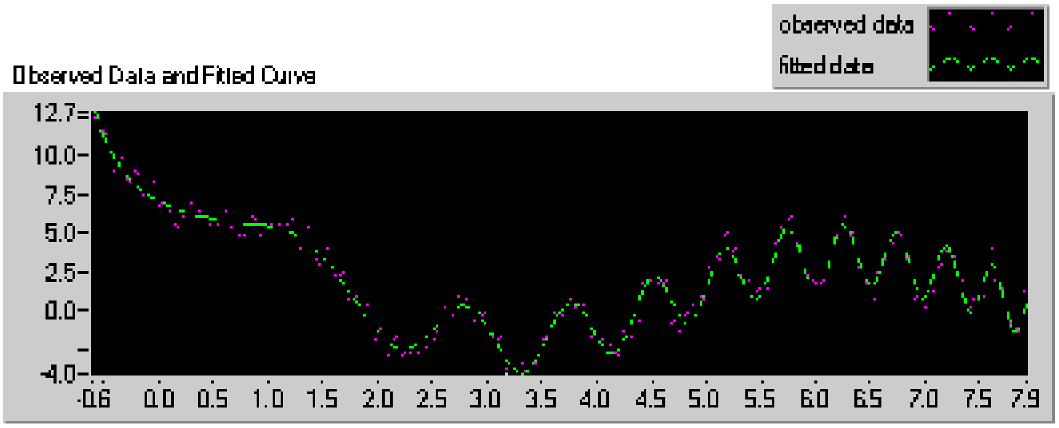
Coefficients				
-0.0298	2.1670	1.0301	3.9226	0.0000

The resulting equation is thus

$y = 0.0298(1) + 2.1670\sin(x^2) + 1.0301(3\cos(x)) + 3.9226/(x+1) + 0.00(x^4)$

$= 0.0298 + 2.1670\sin(x^2) + 1.0301(3\cos(x)) + 3.9226/(x+1)$

The following graph displays the results.



You will now see the VI in which this particular example has been implemented.

1. Open the **General LS Fit Example** VI from the library `Lvspcex.llb`.
2. Examine the block diagram and the front panel.

   noise amplitude: can change the amplitude of the noise added to the data points. The larger this value, the more the spread of the data points.

   NumData: the number of data points that you want to generate.

   algorithm: provides a choice of six different algorithms to obtain the set of coefficients and the fitted values. In this particular example, there is no significant difference among different algorithms. You can select different algorithms from the front panel to see the results. In some cases, different algorithms may have significant differences, depending on your observed data set.

MSE: gives the mean squared error. The smaller the MSE, the better the fit.

error: gives the error code in case of any errors. If error code = 0, it indicates no error. For a list of error codes, see the appendix.

Coefficients: the calculated values of the coefficients (a_0, a_1, a_2, a_3, and a_4) of the model.

3. Run the VI with progressively larger values of the **noise amplitude**. What happens to the observed data plotted on the graph? What about the MSE?
4. For a fixed value of **noise amplitude**, run the VI by choosing different algorithms from the **algorithm** control. Do you find that any one algorithm is better than the other? Which one gives you the lowest MSE?
5. When you finish, close the VI. Do not save any changes.

End of Exercise 7-4

Exercise 7-5

OBJECTIVE: To predict production costs using the General LS Linear Fit VI.

The VIs that you have seen so far have been used to fit a curve to a function of only one variable. In this exercise, you will use the **General LS Linear Fit** VI to fit a curve to a multivariable function. In particular, the function will have two variables, *X1* and *X2*. You can, however, generalize it to functions of three or more variables.

Suppose you are the manager of a bakery and want to estimate the total cost (in dollars) of a production of baked scones using the quantity produced, *X1*, and the price of one pound of flour, *X2*. To keep things simple, the following five data points form this sample data table.

Cost (dollars)	**Quantity**	**Flour Price**
Y	**X1**	**X2**
$150	295	3.00
$75	100	3.20
$120	200	3.10
$300	700	2.80
$50	60	2.50

You want to estimate the coefficients to the equation:

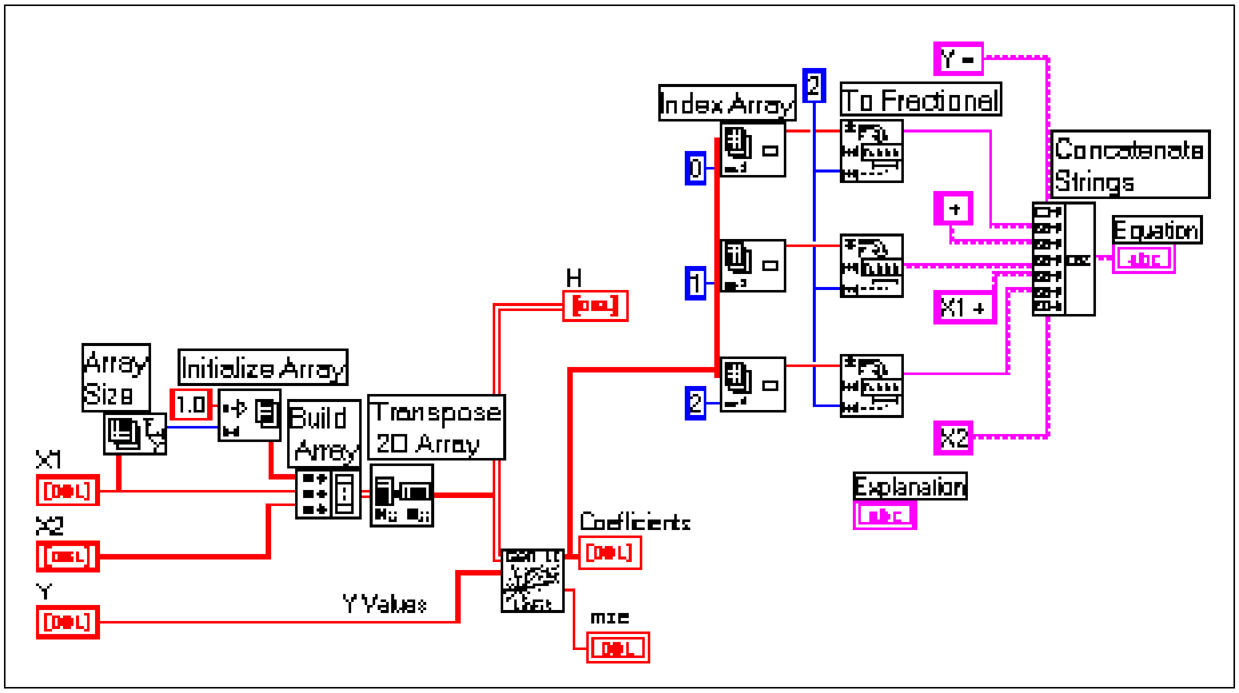
$$Y=a_0 + a_1X1 + a_2X2.$$

You can use the **General LS Linear Fit** VI. It must have the inputs **H**, the observation matrix, and **Y Values**, a vector of values of the LHS of the above equation. Each column of *H* is the observed data for each of the independent variables associated with the coefficients a_0, a_1, and a_2. Note that the first column is one because the coefficient a_0 is not associated with any independent variable. Thus, *H* should be filled in as:

$$H = \begin{bmatrix} 1 & 295 & 3.00 \\ 1 & 100 & 3.20 \\ 1 & 200 & 3.10 \\ 1 & 700 & 2.80 \\ 1 & 60 & 2.50 \end{bmatrix}$$

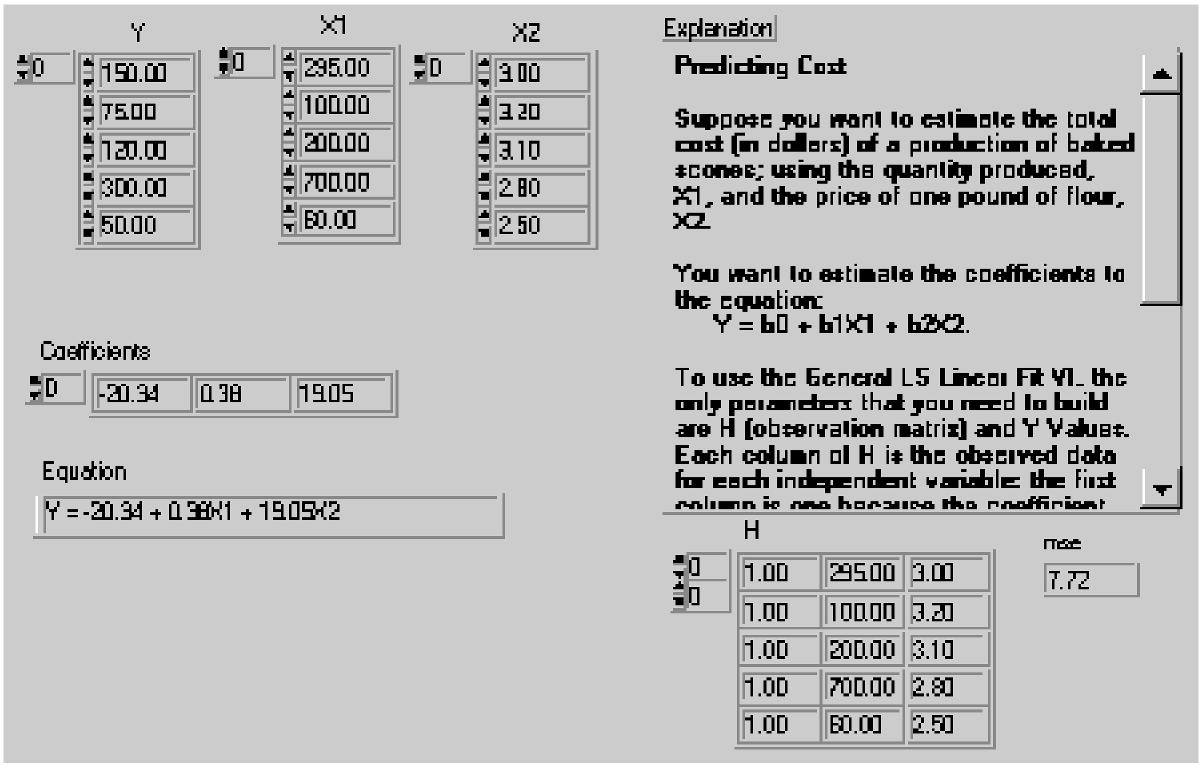
In LabVIEW (or BridgeVIEW), the observed data would normally appear in three arrays (*Y*, *X1*, and *X2*). The following block diagram demonstrates how to build *H*.

Block Diagram



1. Open the **Predicting Cost** VI from the library `Lvspcex.llb`. The values of Y, X1, and X2 have already been entered into the corresponding controls on the front panel.
2. Examine the block diagram. Be sure you understand how to build the matrix *H*. Most of the rest of the block diagram is used to build the string that displays the equation of the functional model. You do not need to worry about the rest of the diagram for now.
3. Switch to the front panel and run the VI. Check to see whether the matrix *H* was created correctly.

Front Panel



Y	X1	X2
150.00	295.00	3.00
75.00	100.00	3.20
120.00	200.00	3.10
300.00	700.00	2.80
50.00	80.00	2.50

Coefficients

-20.34	0.38	19.05

Equation

Y = -20.34 + 0.38X1 + 19.05X2

Explanation

Predicting Cost

Suppose you want to estimate the total cost (in dollars) of a production of baked scones; using the quantity produced, X1, and the price of one pound of flour, X2.

You want to estimate the coefficients to the equation:
Y = b0 + b1X1 + b2X2.

To use the General LS Linear Fit VI, the only parameters that you need to build are H (observation matrix) and Y Values. Each column of H is the observed data for each independent variable; the first column is one because the coefficient

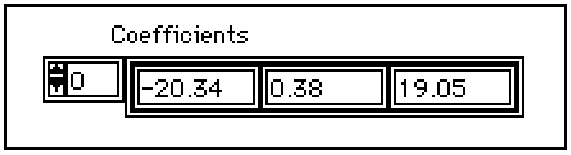
H

1.00	295.00	3.00
1.00	100.00	3.20
1.00	200.00	3.10
1.00	700.00	2.80
1.00	80.00	2.50

mse

7.72

After running the **Predicting Cost VI**, the following coefficients are obtained.



Coefficients

-20.34	0.38	19.05

and the resulting equation for the total cost of scone production is therefore:

Y=-20.34 + 0.38X1 + 19.05X2

4. Experiment with different values of X2 (flour price).
5. When you finish, close the VI. Do not save any changes.

End of Exercise 7-5

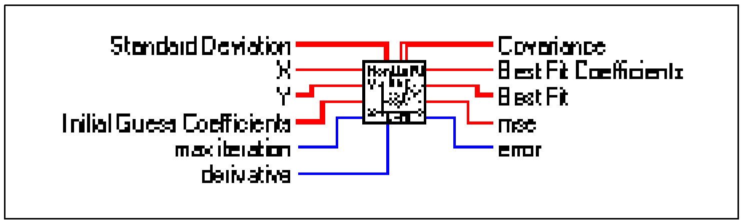
C. Nonlinear Lev-Mar Fit

So far, you have seen VIs that are used when there is a linear relationship between *y* and the coefficients a_0, a_1, a_2, ... However, when a nonlinear relationship exists, you can use the **Nonlinear Lev-Mar Fit** VI to determine the coefficients. This VI uses the Levenberg-Marquardt method, which is very robust, to find the coefficients **A** = {a_0, a_1, a_2, ..., a_k} of the nonlinear relationship between **A** and y[i]. The VI assumes that you have prior knowledge of the nonlinear relationship between the *x* and *y* coordinates.

As a preliminary step, you need to specify the nonlinear function in the **Formula Node** on the block diagram of one of the subVIs of the **Nonlinear Lev-Mar Fit** VI. This particular subVI is the **Target Fnc and Deriv NonLin** VI. You can access the **Target Fnc and Deriv NonLin** VI by selecting it from the menu that appears when you select **Project » This VI's SubVIs**.

Note: ***When using the Nonlinear Lev-Mar Fit VI, you also need to specify the nonlinear function in the Formula Node on the block diagram of the Target Fnc and Deriv NonLin VI.***

The connections to the **Nonlinear Lev-Mar Fit** VI are shown below:



X and **Y** are the input data points x[i] and y[i].

Initial Guess Coefficients is your initial guess as to what the coefficient values are. The coefficients are those used in the formula that you entered in the **Formula Node** of the **Target Fnc and Deriv NonLin** VI. Using the **Nonlinear Lev-Mar Fit** VI successfully sometimes depends on how close your initial guess coefficients are to the actual solution. Therefore, it is always worth taking the time and effort to obtain a good initial guess to the solution from any available resource.

For now, you can leave the other inputs to their default values. For more information on these inputs, see the LabVIEW (or BridgeVIEW) *Analysis VI Reference Manual*.

Best Fit Coefficients: the values of the coefficients (a_0, a_1, ...) that best fit the model of the experimental data.

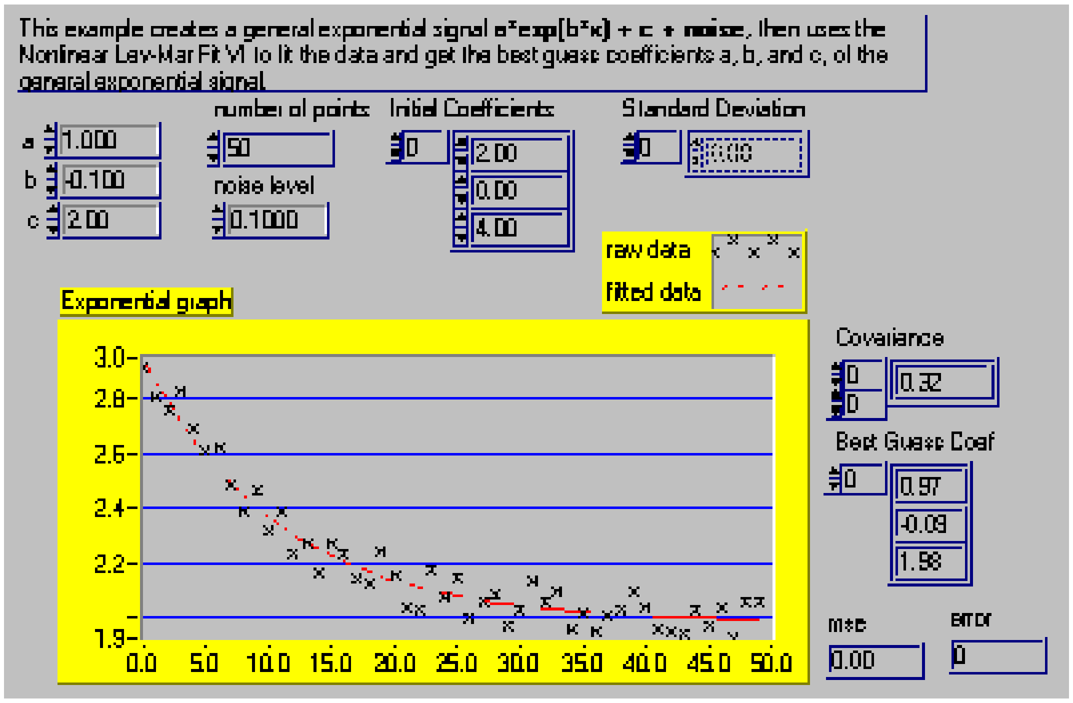
Exercise 7-6

OBJECTIVE: To create a general exponential signal a*exp(b*x) + c + noise. Then, use the Nonlinear Lev-Mar Fit VI to fit the data and get the best guess coefficients a, b, and c of the general exponential signal.

In this exercise, you will see how to use the **Nonlinear Lev-Mar Fit** VI to determine the coefficients *a*, *b*, and *c*, of a nonlinear function given by a*exp(b*x) + c.

1. Open the **Nonlinear Lev-Mar Exponential Fit** VI from the library `Lvspcex.llb`. The front panel is shown below.

Front Panel



This example creates a general exponential signal a*exp[b*x] + c + noise, then uses the Nonlinear Lev-Mar Fit VI to fit the data and get the best guess coefficients a, b, and c, of the general exponential signal.

a 1.000
b -0.100
c 2.00

number of points 50
noise level 0.1000

Initial Coefficients 0 | 2.00 | 0.00 | 4.00

Standard Deviation 0 | 0.00

Covariance 0 | 0 | 0.32

Best Guess Coef 0 | 0.97 | -0.09 | 1.98

mse 0.00
error 0

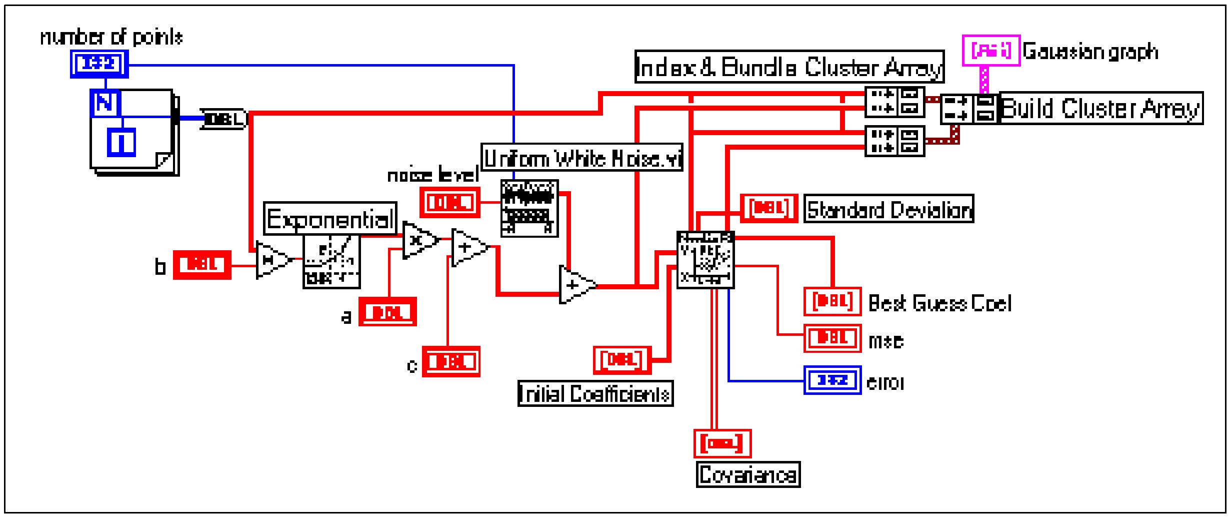
The **a**, **b**, and **c** controls determine the actual values of the coefficients *a*, *b* and *c*. The **Initial Coefficients** control is your educated guess as to the actual values of *a*, *b* and *c*. Finally, the **Best Guess Coef** indicator gives you the values of *a*, *b* and *c* calculated by the **Nonlinear Lev-Mar Fit** VI. To simulate a more practical example, you also add noise to this equation, thus making it of the form:

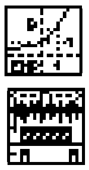
a*exp(b*x) + c + noise

The **noise level** control adjusts the noise level. Note that the actual values of *a*, *b*, and *c* being chosen are +1.0, -0.1 and 2.0. In the **Initial Coefficients** control, the default guess for these is *a* = 2.0, *b* = 0, and *c* = 4.0.

2. Examine the block diagram.

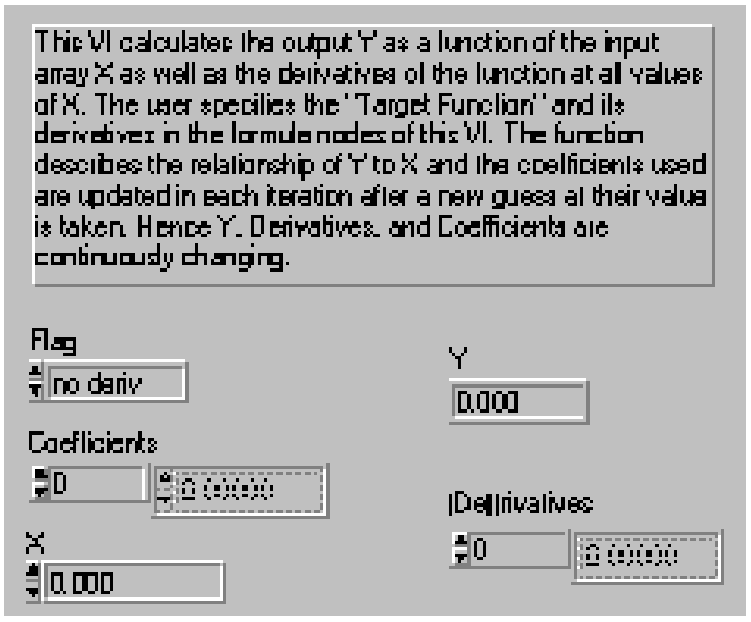
Block Diagram





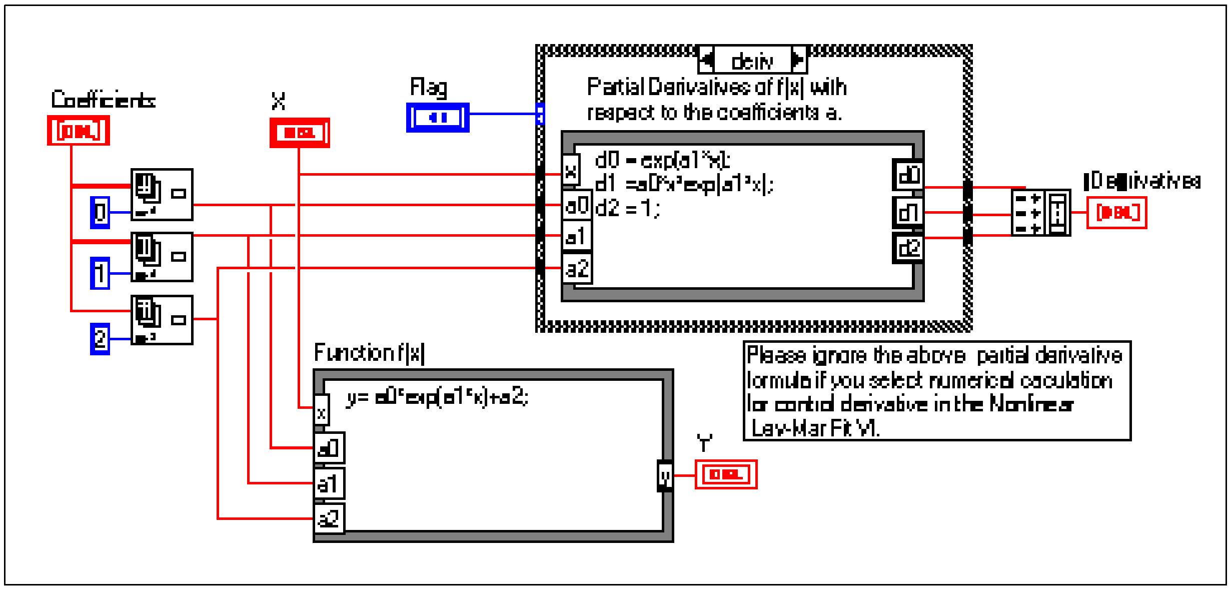
The data samples of the exponential function are simulated using the **Exponential** VI (**Numeric » Logarithmic** subpalette) and uniform white noise is added to the samples with the help of the **Uniform White Noise** VI (**Analysis » Signal Generation** subpalette).

3. From the **Project** menu, select **Unopened SubVIs » Target Fnc and Deriv NonLin** VI. The front panel of the **Target Fnc and Deriv NonLin** VI opens, as shown below.



This VI calculates the output 'Y' as a function of the input array X as well as the derivatives of the function at all values of X. The user specifies the 'Target Function' and its derivatives in the formula nodes of this VI. The function describes the relationship of 'Y' to X and the coefficients used are updated in each iteration after a new guess at their value is taken. Hence Y, Derivatives, and Coefficients are continuously changing.

4. Switch to the block diagram.


Coefficients
X
Flag
deriv
Partial Derivatives of f(x) with respect to the coefficients a.
d0 = exp(a1*x);
d1 = a0*x*exp(a1*x);
d2 = 1;
Derivatives
Function f(x)
y= a0*exp(a1*x)+a2;
Please ignore the above partial derivative formula if you select numerical caculation for partial derivative in the Nonlinear Lev-Mar Fit VI.
Y

Observe the formula node at the bottom. It has the form of the function whose parameters (a0, a1, and a2) you are trying to evaluate.

5. Close the front panel and the block diagram of the **Target Fnc and Deriv NonLin** VI.
6. Run the **NonLinear Lev-Mar Exponential Fit** VI. Note that the values of the coefficients returned in **Best Guess Coef** are very close to the actual values entered in the **Initial Coefficients** control. Also note the value of the **mse**.
7. Increase the **noise level** from 0.1 to 0.5. What happens to the **mse** and the coefficient values in **Best Guess Coef**? Why?
8. Change the **noise level** back to 0.1 and the **Initial Coefficients** to 5.0, -2.0, and 10.0, and run the VI. Note the values returned in the **Best Guess Coef** and the **mse** indicators.
9. With the **noise level** still at 0.1, change your guess of the **Initial Coefficients** to 5.0, 8.0, and 10.0, and run the VI. This time, your guess is further away than the one you chose in step 4. Note the error! This goes to show how important it is to have a reasonably educated guess for the coefficients.
10. When you finish, close the VI. Do not save any changes.

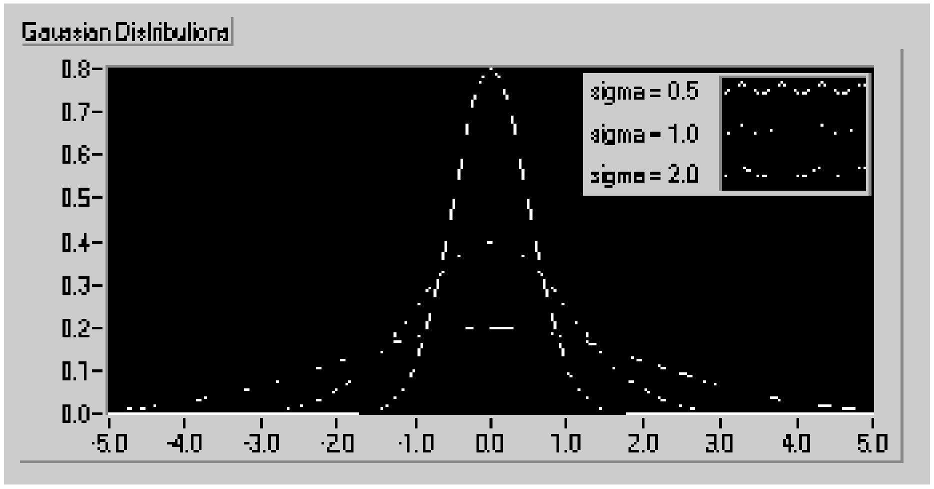
End of Exercise 7-6

D. Fitting a Curve to Gaussian (Normal) Data Points

Real-world data is very often Gaussian distributed. That means that many of the data points lie close to a particular value, known as the mean, and the number of data points is smaller as you get further away from the mean. The mathematical description of a Gaussian (also known as a Normal) distribution is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \qquad (3)$$

where μ is the *mean* and σ is the *standard deviation*. The following figure shows the Gaussian distribution with μ = 0 and σ = 0.5, 1.0, and 2.0.

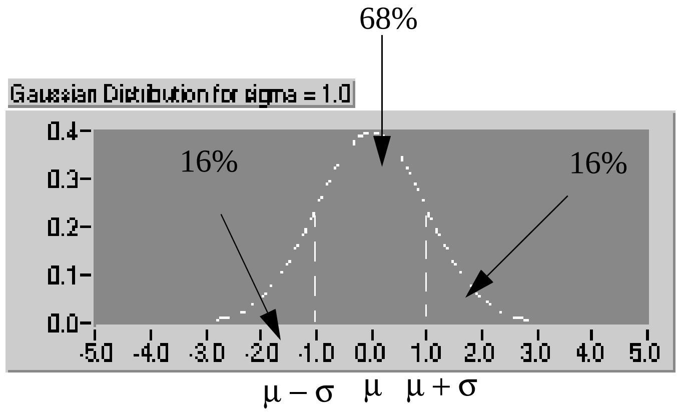


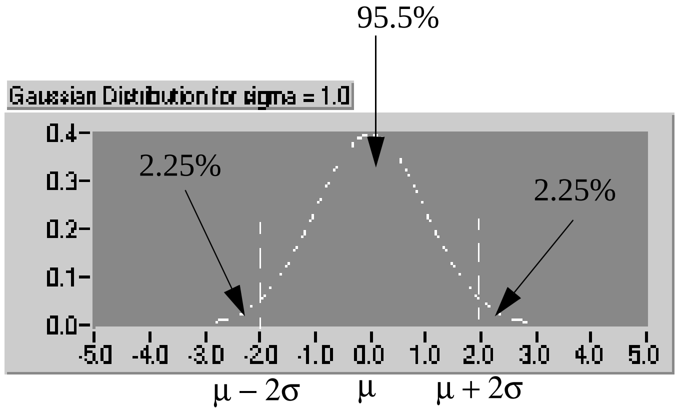
As seen from the figure, the curve is bell shaped and is symmetric about the mean, μ. The peak of the curve occurs at μ. The standard deviation, σ, determines the "spread" of the curve around the mean. The smaller the value of σ, the more concentrated the curve around the mean, the higher the peak at the mean, and the steeper the descents on both sides.

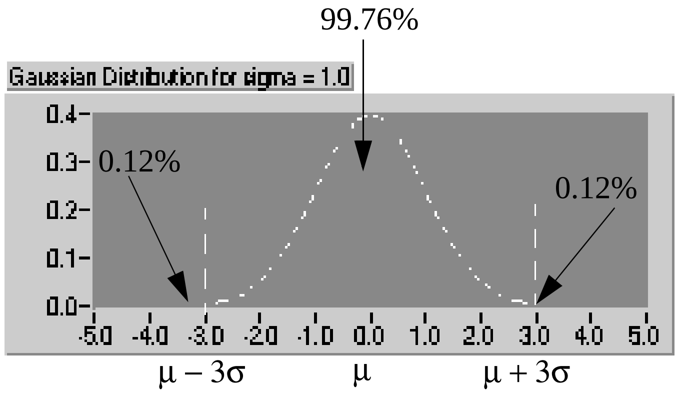
If you have data that is Normally distributed, you will find that the standard deviation is an important parameter in determining the limits within which a certain percentage of your data values are expected to occur. For example,

1. About two-thirds of the values will lie between μ-σ and μ+σ.
2. About 95% of the values will lie between μ-2σ and μ+2σ.

3. About 99.75% of the values will lie between μ-3σ and μ+3σ. Thus, you see that almost all the data values lie between μ-3σ and μ+3σ. This is illustrated in the figures below:







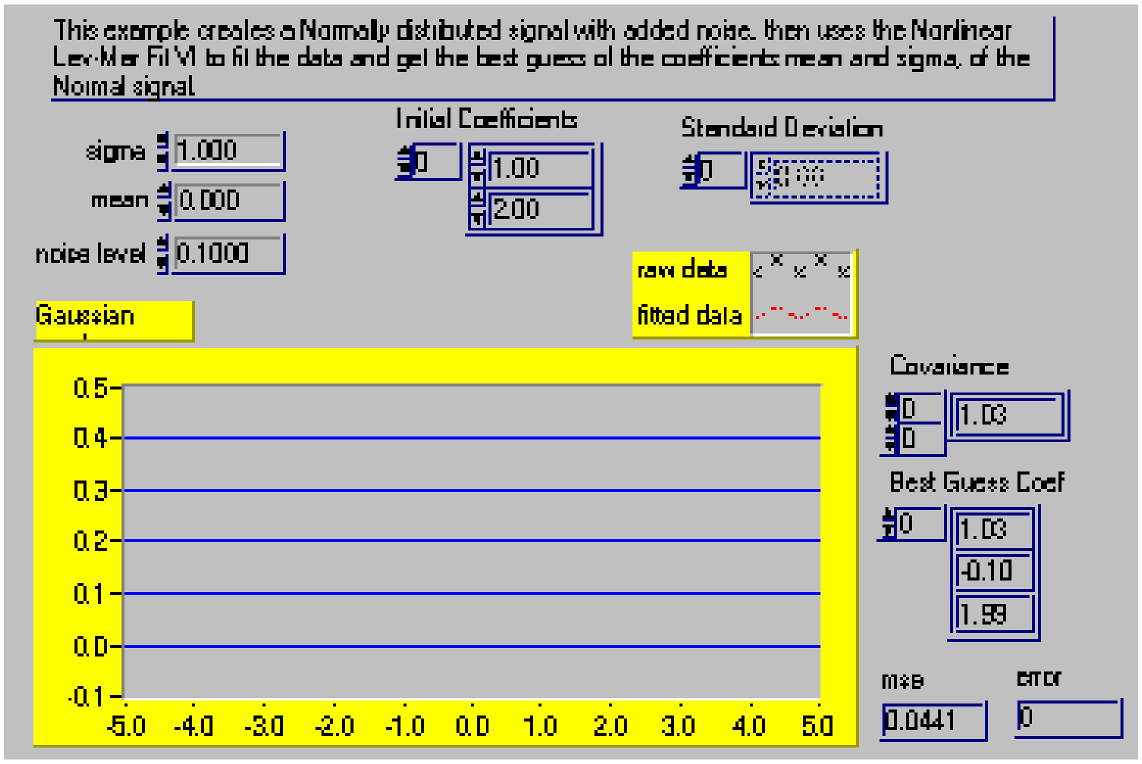
Notice that the two parameters that completely describe Gaussian data are the *mean* and the *standard deviation* of the data. If you believe your data has a Gaussian distribution, you could determine its mean and standard deviation. This has numerous applications, such as determining whether:

- The dimensions of products being manufactured (for example, thickness of plates) are to within specified limits.
- The values of components (for example, resistance of resistors) are to within a specified tolerance.

Exercise 7-7

Objective: **To fit a curve to noisy Gaussian data.**

1. Open the **Normal (Gaussian) Fit** VI from the library `Lvspcex.llb`.



This example creates a Normally distributed signal with added noise, then uses the Nonlinear Lev-Mar Fit VI to fit the data and get the best guess of the coefficients mean and sigma, of the Normal signal.

sigma 1.000
mean 0.000
noise level 0.1000

Initial Coefficients 0 1.00 2.00

Standard Deviation 0 0.00

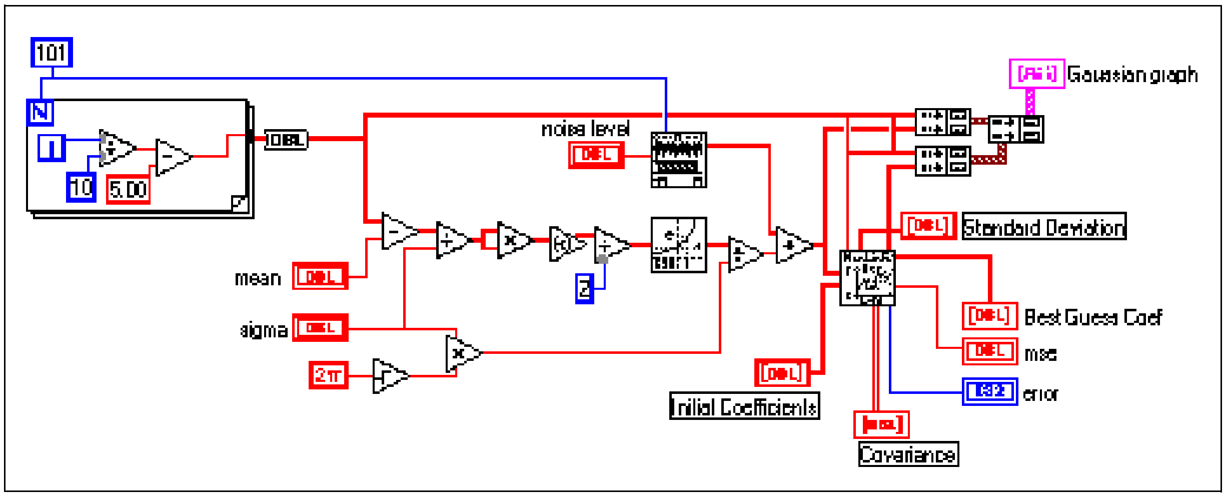
Covariance 0 0 1.03

Best Guess Coef 0 1.03 -0.10 1.99

mse 0.0441
error 0

2. Switch to the block diagram.

Block Diagram

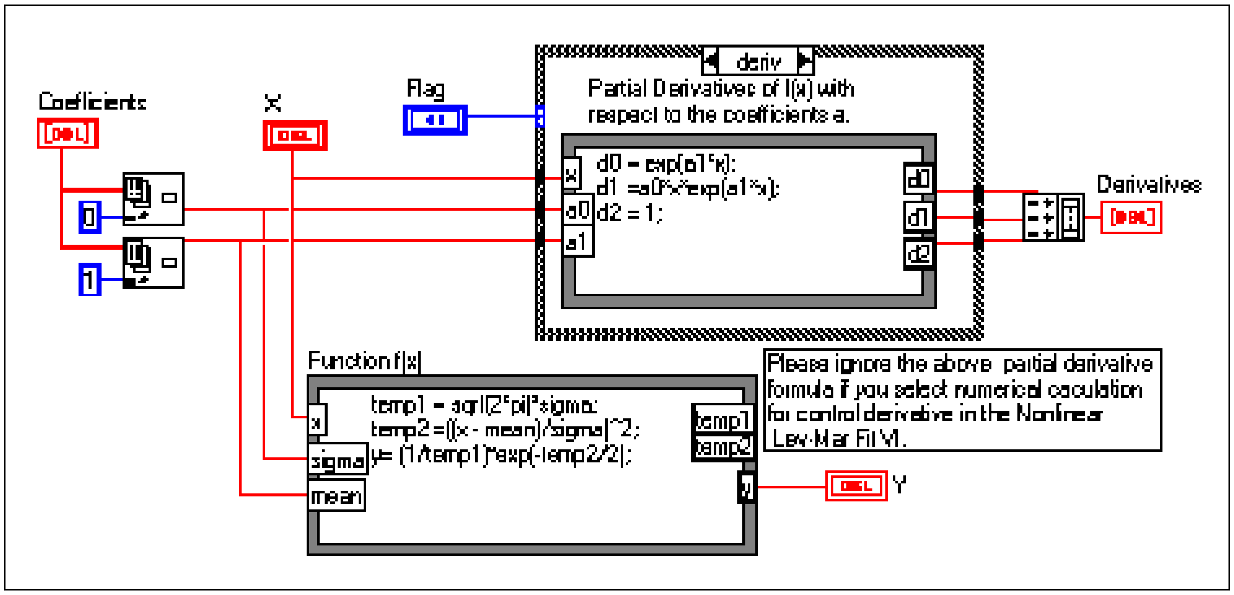


Observe that the parameters *a* and *b* in the **Nonlinear Lev-Mar Exponential Fit** VI of the previous exercise have been replaced by *mean* and *sigma*. Note that the controls for the parameter *c* and the **number of points** have been removed. The For Loop generates the range of the data to lie between -5.0 and +5.0. Most of the additions on the block diagram have to do with implementing equation (3) on page 7-24. Uniform white noise is then added to the Gaussian data generated.

3. Select **Project » Unopened SubVIs » Target Fnc & Deriv Nonlin** VI.

4. Modify the block diagram of the **Target Fnc & Deriv Nonlin** VI as shown below.

Block Diagram


Coefficients
0
1
X
Flag
deriv
Partial Derivatives of f(x) with respect to the coefficients a.
x
a0
a1
d0
d1
d2
Derivatives
Function f(x)
x
sigma
mean
temp1
temp2
y
Y
Please ignore the above partial derivative formula if you select numerical caculation for control derivative in the Nonlinear Lev-Mar Fit VI.

```
d0 = exp(a1*x);
d1 =a0*x*exp(a1*x);
d2 = 1;
```

```
temp1 = sqrt(2*pi)*sigma;
temp2 =((x - mean)/sigma)^2;
y= (1/temp1)*exp(-temp2/2);
```

5. Go back to the **Normal (Gaussian) Fit** VI and enter the following values in the controls on the front panel:

sigma	1.0
mean	0.0
noise level	0.1

6. Clear the **Initial Coefficients** array by popping up on it with the right mouse button and selecting **Data Operations » Empty Array**. Set the values in **Initial Coefficients** to 2.0 and 2.0. These are your guesses of sigma and mean, the actual values of which you chose as 1.0 and 0.0 in the corresponding controls.
7. Run the VI several times and observe the values of sigma and mean in the **Best Guess Coeff** indicator. In the plot, you can also see the spread of the data and the fitted Gaussian curve.
8. Change the value of sigma to 1.0 and run the VI several times. Each time observe the fitted curve, the mse, and the values of the **Best Guess Coeff**.
9. Change Initial Coefficients [0] (the guess for sigma) to 50.0 and run the VI several times. Does the VI obtain a good estimate of sigma in the array **Best Guess Coeff**?
10. Change Initial Coefficients [0] (the guess for sigma) to 500.0 and again run the VI several times. Now how accurate is the estimate?

11. When you finish, save and close the VI.

In this exercise and the previous one, you had to open the **Target Fnc and Deriv NonLin** VI and enter equations in the formula node. This requires a certain amount of work, as well as an understanding of the formula node, on your part. A much simpler and easier method is to have the flexibility of entering the formulas directly on the front panel, without accessing the block diagram. As you will see in a later lesson, this is possible by using the G Math Toolkit, which is a mathematics package (written entirely in G) that is extremely useful for solving differential equations, optimization, and a wide variety of other common math problems.

End of Exercise 7-7

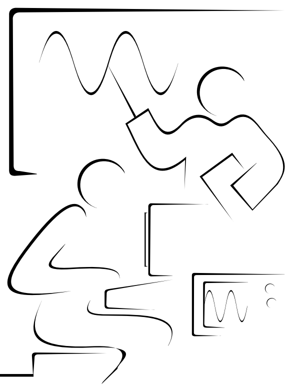
Summary

- Curve fitting is useful in fitting an equation to a set of data points. Some of the practical applications of curve fitting are to remove measurement noise, to fill in missing data points, interpolation, extrapolation, and integration and differentiation of digital data.
- In particular, you have learned how to use the **Linear, Exponential**, **Polynomial**, **General Linear LS Fit**, and **Nonlinear Lev-Mar Fit** VIs to perform several different types of linear and nonlinear fits.
- The MSE is a useful criterion in determining the fit accuracy.

Review Questions

1. Name five applications of curve fitting.
2. Which curve fitting VI would you use to determine the parameters (denoted by the a_i...i integer) of the following models?
   a. $y = a_0 \exp(a_1 x)$
   b. $y = a_0 \exp(a_1 x) + a_2$
   c. $y = a_1 x + a_2 x^2$
   d. $y = a_1 \sin(x) + a_2 \cos(x) = \frac{a_3}{x + 1}$
   e. $t = a_1 y + a_2 x^2$ where x and y are different variables

Notes



Lesson 8 Linear Algebra

Introduction

In this lesson, you will learn the basic theory behind the Linear Algebra VIs in the Analysis library and how these VIs can be used in different applications. Matrix computations, such as matrix-matrix multiplication and many others discussed throughout this lesson, form a significant component of linear algebra and are very important in analysis. The VIs discussed in this lesson form the basis of different algorithms used in many DSP, control, and measurement applications. Therefore, it is important that you completely understand the theory and different VIs discussed in this lesson.

You Will Learn:

A. About linear systems and matrix analysis.

B. About basic matrix operations and eigenvalue-eigenvector problems.

C. About the inverse of a matrix and solving systems of linear equations.

D. About matrix factorization.

A. Linear Systems and Matrix Analysis

Systems of linear algebraic equations arise in many applications that involve scientific computations such as signal processing, computational fluid dynamics, and others. Such systems may occur naturally or may be the result of approximating differential equations by algebraic equations.

Types of Matrices

Whatever the application, it is always necessary to find an accurate solution for the system of equations in a very efficient way. In matrix-vector notation, such a system of linear algebraic equations has the form $Ax = b$, where A is an $n \times n$ matrix, b is a given vector consisting of n elements, and x is the unknown solution vector to be determined. A matrix is represented by a 2D array of elements. These elements may be real numbers, complex numbers, functions, or operators. The matrix A shown below is an array of m rows and n columns with $m \times n$ elements.

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} \end{bmatrix}$$

Here, $a_{i,j}$ denotes the $(i,j)^{th}$ element located in the i^{th} row and the j^{th} column. In general, such a matrix is called a *rectangular matrix*. When $m = n$, so that the number of rows is equal to the number of columns, it is called a *square matrix*. An $m \times 1$ matrix (m rows and one column) is called a *column vector*. A *row vector* is a $1 \times n$ matrix (1 row and n columns). If all the elements other than the diagonal elements are zero (that is, $a_{i,j} = 0$, $i \neq j$), such a matrix is called a *diagonal matrix*. For example,

$$A = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{bmatrix}$$

is a diagonal matrix. A diagonal matrix with all the diagonal elements equal to one is called an *identity matrix*, also known as *unit matrix*. If all the elements below the main diagonal are zero, then the matrix is known as an *upper triangular matrix*. On the other hand, if all the elements above the main diagonal are zero, then the matrix is known as a *lower triangular matrix*. When all the elements are real numbers, the matrix is referred to as a *real matrix*. On the other hand, when at least one of the elements of the matrix is a complex number, the matrix is referred to as a *complex matrix*. To make things simpler to understand, you will work mainly with real matrices in this lesson. However, for the adventurous, there are also some exercises involving complex matrices.

Determinant of a Matrix

One of the most important attributes of a matrix is its *determinant*. In the simplest case, the determinant of a 2 x 2 matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is given by $ad - bc$. The determinant of a square matrix is formed by taking the determinant of its elements. For example, if

$$A = \begin{bmatrix} 2 & 5 & 3 \\ 6 & 1 & 7 \\ 1 & 6 & 9 \end{bmatrix}$$

the determinant of **A**, denoted by $|A|$, is

$$|A| = \left|\begin{bmatrix} 2 & 5 & 3 \\ 6 & 1 & 7 \\ 1 & 6 & 9 \end{bmatrix}\right| = \left(2\left|\begin{bmatrix} 1 & 7 \\ 6 & 9 \end{bmatrix}\right| - 5\left|\begin{bmatrix} 6 & 7 \\ 1 & 9 \end{bmatrix}\right| + 3\left|\begin{bmatrix} 6 & 1 \\ 1 & 6 \end{bmatrix}\right|\right) = 2(-33) - 5(47) + 3(35)$$

=-196

The determinant tells many important properties of the matrix. For example, if the determinant of the matrix is zero, then the matrix is *singular*. In other words, the above matrix (with nonzero determinant) is *nonsingular*. You will revisit the concept of singularity later in section C, when the lesson discusses the solution of linear equations and matrix inverses.

Transpose of a Matrix

The *transpose* of a real matrix is formed by interchanging its rows and columns. If the matrix **B** represents the transpose of **A**, denoted by $\mathbf{A}^T$, then $b_{j,i}=a_{i,j}$. For the matrix **A** defined above,

$$B = A^T = \begin{bmatrix} 2 & 6 & 1 \\ 5 & 1 & 6 \\ 3 & 7 & 9 \end{bmatrix}$$

In case of complex matrices, complex conjugate transposition is defined. If the matrix **D** represents the *complex conjugate transpose*[1] of a complex matrix **C**, then

$$D = C^H \Rightarrow d_{i,j} = c^*_{j,i}$$

That is, the matrix **D** is obtained by replacing every element in **C** by its complex conjugate and then interchanging the rows and columns of the resulting matrix.

1. Complex Conjugate: if $a = x + iy$, complex conjugate $a^* = x - iy$

A real matrix is called a *symmetric matrix* if the transpose of the matrix is equal to the matrix itself. The example matrix **A** is not a symmetric matrix. If a complex matrix **C** satisfies the relation $C = C^H$, then **C** is called a *Hermitian matrix*.

Obtaining One Vector as a Linear Combination of Other Vectors (Linear Independence)

A set of vectors $x_1, x_2, ..., x_n$ is said to be *linearly dependent* if and only if there exist scalars $\alpha_1, \alpha_2, ..., \alpha_n$, not all zero, such that

$$\alpha_1 x_1 + \alpha_2 x_2 + \dots + \alpha_n x_n = 0$$

In simpler terms, if one of the vectors can be written in terms of a linear combination of the others, then the vectors are said to be linearly dependent.

If the only set of α_i for which the above equation holds is $\alpha_1 = 0$, $\alpha_2 = 0$, ..., $\alpha_n = 0$, the set of vectors $x_1, x_2, ..., x_n$ is said to be *linearly independent*. So, in this case, none of the vectors can be written in terms of a linear combination of the others. Given any set of vectors, the above equation always holds for $\alpha_1 = 0$, $\alpha_2 = 0$, ..., $\alpha_n = 0$. Therefore, to show the linear independence of the set, you must show that $\alpha_1 = 0$, $\alpha_2 = 0$, ..., $\alpha_n = 0$ is the only set of α_i for which the above equation holds.

For example, first consider the vectors

$$x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad y = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

Notice that $\alpha_1 = 0$ and $\alpha_2 = 0$ are the only values for which the relation $\alpha_1 x + \alpha_2 y = 0$ holds true. Hence, these two vectors are linearly independent of each other. Now, consider vectors

$$x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad y = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

Notice that, if $\alpha_1 = -2$ and $\alpha_2 = 1$, then $\alpha_1 x + \alpha_2 y = 0$. Therefore, these two vectors are linearly dependent on each other. You must completely understand this definition of linear independence of vectors to fully appreciate the concept of the *rank* of the matrix as discussed next.

How Can You Determine Linear Independence? (Matrix Rank)

The *rank* of a matrix A, denoted by $\rho(A)$, is the maximum number of linearly independent columns in A. If you look at the example matrix A, you will find that all the columns of A are linearly independent of each other. That is, none of the columns can be obtained by forming a linear combination of the other columns. Hence, the rank of the matrix is 3. Consider one more example matrix, *B*, where

$$B = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 2 & 3 \\ 2 & 0 & 2 \end{bmatrix}$$

This matrix has only two linearly independent columns, because the third column of *B* is linearly dependent on the first two columns. Hence, the rank of this matrix is 2. It can be shown that the number of linearly independent columns of a matrix is equal to the number of independent rows. So, the rank can never be greater than the smaller dimension of the matrix. Consequently, if *A* is an $n \times m$ matrix, then

$$\rho(A) \leq min(n, m)$$

where *min* denotes the minimum of the two numbers. In matrix theory, the rank of a square matrix pertains to the highest order nonsingular matrix that can be formed from it. Remember from the earlier discussion that a matrix is singular if its determinant is zero. So, the rank pertains to the highest order matrix that you can obtain whose determinant is not zero. For example, consider a 4 x 4 matrix

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 1 & 2 \\ 1 & 1 & 0 & 2 \end{bmatrix}$$

For this matrix, $det(B) = 0$, but

$$\left| \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & -1 \\ 1 & 0 & 1 \end{bmatrix} \right| = -1$$

Hence, the rank of *B* is 3. A square matrix has full rank if and only if its determinant is different from zero. Matrix *B* is not a full-rank matrix.

"Magnitude" (Norms) of Matrices

You must develop a notion of the "magnitude" of vectors and matrices to measure errors and sensitivity in solving a linear system of equations. As an example, these linear systems can be obtained from applications in control systems and computational fluid dynamics. In two dimensions, for example, you cannot compare two vectors $\boldsymbol{x} = \begin{bmatrix} x1 & x2 \end{bmatrix}$ and $y = \begin{bmatrix} y1 & y2 \end{bmatrix}$, because you might have $x1 > y1$ but $x2 < y2$. A vector norm is a way to assign a scalar quantity to these vectors so that they can be compared with each other. It is similar to the concept of magnitude, modulus, or absolute value for scalar numbers.

There are several ways to compute the norm of a matrix. These include the *2-norm* (Euclidean norm*)*, the *1-norm*, the *Frobenius norm* (F-norm), and the *Infinity norm* (inf-norm). Each norm has its own physical interpretation. Consider a unit ball containing the origin. The Euclidean norm of a vector is simply the factor by which the ball must be expanded or shrunk to encompass the given vector exactly. This is shown in the figures below:

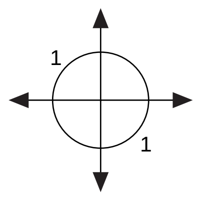


Figure 1a

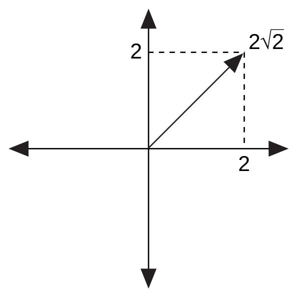


Figure 1b

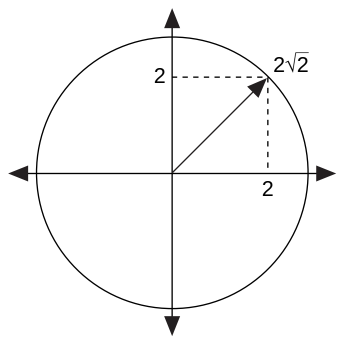


Figure 1c

Figure 1a shows a unit ball of radius = 1 unit. Figure 1b shows a vector of length $\sqrt{2^2 + 2^2} = \sqrt{8} = 2\sqrt{2}$. As shown in figure 1c, the unit ball must be expanded by a factor of $2\sqrt{2}$ before it can exactly encompass the given vector. Hence, the Euclidean norm of the vector is $2\sqrt{2}$.

The norm of a matrix is defined in terms of an underlying vector norm. It is the maximum relative stretching that the matrix does to any vector. With the vector *2-norm*, the unit ball expands by a factor equal to the norm. On the other hand, with the matrix *2-norm*, the unit ball may become an ellipsoidal (ellipse in 3D), with some axes longer than others. The longest axis determines the norm of the matrix.

Some matrix norms are much easier to compute than others. The *1-norm* is obtained by finding the sum of the absolute value of all the elements in each column of the matrix. The largest of these sums is called the 1-norm. In mathematical terms, the 1-norm is simply the maximum absolute column sum of the matrix.

$$\|\boldsymbol{A}\|_1 = max_j \sum_{i=0}^{n-1} |a_{i,j}|$$

For example, $A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$

then $\|A\|_1 = max(3, 7) = 7$. The *inf-norm* of a matrix is the maximum absolute row sum of the matrix

$$\|\boldsymbol{A}\|_\infty = max_i \sum_{i=0}^{n-1} |a_{i,j}|$$

In this case, you add the magnitudes of all elements in each row of the matrix. The maximum value that you get is called the inf-norm. For the above example matrix, $\|\boldsymbol{A}\|_\infty = max(4, 6) = 6$.

The *2-norm* is the most difficult to compute because it is given by the largest singular value of the matrix. Singular values are discussed in Section D, and Exercise 8-9 verifies the validity of the above statement.

Determining Singularity (Condition Number)

Whereas the norm of the matrix provides a way to measure the magnitude of the matrix, the *condition number* of a matrix is a measure of how close the matrix is to being singular. The condition number of a square nonsingular matrix is defined as

$$cond(\boldsymbol{A}) = \|\boldsymbol{A}\|_p \cdot \|A^{-1}\|_p$$

where *p* can be one of the four norm types discussed above. For example, to find the condition number of a matrix *A*, you can find the 2-norm of *A*, the 2-norm of the inverse[1] of the matrix *A*, denoted by A^{-1}, and then multiply them together. As mentioned earlier, the 2-norm is difficult to calculate on paper. You can use the **Matrix Norm** VI from the Analysis library to compute the 2-norm. For example,

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, A^{-1} = \begin{bmatrix} -2 & 1 \\ 1.5 & -0.5 \end{bmatrix}, \|A\|_2 = 5.4650, \|A^{-1}\|_2 = 2.7325, cond(A) = 14.9331$$

1. The inverse of a square matrix A is a square matrix B such that *AB=I*, where I is the identity matrix. Matrix inverses and their applications are described in more detail later in section C.

The condition number can vary between 1 and infinity. A matrix with a large condition number is nearly singular, while a matrix with a condition number close to 1 is far from being singular. The matrix *A* above is nonsingular. However, consider the matrix $B = \begin{bmatrix} 1 & 0.99 \\ 1.99 & 2 \end{bmatrix}$.
The condition number of this matrix is 47168, and hence the matrix is close to being singular. As you might recall, a matrix is singular if its determinant is equal to zero. However, the determinant is not a good indicator for assessing how close a matrix is to being singular. For the matrix *B* above, the determinant (0.0299) is nonzero; however, the large condition number indicates that the matrix is close to being singular. Remember that the condition number of a matrix is always greater than or equal to one; the latter being true for identity and permutation matrices[1]. The condition number is a very useful quantity in assessing the accuracy of solutions to linear systems.

In this section, you have become familiar with some basic notation and fundamental matrix concepts such as determinant of a matrix and its rank. The following exercise should help you further understand these terms, which will be used frequently throughout the rest of the lesson.

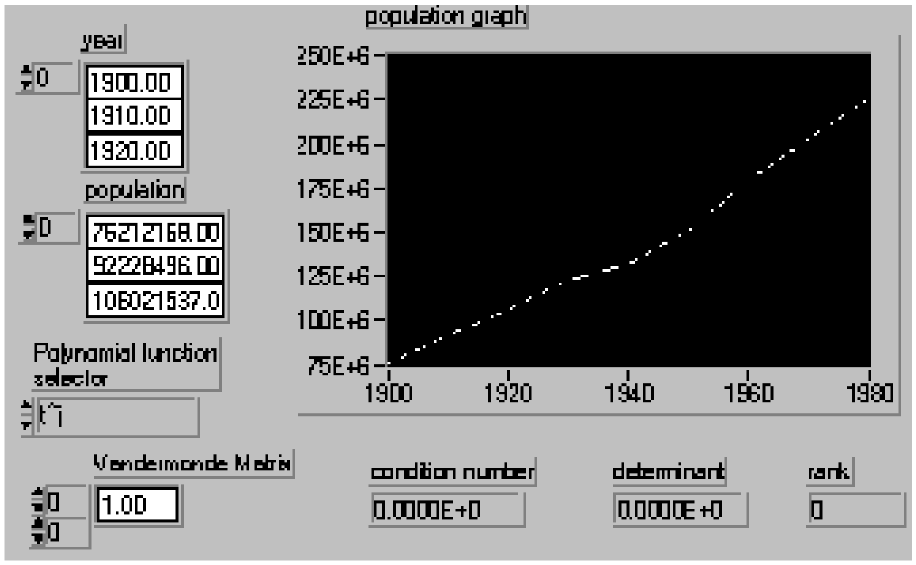
1. A *permutation matrix* is an identity matrix with some rows and columns exchanged.

Exercise 8-1

Objective: To construct matrices, compute condition number, rank, and determinant, and observe how the condition number affects the accuracy of the solution.

In this exercise, you will complete a VI that constructs four different matrices of size 9×9. You will determine the condition number, rank, and determinant of these matrices. You will then solve a curve fitting problem and observe how the condition number of these matrices affects the accuracy of the final solution.

1. Open the **Construct Matrices** VI from `Lvspcex.llb`.



year: 0 — 1900.00, 1910.00, 1920.00

population: 0 — 76212168.00, 92228496.00, 106021537.0

Polynomial function selector: t^j

Vandermonde Matrix: 0, 0 — 1.00

condition number: 0.0000E+0

determinant: 0.0000E+0

rank: 0

2. The array **population** (nine elements) contains the population data for the United States for the years 1900 to 1980, at intervals of 10 years. This data has been plotted for you in the graph labeled **population graph**. To find the population in any one of the intermediate years, you need to interpolate between these nine data points. This interpolation can be achieved by first fitting a curve to these data points and then using the curve to obtain the intermediate values. This curve is represented by a unique polynomial of degree eight that interpolates these nine data points, but that polynomial can be represented in many different ways. Consider the following four ways to represent the individual terms in the polynomial:

   (i) $b_j(t) = t^j$

   (ii) $b_j(t) = (t - 1900)^j$

   (iii) $b_j(t) = (t - 1940)^j$

   (iv) $b_j(t) = ((t - 1940)/40)^j$

For example, the actual polynomial in case (i) is given by

$$y = a_0 + a_1 t + a_2 t^2 + \ldots$$

where the a_i are the curve parameters to be determined.

To determine these parameters, it is necessary to solve a linear system of equations $va = y$, where v is a matrix, $a = [a_0, a_1, \ldots]$, and y is the population vector.

For each of these four representations, you can generate the matrix V, where the (i,j)th element of this matrix is given by

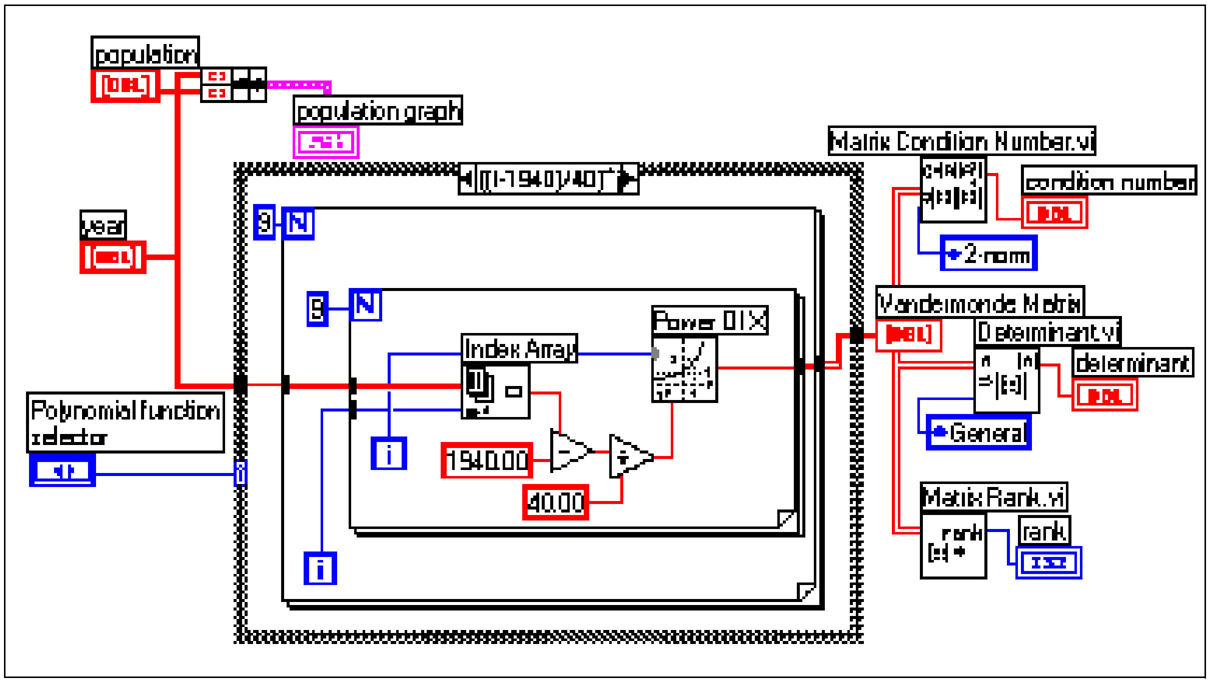
$$v_{i,j} = b_j(t[i])$$

Such a matrix V is referred to as the *Vandermonde matrix*. For example, if you are using the polynomial representation (i) above, the Vandermonde matrix will look like

$$\boldsymbol{V} = \begin{bmatrix} 1 & t_0 & t_0^2 & \ldots & t_0^{n-1} \\ 1 & t_1 & t_1^2 & \ldots & t_1^{n-1} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & t_{n-1} & t_{n-1}^2 & \ldots & t_{n-1}^{n-1} \end{bmatrix}$$

where t_i (0 ð i ð n-1) is the ith element of the vector “year.”

Block Diagram



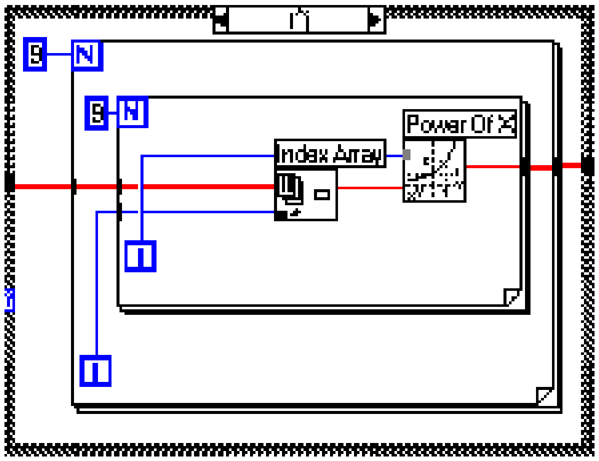
3. Open the block diagram for this VI and complete it as shown above.

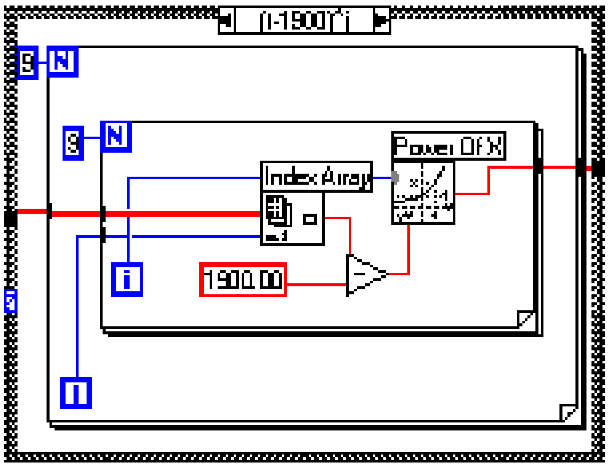
Matrix Condition Number VI (**Analysis » Linear Algebra » Advanced Linear Algebra** subpalette). In this exercise, this function computes the matrix condition number.

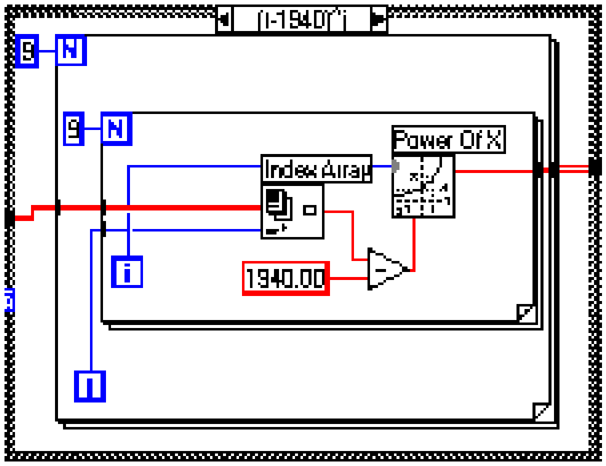
Determinant VI (**Analysis » Linear Algebra** subpalette). In this exercise, this function computes the matrix determinant.

Matrix Rank VI (**Analysis » Linear Algebra » Advanced Linear Algebra** subpalette). In this exercise, this function computes the matrix rank.

Bundle function (**Cluster** subpalette) In this exercise, this function assembles the population array and the year array to plot the population array against the year array.

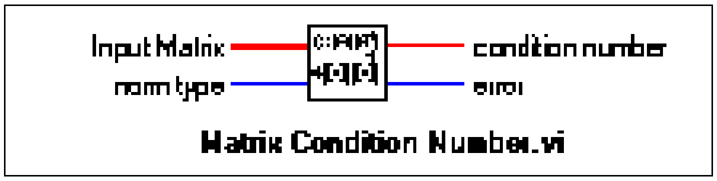




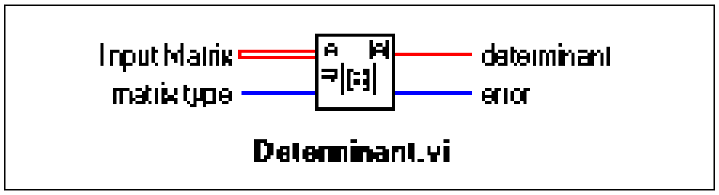


4. The subdiagrams in Case *t^j*, *(t-1900)^j*, *(t-1940)^j*, and *((t-1940)/40)^j* construct the matrices corresponding to the polynomial functions *(i)*, *(ii)*, *(iii)*, and *(iv)*, respectively. The subdiagrams in the first three cases, as shown above, are already

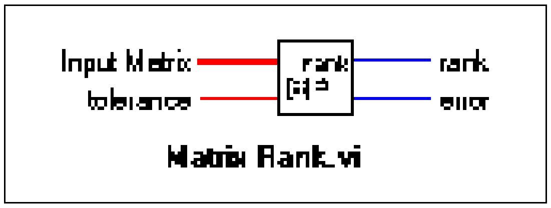
built for you. You need only complete the subdiagram in Case *((t-1940)/40)^j*.



5. The **Matrix Condition Number** VI has an input called **norm type**. Earlier in the section, you looked at different types of norms. This VI can compute the condition number using four different norm types. In this exercise, you will set **norm type** = 0, which is the 2-norm.



6. The **Determinant** VI has an input called **matrix type**. In this exercise, you will set **matrix type** = 0 (general matrix).



7. The **Matrix Rank** VI has an input called **tolerance**. Leave this terminal unconnected, using the default value for this exercise.
8. Return to the front panel and run the VI. Choose different polynomial functions using the **polynomial function selector** control.
9. Look at the condition numbers of the four matrices. Which is very close to being singular?
10. Save and close this VI.

11. You will now use the matrices (computed in the **Construct Matrices** VI) to calculate the population for any year between 1900 and 1980. To do so, open the **Compute Population** VI from `Lvspcex.llb`. This VI computes a polynomial to interpolate the data values to the population data. It then computes the population for a specified year using the Horner's nested evaluation scheme[1].
12. Return to the front panel for the VI opened in step 10. Set the "choose year" control to 1950. Using each of the four polynomial functions, run the VI to compute the population for this year. The red dot on the **population graph** shows the population value for the year chosen. Which of these values is closest to the true value of 151,325,798, according to the 1950 census?
13. Save the VI and close it.

End of Exercise 8-1

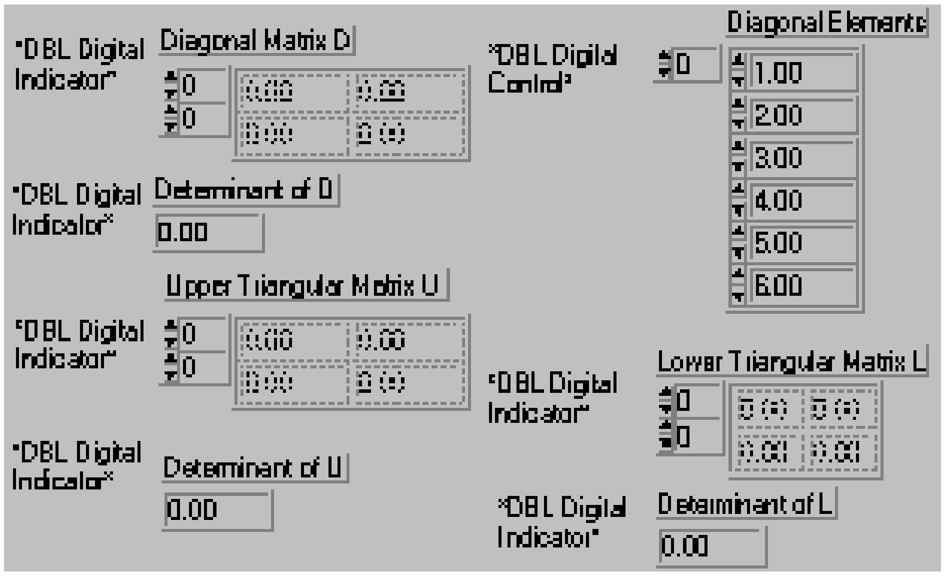
1. For more information on this method, refer to *Scientific Computing, An Introduction Survey* by M.T. Heath, McGraw-Hill, 1997.

Exercise 8-2

Objective: To study special matrices.

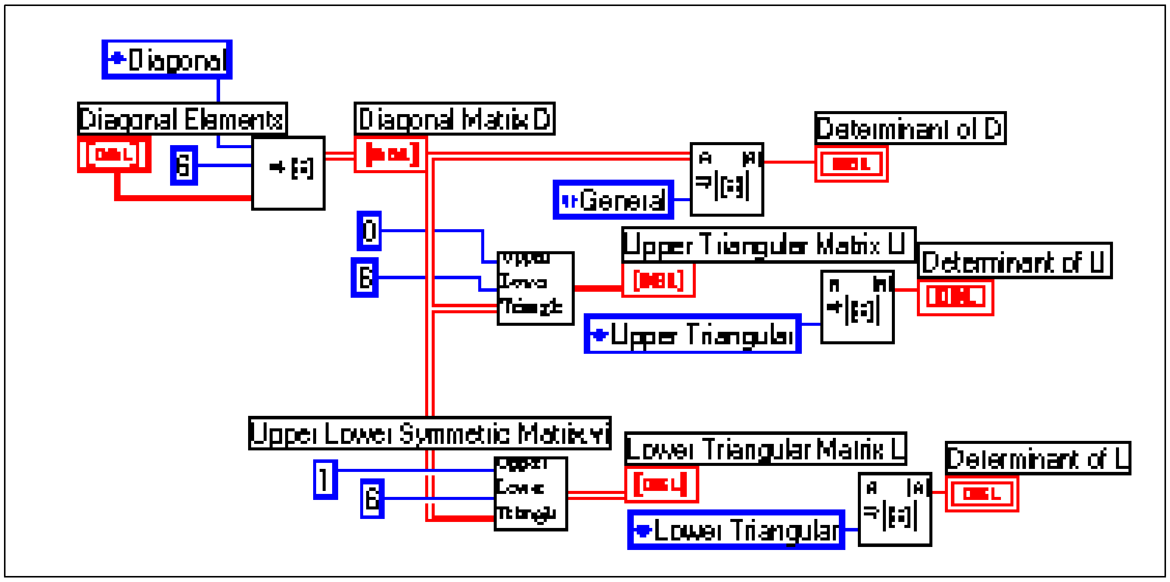
In this exercise, you will learn to use the **Create Special Matrix** VI in the Analysis library. Examine the different types of special matrices that this VI creates. Note that this VI also generates the Vandermonde matrix used in the previous exercise.

Front Panel

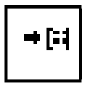


1. Build the front panel as shown above. You can resize the two-dimensional matrices D, U, and L to see all the elements in the matrix.

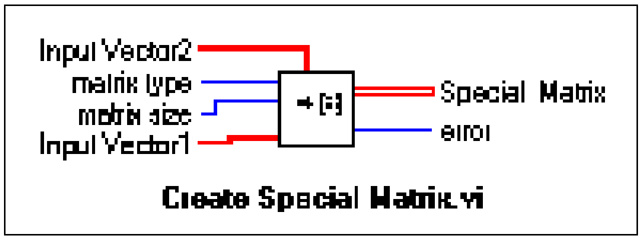
Block Diagram



2. Build the block diagram as shown above.



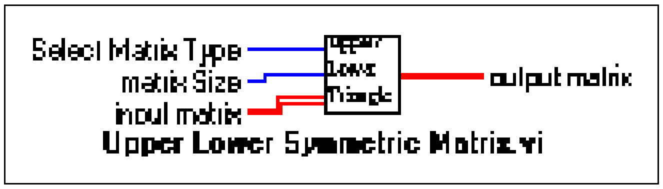
Create Special Matrix VI (**Analysis » Linear Algebra » Advanced Linear Algebra** subpalette). In this exercise, this function creates special matrices. You will complete it in steps 4, 5, and 6.



3. Construct **Diagonal Matrix D** using the **Create Special Matrix** VI from the **Analysis » Linear Algebra » Advanced Linear Algebra** subpalette. The **matrix size** input determines the dimension size of the output **Special Matrix**. Choose **matrix size** = 6.
4. The **matrix type** input determines the type of matrix that is generated at the output **Special Matrix**. Choose **matrix type** = Diagonal.
5. Enter the diagonal elements 1, 2, 3, 4, 5, and 6 in the **Diagonal Elements** control on the front panel. Connect this control to the **Input Vector1** terminal. The **Input Vector1** terminal is the input to construct a special matrix depending on the **matrix type** chosen.
6. Return to the front panel and run the VI.
7. Compute the determinant of this matrix as you did in the earlier exercise. Do you find anything interesting about this determinant

value? If yes, what is it? If no, take a close look at the diagonal elements of the matrix *D*.

8. Select the **Upper Lower Symmetric Matrix** VI from `Lvspcex.llb`. Set **Select Matrix Type** = 0 (upper triangular matrix). Set **matrix Size** = 6. Wire the matrix *D* to the **input matrix** terminal. The **output matrix** *U* is an upper triangular matrix.



9. Compute the determinant of this matrix. Do you find anything interesting about this determinant value? If yes, what is it?
10. Choose **matrix type** = 1 (lower triangular matrix). Set **matrix Size** equal to 6. Wire the matrix *D* to the **input matrix** terminal. The **output matrix** *L* is a lower triangular matrix. Repeat step 6.
11. Save the VI as **Special Matrix.vi** and close it.

Note: ***The determinant for all the three matrices is equal to the product of the diagonal elements of the matrices.***

End of Exercise 8-2

B. Basic Matrix Operations and Eigenvalues-Eigenvector Problems

In this section, consider some very basic matrix operations. Two matrices, *A* and *B*, are said to be equal if they have the same number of rows and columns and their corresponding elements are all equal. Multiplication of a matrix *A* by a scalar α is equal to multiplication of all its elements by the scalar. That is,

$$\boldsymbol{C} = \alpha\boldsymbol{A} \Rightarrow c_{i,j} = \alpha a_{i,j}$$

For example,

$$2\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

Two (or more) matrices can be added or subtracted if and only if they have the same number of rows and columns. If both matrices *A* and *B* have *m* rows and *n* columns, then their sum *C* is an *m*-by-*n* matrix defined as $C = A \pm B$, where $c_{i,j} = a_{i,j} \pm b_{i,j}$. For example,

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 2 & 4 \\ 5 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 8 & 5 \end{bmatrix}$$

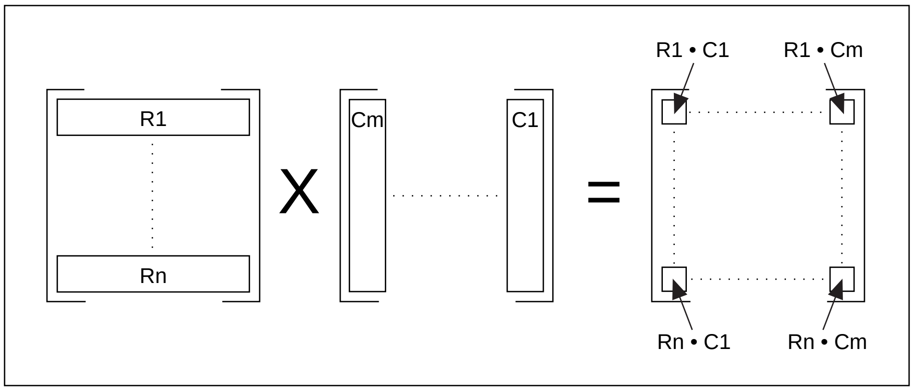
For multiplication of two matrices, the number of columns of the first matrix must be equal to the number of rows of the second matrix. If matrix *A* has *m* rows and *n* columns and matrix *B* has *n* rows and *p* columns, then their product *C* is an *m*-by-*p* matrix defined as $C = AB$, where

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$$

For example,

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 2 & 4 \\ 5 & 1 \end{bmatrix} = \begin{bmatrix} 12 & 6 \\ 26 & 16 \end{bmatrix}$$

So, you multiply the elements of the first row of *A* by the corresponding elements of the first column of *B* and add all the results to get the elements in the first row and first column of *C*. Similarly, to calculate the element in the ith row and the jth column of *C*, multiply the elements in the ith row of *A* by the corresponding elements in the jth column of *C*, and then add them all. This is shown pictorially as:



Matrix multiplication, in general, is not commutative. That is, $AB \neq BA$. Also, remember that multiplication of a matrix by an identity matrix results in the original matrix.

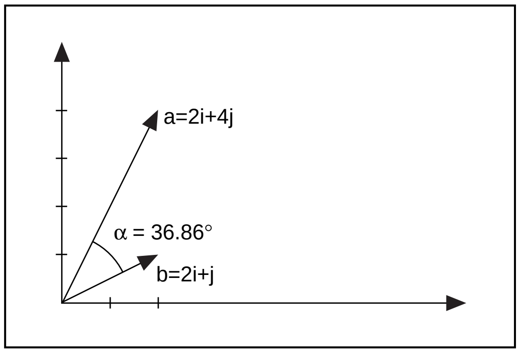
Dot Product and Outer Product

If *X* represents a vector and *Y* represents another vector, then the *dot product* of these two vectors is obtained by multiplying the corresponding elements of each vector and adding the results. This is denoted by

$$X \bullet Y = \sum_{i=0}^{n-1} x_i y_i$$

where *n* is the number of elements in *X* and *Y*. Note that both vectors must have the same number of elements. The dot product is a scalar quantity, and has many practical applications.

For example, consider the vectors $a = 2i + 4j$ and $b = 2i + j$ in a two-dimensional rectangular coordinate system.



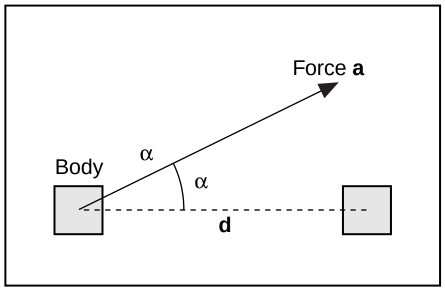
The dot product of these two vectors is given by

$$d = \begin{bmatrix}2\\4\end{bmatrix} \bullet \begin{bmatrix}2\\1\end{bmatrix} = (2\times2) + (4\times1) = 8$$

The angle α between these two vectors is given by

$$\alpha = inv\cos\left(\frac{a \bullet b}{|a||b|}\right) = inv\cos\left(\frac{8}{10}\right) = 36.86^{o},$$

where $|a|$ denotes the magnitude of a.



As a second application, consider a body on which a constant force *a* acts. The work W done by *a* in displacing the body is defined as the product of |*d*| and the component of *a* in the direction of displacement *d*. That is,

$$W = |a||d|\cos\alpha = a \bullet d$$

On the other hand, the *outer product* of these two vectors is a matrix. The *(i,j)*th element of this matrix is obtained using the formula

$$a_{i,j} = x_i \times y_j$$

For example, $\begin{bmatrix}1\\2\end{bmatrix} \times \begin{bmatrix}3\\4\end{bmatrix} = \begin{bmatrix}3 & 4\\6 & 8\end{bmatrix}$

Eigenvalues and Eigenvectors

To understand eigenvalues and eigenvectors, start with the classical definition. Given an $n \times n$ matrix *A*, the problem is to find a scalar λ and a nonzero vector *x* such that

$$Ax = \lambda x$$

Such a scalar λ is called an *eigenvalue*, and *x* is a corresponding *eigenvector*.

Calculating the eigenvalues and eigenvectors are fundamental principles of linear algebra and allow you to solve many problems such as systems of differential equations when you understand what they represent. Consider an eigenvector *x* of a matrix *A* as a nonzero vector that does not rotate when *x* is multiplied by *A* (except perhaps to point in precisely the opposite direction). *x* may change length or reverse its direction, but it will not turn sideways. In other words, there is some scalar constant λ such that the above equation holds true. The value λ is an *eigenvalue* of *A*.

Consider the following example. One of the eigenvectors of the matrix *A*, where $A = \begin{bmatrix} 2 & 3 \\ 3 & 5 \end{bmatrix}$, is $x = \begin{bmatrix} 0.62 \\ 1.00 \end{bmatrix}$. Multiplying the matrix *A* and the vector *x* simply causes the vector *x* to be expanded by a factor of 6.85. Hence, the value 6.85 is one of the eigenvalues of the matrix *A*. For any constant α, the vector αx is also an eigenvector with eigenvalue λ, because

$$A(\alpha x) = \alpha A x = \lambda \alpha x$$

In other words, an eigenvector of a matrix determines a direction in which the matrix expands or shrinks any vector lying in that direction by a scalar multiple, and the expansion or contraction factor is given by the corresponding eigenvalue. A *generalized* eigenvalue problem is to find a scalar λ and a nonzero vector *x* such that

$$Ax = \lambda Bx$$

where *B* is another $n \times n$ matrix.

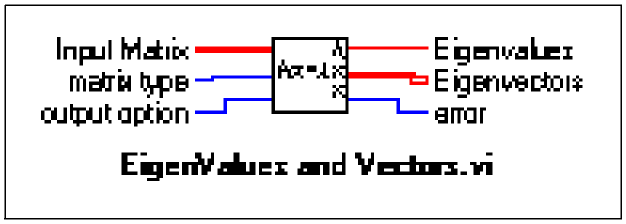
The following are some important properties of eigenvalues and eigenvectors:

- The eigenvalues of a matrix are not necessarily all distinct. In other words, a matrix can have multiple eigenvalues.
- All the eigenvalues of a real matrix need not be real. However, complex eigenvalues of a real matrix must occur in complex conjugate pairs.

- The eigenvalues of a diagonal matrix are its diagonal entries, and the eigenvectors are the corresponding columns of an identity matrix of the same dimension.
- A real symmetric matrix always has real eigenvalues and eigenvectors.
- As discussed earlier, eigenvectors can be scaled arbitrarily.

There are many practical applications in the field of science and engineering for an eigenvalue problem. For example, the stability of a structure and its natural modes and frequencies of vibration are determined by the eigenvalues and eigenvectors of an appropriate matrix. Eigenvalues are also very useful in analyzing numerical methods, such as convergence analysis of iterative methods for solving systems of algebraic equations, and the stability analysis of methods for solving systems of differential equations.

The LabVIEW/BridgeVIEW **EigenValues and Vectors** VI is shown below. The **Input Matrix** is an N-by-N real square matrix. **Matrix type** determines the type of the input matrix. **Matrix type** could be *0*, indicating a *general matrix*[1], or 1, indicating a *symmetric matrix*. A symmetric matrix always has real eigenvalues and eigenvectors.



Output option determines what needs to be computed. Output option = 0 indicates that only the eigenvalues need to be computed. Output option = 1 indicates that both the eigenvalues and the eigenvectors should be computed. It is computationally very expensive to compute both the eigenvalues and the eigenvectors. So, it is important that you use the output option control in the **EigenValues and Vectors** VI very carefully. Depending on your particular application, you might just want to compute the eigenvalues or both the eigenvalues and the eigenvectors. Also, a symmetric matrix needs less computation than an unsymmetric matrix. So, choose the matrix type control carefully.

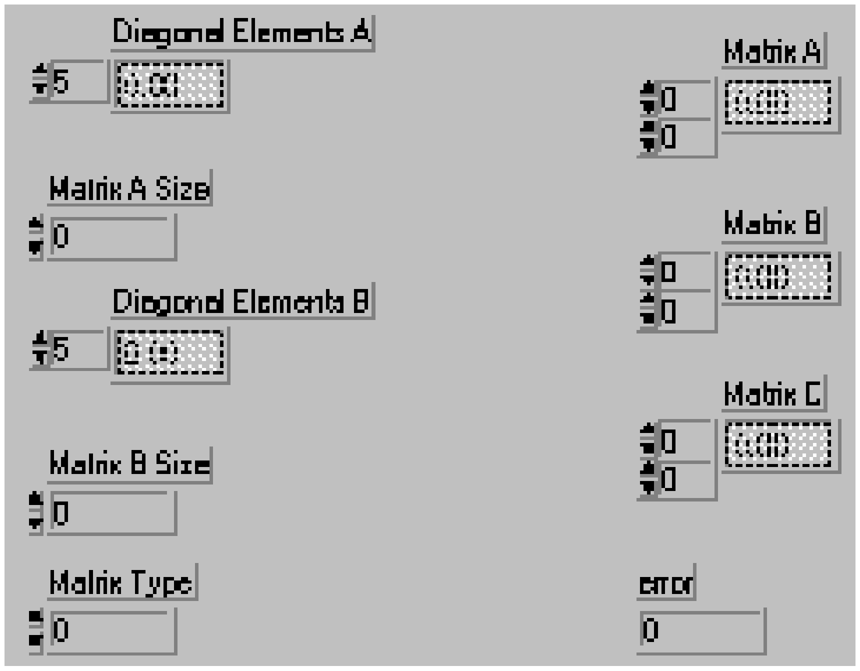
In this section, you learned about some basic matrix operations and the eigenvalues-eigenvectors problem. The next example introduces some VIs in the Analysis library that perform these operations.

1. General Matrix: A matrix with no special property such as symmetry or triangular structure.

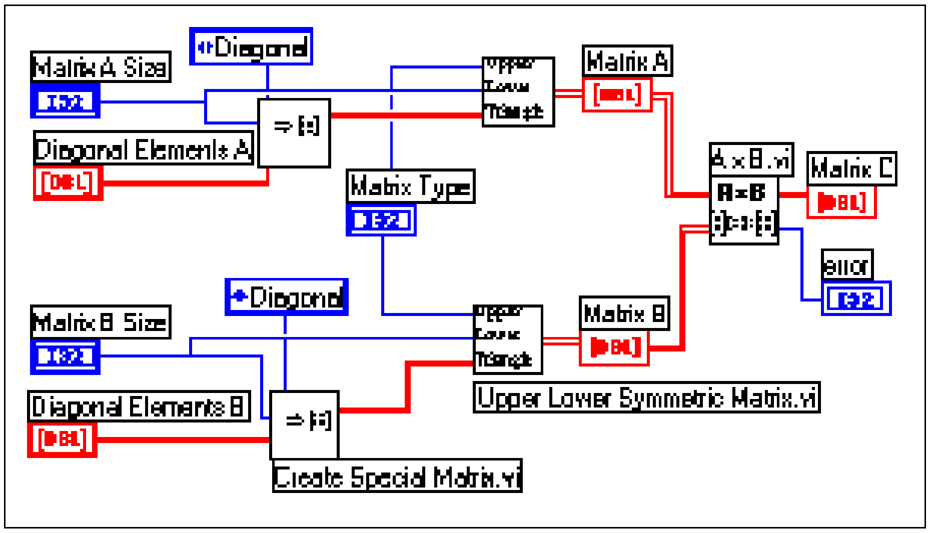
Exercise 8-3

Objective: To learn basic matrix manipulations

You will build a VI that will help you further understand the basic matrix manipulations discussed in the first part of the previous section. You will also learn some very interesting matrix properties.



1. Build the front panel as shown above. You can resize the two-dimensional matrices *A*, *B*, and *C* to see all the elements in the matrix.
2. In this exercise, you will experiment with three different types of matrices, namely upper triangular, lower triangular, and symmetric matrix. You will use the **Upper Lower Symmetric Matrix** VI that you used in the previous exercise.



3. Build the block diagram as shown above.

Create Special Matrix VI (**Analysis » Linear Algebra » Advanced Linear Algebra** subpalette). In this exercise, this function creates special matrices.

AxB VI (**Analysis » Linear Algebra** subpalette) In this exercise, this function multiplies two matrices).

4. Create a diagonal matrix using the **Create Special Matrix** VI. Select the **Upper Lower Symmetric Matrix** VI from the library `Lvspcex.llb`. This VI will create the appropriate type of matrix depending on the **Matrix Type** control.
5. Select the **A x B** VI from the **Analysis » Linear Algebra** subpalette. You will use this VI to multiply matrix *A* and matrix *B* and the result of this multiplication is stored in matrix *C*.
6. Return to the front panel. Choose **Matrix A Size** = 6 and **Matrix B Size** = 6. Set the **Diagonal Elements A** and **Diagonal Elements B** controls for both the matrices to some value (similar to what you did in the previous exercise). The size of this array is equal to the matrix size.
7. Choose **Matrix Type** = 0 (Upper Triangular Matrix). Run the VI. Observe the structure of the upper triangular matrices *A* and *B*. **Matrix *C*** is the product of these two matrices. Do you find anything interesting about the structure of this matrix?
8. Now choose **Matrix Type** = 1 (Lower Triangular Matrix). Run the VI. Observe the structure of the two lower triangular matrices *A*

and *B*. **Matrix *C*** is the product of these two matrices. Do you find anything interesting about the structure of this matrix?

9. Now choose **Matrix Type** = 2 (Symmetric Matrix). Run the VI. Observe the structure of the two symmetric matrices *A* and *B*. Matrix *C* is the product of these two matrices. Do you find anything interesting about the structure of this matrix?
10. Change the **Matrix A Size** to 5. Keep the **Matrix B Size** at 6. Run the VI. Why did you get an error[1]?
11. Save the VI as **Matrix Multiplication.vi** and close it.

End of Exercise 8-3

1. Description of Error Code -20039: The number of columns in the first matrix is not equal to the number of rows in the second matrix or vector.

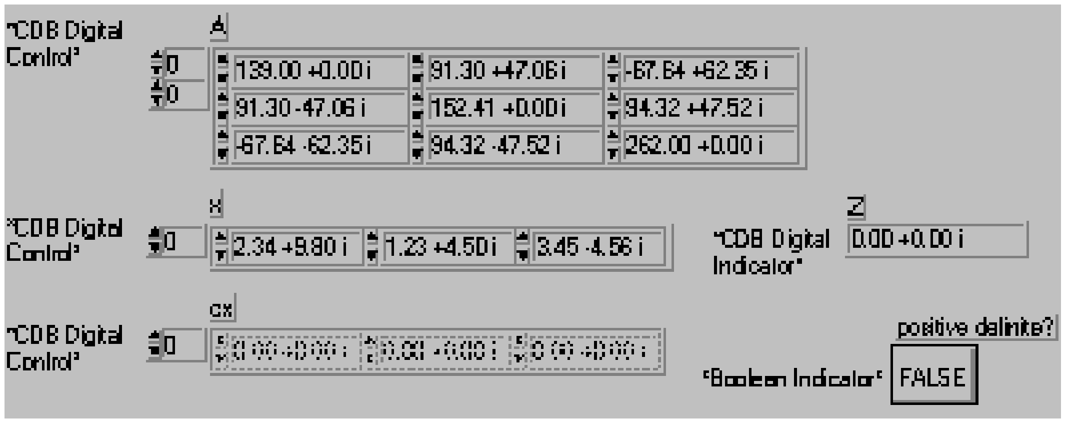
Exercise 8-4

Objective: To study positive definite matrices.

In many applications, it is advantageous to determine if your matrix is *positive definite*. This is because if your matrix is indeed positive definite, you can save a significant amount of computation time when using VIs to compute determinants, solve linear systems of equations, or compute the inverse of a matrix. All these VIs let you choose the matrix type, and properly identifying the matrix can significantly improve performance.

In this exercise, you will learn about complex positive definite matrices. A complex matrix is positive definite if and only if it is Hermitian; that is, $A = A^H$, and the quadratic form $X^H AX > 0$ for all nonzero vectors X.

Front Panel



"CDB Digital Control" A

139.00 +0.00 i	91.30 +47.06 i	-67.64 +62.35 i
91.30 -47.06 i	152.41 +0.00 i	94.32 +47.52 i
-67.64 -62.35 i	94.32 -47.52 i	262.00 +0.00 i

"CDB Digital Control" x: 2.34 +9.80 i, 1.23 +4.50 i, 3.45 -4.56 i

"CDB Digital Indicator" z: 0.00 +0.00 i

"CDB Digital Control" ax: 0.00 +0.00 i, 0.00 +0.00 i, 0.00 +0.00 i

"Boolean Indicator" positive definite?: FALSE

1. Build the matrix *A* and the rest of the front panel as shown in the front panel above.
2. Switch to the block diagram and use the **Test Complex Positive Definite** VI to check if this matrix is positive definite.

Test Complex Positive Definite VI (**Analysis » Linear Algebra » Complex Linear Algebra » Advanced Complex Linear Algebra** subpalette). Use this function to check if the input matrix *A* is positive definite.

3. Enter the vector *X* shown below in the array of controls *x*.

$$x = \begin{bmatrix} 2.34 + 9.8i \\ 1.23 + 4.5i \\ 3.45 - 4.56i \end{bmatrix}$$

4. Enter the complex conjugate *CX* of *X* in the array of controls *cx*. (This part has not been completed in the front panel above.)

   Hint: Complex Conjugate of *a+ib* is *a-ib*

   Complex A x Vector VI (**Analysis » Linear Algebra » Complex Linear Algebra** subpalette). This function multiplies a complex input matrix and a complex input vector.

5. Switch to the block diagram. Compute the complex matrix vector multiplication $Y = AX$.

   Hint: Use the **Complex A x Vector** VI

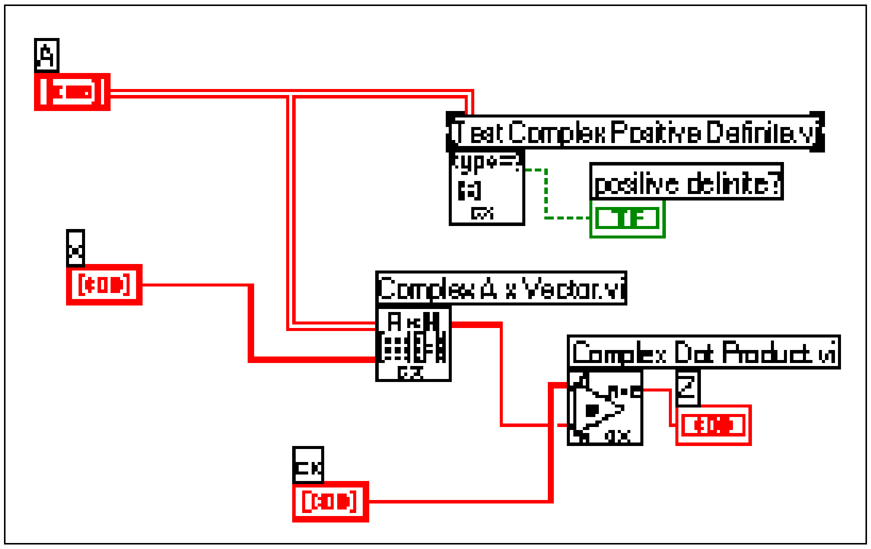
   Complex Dot Product VI (**Analysis » Linear Algebra » Complex Linear Algebra** subpalette) Use this function to compute the dot product of two complex vectors.

6. Compute the complex dot product of the vectors *CX* and *Y*. That is, $Z = CX \bullet Y$.

   Hint: Use the **Complex Dot Product** VI.

7. You have now computed the product $Z = X^H AX$. What is the value of Z? Does this result verify the above definition of complex positive definite matrices?

8. Compare your block diagram with the diagram shown below.



9. Save the VI as **Positive Definite Matrix.vi** and close it.

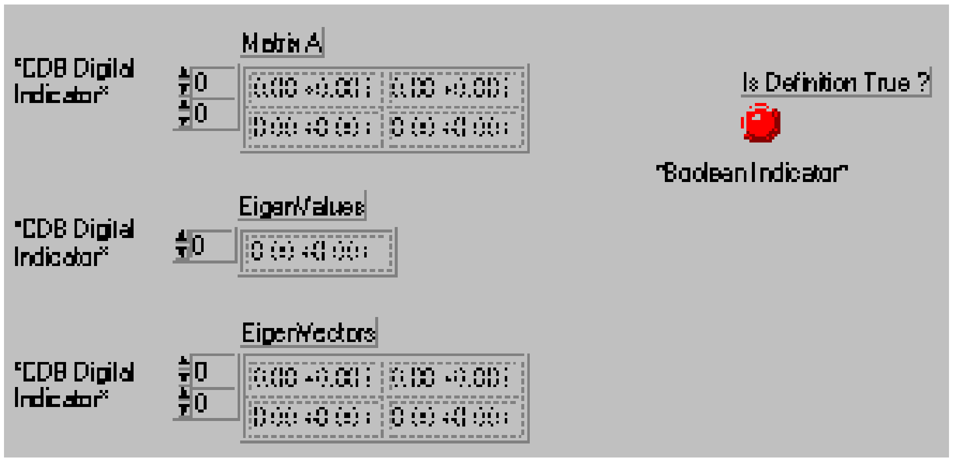
End of Exercise 8-4

Exercise 8-5

Objective: To compute the eigenvalues and eigenvectors of a real matrix.

As explained earlier, the eigenvalues-eigenvector problem is widely used in a number of different practical applications. For example, in the design of control systems, the eigenvalues help determine whether the system is stable or unstable. If all the eigenvalues have nonpositive real parts, the system is stable. However, if any of the eigenvalues have a positive real part, it means that the system is unstable. If the system is unstable, you can design a feedback system to obtain the desired eigenvalues and ensure stability of the overall system.

In this exercise, you will use the **EigenValues and Vectors** VI to compute all the eigenvalues and eigenvectors of a real matrix. You will also learn an alternative definition of eigenvalues and numerically verify this definition.



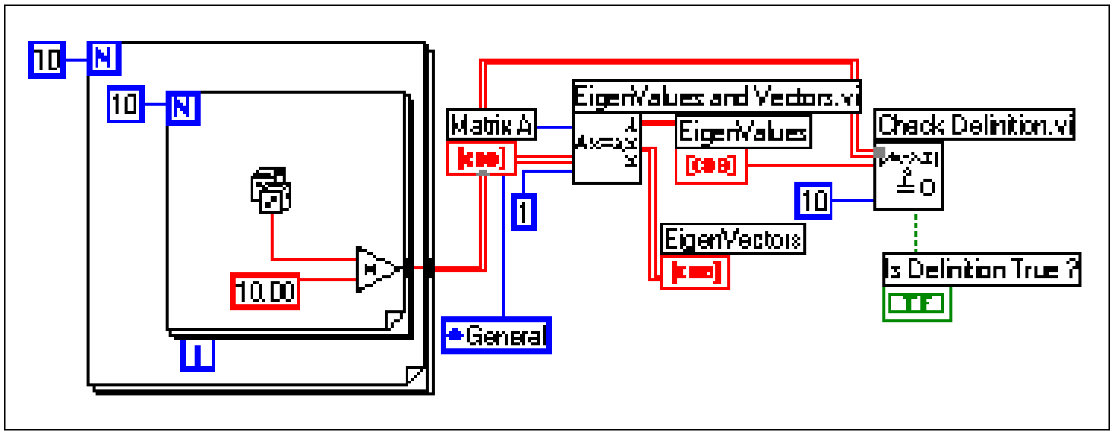
1. Build the front panel as shown above. You can resize the controls to view all the elements in the matrices *A* and *EigenVectors* and the array *EigenValues*. The size of the matrix *A* is 10×10.

   Eigenvalues is a one-dimensional complex array of size 10 containing all the computed eigenvalues of the input matrix.

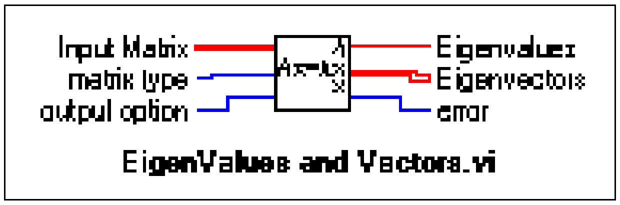
   Eigenvectors is a 10×10 complex matrix containing all the computed eigenvectors of the input matrix. The ith column of *Eigenvectors* is the eigenvector corresponding to the ith component of the *Eigenvalues* vector.

2. Build the block diagram as shown below.

EigenValues and Vectors VI (**Analysis » Linear Algebra** subpalette) In this exercise, you will use this VI to compute the eigenvalues and the eigenvectors of the input matrix.



3. Matrix *A* is a real matrix consisting of randomly generated numbers. Use the **EigenValues and Vectors** VI from the **Analysis » Linear Algebra** subpalette to compute both the eigenvalues and the eigenvectors of this matrix. Remember to set the output option control to 1. Choose matrix type = *General*.



4. Earlier in this section, you looked at the classical definition of eigenvalues and eigenvectors. A different and widely used definition of eigenvalues is as follows. The eigenvalues of *A* are the values λ such that $det(A - \lambda I) = 0$, where *det* stands for the determinant of the matrix. In this exercise, you will numerically verify the validity of this definition. Select the **Check Definition** VI from `Lvspcex.llb`. The round LED on the front panel will glow green if the definition is true and turn to red if the definition fails.

5. Run the VI several times. You will notice that the eigenvalues can be real or complex numbers although the matrix is purely real. The same holds true for the eigenvectors also. Furthermore, you will notice that complex eigenvalues of a real matrix always occur in complex conjugate pairs (that is, if $\alpha + i\beta$ is an eigenvalue of a real matrix, so is $\alpha - i\beta$).
6. Did the definition stated above in step 3 always hold true?
7. Save the VI as **My EigenValues and Vectors.vi** and close it.

End of Exercise 8-5

C. Matrix Inverse and Solving Systems of Linear Equations

The *inverse*, denoted by A^{-1}, of a square matrix *A* is a square matrix such that

$$A^{-1}A = AA^{-1} = I$$

where *I* is the identity matrix. The inverse of a matrix exists if and only if the determinant of the matrix is not zero (that is, it is nonsingular). In general, you can find the inverse of only a square matrix. You can, however, compute the *pseudoinverse* of a rectangular matrix, as discussed later in section D.

Solutions of Systems of Linear Equations

In matrix-vector notation, a system of linear equations has the form $Ax = b$, where *A* is a $n \times n$ matrix and *b* is a given *n*-vector. The aim is to determine *x*, the unknown solution *n*-vector. There are two important questions to be asked about the existence of such a solution. Does such a solution exist, and if it does is it unique? The answer to both of these questions lies in determining the singularity or nonsingularity of the matrix *A*.

As discussed earlier, a matrix is said to be singular if it has any one of the following equivalent properties:

- The inverse of the matrix does not exist.
- The determinant of the matrix is zero.
- The rows (or columns) of *A* are linearly dependent.
- $Az = 0$ for some vector $z \neq 0$.

Otherwise, the matrix is nonsingular. If the matrix is nonsingular, its inverse A^{-1} exists, and the system $Ax = b$ has a unique solution: $x = A^{-1}b$ regardless of the value for *b*. On the other hand, if the matrix is singular, then the number of solutions is determined by the right-side vector *b*. If *A* is singular and $Ax = b$, then $A(x + \Upsilon z) = b$ for any scalar Υ, where the vector z is as in the last definition above. Thus, if a singular system has a solution, then the solution cannot be unique.

It is not a good idea to explicitly compute the inverse of a matrix, because such a computation is prone to numerical inaccuracies. Therefore, it is not a good strategy to solve a linear system of equations by multiplying the inverse of the matrix *A* by the known right-side vector. The general strategy to solve such a system of equations is to transform the original system into one whose solution is the same as that of the original system, but is easier to compute. One way to do so is to use the Gaussian Elimination[1] technique. The three basic steps involved in the Gaussian Elimination technique are as follows. First, express the matrix A as a product $A = LU$ where L is a unit lower triangular matrix and U is an upper triangular matrix. Such a factorization is known as LU factorization. Given this, the linear system $Ax = b$ can be expressed as $LUx = b$. Such a system can then be solved by first solving the lower triangular system $Ly = b$ for y by *forward-substitution*. This is the second step in the Gaussian Elimination technique. For example, if

$$L = \begin{bmatrix} a & 0 \\ b & c \end{bmatrix} \qquad y = \begin{bmatrix} p \\ q \end{bmatrix} \qquad b = \begin{bmatrix} r \\ s \end{bmatrix}$$

then $p = \frac{r}{a}, q = \frac{(s-bp)}{c}$. The first element of y can be easily determined due to the lower triangular nature of the matrix L. Then you can use this value to compute the remaining elements of the unknown vector sequentially. Hence, the name forward-substitution. The final step involves solving the upper triangular system $Ux = y$ by *back-substitution*. For example, if

$$U = \begin{bmatrix} a & b \\ 0 & c \end{bmatrix} \qquad x = \begin{bmatrix} m \\ n \end{bmatrix} \qquad y = \begin{bmatrix} p \\ q \end{bmatrix}$$

then $n = \frac{q}{c}, m = \frac{(p-bn)}{a}$. In this case, this last element of x can be easily determined and then used to determine the other elements sequentially. Hence, the name back-substitution. So far, this lesson has discussed the case of square matrices. Because a nonsquare matrix is necessarily singular, the system of equations must have either no solution or a nonunique solution. In such a situation, you usually find a unique solution x that satisfies the linear system in an approximate sense.

The Analysis library includes VIs for computing the inverse of a matrix, computing LU decomposition of a matrix, and solving a system of linear equations. It is important to identify the input matrix properly, as it helps avoid unnecessary computations, which in turn helps to minimize numerical inaccuracies. The four possible matrix types are general matrices, positive definite matrices[2], and lower and upper triangular matrices. If the input matrix is square, but does not have a full rank (a *rank-deficient matrix*), then the VI finds the *least square* solution x. The least square solution is the one that minimizes the norm of $Ax - b$. The same holds true also for nonsquare matrices.

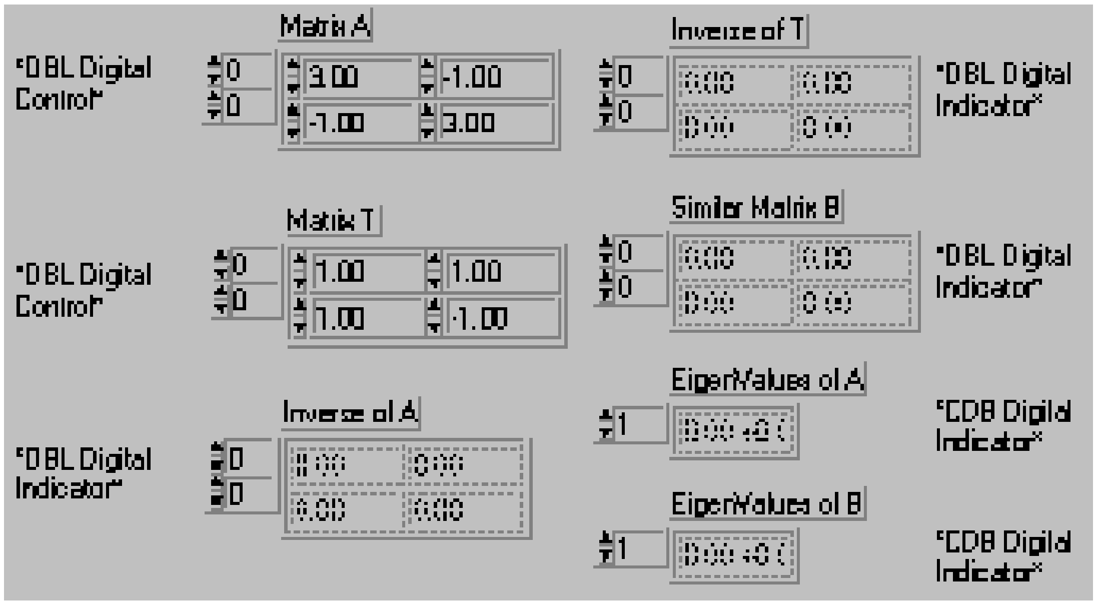
1. For more information on Gaussian Elimination, see *Matrix Computations* by G.H. Golub and C.F. Van Loan. The John Hopkins University Press, Baltimore, 1989.
2. A real matrix is positive definite if and only if it is symmetric and the quadratic form $X^T AX > 0$ for all nonzero vectors X.

Exercise 8-6

Objective: To compute the inverse of a matrix.

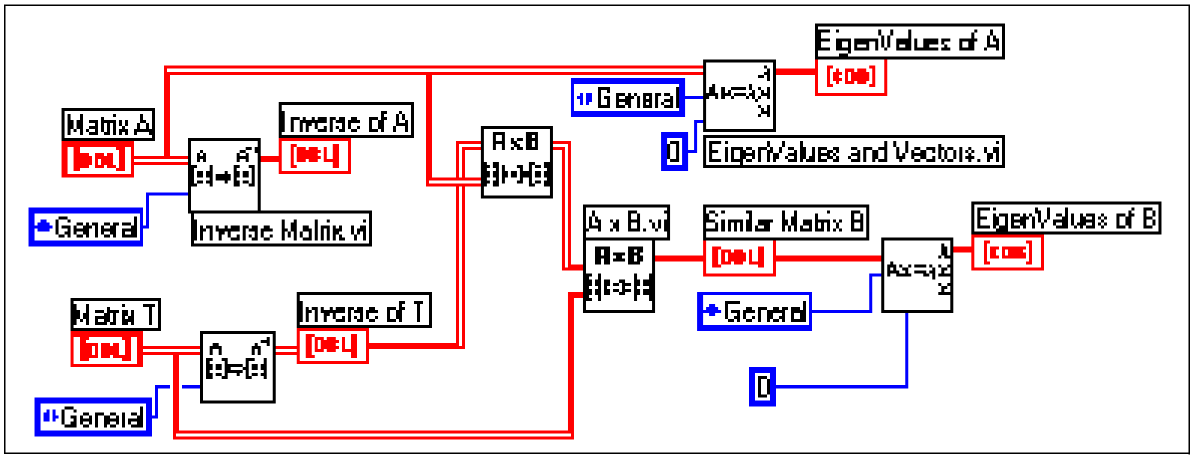
You will build a VI that computes the inverse of a matrix *A*. Further, you will compute a matrix *B*, which is *similar* to matrix *A*. A matrix *B* is similar to a matrix *A* if there is a nonsingular matrix *T* such that $B = T^{-1}AT$ so that *A* and *B* have the same eigenvalues. You will verify this definition of similar matrices.

Front Panel



1. Build the front panel as shown above. Matrix *A* is a 2×2 real matrix. Matrix *T* is a 2×2 nonsingular matrix that will be used to construct the similar matrix *B*.

Block Diagram



2. Construct the block diagram as shown above.

Inverse Matrix VI (**Analysis » Linear Algebra** subpalette). In this exercise, this function computes the inverse of the input matrix *A*.

AxB VI (**Analysis » Linear Algebra** subpalette). In this exercise, this function multiplies two two-dimensional input matrices.

EigenValues and Vectors VI (**Analysis » Linear Algebra** subpalette). In this exercise, this VI computes the eigenvalues and eigenvectors of the input matrix.

3. Return to the front panel and run the VI. Check if the eigenvalues of *A* and the similar matrix *B* are the same.

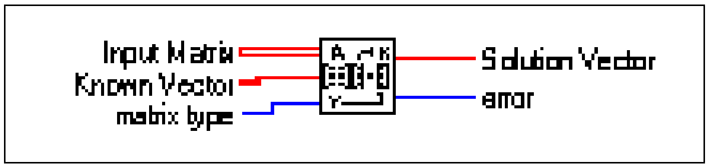
4. Save the VI as **Matrix Inverse.vi** and close it.

End of Exercise 8-6

Exercise 8-7

Objective: To solve a system of linear equations.

Many practical applications require you to solve a system of linear equations. A very important area of application is related to military defense. This includes analysis of electromagnetic scattering and radiation from large targets, performance analysis of large radomes, and design of aerospace vehicles having low radar cross sections (the stealth technology). A second area of application is in the design and modeling of wireless communication systems such as hand-held cellular phones. This list of applications goes on and on, and therefore it is very important for you to properly understand how to use the VIs in the Analysis library to solve a linear system of equations.



1. Use the **Solve Linear Equations** VI in the **Analysis » Linear Algebra** subpalette to solve the system of equations $Ax = b$ where the **Input Matrix** *A* and the **Known Vector** *b* are

$$A = \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -1 & 7 \end{bmatrix}, b = \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix}$$

   Choose matrix type equal to general.

2. Use **A x Vector.vi** to multiply the matrix *A* and the vector *x* (output of the above operation) and check if the result is equal to the vector *b* above.
3. Save the VI as **Linear System.vi** and close it.

End of Exercise 8-7

D. Matrix Factorization

The previous section discussed how a linear system of equations can be transformed into a system whose solution is simpler to compute. The basic idea was to factorize the input matrix into the multiplication of several, simpler matrices. You looked at one such technique, the *LU decomposition* technique, in which you factorized the input matrix as a product of upper and lower triangular matrices. Other commonly used factorization methods are *Cholesky*, *QR*, and the *Singular Value Decomposition (SVD)*. You can use these factorization methods to solve many matrix problems, such as solving linear system of equations, inverting a matrix, and finding the determinant of a matrix.

If the input matrix *A* is symmetric and positive definite, then an LU factorization can be computed such that $A = U^TU$, where *U* is an upper triangular matrix. This is called *Cholesky factorization*. This method requires only about half the work and half the storage compared to LU factorization of a general matrix by Gaussian elimination. As you saw earlier in Exercise 1-4, it is easy to determine if a matrix is positive definite by using the **Test Positive Definite** VI in the Analysis library.

A matrix *Q* is *orthogonal* if its columns are *orthonormal*. That is, if $Q^TQ = I$, the identity matrix. *QR factorization* technique factors a matrix as the product of an orthogonal matrix *Q* and an upper triangular matrix *R*. That is, $A = QR$. QR factorization is useful for both square and rectangular matrices. A number of algorithms are possible for QR factorization, such as the *Householder transformation*, the *Givens transformation* and the *fast Givens transformation*.

The singular value decomposition (SVD) method decomposes a matrix into the product of three matrices: $A = USV^T$. *U* and *V* are orthogonal matrices. *S* is a diagonal matrix whose diagonal values are called the *singular values* of *A*. The singular values of *A* are the nonnegative square roots of the eigenvalues of A^TA, and the columns of *U* and *V*, which are called left and right singular vectors, are orthonormal eigenvectors of AA^T and A^TA, respectively. SVD is useful for solving analysis problems such as computing the rank, norm, condition number, and pseudoinverse of matrices. The following section discusses this last application.

Pseudoinverse

The pseudoinverse[1] of a scalar σ is defined as $1/\sigma$ if $\sigma \neq 0$, and zero otherwise. You can now define the pseudoinverse of a diagonal matrix by transposing the matrix and then taking the scalar pseudoinverse of each

1. In case of scalars, pseudoinverse is the same as the inverse.

entry. Then the pseudoinverse of a general real $m \times n$ matrix *A*, denoted by $A^{\dagger}$, is given by

$$A^{\dagger} = VS^{\dagger}U^{T}$$

Note that the pseudoinverse exists regardless of whether the matrix is square or rectangular. If *A* is square and nonsingular, the pseudoinverse is the same as the usual matrix inverse. The Analysis library includes a VI for computing the pseudoinverse of real and complex matrices.

Exercise 8-8

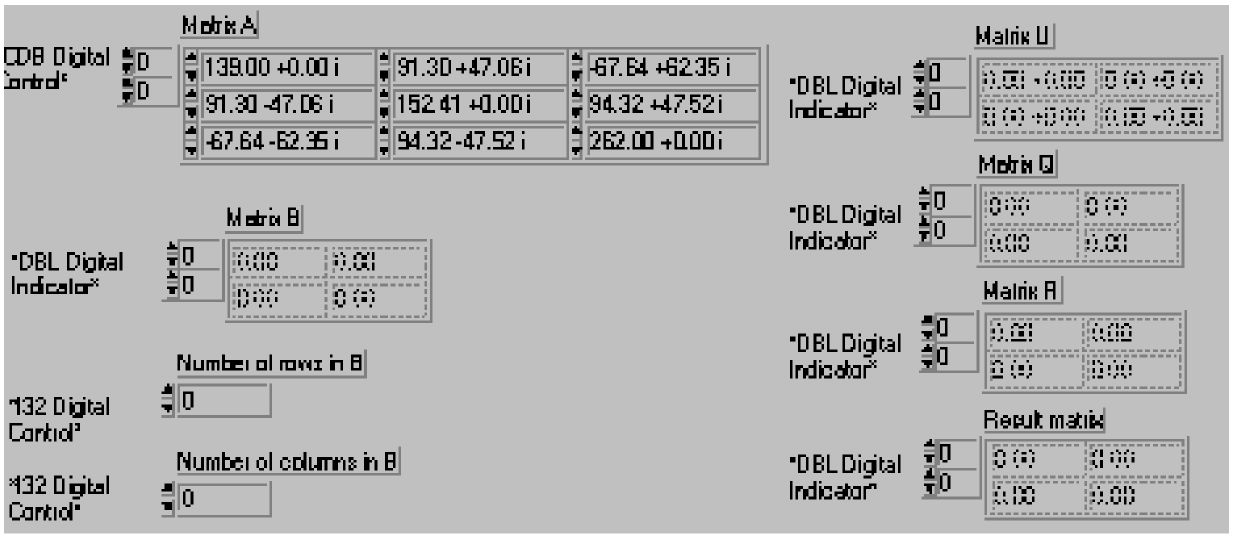
Objective: To compute Cholesky decomposition and QR decomposition.

Exercise 1-4 discussed complex positive definite matrices. You verified that the matrix A

$$A = \begin{bmatrix} 139 & 91.30 + 47.06i & -67.64 + 62.35i \\ 91.30 - 47.06i & 152.41 & 94.32 + 47.52i \\ -67.64 - 62.35i & 94.32 - 47.52i & 262.00 \end{bmatrix}$$

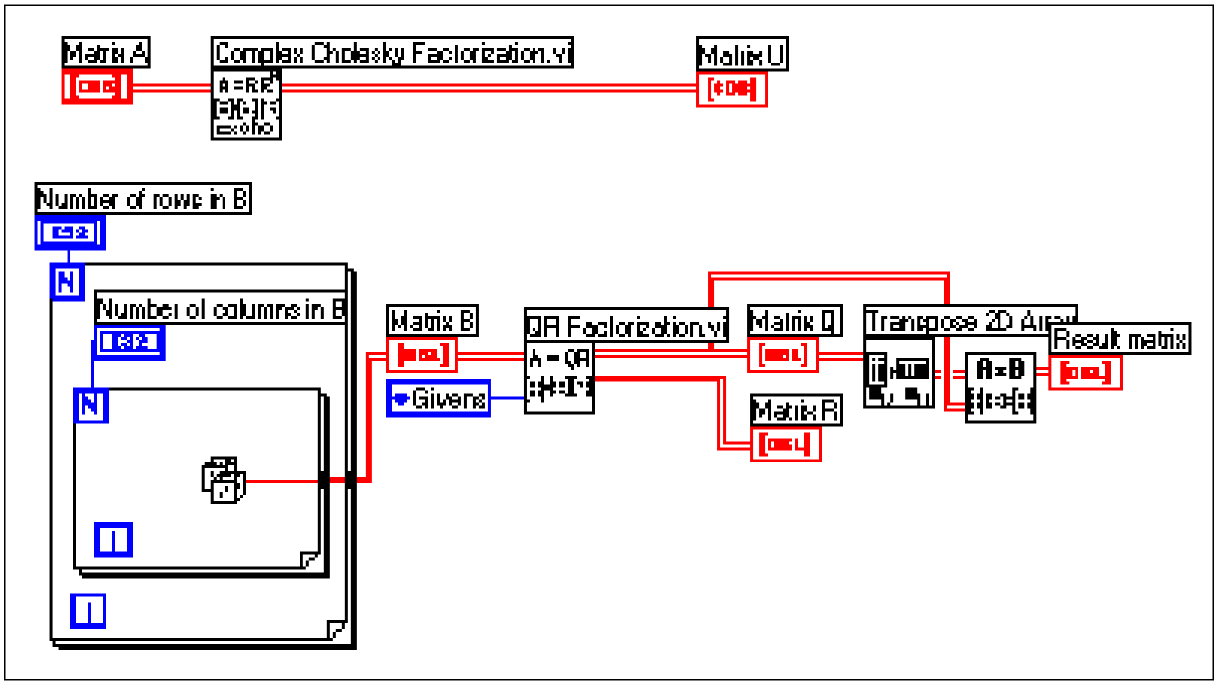
is a positive definite matrix. In this exercise, compute the Cholesky decomposition of this matrix. Also, compute the QR decomposition of a matrix B.

Front Panel



1. Build the front panel as shown above. You can resize the array of controls to see all the elements of the matrix.
2. The matrix U is an upper triangular matrix that is the result of the Cholesky Decomposition of the matrix A.
3. The matrix B is a rectangular matrix. That is, the number of rows m is different from the number of columns n. The result of the QR factorization is an $m \times m$ orthogonal matrix Q and an upper triangular matrix R of size $m \times n$.
4. In this exercise, you will also verify the definition of orthogonal matrices, $Q^T Q = I$. The **Result matrix** contains the product of the transpose of the orthogonal matrix and the orthogonal matrix itself.

Block Diagram



5. Build the block diagram as shown above

Complex Cholesky Factorization VI (**Analysis » Linear Algebra » Complex Linear Algebra » Advanced Complex Linear Algebra** subpalette). In this exercise, this function computes the Cholesky decomposition of the positive definite input complex matrix.

QR Factorization VI (**Analysis » Linear Algebra » Advanced Linear Algebra** subpalette). In this exercise, this function computes the QR factorization of the input real matrix.

6. Select the **Complex Cholesky Factorization** VI from the **Analysis » Linear Algebra » Complex Linear Algebra » Advanced Complex Linear Algebra** subpalette. The output terminal of this VI is connected to matrix *U*.
7. Generate a random matrix *B*. Select the **QR Factorization** VI from the **Analysis » Linear Algebra » Advanced Linear Algebra** subpalette. Connect the matrix *B* as the input matrix for QR factorization. Connect the *Q* and *R* outputs of this VI to matrix *Q* and matrix *R*, respectively.
8. Compute the transpose of the matrix *Q* and then multiply this transpose with the original matrix *Q*. The result of this operation is connected to the result matrix.

9. Choose a value for the number of rows and the number of columns. Run the VI. Notice the upper triangular structure of the output of the Cholesky decomposition and the *R* output of the QR factorization. Notice the structure of the orthogonal matrix *Q*. You can run the VI a number of times to generate different matrices and check if this definition is true.
10. Save the VI as **QR Factor.vi** and close it.

End of Exercise 8-8

Exercise 8-9 (Optional)

Objective: To compute singular value decomposition.

In this exercise, you will compute the singular value decomposition of the following matrix.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

Use the **SVD Factorization** VI from the **Analysis » Linear Algebra » Advanced Linear Algebra** subpalette. Compute the rank of the matrix *A* using the **Matrix Rank** VI (**Analysis » Linear Algebra » Advanced Linear Algebra** subpalette). Do you see an interesting relation between the rank of this matrix and the number of nonzero singular values? (The singular values are stored in the one-dimensional array *S*.) Now compute the 2-norm of this matrix using the **Matrix Norm** VI (**Analysis » Linear Algebra » Advanced Linear Algebra** subpalette). Do you see an interesting relation between this number and the largest singular value of the matrix?

Note: ***The rank of a matrix is equal to the number of nonzero singular values, which in this example is equal to 2. Also, as discussed at the beginning of this lesson, the 2-norm of a matrix is equal to its largest singular value.***

End of Exercise 8-9

Summary

- A matrix can be considered as a two-dimensional array of *m* rows and *n* columns. Determinant, rank, and condition number are some important attributes of a matrix.
- The condition number of a matrix affects the accuracy of the final solution.
- The determinant of a diagonal matrix, an upper triangular matrix, or a lower triangular matrix is the product of its diagonal elements.
- Two matrices can be multiplied only if the number of columns of the first matrix is equal to the number of rows in the second matrix.
- An eigenvector of a matrix is a nonzero vector that does not rotate when the matrix is applied to it. Similar matrices have the same eigenvalues.
- The existence of a unique solution for a system of equations depends on whether the matrix is singular or nonsingular.

Review Questions

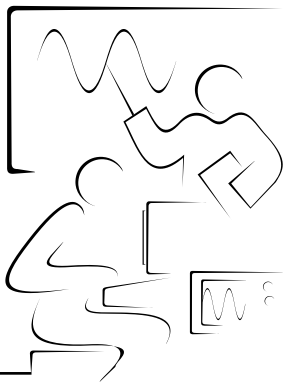
1. For the matrix given by $A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 2 & 3 \\ 2 & 0 & 1 \end{bmatrix}$, calculate its rank, determinant, 1-norm, and inf-norm.

   Which LabVIEW/BridgeVIEW VI(s) could you use to check your answer?
2. The condition number of a matrix *B* is 15.3, and that of a matrix *C* is 30,532. Which of these matrices, *B* or *C*, is closer to being singular?
3. Which of the following is true, and which is false?
   a. The eigenvalues of a real matrix are always real.
   b. The rank of an *m* x *n* matrix could at most be equal to the larger of *m* or *n* (*m* ¦ *n*).
4. For the two vectors given by *x* = [1,2] and *y* = [3,4], calculate:
   a. Their dot product
   b. Their outer product
   c. The angle between the two vectors

   Which LabVIEW/BridgeVIEW VI(s) could you choose to check your answer?
5. Which VI could you use to check if a matrix is positive definite? Why might you want to make such a check?
6. Why is matrix factorization important? Which are the matrix factorization VIs available in LabVIEW/BridgeVIEW?

Notes

Notes



Lesson 9
Probability and Statistics

Introduction

In this lesson, you will learn some of the fundamental concepts in probability and statistics such as mean or average, variance, histogram, and others. The lesson describes different LabVIEW/BridgeVIEW VIs that compute these quantities and show how they can be used in different applications.

You Will Learn:

A. What the terms *probability* and *statistics* mean, and how are they relevant in different areas of applications.

B. About the most commonly used concepts in statistics and how to use the Statistics VIs.

C. About the most commonly used concepts in probability and how to use the Probability VIs.

A. Probability and Statistics

We live in an information age in which facts and figures form an important part of life. Statements such as “There is a 60% chance of thunderstorms,” “Joe was ranked among the top five in the class,” “Michael Jordan has an average of 30 points this season,” and so on are common. These statements give a lot of information, but we seldom think about how this information was obtained. Was there a lot of data involved in obtaining this information? If there was, how did someone condense it to single numbers such as *60% chance* and *average of 30 points* or terms such as *top five*. The answer to all these questions brings up the very interesting field of statistics.

First, consider how information (data) is generated. Consider the statistics of part of the 1997 basketball season. Michael Jordan of the Chicago Bulls played 51 games, scoring a total of 1568 points. This includes the 45 points he posted, including the game-winning buzzer three-pointer, in a 103-100 victory over the Charlotte Hornets; his 36 points in an 88-84 victory over the Portland Trail Blazers; a season high of 51 points in an 88-87 victory over the New York Nicks; 45 points, seven rebounds, five assists, and three steals in a 102-97 victory over the Cleveland Cavaliers; and his 40 points, six rebounds, and six assists in a 107-104 victory over the Milwaukee Bucks. The point is not that Jordan is a great player, but that a single player can generate lots of data in a single season. The question is, how do you condense all this data so that it brings out all the essential information and is yet easy to remember? This is where the term *statistics* comes into the picture.

To condense all the data, single numbers must make it more intelligible and help draw useful inferences. For example, consider the number of points that Jordan scored in different games. It is difficult to remember how many points he scored in each game. But if you divide the total number of points that Jordan scored (1568) by the number of games he has played (51), you have a single number of 30.7 and can call it points per game *average*.

Suppose you want to rate Jordan’s free throw shooting skills. It might be difficult to do so by looking at his performance in each game. However, you can divide the number of free throws he has scored in all the games by the total number of free throws he was awarded. This shows he has a free throw *percentage* of 84.4%. You can obtain this number for all the NBA players and then rank them. Thus, you can condense the information for all the players into single numbers representing free throw percentage, points per game, and three-point average. Based on this information, you can rank players in different categories. You can further weight these different numbers and come up with a single number for each player. These single numbers can then help in judging the Most Valuable Player (MVP) for the season. Thus, in a broad sense, the term statistics implies different ways to summarize data to derive useful and important information from it.

The next question is, what is probability? You have looked at ways to summarize lots of data into single numbers. These numbers then help draw conclusions for the present. For example, looking at Jordan's statistics for the 1996 season helped the NBA officials elect him the MVP for that season. It also helped people to infer that he is one of the best players in the game. But can you say anything about the future? Can you measure the degree of accuracy in the inference and use it for making future decisions? The answer lies in the theory of probability. Whereas, in laymen's terms, one would say that it is *probable* that Jordan will continue to be the best in the years to come, you can use different concepts in the field of probability, as discussed later in this lesson, to make more quantitative statements.

In a completely different scenario, there may be certain experiments whose outcomes cannot be predetermined, but certain outcomes may be more probable. This once again leads to the notion of probability. For example, if you flip an unbiased coin in the air, what is the chance that it will land heads up? The chance or probability is 50 %. That means, if you repeatedly flip the coin, half the time it will land heads up. Does this mean that 10 tosses will result in exactly five heads? Will 100 tosses result in exactly 50 heads? Probably not. But in the long run, the probability will work out to be 0.5.

To summarize, whereas statistics allows you to summarize data and draw conclusions for the present, probability allows you to measure the degree of accuracy in those conclusions and use them for the future.

B. Statistics

In this section, you will look at different concepts and terms commonly used in statistics and see how to use the Analysis VIs in different applications.

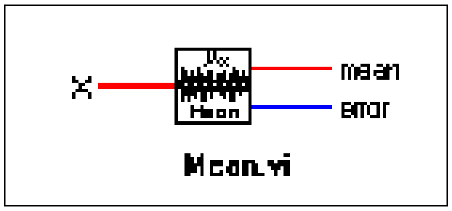
Mean

Consider a data set *X* consisting of *n* samples $x_0, x_1, x_2, x_3, \ldots, x_{n-1}$. The mean value (a.k.a. average) is denoted by $\bar{x}$ and is defined by the formula

$$\bar{x} = \frac{1}{n}(x_0 + x_1 + x_2 + x_3 + \ldots + x_{n-1})$$

In other words, it is the sum of all the sample values divided by the number of samples. As you saw in the Michael Jordan example above, the data set consisted of 51 samples. Each sample was equal to the number of points that Jordan scored in each game. The total of all these points was 1568, divided by the number of samples (51) to get a mean or average value of 30.7.

The input-output connections for the **Mean** VI are shown below.



Median

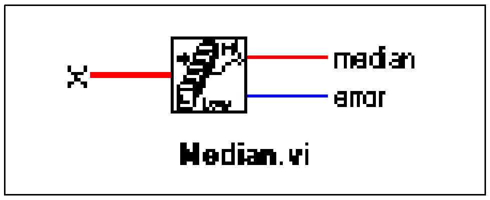
Let $S = \{s_0 s_1, s_2, \ldots, s_{n-1}\}$ represent the sorted sequence of the data set *X*. The sequence can be sorted either in the ascending order or in descending order. The median of the sequence is denoted by x_{median} and is obtained by the formula

$$x_{median} = \begin{cases} s_i & \text{n is odd} \\ 0.5(s_{k-1} + s_k) & \text{n is even} \end{cases}$$

where $i = \frac{n-1}{2}$ and $k = \frac{n}{2}$.

In words, the median of a data sequence is the *midpoint* value in the sorted version of that sequence. For example, consider the sequence $\{5, 4, 3, 2, 1\}$ consisting of five (odd number) samples. This sequence is already sorted in the descending order. In this case, the median is the midpoint value, 3. Consider a different sequence $\{1, 2, 3, 4\}$ consisting of four (even number) samples. This sequence is already sorted in the ascending order. In this case, there are two midpoint values, 2 and 3. As per the formula above, the median is equal to $0.5 \times (2 + 3) = 2.5$. If a student X scored 4.5 points on a test and another student Y scored 1 point on the same test, the median is a very useful quantity for making qualitative statements such as “X lies in the top half of the class” or “Y lies in the bottom half of the class.”

The input-output connections for the **Median** VI are shown below.



Sample Variance

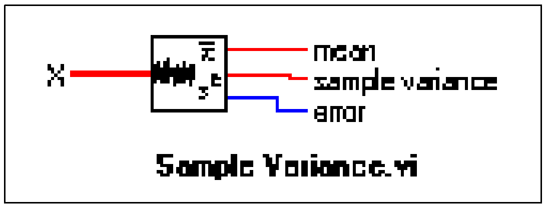
The sample variance of the data set X consisting of n samples is denoted by s^2 and is defined by the formula

$$s^2 = \frac{1}{n-1}[(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \ldots + (x_n - \bar{x})^2]$$

where $\bar{x}$ denotes the mean of the data set. Hence, the sample variance is equal to the sum of the squares of the deviations of the sample values from the mean divided by n-1.

Note: ***The above formula does not apply for n = 1. However, it does not mean anything to compute the sample variance if there is only one sample in the data set.***

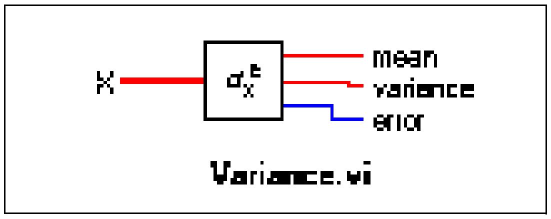
The input-output connections for the **Sample Variance** VI are shown below.



In other words, the sample variance measures the spread or dispersion of the sample values. If the data set consists of the scores of a player from different games, the sample variance can be used as a measure of the consistency of the player. It is always positive, except when all the sample values are equal to each other and in turn equal to the mean.

There is one more type of variance called population variance. The formula to compute population variance is similar to the one above to compute sample variance, except for the (n-1) in the denominator replaced by n.

The input-output connections for the **Variance** VI are shown below.



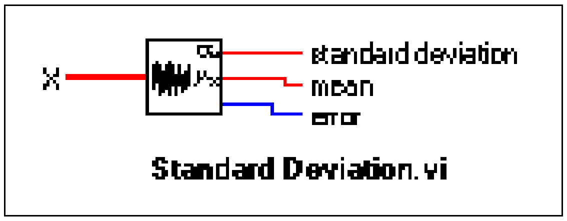
The **Sample Variance** VI computes sample variance, whereas the **Variance** VI computes the population variance. Whereas statisticians and mathematicians prefer to use the latter, engineers prefer to use the former. It really does not matter for large values of n, say $n \geq 30$.

☞ **Note:** ***Use the proper type of VI suited for your application.***

Standard Deviation

The positive square root of the sample variance s^2 is denoted by s and is called the standard deviation of the sample.

The input-output connections for the **Standard Deviation** VI are shown below.



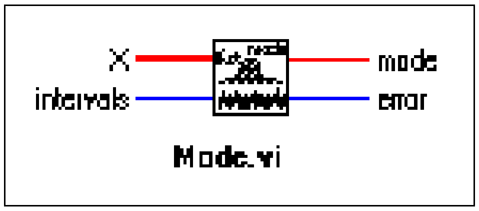
Mode

The mode of a sample is a sample value that occurs most frequently in the sample. For example, if the input sequence *X* is

$X = \{0, 1, 3, 3, 4, 4, 4, 5, 5, 7\}$

then the mode of *X* is 4, because that is the value that most often occurs in *X*.

The input-output connections for the **Mode** VI are shown below.



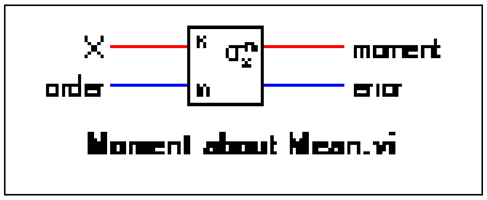
Moment About Mean

If X represents the input sequence with n number of elements in it, and $\bar{x}$ is the mean of this sequence, then the m^{th}-order moment can be calculated using the formula

$$\sigma_x^{\ m} = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \bar{x})^m$$

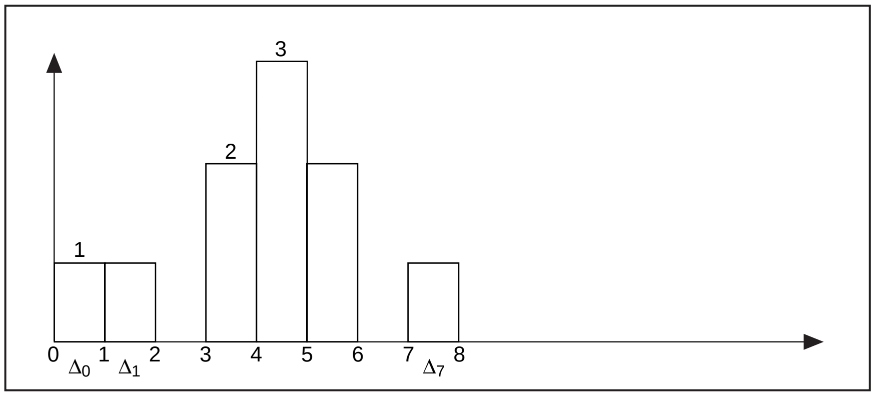
In other words, the moment about mean is a measure of the deviation of the elements in the sequence from the mean. Note that for $m = 2$, the moment about mean is equal to the population variance.

The input-output connections for the **Moment About Mean** VI are shown below.



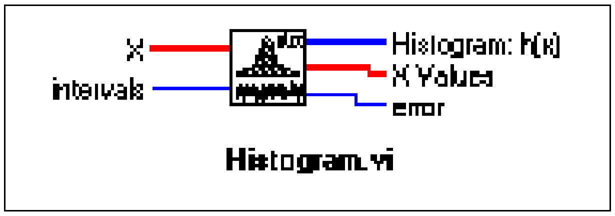
Histogram

So far, this lesson has discussed different ways to extract important features of a data set. The data is usually stored in a table format, which many people find difficult to grasp. It is generally useful to display the data in some form. The visual display of data helps us gain insights into the data. Histogram is one such graphical method for displaying data and summarizing key information. Consider a data sequence $X = \{0, 1, 3, 3, 4, 4, 4, 5, 5, 8\}$. Divide the total range of values into 8 intervals. These intervals are 0-1, 1-2, 2-3, ..., 7-8. The histogram for the sequence X then plots the number of data samples that lie in that interval, not including the upper boundary.



The figure above shows that one data sample lies in the range 0-1 and 1-2, respectively. However, there is no sample in the interval 2-3. Similarly, two samples lie in the interval 3-4, and three samples lie in the range 4-5. Examine the data sequence *X* above and be sure you understand this concept.

There are different ways to compute data for a histogram. Next you will see how it is done in the **Histogram** VI using the sequence *X*.



As shown above, the inputs to this VI are the **input sequence** *X* and the **number of intervals** *m*. The VI obtains **Histogram:h(x)** as follows. It scans *X* to determine the range of values in it. Then the VI establishes the interval width, Δx, according to the specified value of *m*

$$\Delta x = \frac{max - min}{m}$$

where *max* is the maximum value found in *X*, *min* is the minimum value found in *X*, and *m* is the specified number of intervals.

Let $m = 8$. Then $\Delta x = \frac{8-0}{8} = 1$

Let χ represent the output sequence *X* Values. The histogram is a function of *X*. This VI evaluates the elements of χ using

$$\chi_i = min + 0.5\Delta x + i\Delta x \qquad for \qquad i = 0, 1, 2, \ldots, m-1$$

For this example, $\chi_0 = 0.5, \chi_1 = 1.5, \ldots, \chi_7 = 7.5$.

The VI then defines the i^{th} interval to be in the range of values from $\chi_i - 0.5\Delta x$ up to but not including $\chi_i + 0.5\Delta x$,

$$\Delta_i = [(\chi_i - 0.5\Delta x), (\chi_i + 0.5\Delta x)), for \qquad i = 0, 1, 2, \ldots, m-1$$

and defines the function $y_i(x) = 1$ for *x* belonging to Δ_i and zero elsewhere. The function has unity value if the value of *x* falls within the specified interval, not including the boundary. Otherwise, it is zero. Notice that the interval is centered about χ_i and its width is Δ_x. If a value is equal to max, it is counted as belonging to the last interval.
For our example, $\Delta_0 = [0, 1), \Delta_1 = [1, 2), \ldots, \Delta_7 = [7, 8)$ and as an example $y_0(0) = 1$ and $y_0(1) = y_0(3) = y_0(4) = y_0(5) = y_0(8) = 0$.

Finally, the VI evaluates the histogram sequence *H* using

$$h_i = \sum_{j=0}^{n-1} y_i(x_j) \qquad for \qquad i = 0, 1, 2, \ldots, m-1$$

where h_i represents the elements of the output sequence *Histogram: h(X)* and *n* is the number of elements in the input sequence *X*. For this example, $h_0 = 1, h_4 = 3, \ldots, h_7 = 1$.

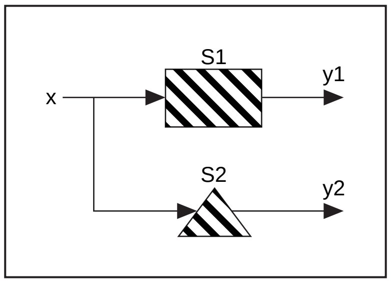
The Analysis library also has a **General Histogram** VI that is more advanced than the **Histogram** VI. Please refer to the *LabVIEW Analysis VI Reference Manual* for detailed information.

Mean Square Error (MSE)

If X and Y represent two input sequences, the mean square error is the average of the sum of the square of the difference between the corresponding elements of the two input sequences. The following formula is used to find the mse.

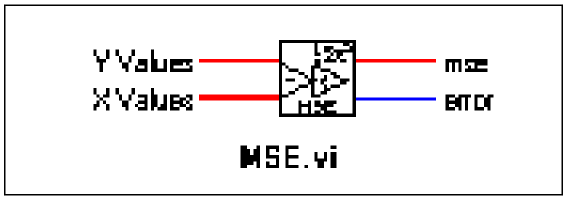
$$mse = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - y_i)^2$$

where *n* is the number of data points.



Consider a digital signal x fed to a system, S1. The output of this system is y1. Now you acquire a new system, S2, which is theoretically known to generate the same result as S1 but has two times faster response time. Before replacing the old system, you want to be absolutely sure that the output response of both the systems is the same. If the sequences y1 and y2 are very large, it is difficult to compare each element in the sequences. In such a scenario, you can use the **MSE** VI to calculate the mean square error (mse) of the two sequences y1 and y2. If the mse is smaller than an acceptable tolerance, the system S1 can be reliably replaced by the new system S2.

The input-output connections for the **MSE** VI are shown below.



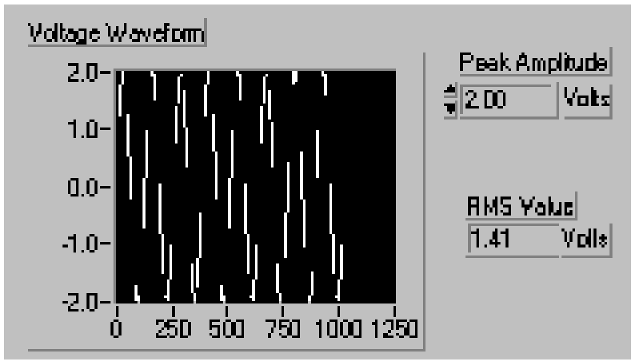
Root Mean Square (RMS)

The root mean square Ψ_x of a sequence X is the positive square root of the mean of the square of the input sequence. In other words, you can square the input sequence, take the mean of this new squared sequence, and then take the square root of this quantity. The formula used to compute the rms value is

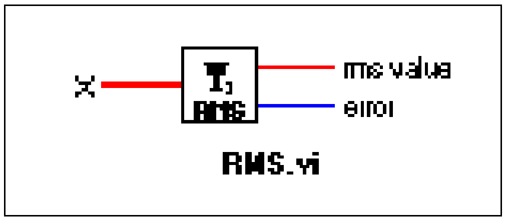
$$\Psi_x = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1} x_i^2}$$

where *n* is the number of elements in *X*.

RMS is a widely used quantity in the case of analog signals. For a sine voltage waveform, if V_p is the peak amplitude of the signal, then the root mean square voltage V_{rms} is given by $\frac{V_p}{\sqrt{2}}$. The following figure shows a voltage waveform of peak amplitude = 2 V and the RMS value of $\sqrt{2} \approx 1.41$ V computed using the **RMS** VI from the Analysis library.



The input-output connections for the **RMS** VI are shown below.

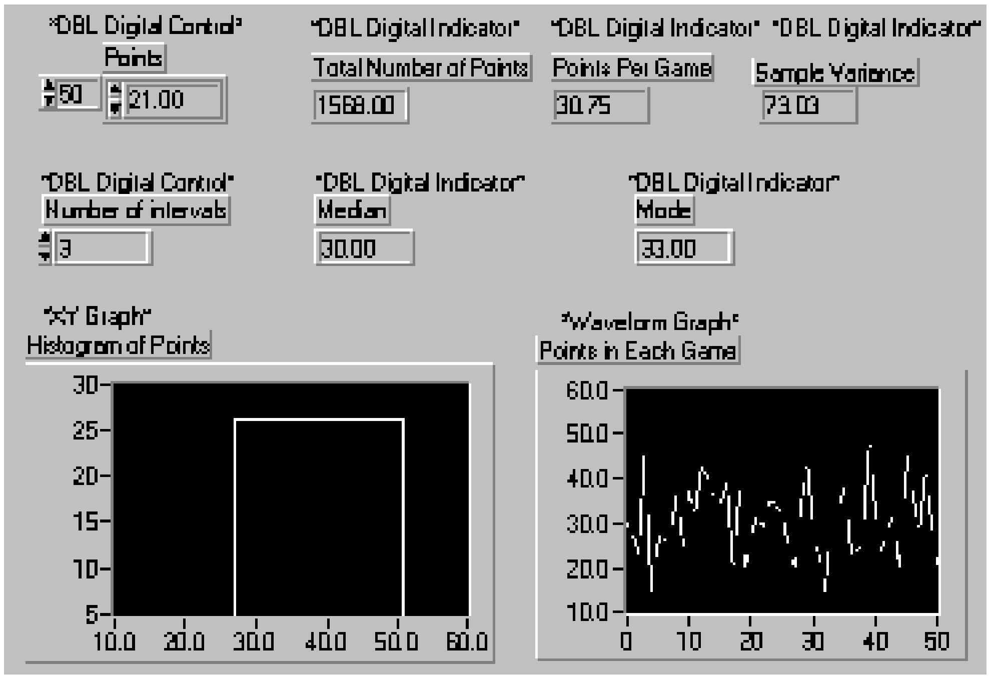


Exercise 9-1

Objective: To use different Statistics VIs in an example using Michael Jordan's basketball scores.

In this exercise, you will learn how to use different Statistics VIs in an interesting example. The data set consists of the number of points scored by Michael Jordan of the Chicago Bulls in each of the 51 games he played during part of the 1997 NBA season. You will use some of the concepts discussed in the previous section to decipher this information and create single numbers that are easy to remember and yet reveal all the information that the entire data set provides.

Front Panel



1. Open the **Statistics** VI from `Lvspcex.llb`.
2. Open the front panel as shown above. The **Points** control is the data set consisting of the number of points scored by Jordan in each game this season. The number of games he has played this season to date is 51.

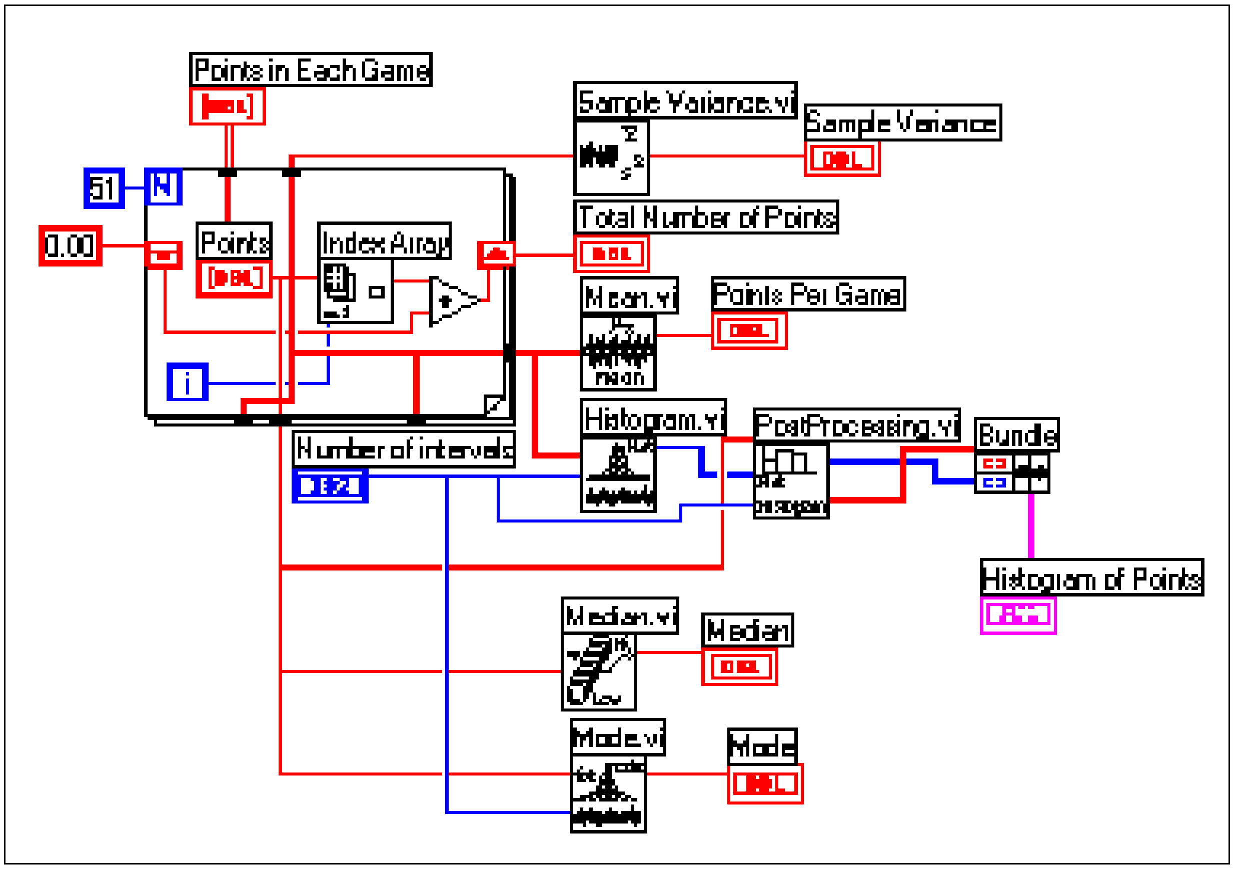
   The **Total Number of Points** digital indicator is the sum of all the elements in the given data set.

   The **Points Per Game** digital indicator is the average of Jordan's scores this season.

   The **Sample Variance**, **Median**, and **Mode** digital indicators are the sample variance, median, and mode of the data set.

Histogram of Points is an XY graph that shows the distribution of points in different intervals. The number of intervals is set using the Number of intervals digital control. **Points in Each Game** is a waveform graph that plots the number of points scored in each game.

Block Diagram



3. Build the diagram shown above:

Sample Variance VI (**Analysis » Probability and Statistics** subpalette). In this exercise, this function computes the sample variance of the data set **Points**.

Mean VI (**Analysis » Probability and Statistics** subpalette). In this exercise, this function computes the mean value (average) of the data set **Points**.

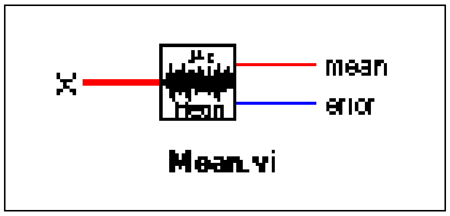
Histogram VI (**Analysis » Probability and Statistics** subpalette). In this exercise, this function computes the histogram of the data set **Points**.

Median VI (**Analysis » Probability and Statistics** subpalette). In this exercise, this function computes the median of the data set **Points**.

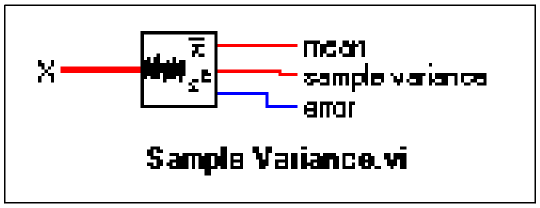
Mode VI (**Analysis » Probability and Statistics** subpalette). In this exercise, this function computes the mode of the data set **Points**.

Bundle function (**Cluster** subpalette) In this exercise, this function assembles the outputs of the **Post Processing** VI (explained later) to plot on the XY graph **Histogram of Points.**

4. Follow the instructions in steps 4 through 9 and build the block diagram as shown above.
5. You will compute the Points Per Game (Jordan's average) by using the **Mean** VI in the Analysis library. Connect the **Points** control to the input terminal **X** and the output mean to the **Points Per Game** indicator.

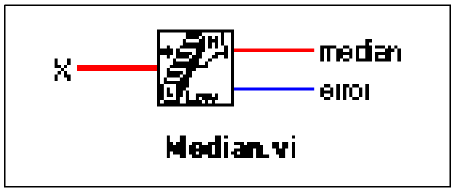


6. You will compute the Sample Variance using the **Sample Variance** VI in the Analysis library. Once again, connect the **Points** control to the input terminal **X** and the output sample variance to the **Sample Variance** indicator.

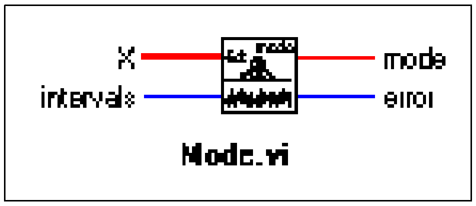


Note: ***This VI also computes the mean of the data set.***

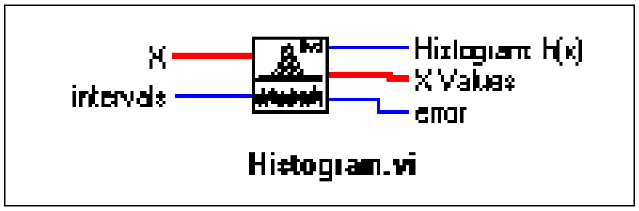
7. You will compute the Median using the **Median** VI in the Analysis library. Connect the **Points** control to the input terminal **X** and the output median to the **Median** indicator.



8. You will compute the Mode using the **Mode** VI in the Analysis library. Connect the **Points** control to the input terminal **X**. Choose the number of intervals equal to 3. Connect the output mode to the **Mode** indicator.

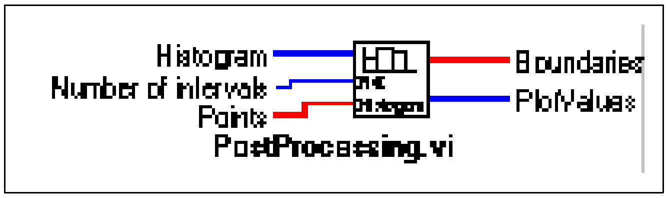


9. You first will compute the data for the histogram using the **Histogram** VI. Connect the control **Points** to the input terminal **X** and set the number of intervals equal to 3. This VI generates histogram values and X values, which are the midpoints of the different intervals as discussed above. You can plot the histogram by using the X values for the X axis and the histogram values for the Y axis. If you are interested, try doing this and observe the histogram.



10. Generally, you may want to view the histogram in a different way. Select the **PostProcessing** VI from `Lvspcex.llb`. Connect the **Points** control to the **Points** input terminal, the histogram output of the **Histogram** VI to the **Histogram** input terminal, and set the number of intervals equal to 3 (same value you chose earlier for the **Histogram** VI). This VI generates the data for plotting the histogram in a better way. The X axis values are stored in the **Boundaries** output terminal, and the Y axis values are stored in the **PlotValues**

output terminal. You can use the output of this VI for plotting the histogram on the **Histogram of Points** XY graph.



11. Return to the front panel. Set the number of intervals equal to 3 and run the VI. Study the different output values. See how the histogram (on the left) provides more information than just plotting the points in each game (on the right).
12. Save the VI and close it.

End of Exercise 9-1

C. Probability

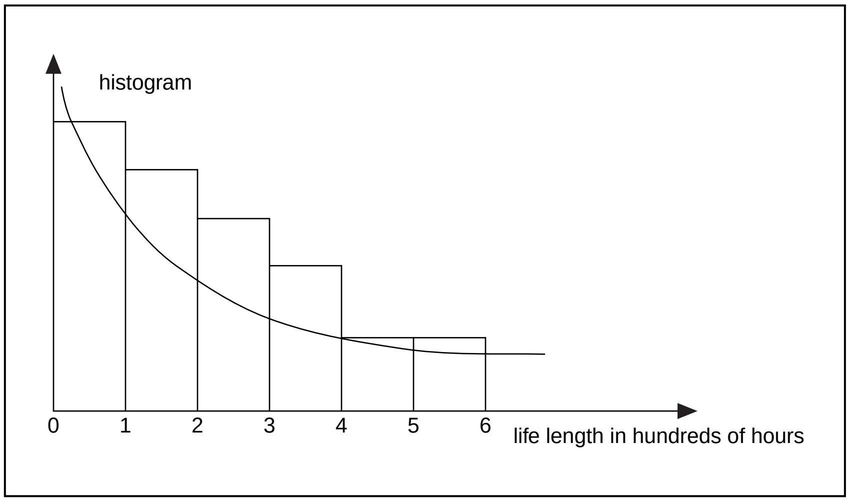
In any random experiment, there is always a chance that a particular event will or will not occur. A number between 0 and 1 is assigned to measure this chance, or probability, that a particular event occurs. If you are absolutely sure that the event will occur, its probability is 100% or 1.0, but if you are sure that the event will not occur, its probability is 0.

Consider a simple example. If you roll a single unbiased die, there are six possible events that can occur—either a 1, 2, 3, 4, 5, or 6 can result. What is the probability that a 2 will result? This probability is one in six, or 0.16666. You can define probability in simple terms as: The probability that an event A will occur is the ratio of the number of outcomes favorable to A to the total number of equally likely outcomes.

Random Variables

Many experiments generate outcomes that you can interpret in terms of real numbers. Some examples are the number of cars passing a stop sign during a day, number of voters favoring candidate A, and number of accidents at a particular intersection. The values of the numerical outcomes of this experiment can change from experiment to experiment and are called random variables. Random variables can be discrete (if they can take on only a finite number of possible values) or continuous. As an example of the latter, weights of patients coming into a clinic may be anywhere from, say, 80 to 300 pounds. Such random variables can take on any value in an interval of real numbers. Given such a situation, suppose you want to find the probability of encountering a patient weighing exactly 172.39 pounds. You will see how to calculate this probability next using an example.

Consider an experiment to measure the life lengths *x* of 50 batteries of a certain type. These batteries are selected from a larger population of such batteries. The histogram for observed data is shown below.



This figure shows that most of the life lengths are between zero and 100 hours, and the histogram values drop off smoothly as you look at larger life lengths.

You can approximate the histogram shown above by an exponentially decaying curve. You could take this function as a mathematical model for the behavior of the data sample. If you want to know the probability that a randomly selected battery will last longer than 400 hours, this value can be approximated by the area under the curve to the right of the value 4. Such a function that models the histogram of the random variable is called the *probability density function*.

To summarize all the information above in terms of a definition, a random variable *X* is said to be *continuous* if it can take on the infinite number of possible values associated with intervals of real numbers, and there is a function *f(x)*, called the *probability density function*, such that

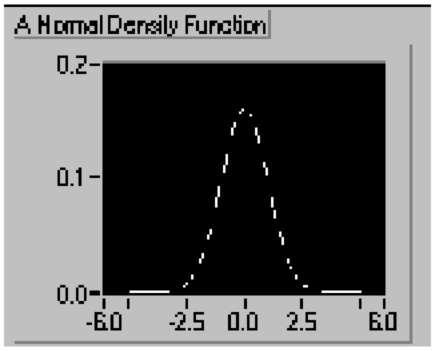
1. $f(x) \geq 0$ for all x
2. $\int_{-\infty}^{\infty} f(x)dx = 1$
3. $P(a \leq X \leq b) = \int_{a}^{b} f(x)dx$

Notice from equation (3) above that for a specific value of the continuous random variable, that is for *X=a*, $P(X = a) = \int_{a}^{a} f(x)dx = 0$. It should not be surprising that you assign a probability of zero to any specific value, because there are an infinite number of possible values that the random variable can take. Therefore, the chance that it will take on a specific value $X = a$ is extremely small.

The previous example used the exponential function model for the probability density function. There are a number of different choices for this function. One of these is the Normal Distribution, discussed below.

Normal Distribution

The normal distribution is one of the most widely used continuous probability distributions. This distribution function has a symmetric bell



shape, as shown above. The curve is centered at the mean value $\bar{x} = 0$, and its spread is measured by the variance $s^2 = 1$. These two parameters completely determine the shape and location of the normal density function, whose functional form is given by

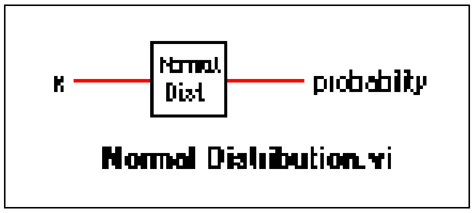
$$f(x) = \frac{1}{s\sqrt{2\pi}} e^{-(x-\bar{x})^2/(2s^2)}$$

Suppose a random variable Z has a normal distribution with mean equal to zero and variance equal to one. This random variable is said to have *standard normal distribution.*

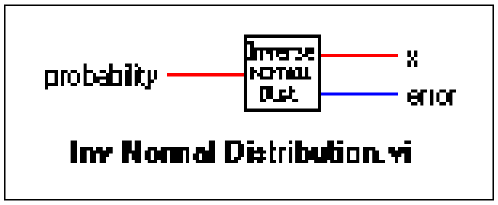
The **Normal Distribution** VI computes the one-sided probability, *p*, of the normally distributed random variable *x*.

$$p = Prob(X \le x)$$

where *X* is a standard normal distribution with the mean value equal to zero and variance equal to one, *p* is the probability and *x* is the value.



Suppose you conduct an experiment in which you measure the heights of adult males. You conduct this experiment on 1000 randomly chosen men and obtain a data set S. The histogram distribution has many measurements clumped closely about a mean height, with relatively few very short and very tall males in the population. Therefore, the histogram can be closely approximated by a normal distribution. Now suppose that, among a different set of 1000 randomly chosen males, you want to find the probability that the height of a male is greater than or equal to 170 cm. You can use the **Normal Distribution** VI to find this probability. Set the input $x = 170$. Thus, the choice of the probability density function is fundamental to obtaining a correct probability value.



The **Inverse Normal Distribution** VI performs exactly the opposite function as the **Normal Distribution** VI. Given a probability *p*, it finds the values *x* that have the chance of lying in a normally distributed sample. For example, you might want to find the heights that have a 60% chance of lying in a randomly chosen data set.

As mentioned earlier, there are different choices for the probability density function. The well-known and widely used ones are the Chi-Square distribution, the F distribution, and the T-distribution[1]. The Analysis library includes VIs that compute the one-sided probability for these different types of distributions. In addition, it also has VIs that perform the inverse operation.

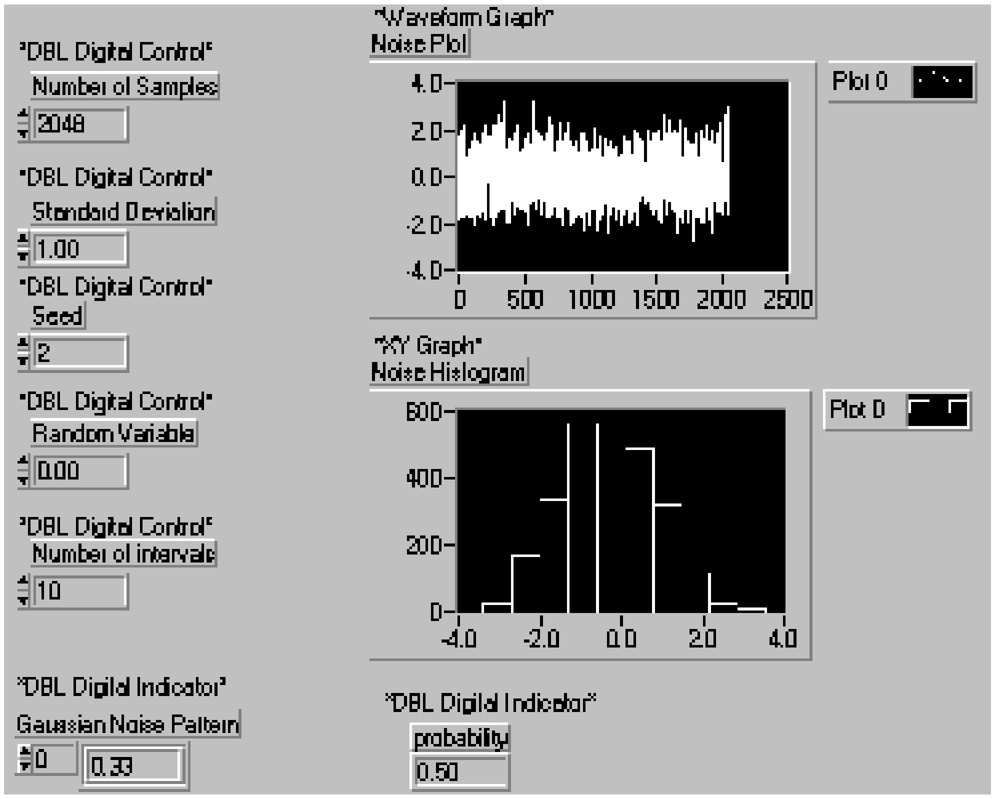
1. Interested readers should refer to the latest edition of the Schaum's Outline Series on Theory and Problems of Probability and Statistics by Murray Spiegel, McGraw-Hill, Inc., 1975, for detailed discussion on these three types of distributions.

Exercise 9-2

Objective: To understand key probability concepts.

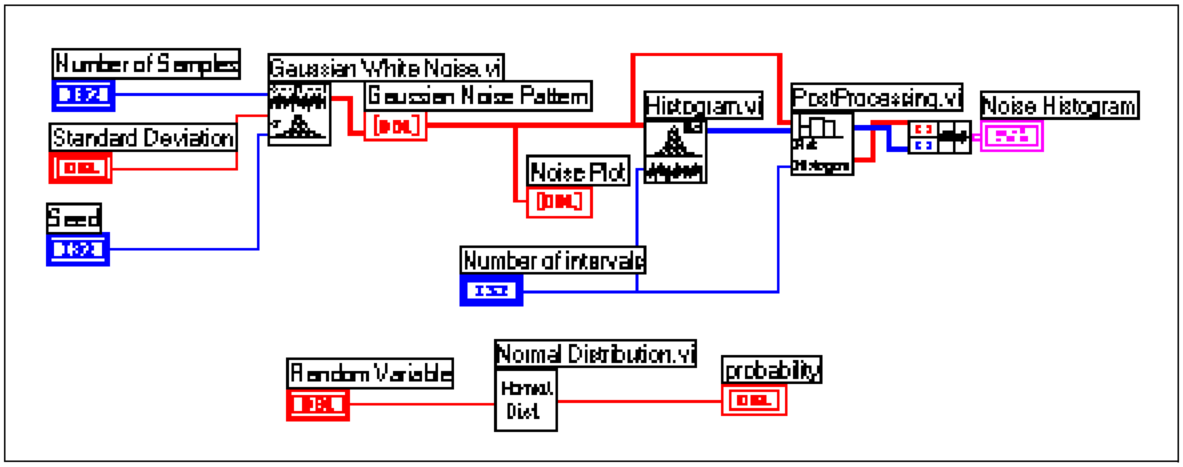
In this exercise, you will first generate a data sample with standard normal distribution and then use the **Normal Distribution** VI to check the probability of a random variable *x*.

Front Panel



1. Build the front panel as shown above. **NoisePlot** is a waveform graph, whereas **NoiseHistogram** is an XY graph.

Block Diagram



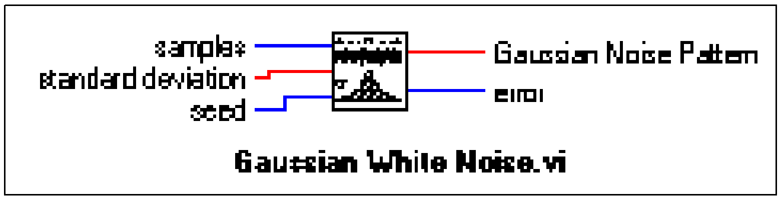
2. Build the block diagram as shown above. The **Gaussian White Noise** VI generates a Gaussian-distributed pattern with mean value equal to 0 and standard deviation set by the user using the input standard deviation. **Samples** is the number of samples of the Gaussian noise pattern. **Seed** is the seed value used to generate the random noise.

Gaussian White Noise VI (**Analysis » Signal Generation** subpalette). In this exercise, this function generates a Gaussian white noise pattern.

Histogram VI (**Analysis » Probability and Statistics** subpalette). In this exercise, this function computes the histogram of the Gaussian noise pattern.

Normal Distribution VI (**Analysis » Probability and Statistics** subpalette). In this exercise, this function computes the one-sided probability of the normally distributed random variable **Random Variable**.

Connect the Gaussian noise pattern to the waveform graph **Noise Plot**.



3. You will compute the histogram of the Gaussian noise pattern using the **Histogram** VI used in the previous exercise.

4. As discussed earlier, do some postprocessing to plot the histogram in a different way. Select the **PostProcessing** VI from `Lvspcex.llb`.
5. Bundle the output of this VI and connect it to the **Noise Histogram**.
6. Select the **Normal Distribution** VI. Connect the **Random Variable** control to the input terminal and connect the output to the probability indicator.
7. Return to the front panel. Set the **Number of Samples** control to 2048, **Standard Deviation** to 1, **Seed** to 2 and **Number of intervals** to 10. Run the VI.
8. You will see the Gaussian white noise on the **Noise Plot** graph. It is difficult to tell much from this plot. However, the histogram plot for the same noise pattern provides a lot of information. It shows that most of the samples are centered around the mean value of zero. From this histogram, you can approximate this noise pattern by a **Normal Distribution** function (Gaussian distribution). Because the mean value is zero and you set the standard deviation equal to one, the probability density function is actually a standard normal distribution.

Note: ***It is very important that you carefully choose the proper type of distribution function to approximate your data. In this example, you actually plotted the histogram to make this decision. Many times, you can make an intelligent decision based solely on prior knowledge of the behavior and characteristics of the data sample.***

9. Return to the front panel and enter a value for **Random Variable**. This VI will compute the one-sided probability of this normally distributed random variable. Remember, you have assumed that the variable is normally distributed by looking at the histogram.
10. Save the VI as **Probability.vi** and close it.

End of Exercise 9-2

Summary

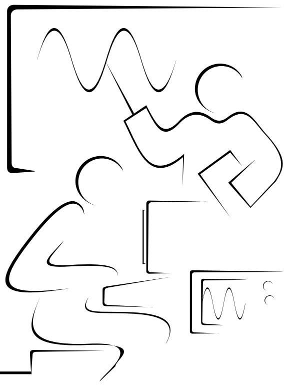
- Different concepts in statistics and probability help decipher information and data to make intelligent decisions.
- Mean, median, sample variance, and mode are some of the statistics techniques to help in making inferences from a sample to a population.
- Histograms are widely used as a simple but informative method of data display.
- Using the theory of probability, you can make inferences from a sample to a population and then measure the degree of accuracy in those inferences.

Review Questions

1. What is the difference between probability and statistics? Which VI(s) would you use in each case?
2. What is the difference between:
   a. Mode and median?
   b. Sample variance and population variance?

   Which VI(s) would you use in each case?
3. Name some real-world practical applications of using:
   a. Histograms
   b. The Gaussian probability density function
4. What is the difference between the **Normal Distribution** VI and the **Inv Normal Distribution** VI?

Notes



Lesson 10
Digital Filter Design Toolkit

Introduction

In this lesson, you will learn how to use the Digital Filter Design (DFD) Toolkit to design FIR and IIR filters to meet required specifications. You will also see how to use the DFD toolkit to analyze your filter design in terms of its frequency response, impulse and step responses, and its pole-zero plot.

You Will Learn:

A. About digital filters.
B. About the Digital Filter Design Toolkit.
C. About designing IIR filters.
D. About designing FIR filters.
E. How to use the Digital Filter Design Toolkit to analyze your filter design.

A. Review of Digital Filters

In a previous lesson, you learned about the basic theory behind the operation of digital filters. Below is a brief review of some of the important information you need to know to use the Digital Filter Design (DFD) Toolkit.

Filtering

Filtering is one of the most common signal processing techniques and is the process by which the frequency content of a signal is altered. Some of the practical applications of filtering are in the bass and treble controls of your stereo to adjust the frequency response, in the tuning circuits of your radio and television receivers to select a particular channel, in telephone handsets to limit the frequency content of the sound signals to 3 KHz, and many others in the audio, telecommunications, geophysics, and medical fields.

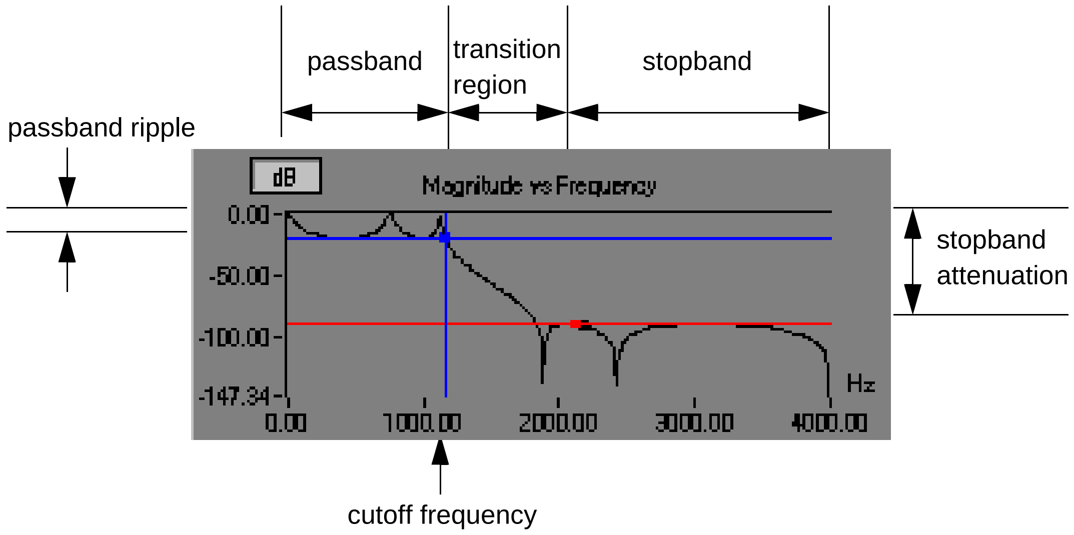
Why Digital Filters?

Until the advent of the computer age, filtering was in analog form using resistors, inductors, and capacitors. Both the input and the output of the filter were analog signals. Designing analog filters is a specialized task requiring a good mathematical background and proper understanding of the filtering process. However, with the widespread use of computers, digital representation and processing of signals gained immense popularity due to the numerous advantages that digital signals have over their analog counterparts. Because of this, analog filters have gradually been replaced by digital filters. The advantages of digital filters over analog filters are:

- They are software programmable, and so are easy to “build” and test.
- They require only the arithmetic operations of multiplication and addition/subtraction and so are easier to implement.
- They are stable (do not change with time nor temperature) and predictable.
- They do not drift with temperature or humidity or require precision components.
- They have a superior performance-to-cost ratio.
- They do not suffer from manufacturing variations or aging.

Filter Response Characteristics

The range of frequencies that a filter passes through it is known as the *passband,* whereas the range of frequencies that are attenuated is known as the *stopband*. Between the passband and the stopband is a *transition* region where the gain falls from one (that is, 0 dB in the passband) to zero or a very small value (in the stopband). The passband, stopband, and the transition region for a lowpass filter are shown below.

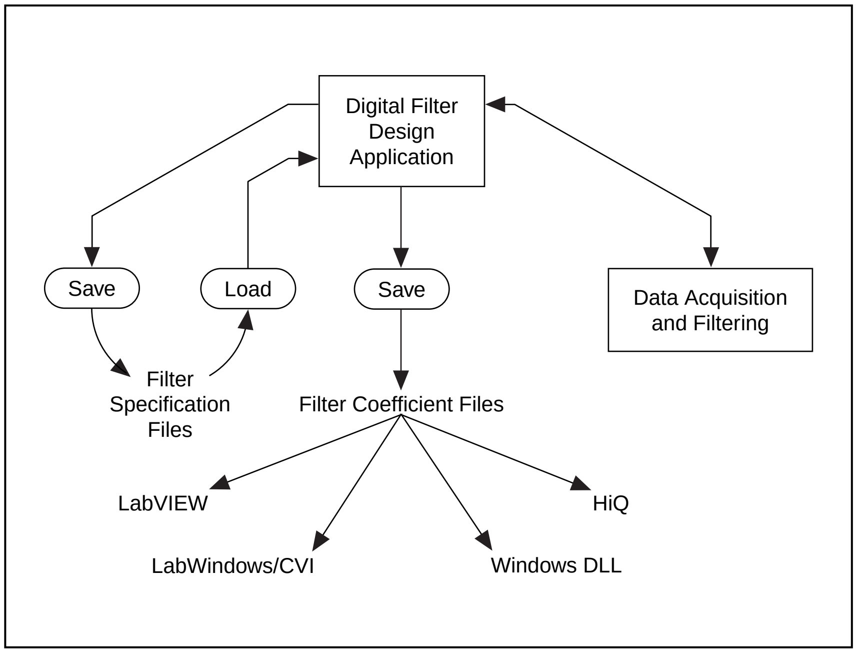


The figure above also shows the *passband ripple*, the *stopband attenuation*, and the *cut-off frequency*, three specifications that are needed in designing digital filters. The passband ripple (in dB) is the maximum deviation in the passband from 0 dB, whereas the stopband attenuation is the minimum attenuation (in dB) in the stopband. In the DFD toolkit, the passband ripple is also referred to as the *passband response*.

B. The Digital Filter Design Toolkit

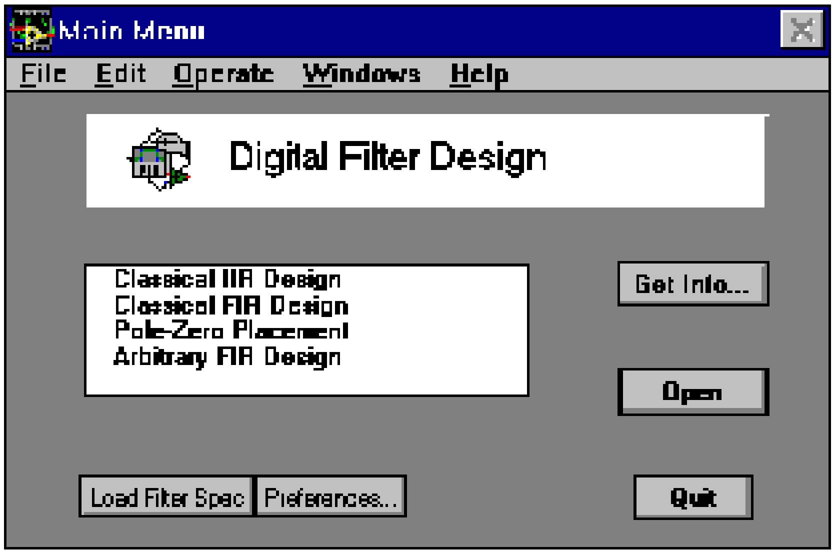
Lesson 6 introduced the Digital Filter VIs available in the LabVIEW/BridgeVIEW Advanced Analysis library. Using these VIs is one way to design your digital filters. However, National Instruments also provides the Digital Filter Design (DFD) toolkit, which is a complete filter design and analysis tool you can use to design digital filters to meet your precise filter specifications. You can graphically design your IIR and FIR filters, interactively review filter responses, save your filter design work, and load your design work from previous sessions. If you have a National Instruments data acquisition (DAQ) device, you can perform real-world filter testing from within the DFD application. You can view the time waveforms or the spectra of both the input signal and the filtered output signal while simultaneously redesigning your digital filters.

After you design your digital filter, you can save the filter coefficients to a file on your drive. The filter coefficient files can then be loaded for later implementation by LabVIEW, BridgeVIEW, LabWindows/CVI, or any other application. The following diagram shows you the conceptual overview of the DFD Toolkit.


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

Main Menu

When you launch the DFD application, you get the following panel, which is referred to as the Main Menu:



Design Options

From the Main Menu, you can choose any of the following four methods of designing digital filters:

- *Classical IIR Design*—for designing IIR filters by specifying the frequency response characteristics.
- *Classical FIR Design*—for designing FIR filters by specifying the frequency response characteristics.
- *Pole-Zero Placement*—for designing either IIR or FIR filters by adjusting the location of the poles and zeros (in the z-plane) of the filter transfer function.
- *Arbitrary FIR Design*—for designing FIR filters by specifying the gain of the filter at selected (two or more) frequencies.

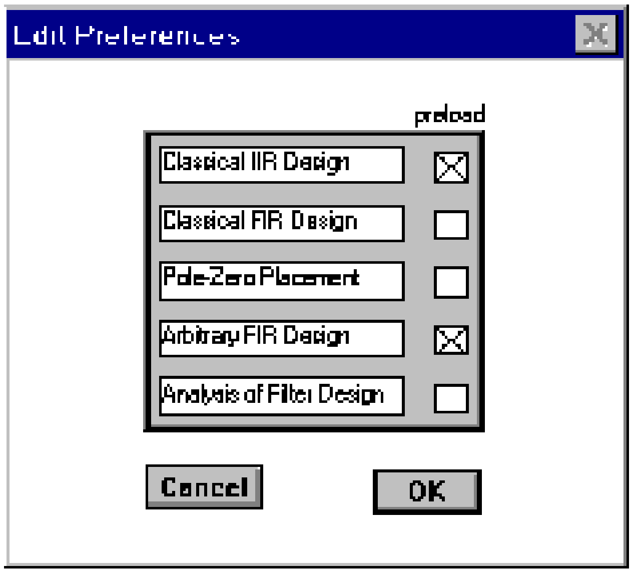
If you double-click on one of the four design selections (or single-click on a selection and then click on the **Open** button) in the Main Menu, the DFD application loads and runs the selected design panel, in which you can design your filter.

Loading Previously Saved Specifications

You can also load a previously designed filter specification file directly from the Main Menu by clicking on the **Load Filter Spec** button. You will then be prompted to select the filter specification file that you saved during a previous design work.

Customizing the DFD Application

By clicking on the **Preferences...** button in the Main Menu, you can edit your DFD application preferences for future design sessions.



Edit Preferences

preload

Classical IIR Design ☒
Classical FIR Design ☐
Pole-Zero Placement ☐
Arbitrary FIR Design ☒
Analysis of Filter Design ☐

The selections in the window above tell the DFD application to preload one or more of the filter design panels into memory when it is started. Preloading filter designs increases the time taken for the Main Menu to open. However, when you select a particular design panel from the Main Menu, the corresponding design panel opens almost immediately. If you have limited amount of memory on your computer, you may want to reconsider how many (if any) of the design panels you preload into memory.

Quitting the DFD Application

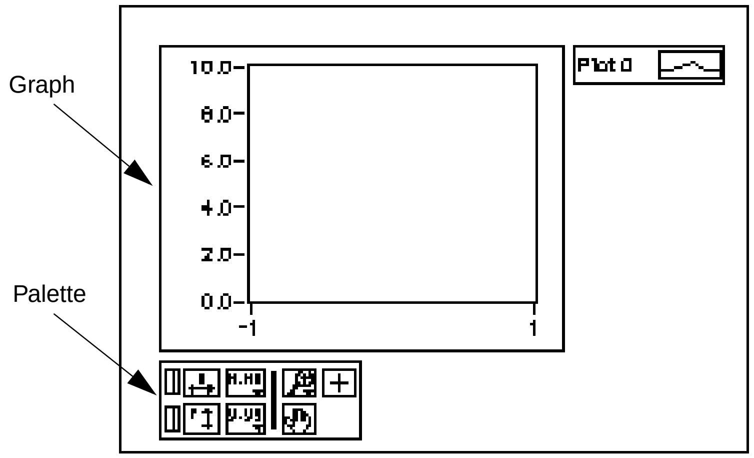
Choose the **Quit** button to exit the DFD application.

Manipulating the Graphical Display

Each design panel has a graphical display showing you the frequency response of the filter that you are designing. The graphical displays provide you with considerable flexibility in adjusting the magnitude response that you are seeing. Before you move on to the next section (Designing IIR Filters), you should become familiar with the display options.

Panning and Zooming Options

The graph palette has controls for panning (scrolling the display area of a graph) and for zooming in and out of sections of the graph. The graph palette is included with many DFD graphs. A graph with its accompanying graph palette is shown below.



If you press the x autoscale button, shown at left, the DFD application autoscales the X data of the graph. If you press the y autoscale button, shown at left, the DFD application autoscales the Y data of the graph. If you want the graph to autoscale either of the scales continuously, click on the lock switch, shown at the left, to lock autoscaling on.

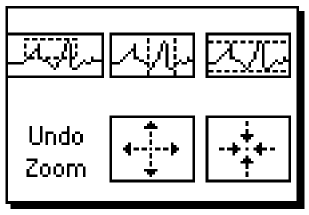
The scale format buttons, shown left, give you run-time control over the format of the X and Y scale markers, respectively.

You use the remaining three buttons to control the operation mode for the graph.

Normally, you are in standard operate mode, indicated by the plus or crosshatch. In operate mode, you can click in the graph to move cursors around.

The panning tool switches to a mode in which you can scroll the visible data by clicking and dragging sections of the graph.

The zoom tool zooms in on a section of the graph by dragging a selection rectangle around that section. If you click on the zoom tool, you get a pop-up menu you can use to choose some other methods of zooming. This menu is shown below.



A description of each of these options follows.

Zoom by rectangle.

Zoom by rectangle, with zooming restricted to x data (the y scale remains unchanged.

Zoom by rectangle, with zooming restricted to y data (the x scale remains unchanged).

Undo Zoom

Undo last zoom. Resets the graph to its previous setting.

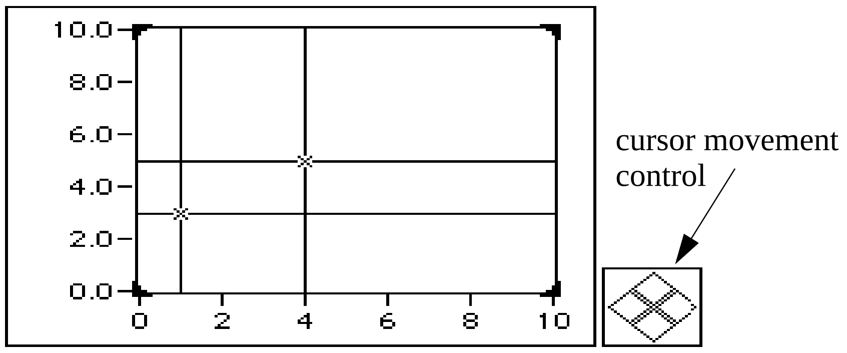
Zoom in about a point. If you hold down the mouse on a specific point, the graph continuously zooms in until you release the mouse button.

Zoom out about a point. If you hold down the mouse on a specific point, the graph continuously zooms out until you release the mouse button.

Note: ***For the last two modes, you can zoom in and zoom out about a point. Shift-clicking zooms in the other direction.***

Graph Cursors

Below are illustrations of a waveform graph showing two cursors and the cursor movement control.



You can move a cursor on a graph or chart by dragging it with the Operating tool, or by using the cursor movement control. Clicking the arrows on the cursor movement control causes all cursors selected to move in the specified direction. You select cursors by moving them on the graph with the Operating tool.

C. Designing IIR Filters

You have seen that IIR filters are digital filters whose impulse response is infinitely long. In practice, the impulse response decays to a very small value in a finite amount of time. The output of an IIR filter depends on both the previous and past inputs and the past outputs. For calculating the current output of the IIR filter, let (N_y - 1) be the number of past outputs and let (N_x - 1) be the number of past inputs. Denoting the inputs to the filter as x[.] and the outputs as y[.], the equation for the output of an IIR filter can be written as:

$$a_0 y[i] = -a_1 y[i-1] - a_2 y[i-2] + -\ldots - a_{N_y-1} y[i-(N_y-1)] + \\ b_0 x[i] + b_1 x[i-1] + b_2 x[i-2] + \ldots + b_{N_x-1} x[i-(N_x-1)]$$

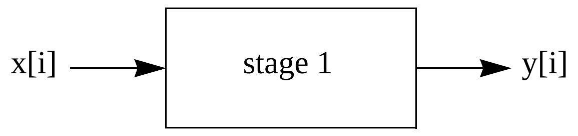
$$y[i] = \frac{1}{a0}\left(-\sum_{j=1}^{Ny-1} a[j]y[i-j] + \sum_{k=0}^{Nx-1} b[k]x[i-k]\right) \quad (1)$$

In the above equation, the b[k] are known as the *forward* coefficients and the a[j] are known as the *reverse* coefficients. The output sample at the present sample index *i* is the sum of scaled present and past inputs (*x*[i] and *x*[i-k] when *k* ¦ 0) and scaled past outputs (*y*[i-j]). Usually N_x is equal to N_y, and this value is known as the *order* of the filter.

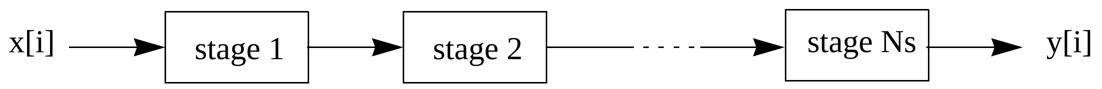
Implementation of IIR filters

IIR filters implemented in the form given by equation (1) are known as *direct form* IIR filters. Direct form implementations are usually sensitive to errors due to the number of bits used to represent the values of the coefficients (quantization error) and to the precision used in performing the computations. In addition, a filter designed to be stable can become unstable when the number of coefficients (that is, the *order* of the filter) is increased.

A less sensitive implementation is obtained by breaking up the higher order direct form implementation into an implementation that has several cascaded filter stages, but where each filter in the cascade is of a lower order, as shown in the figure below:



(a) Direct Form (of higher order)



(b) Cascaded Direct Form Filter Stages (each filter is of lower order than the filter in (a))

Typically, each lower order filter stage in the cascade form is a second-order stage. Each second-order stage can be implemented in the *direct form*, where you must maintain two past inputs (x[i-1] and x[i-2]) and two past outputs (y[i-1] and y[i-2]). The output of the filter is calculated using the equation

$$y[i] = b_0x[i] + b_1x[i-1] + b_2x[i-2] - a_1y[i-1] - a_2y[i-2] \qquad (2)$$

The implementation can also be done in the more efficient *direct form II*, where you maintain two internal states (s[i-1] and s[i-2]). The output of the filter is then calculated as follows:

$$s[i] = x[i] - a_1s[i-1] - a_2s[i-2] \qquad (3)$$

$$y[i] = b_0s[i] + b_1s[i-1] + b_2s[i-2]$$

The *direct form II* is a more efficient structure because it uses less memory. It needs to store only two past internal states (s[i-1] and s[i-2]), whereas the *direct form* structure needs to store four past values (x[i-1], x[i-2], y[i-1] and y[i-2]).

IIR Filter Designs

Depending on whether the ripple in the filter’s frequency response lies in the passband and/or the stopband, IIR filters are classified as follows:

- Butterworth: no ripple in either the passband nor the stopband.
- Chebyshev: ripples only in the passband
- Inverse Chebyshev: ripples only in the stopband.
- Elliptic: ripples in both the passband and the stopband.

The advantage of using Butterworth filters is for applications where you want a smooth filter response and no ripples. However, a higher order Butterworth filter (as compared to Chebyshev, Inverse Chebyshev, or Elliptic) is generally required for the same filter specifications. This increases the processing time for Butterworth filters.

The advantage of Chebyshev and Inverse Chebyshev filters over Butterworth filters is their sharper transition band for the same order filter. On deciding which of these two types of filters to use, the advantage of Inverse Chebyshev filters over Chebyshev filters is that they distribute the ripples in the stopband instead of in the passband.

Because elliptic filters distribute the ripples in both the passband and the stopband, they can usually be implemented with the smallest order for the same filter specifications. Hence, they have faster execution speeds than either of the other filters.

Applications of IIR Filters

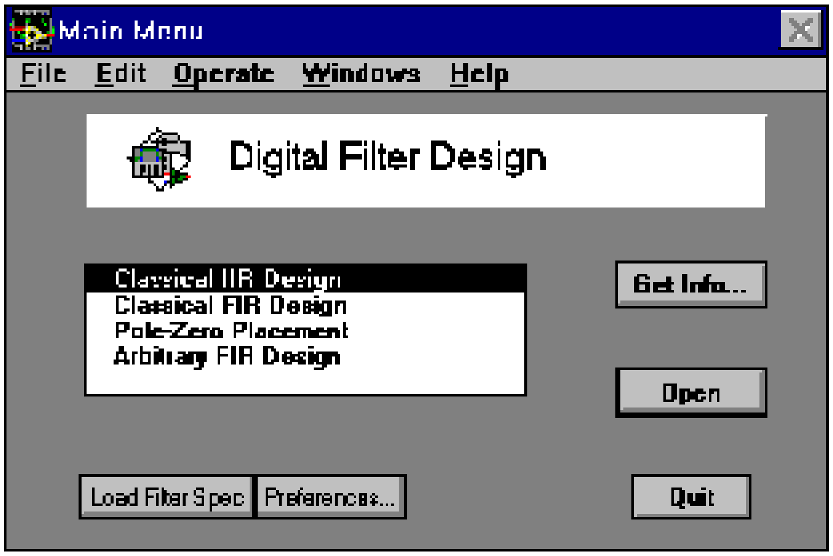
The advantage of IIR filters over FIR filters is that IIR filters usually require fewer filter coefficients to perform similar filtering operations. Thus, they execute much faster and do not require extra memory, because they execute in place.

The disadvantage of IIR filters is that they have nonlinear phase characteristics. Hence, if your application requires a linear phase response, then you should use an FIR filter instead. However, for applications where phase information is not necessary, such as in simple signal monitoring, then IIR filters can be used. Thus, the bandpass filters in real time octave analyzers are commonly IIR filters, because of their faster speed and also because it is necessary to determine the distribution of sound power over several frequency bands, but there is no need to determine the phase of the signal. The applications of such octave analyzers where phase information is not important are vibration tests of aircraft and submarines, testing of appliances, etc.

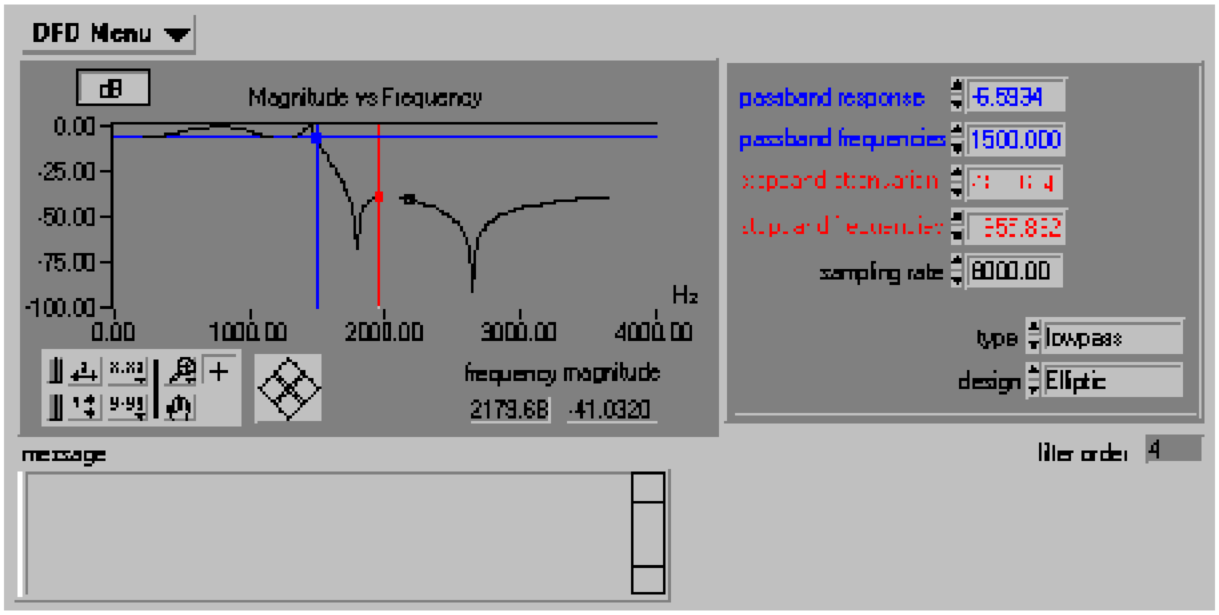
Exercise 10-1

Objective: To design an IIR bandpass filter for use in an octave analyzer.

1. Launch the **Digital Filter Design Toolkit** application. The Main Menu panel opens as shown.



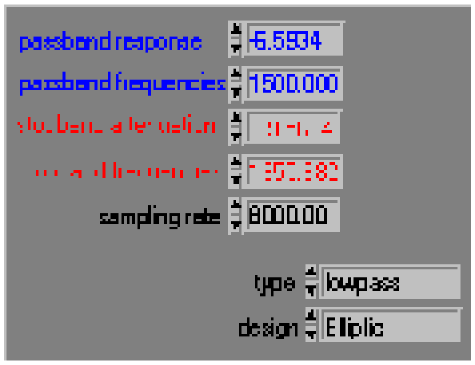
2. In the Main Menu, select Classical IIR Design and click on the **Open** button. The design panel of the Classical IIR Design opens as shown.



frequency 2179.68, magnitude -41.0320

passband response -6.5934
passband frequencies 1500.000
stopband attenuation [illegible]
stopband frequencies [illegible]
sampling rate 8000.00
type lowpass
design Elliptic
filter order 4

On the left, a plot shows the Magnitude vs. Frequency response characteristic of the filter you design. The specifications for your filter can be entered in the text entry portion at the upper right side of the design panel.



passband response -6.5934
passband frequencies 1500.000
stopband attenuation [illegible]
[illegible] [illegible]
sampling rate 8000.00
type lowpass
design Elliptic

The **passband response** is the minimum gain allowed in the passband. This is represented by the horizontal blue cursor line in the **Magnitude vs. Frequency** plot. With the reference at 0 dB, it is also the same as the **passband ripple**.

The **passband frequencies** determine the frequency edges of the passband. For lowpass and highpass filters, you have only one frequency edge. For bandpass and bandstop filters, you will have two. These frequencies are represented by the vertical blue lines in the Magnitude vs. Frequency plot.

The **stopband attenuation** is the minimum attenuation in the stopband. The horizontal red cursor line represents this attenuation in the **Magnitude vs. Frequency** plot.

The **stopband frequencies** determine the frequency edges of the stopband. For lowpass and highpass filters, you have only one frequency edge. For bandpass and bandstop filters, you will have two. These frequencies are represented by the vertical red lines in the **Magnitude vs. Frequency** plot.

The **sampling rate** control specifies the sampling rate in samples per second (Hz).

The **type** control specifies one of the four classical filter types:

- Lowpass
- Highpass
- Bandpass
- Bandstop

The **design** control specifies one of the four classical filter design algorithms:

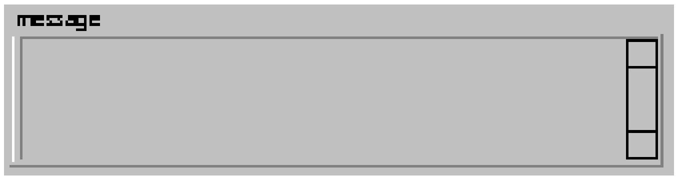
- Butterworth
- Chebyshev

- Inverse Chebyshev
- Elliptic

Below the text entry portion is an indicator showing the order of the IIR filter.

The DFD application automatically estimates the filter order to be the lowest possible order that meets or exceeds the desired filter specifications.

At the bottom left of the Classical IIR Design panel is the **message** window where error messages are displayed.



You will use the Classical IIR Design panel to design an IIR bandpass filter that can be used in an octave analyzer. Octave analyzers (see the lesson on the Third-Octave Analyzer Toolkit) are used in applications where you need to determine how the signal power is distributed over a particular frequency range. These applications include the fields of architectural acoustics, noise and vibration tests in aircraft and submarines, testing of household appliances, etc.

An octave analyzer uses bandpass filters to separate the signal power into several frequency bands. The American National Standards Institute requires that these filters adhere to certain specifications. Some specifications for one of these filters are:

fp1 = 890.90 Hz

fp2 = 1122.46 Hz

maximum passband ripple ð 50 millibels

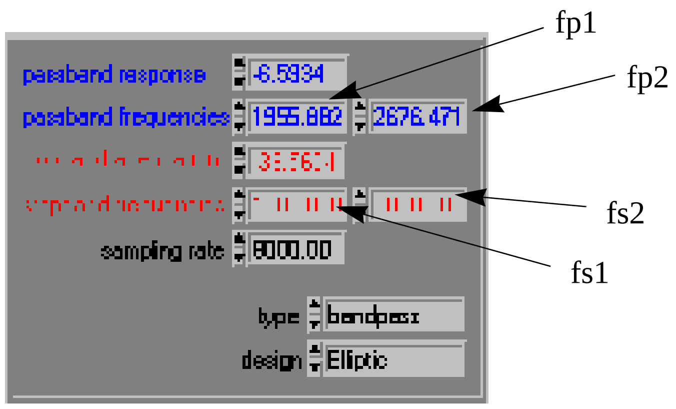
fs1 = 120.48

fs2 = 8300

stopband attenuation Š 65 dB

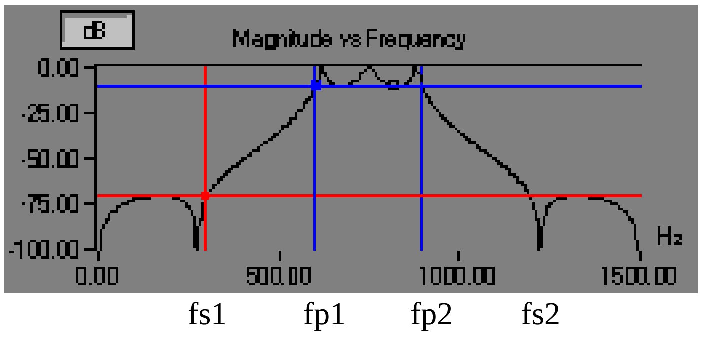
Because the purpose of the bandpass filter is to determine the level of sound power in a particular frequency band, and the phase information in the signal is not being used, it is not necessary for the filter to be linear phase. Hence, you can choose an IIR filter for this application. You will use the DFD toolkit to design the IIR filter to meet these specifications.

3. Change the type control in the text entry box to *bandpass*.



The explanation of the controls in the text entry box is the same as before, except note that because you have selected a bandpass filter, there are two controls for the **passband frequencies** and the **stopband frequencies**. They are denoted by fp1, fp2, fs1, and fs2, as shown in the following figure, and bear the following relationship:

fs1 < fp1 < fp2 < fs2

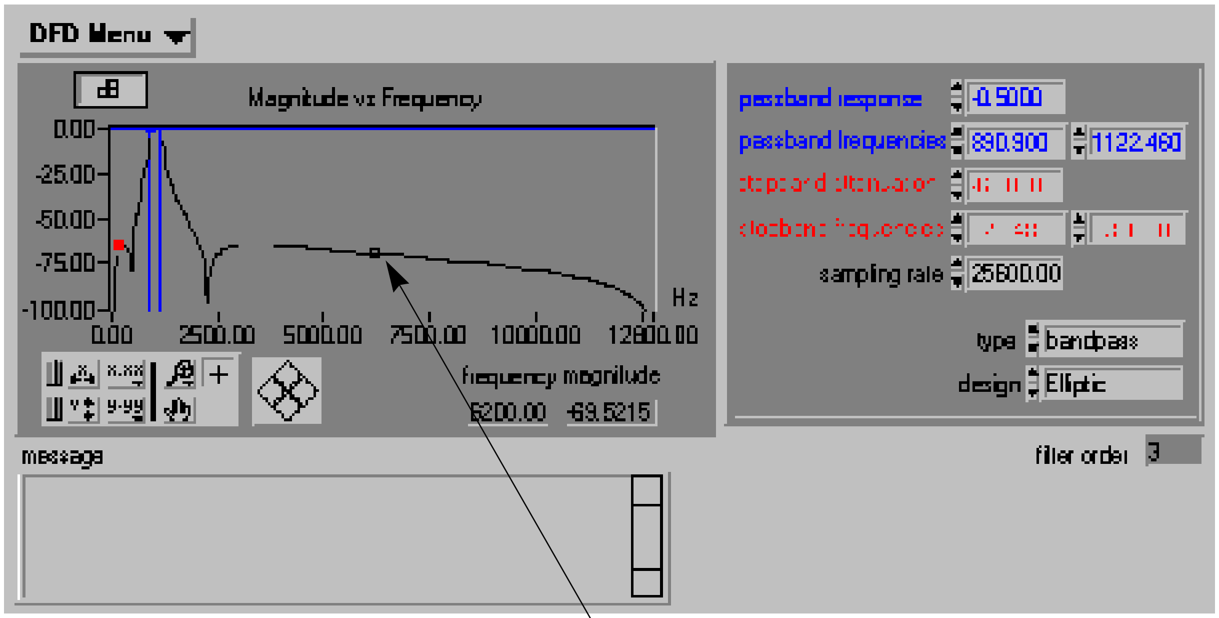


passband and stopband frequencies for a bandpass filter

4. Looking at the specifications in step 2, enter the following values in the controls in the text entry box

passband response	-0.5
passband frequencies	890.90 and 1122.46
stopband attenuation	-65
stopband frequencies	120.48 and 8300
sampling rate	25600
type	bandpass
design	elliptic

Note that 50 millibels = $\frac{50}{1000}$ Bels = $\frac{50}{100}$ deciBels = 0.5 dB. That is why you entered the passband response as -0.5 dB.



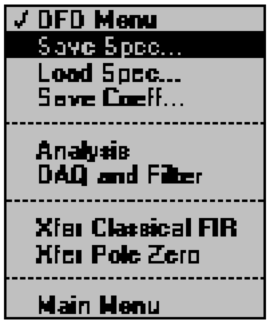
frequency 6200.00
magnitude -69.5215

passband response -0.5000
passband frequencies 890.900 1122.460
stopband attenuation [illegible]
stopband frequencies [illegible] [illegible]
sampling rate 25600.00
type bandpass
design Elliptic
filter order 3

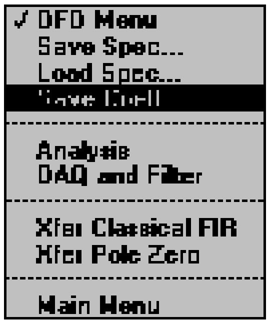
5. On the **Magnitude Vs. Frequency** graph is a tracking square cursor that you can move around. The frequency and the corresponding magnitude of the point where the cursor is placed, is displayed on the frequency and magnitude indicators below the graph. Move the square cursor to the passband region and verify that the attenuation in the passband is never below -0.5 dB, as was specified in the passband response control.

6. The default filter design is Elliptic. Change the filter design one by one to Butterworth, Chebyshev, and Inverse Chebyshev. Note the order of the filter. For the same filter order, which of the four filter designs has the sharpest transition region?

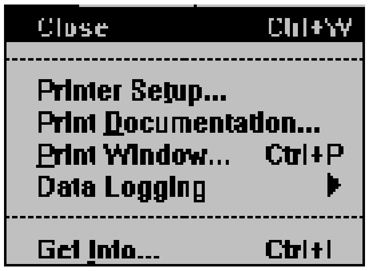
7. You can save the specifications of the filter in a file for later use. From the **DFD** menu, select **Save Spec...** When asked for the name of the file in which to save the filter specifications, type in `bandpass.iir`. In Exercise 10-3, you will load this file and analyze the characteristics of the filter that you have just designed.



8. You can also save the designed filter coefficients in a file for later use with the DFD toolkit, or with other programs. From the **DFD** menu, select **Save Coeff...**. When asked for the name of the file in which to save the filter coefficients, type `bpiir.txt`. Save the file as a *text* file. The appendix gives the format of the text file in which the coefficients are stored.



9. Now that you have saved the filter specifications, and the filter coefficients, you can close the application. Select **File » Close** to close the Classical IIR Design panel. Then select **Quit** in the Main Menu to exit the DFD Toolkit.



End of Exercise 10-1

D. Designing FIR Filters

As opposed to IIR filters, whose output depends on both its inputs and outputs, the output of an FIR filter depends only on its inputs. Because the current output is independent of past outputs, its impulse response is of finite length. The output of a general FIR filter is given by

$$y[i] = b_0x[i] + b_1x[i-1] + b_2x[i-2] + \dots\dots.. + b_Nx[i-N] \qquad (4)$$

where N is the order of the filter and b_0, b_1, ... b_N, are its coefficients.

FIR filters have certain advantages as compared to IIR filters.

- They can achieve linear phase response, and hence they can pass a signal without phase distortion.
- They are always stable. During filter design or development, you do not need to worry about stability concerns.
- FIR filters are simpler and easier to implement.

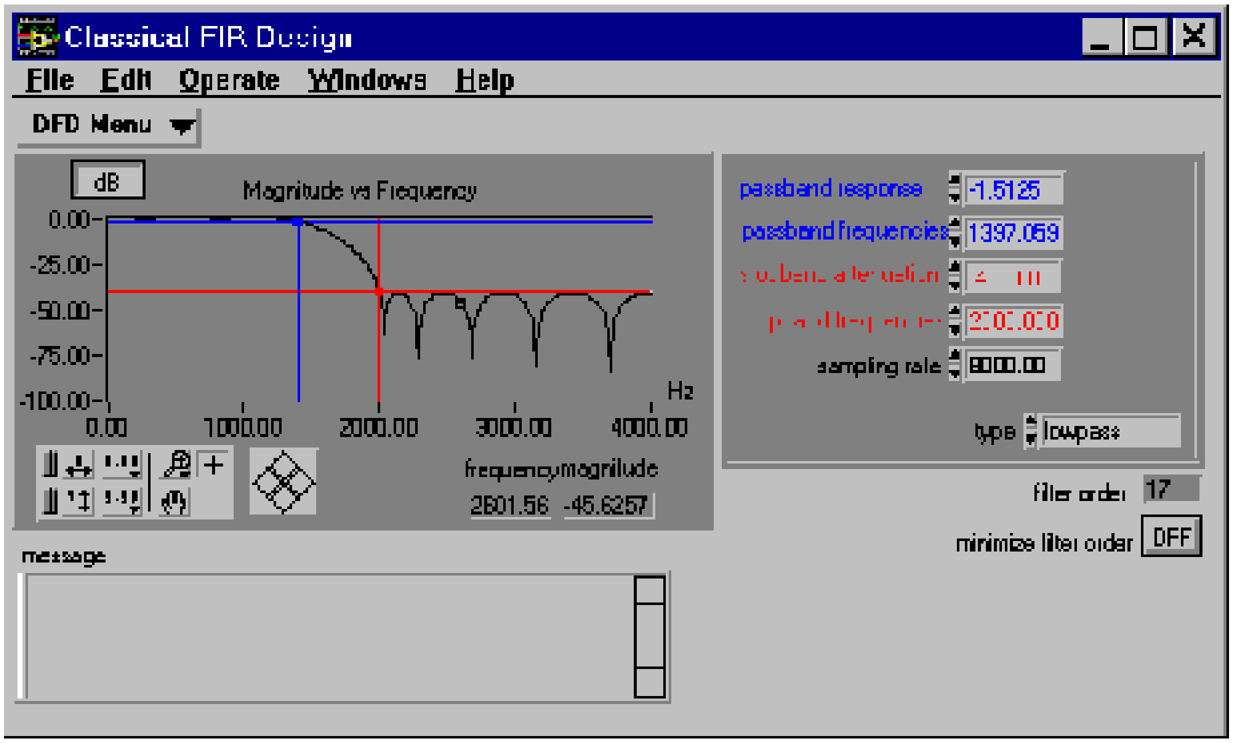
Applications of FIR filters

Many applications require the filters to be linear phase. In this case, you should use FIR filters. However, FIR filters generally need to be of a higher order than IIR filters, to achieve the same magnitude response characteristics. So, if linear phase is not necessary, but speed is an important consideration, you can use IIR filters instead.

In applications where a signal needs to be reconstructed after it has been split up into several frequency bands, it is important that the filter is linear phase. In the reconstruction process, the loss of phase could result in the reconstructed signal being quite different from the original one. An example of such an application is in the Wavelet and Filter Banks Design toolkit, where you reconstruct the original signal (either a 1D waveform or a 2D image) from the wavelet coefficients. The filters used are FIR filters. The applications are in areas where phase information is an important consideration, such as noise removal, data compression, etc.

Classical FIR Design and Arbitrary FIR Design

Using the DFD Toolkit, you can design FIR filters in the Classical FIR Design panel (shown below) which is very similar to the Classical IIR Design panel that you have seen.



frequency/magnitude 2601.56 -45.6257

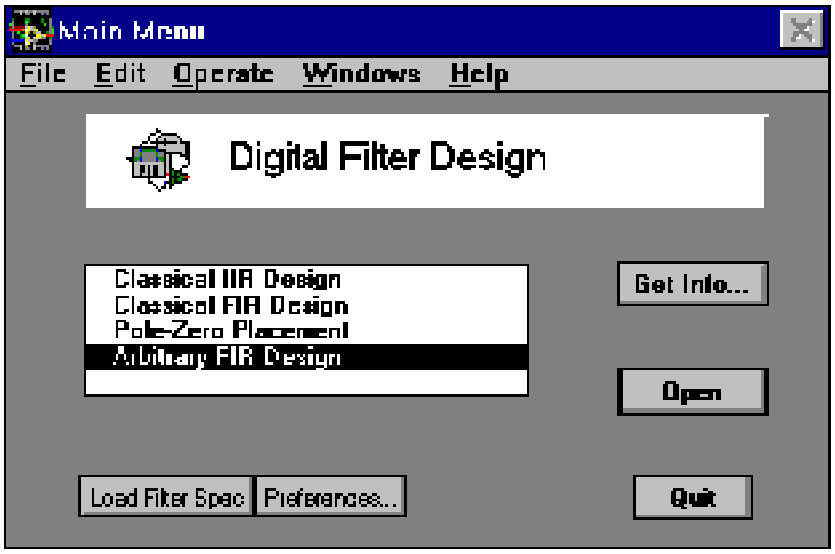
passband response -1.5125
passband frequencies 1397.059
stopband attenuation [illegible]
stopband frequencies [illegible]
sampling rate 8000.00
type lowpass
filter order 17
minimize filter order OFF

The panel includes a graphical interface with the **Magnitude vs. Frequency** graph and cursors on the left side, and a text-based interface with digital controls on the right side. The differences are the absence of the *design* control in the text entry box and the addition of the minimize filter order control below the text entry box.

This button controls 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.

The FIR filters that are designed use the Parks-McClellan equiripple FIR filter design algorithm and include the lowpass, highpass, bandpass, and bandstop types. The Parks-McClellan algorithm minimizes the difference between the desired and actual filter response across the entire frequency range.

You can also design FIR filters with an arbitrary frequency response by selecting Arbitrary FIR Design in the Main Menu.



“Arbitrary” means that you can specify exactly what the magnitude of the filter response should be at specific frequencies. In the next exercise, you will design an FIR filter by specifying an arbitrary frequency response.

Exercise 10-2

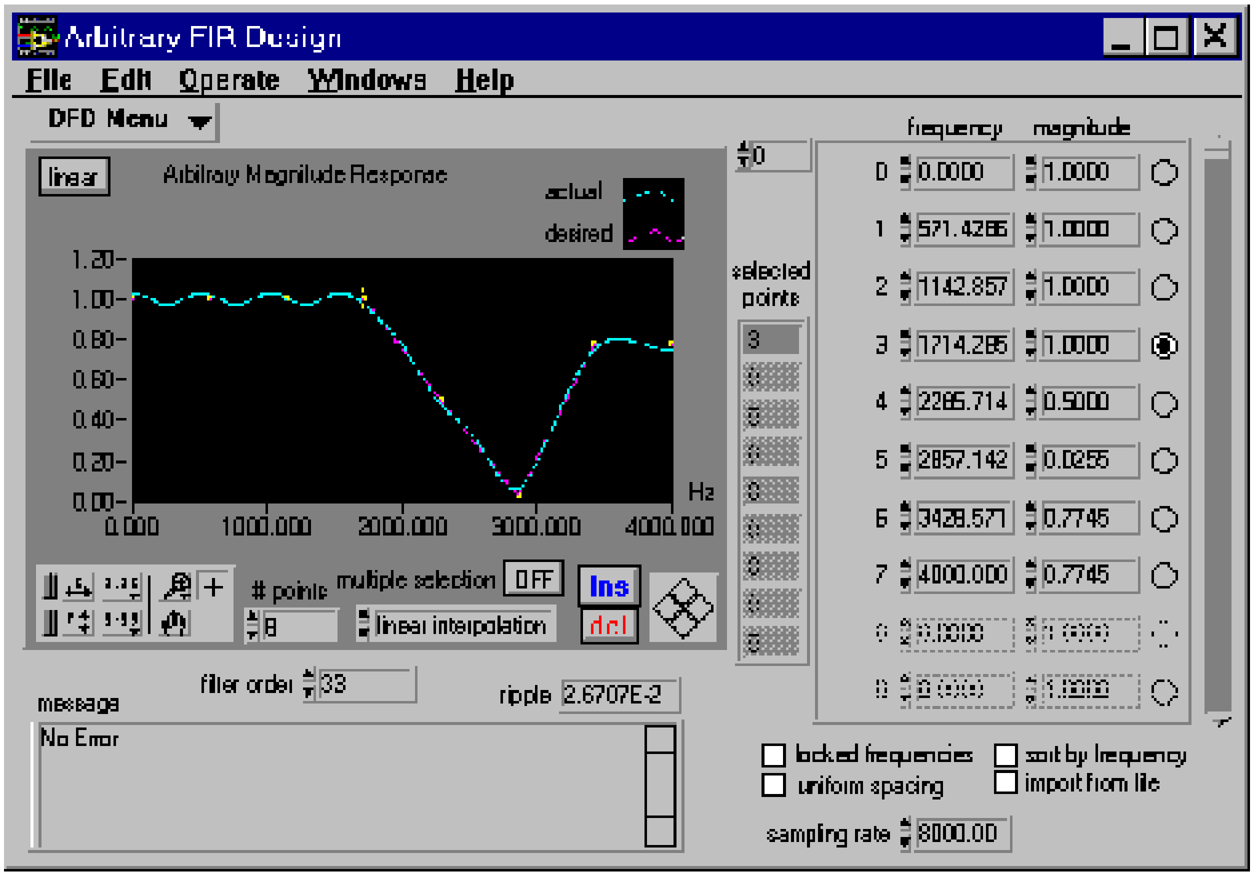
Objective: To design an FIR filter which filters the data according to A-weighting.

The human sense of hearing responds differently to different frequencies and does not perceive sound equally. Certain filters are used to filter the sound applied at their input such that they mimic the human hearing response to audio signals. An application of this is in third-octave analyzers, where, to mimic the response of the human ear, the analyzer output is weighted according to the table shown below:

Frequency (Hz)	Weighting (dB)	Frequency (Hz)	Weighting (dB)
10	-70.4	500	-3.2
12.5	-63.4	630	-1.9
16	-56.7	800	-0.8
20	-50.5	1000	0
25	-44.7	1250	+0.6
31.5	-39.4	1600	+1.0
40	-34.6	2000	+1.2
50	-30.3	2500	+1.3
63	-26.2	3150	+1.2
80	-22.5	4000	+1.0
100	-19.1	5000	+0.5
125	-16.1	6300	-0.1
160	-13.4	8000	-1.1
200	-10.9	10000	-2.5
250	-8.6	12500	-4.3
315	-6.6		
400	-4.8		

This type of weighting is known as *A-weighting*.

1. Launch the DFD Toolkit.
2. Select **Arbitrary FIR Design** in the **Main Menu**. You will access the following design panel:



	frequency	magnitude
0	0.0000	1.0000
1	571.4286	1.0000
2	1142.857	1.0000
3	1714.286	1.0000
4	2285.714	0.5000
5	2857.142	0.0255
6	3428.571	0.7745
7	4000.000	0.7745

# points 8
multiple selection OFF
linear interpolation
filter order 33
ripple 2.6707E-2
message
No Error
☐ locked frequencies ☐ sort by frequency
☐ uniform spacing ☐ import from file
sampling rate 8000.00

The panel includes a graphical interface with the **Magnitude vs. Frequency** graph on the left side and a text-based interface with digital controls on the right side. In the array on the right hand side, you can 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 half the sampling rate. The DFD application then takes this desired response, along with the filter order, and uses the Parks-McClellan algorithm to design an optimal equiripple FIR filter.

The graph below plots the desired and actual magnitude response of the designed FIR filter.

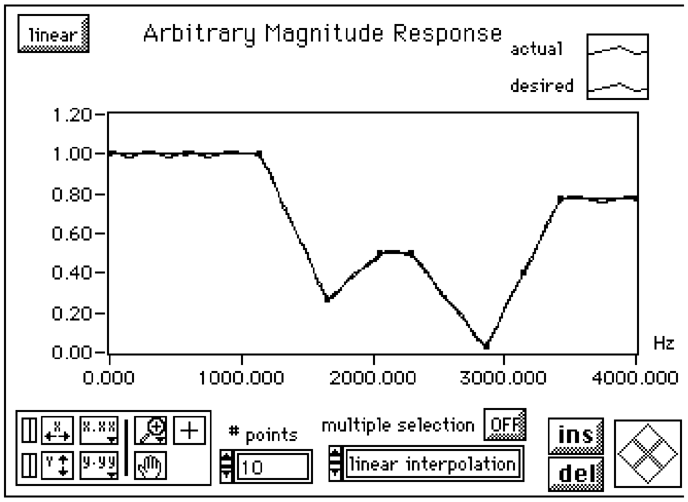


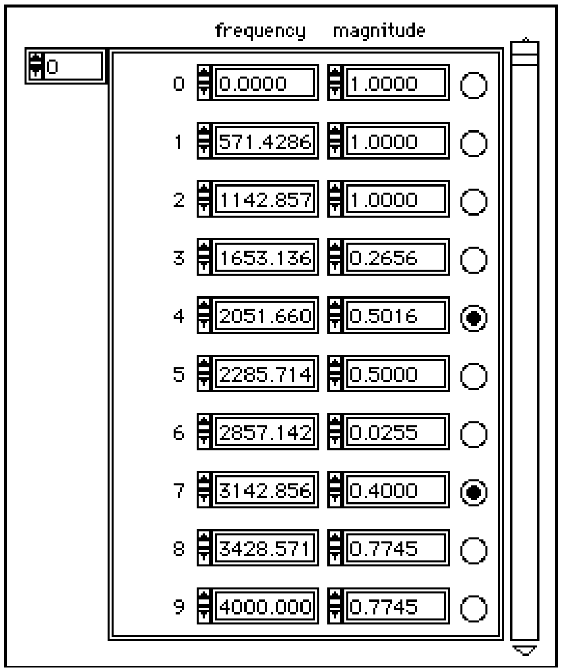
Figure 10-1. Desired and Actual Magnitude Response

The y-axis is in linear or decibel units, depending on how you set the button in the upper left corner of the graph. The x-axis is in Hertz. The full scale ranges from 0.0 to Nyquist (sampling rate/2).

dB

The **dB** button controls the display units (linear or decibel) of all the magnitude controls and displays. These controls and displays include the **Magnitude vs. Frequency** graph (y-axis) and the magnitudes in the array of frequency-magnitude points.

The following array is 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.



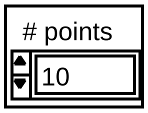
0

	frequency	magnitude	
0	0.0000	1.0000	○
1	571.4286	1.0000	○
2	1142.857	1.0000	○
3	1653.136	0.2656	○
4	2051.660	0.5016	●
5	2285.714	0.5000	○
6	2857.142	0.0255	○
7	3142.856	0.4000	●
8	3428.571	0.7745	○
9	4000.000	0.7745	○

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

You can select points in this array by clicking in the circle to the right of each point. You can then delete the selected points by clicking on the delete button, or move them by clicking on the desired direction diamond in the lower right corner of the **Arbitrary Magnitude Response** graph.

The **# points** control specifies the total number of frequency-magnitude points the DFD application uses to create the desired filter magnitude response.

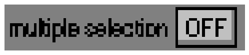


# points
10

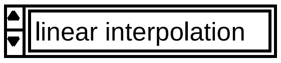
Reducing this number deletes points from the end of the frequency-magnitude array, while increasing this number, inserts the additional number of points to the right of the selected point.

If you want to select more than one frequency-magnitude point on the response graph, you should set the multiple selection button to ON.

Clicking on a point you already selected removes that point from the selection list.



The interpolation control selects the type of interpolation the DFD application uses to generate the desired response from the array of frequency-magnitude points.



Choose linear interpolation to create "flat" filters (lowpass, highpass, bandpass, and bandstop). Choose spline interpolation to create smoothly-varying filters.

ins

To insert a frequency-magnitude point between the selected point and the next point, click on the **ins** button.

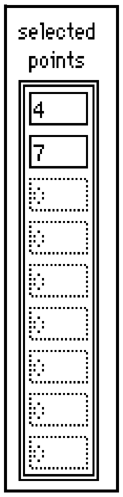
If the selected point is the last point in the frequency-magnitude array, the DFD application inserts the new point between the last two points of the array.

The DFD application inserts new points at halfway along the line connecting the two outer points.

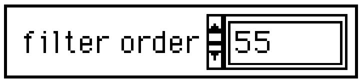
del

To delete the selected frequency-magnitude points, click on the del button. The DFD application deletes all selected points.

These points are the selected frequency-magnitude points. You can select points on the **Arbitrary Magnitude Response** graph by clicking on the point, or directly from the frequency-magnitude array shown at the right by clicking on the circle to the right of each point.

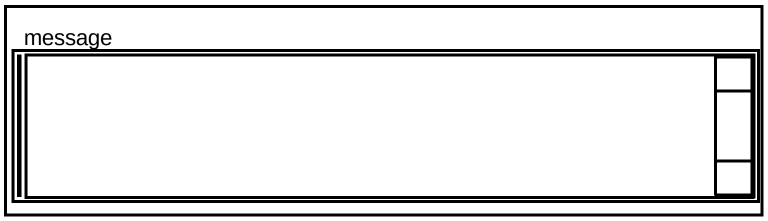


The filter order control specifies the total number of coefficients in the digital FIR filter.

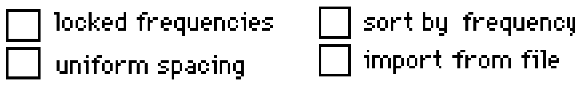


The ripple indicator displays the largest absolute error (linear) between the desired and actual filter responses.

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



The locked frequencies box allows you to lock the present frequency values of the frequency-magnitude points. If you click in this box, you can alter only the magnitude or y-value of the frequency-magnitude points.

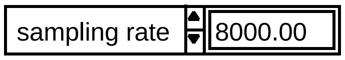


The uniform spacing box is used to space the frequency values of the frequency-magnitude points. If you click in this box, the DFD application spaces the frequency-magnitude points uniformly from 0.0 to sampling rate/2, inclusive.

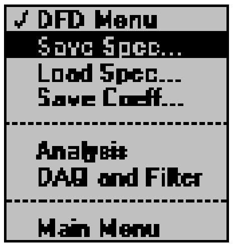
Clicking in the sort by frequency box tells the DFD application to sort the frequency-magnitude points in both the response graph and the array according to ascending frequency. The value of each frequency-magnitude point remains unchanged; however, the point order may change.

Clicking on import from file enables you to import frequency-magnitude points from a text file.

The sampling rate control specifies the sampling rate in samples per second (Hertz).

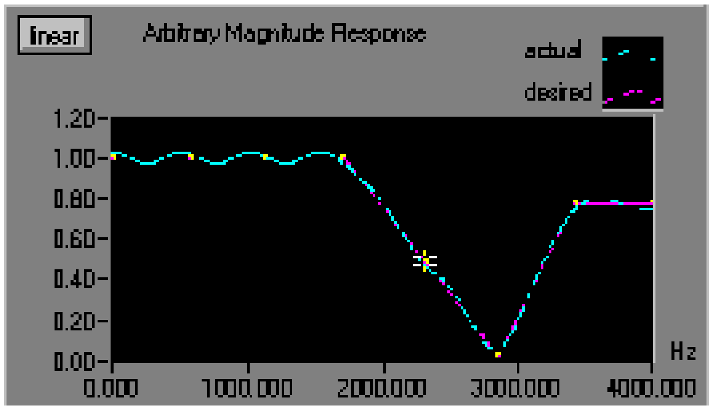


3. The filter specifications shown in the table on page 10-21 have been saved in the `Dfd\Aweight.fir` file. Load these specifications by selecting **DFD Menu » Load Spec...** and choosing the `Aweight.fir` file.
4. You could also have entered these values directly on the front panel and then saved them by selecting **Save Specs...** from the **DFD Menu**.



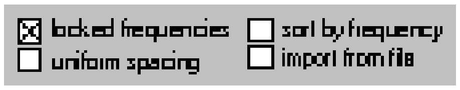
When prompted for a file name, type `Aweight.fir`.

5. With the multiple selection control set to OFF, move some of the points on the **Arbitrary Magnitude Response** graph. To do this, move your cursor close to a point till the cursor changes shape, as shown below:



Hold down the left mouse button and move the point. Observe how the response of the filter changes as the point is moved.

6. Choose *locked frequencies* and try to do the same as in the previous step. Now you should be able to change the magnitude of the selected point, but not its frequency.



7. Change the methods of interpolation between *linear interpolation* and *spline interpolation*. Observe the difference in the shape of the filter response. Linear interpolation is used to create “flat” filters, such as lowpass, highpass, bandpass, and bandstop filters. Spline interpolation is used to create smoothly-varying filters.
8. Close the Arbitrary FIR Design panel by selecting **Close** from the **File** menu.
9. Quit the application by selecting **Quit** from the Main Menu.

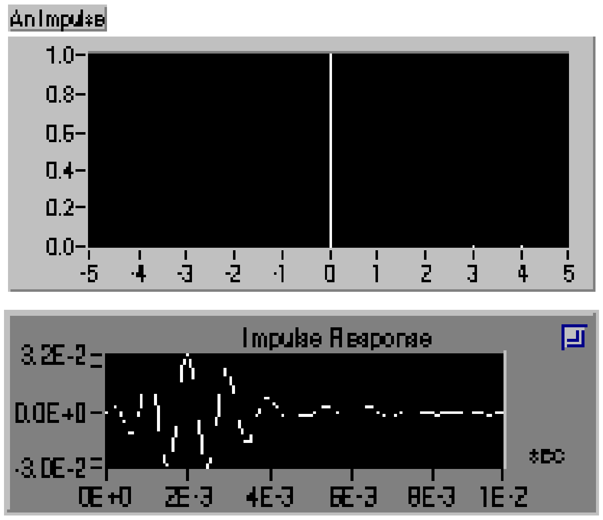
End of Exercise 10-2

E. Analyzing your Filter Design

After designing your IIR or FIR filter, you can analyze your design in several ways. For example, you can see the effect that the filter has on the amplitude and phase of input signals at different frequencies by observing its magnitude and phase responses. Several types of analysis methods are available in the Digital Filter Design Toolkit. These are explained below. You will first look at the response of the filter to special kinds of input signals.

Impulse Response

The output of the filter when the input is an impulse is known as the *impulse response* of the filter. An impulse in the digital world has an amplitude of 1 at index 0 and an amplitude of 0 for all other indices. An impulse and the impulse response are shown in the figures below.

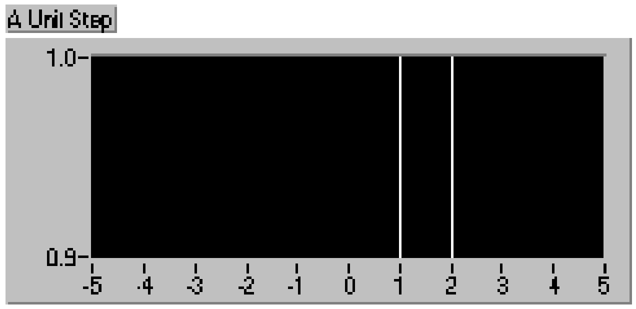


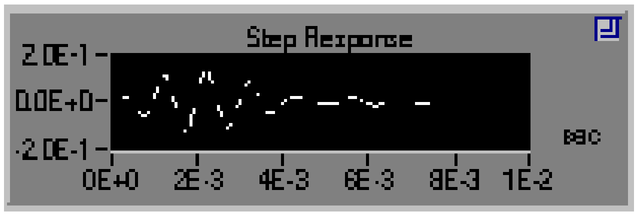
The impulse response has a very special meaning in the case of FIR filters. The impulse response of an FIR filter gives the coefficients of that filter. Thus, the impulse response is a useful method for determining the coefficients of an FIR filter. Furthermore, the number of nonzero terms in the impulse response gives the number of coefficients in the filter. (For an IIR filter, the relationship is much more complicated, and the above discussion does not apply.)

Another use of the impulse response is that, for both FIR and IIR filters, the output of the filter is given by the convolution of the input signal and the impulse response of the filter.

Step Response

The output of the filter when the input is a unit step is known as the *step response* of the filter. A unit step in the digital world has an amplitude of 0 for all negative indices, and an amplitude of 1 at index zero and for all positive indices. A unit step and the step response are shown in the figures below.

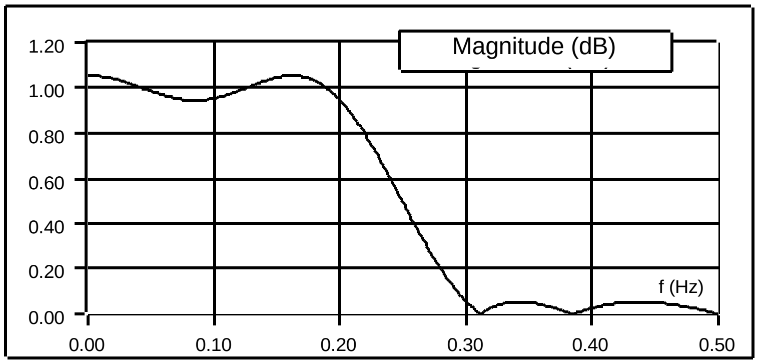


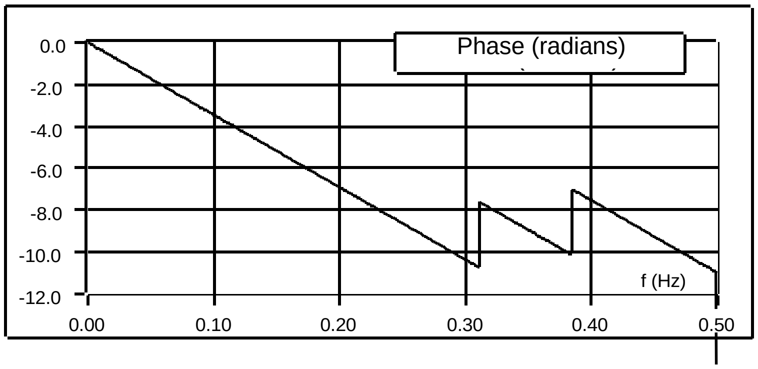


The step response is important if you will use the filter in a control system. You can then see how the parameters of the control system (such as the rise time, overshoot, etc.) are affected by the filter. The step response also shows you how long the filter will take to respond to a sudden change in the input.

Frequency Response (Magnitude Response and Phase Response)

The frequency response in useful in that it shows the effect that the filter has on the amplitude and phase of input signals at different frequencies. Because the filter can affect both the magnitude and the phase of the input signal, the frequency response consists of two parts—the magnitude response and the phase response. An example of these responses is shown below. The x-axis units are normalized in terms of the sampling frequency.





Not only does the frequency response enlighten you as to the effect that the filter has on signals of specific frequencies, it also allows you to determine what happens to arbitrary signals as they pass through the filter. Because most signals can be expressed as a sum of exponentials (sines and cosines), you can break a signal down into its individual components and determine the effect of the filter on those components.

The Z-Domain: Transfer Function H(z) and the Pole-Zero Plot

The *transfer function*, H(z), of a digital filter can be expressed as a ratio of polynomials,

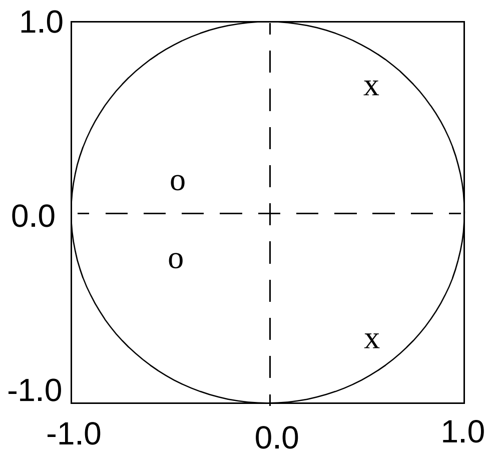
$$H(z) = \frac{N(z)}{D(z)}$$

where N(z) is a numerator polynomial, and D(z) is the denominator polynomial. (For an FIR filter, D(z) = 1.) H(z) is also known as the *z-transform* of the filter.

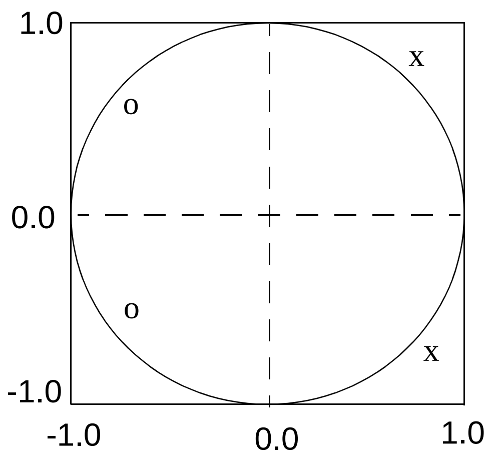
The values of *z* at which N(z) is equal to zero are known as the *zeros* of the filter, because for these values, H(z) is also equal to zero. The values of *z* at which D(z) is equal to zero are known as the *poles* of the filter, because at these values, H(z) is equal to infinity. A plot of the poles and zeros of the filter is known as the *pole-zero plot*. Because *z* is a complex number, the pole-zero plot is shown in terms of the real part of *z* on the *x*-axis and the imaginary part of *z* on the *y*-axis.

The pole-zero plot is useful in determining the stability of the filter. As long as all the poles of the filter have a magnitude less than one, the filter is stable. If any of the poles of the filter have a magnitude greater than one, the filter will be unstable. That means that the output of the filter will continue to grow indefinitely even if the input is no longer applied. The values of *z* for which its magnitude is equal to one is drawn on the pole-zero plot as a circle having its center at the origin and having radius equal to one. Thus, so long as the poles of the filter lie inside the circle, the filter will be stable. If even one of the poles lie outside the circle, the filter will be unstable.

The Digital Filter Design Toolkit gives you both the transfer function of the filter and its corresponding pole-zero plot. The figures below show the pole-zero plots for both a stable and an unstable filter. Each *o* depicts a zero and each *x* depicts a pole.



stable filter



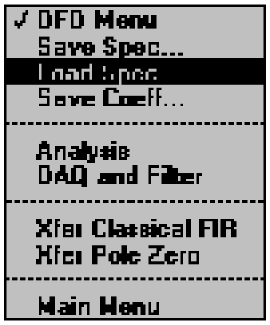
unstable filter

Exercise 10-3

Objective: To analyze the design of an IIR filter.

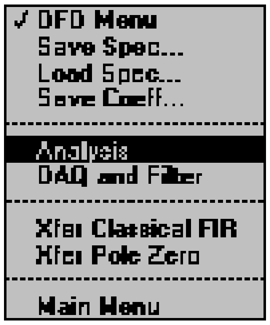
You will load the filter specifications you saved in Exercise 10-1 in the file `bandpass.iir` and see its impulse and step responses, and the pole-zero plot of its transfer function.

1. Open the DFD application.
2. Choose the Classical IIR Design panel.
3. When the Classical IIR Design panel opens, select **Load Spec...** from the **DFD Menu**.

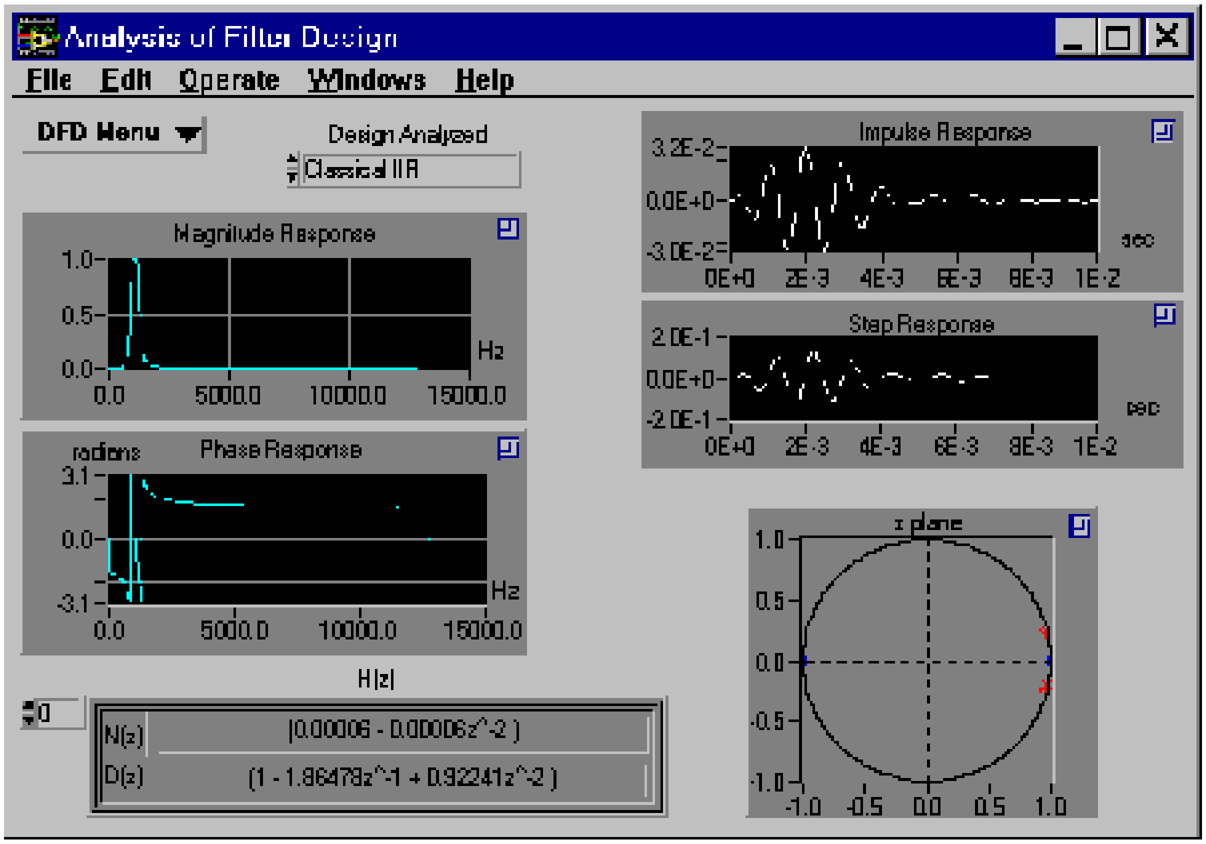


When prompted for the filename, select `bandpass.iir`.

4. From the **DFD** menu, select **Analysis**.



The Analysis of Filter Design window opens, as shown below.


Analysis of Filter Design
File Edit Operate Windows Help
DFD Menu
Design Analyzed
Classical IIR
Magnitude Response
Phase Response
radians
Hz
Impulse Response
Step Response
sec
z plane
H(z)
N(z) (0.00006 - 0.00006z^-2)
D(z) (1 - 1.86478z^-1 + 0.92241z^-2)

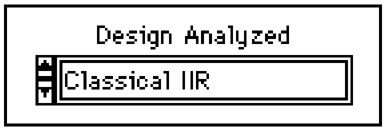
In this panel, you can view the filter magnitude response, phase response, impulse response, step response, and pole-zero plot of the filter you designed in the first exercise. You can also view and print full-screen plots of each response. From the full-screen views, you can save the analysis results to text files.

If you select **DFD Menu » Analysis** from a filter design panel, the Analysis of Filter Design panel uses that particular filter design to compute the various filter responses. You can also analyze any of the four filter designs from the **Design Analyzed** ring selector; the Analysis of Filter Design panel uses the filter parameters from the selected filter design.

DFD Menu

The DFD Menu can be used to load filter designs from previous work, open the DAQ and Filter panel, go to the selected filter design panel, or return to the Filter Design Main Menu.

The **Design Analyzed** control selects which filter control to analyze. If you continue to modify the same filter design that is presently being analyzed, the DFD will recompute all filter responses.



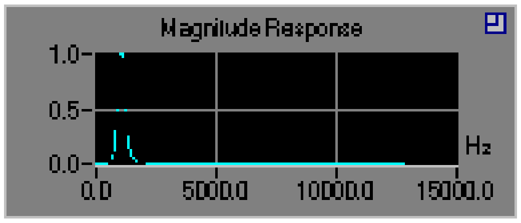
Analysis Displays

zoom box

Each of the five filter plots has a zoom box in the upper right corner. Clicking in this box brings up a full-screen version of that plot. In the full-screen versions of these plots, you can change the units from linear to decibel (magnitude response), from radians to degrees (phase response) or from seconds to samples (impulse and step responses). From each full-screen view, you can save the response data to text files.

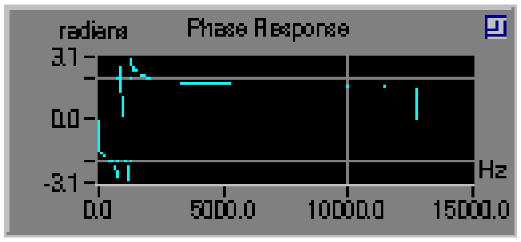
Magnitude Response

5. The magnitude response is the magnitude of the filter's response H(f) as frequency varies from zero to half the sampling rate. Look at the magnitude response of the designed filter. You can see that it is indeed a bandpass filter.



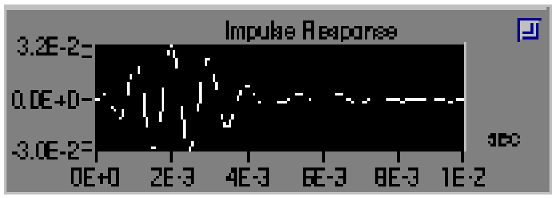
Phase Response

6. The phase response is the phase of the filter's response H(f) as frequency varies from zero to the sampling rate. The following figure illustrates the phase response of the selected filter design. Note that the phase is displayed in radians. You can obtain a display in degrees by clicking on the zoom box of the Phase Response plot. A new window will appear, which will give you the option of choosing between the appropriate display units.



Impulse Response

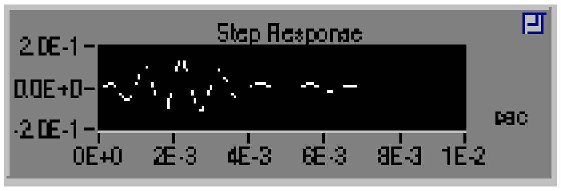
7. The impulse response of a digital filter is the filter’s output when the input is a unit sample sequence (1, 0, 0, ...). The input before the unity sample is also zero. The following figure shows the impulse response of the selected filter design.



   Observe that although it is an IIR filter, the impulse response decays toward zero after a finite amount of time.

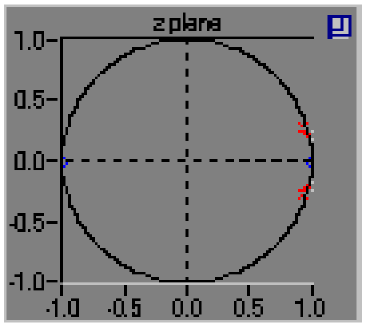
Step Response

8. The step response of a digital filter is the filter’s output when the input is a unit step sequence (1, 1, 1, ...). The input samples before the step sequence are defined as zero. The following figure shows the step response of the designed filter.



Z-Plane Plot

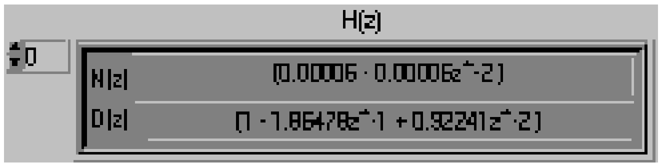
9. The following figure illustrates the z-plane plot of the filter poles and zeros.



   Each pole is represented by a red *x*. Each zero is represented by a blue *o*. Compare this with the **Magnitude Response** of the filter and observe that the zeros are at 0 and π, which correspond to frequencies 0 and Nyquist (fs/2). The location of the poles is about 1000 Hz, which is why there is a peak in the magnitude response around 1000 Hz.

H(z) for IIR Filters

10. H(z) is the z-transform of the designed digital filter.



H(z)

0

N(z) [0.00006 - 0.00006z^-2]

D(z) [1 -1.86478z^-1 +0.92241z^-2]

Previously, you saw that the IIR filters are implemented as cascaded second-order stages. For an IIR filter, H(z) can be represented by a product of fractions of second-order z polynomials.

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

$N_k(z)$ = numerator for stage *k*

$D_k(z)$ = denominator for stage *k*

N_s = 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 corner of the H(z) display.

11. Close the panel by selecting **Close** from the **File** menu. Quit the DFD application.

End of Exercise 10-3

F. Format of Filter Coefficient Text Files

When you save your filter coefficients to a text file, the DFD application generates a readable text file containing all the information you need to implement the designed FIR or IIR digital filter. This section details the format for both FIR and IIR filter coefficient files.

FIR Coefficient File Format

The following table gives an example FIR coefficient text file and description:

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	
-2.002337E-1	
1.516199E-1	
1.516199E-1	
-2.002337E-1	
-5.984635E-2	
1.810791E-1	

coefficient file example	description
-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]

You can implement the FIR filter using equation (4) directly.

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. There are a total of M * S *a* coefficients and (M+1)*S *b* coefficients, where M is the stage order (usually 2) 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.

The following table gives an example IIR coefficient text file and description:

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 a coefficients
3.801467E-1	a1 for stage 1

coefficient file example	description
8.754090E-1	a2 for stage 1
-1.021050E-1	a1 for stage 2
9.492741E-1	a2 for stage 2
8.460304E-1	a1 for stage 3
9.540986E-1	a2 for stage 3
b Coefficients	b coefficients label
9	number of b coefficients
1.514603E-2	b0 for stage 1
0.000000E+0	b1 for stage 1
1.514603E-2	b2 for stage 1
1.000000E+0	b0 for stage 2
6.618322E-1	b1 for stage 2
1.000000E+0	b2 for stage 2
1.000000E+0	b0 for stage 3
1.276187E+0	b1 for stage 3
1.000000E+0	b2 for stage 3

You can implement the IIR filter in cascade stages by using equation (2) (maintaining two past inputs and two past outputs for each stage), or by using the direct form II equations (maintaining two past internal states), as in equation (3).

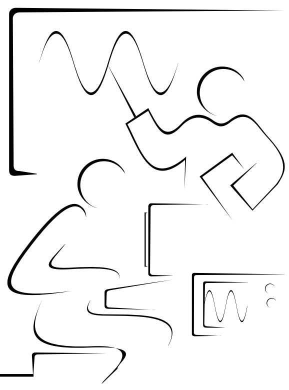
Summary

You have learned that FIR filters are used for applications where you need a linear phase response, such as for applications that require reconstructing the original waveform after filtering, noise removal, and data compression. For applications where phase is not an important consideration (such as for simple signal monitoring) and where faster speeds are necessary, you can use IIR filters.

You saw that the DFD toolkit allows you to interactively design both FIR and IIR filters. The design could be done either by specifying the filter parameters (classical IIR design and classical FIR design), deciding the location of the poles and zeros in the z-plane (pole-zero placement), or arbitrarily specifying the magnitude response characteristics (arbitrary FIR design) of the filter.

After the filter has been designed, you can analyze your filter in terms of its magnitude and phase responses, impulse and step responses, and the pole-zero plot. You can also save the filter coefficients for use in other applications.

Notes



Lesson 11
G Math Toolkit

Introduction

The G Math Toolkit offers a new paradigm for mathematics, *numerical recipes in G,* with hundreds of math VIs for solving differential equations, optimization, root finding, and so on. All VIs in the G Math Toolkit are written in G, so you can quickly modify them for your custom applications. A main feature of the G Math Toolkit is that it adds to LabVIEW and BridgeVIEW the ability to enter complex formulas directly onto the front panel of a VI.

This toolkit is intended for use by scientists, engineers, and mathematicians, and by anyone needing to solve mathematical problems in a simple, quick, and efficient manner. It can also be used as an educational aid by those interested in expanding their knowledge of mathematics.

You Will Learn:

A. About the organization of the G Math Toolkit.

B. About the different types of parser VIs and how to use them to parse formulas entered directly on the front panel.

C. About solving differential equations using the differential equation VIs.

A. Organization of the G Math Toolkit

The G Math Toolkit consists of nine libraries, each specifically suited to solve problems in a particular area of mathematics. These libraries are:

`Parser.llb`: Consists of the VIs that act as an interface between the end user and the programming system. These VIs parse the user-given formula and convert it to a form that can be used for evaluating the results.

`Visualiz.llb`: These are the Data Visualization VIs for plotting and visualizing data in both 2D and 3D. They include advanced methods such as animation, contour plots, and surface cuts.

`Ode.llb`: The VIs in this library solve ordinary differential equations VIs, both numerically and symbolically.

`Zero.llb`: Used for finding the zeros of 1D or *n*D, linear or nonlinear functions (or system of functions).

`Opti.llb`: The optimization VIs that determine local minima and maxima of real 1D or *n*D functions. You can choose between optimization algorithms based on derivatives of the function and others working without these derivatives.

`1Dexplo.llb`: Contains VIs that allow the study of real-valued 1D functions, with and without additional parameters, given in symbolic form.

`2Dexplo.llb`: A collection of VIs that deliver information about 2D functions given in symbolic form, where parameterization is allowed. Extrema (minima and maxima) and partial derivatives can be numerically calculated.

`Function.llb`: These VIs evaluate some common mathematical functions.

`Trans.llb`: A group of VIs that implement some transforms commonly used in mathematics and signal processing.

The G Math libraries consist of more than 100 VIs you can use for solving your mathematics problems. In the next section, you will concentrate on learning more about the parser VIs and use some of them to build a simple arbitrary waveform generator. The importance of the VIs in the parser library is in enabling users to enter formulas directly on the front panel. Thus, the user can enter a formula on the front panel and the arbitrary waveform generator will generate and plot the corresponding signal.

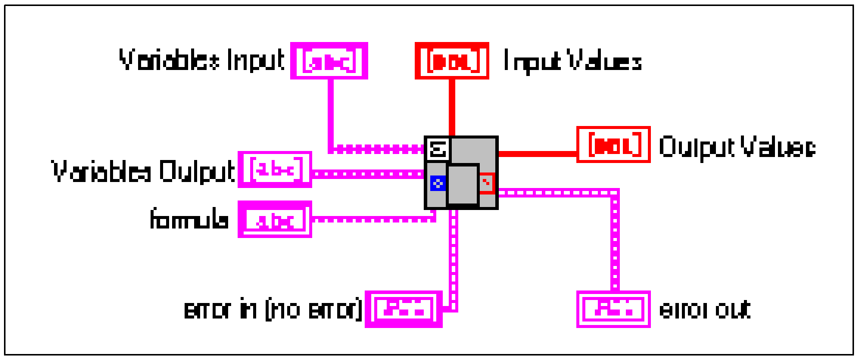
B. Parser VIs

The VIs in the Parser library act as the interface between the user and the VIs in the other libraries. The formulas entered on the front panel can have any number of variables. The formulas are first parsed to determine the variables and the values to be assigned to them. They are then evaluated to a number by substituting numeric values for the variables.

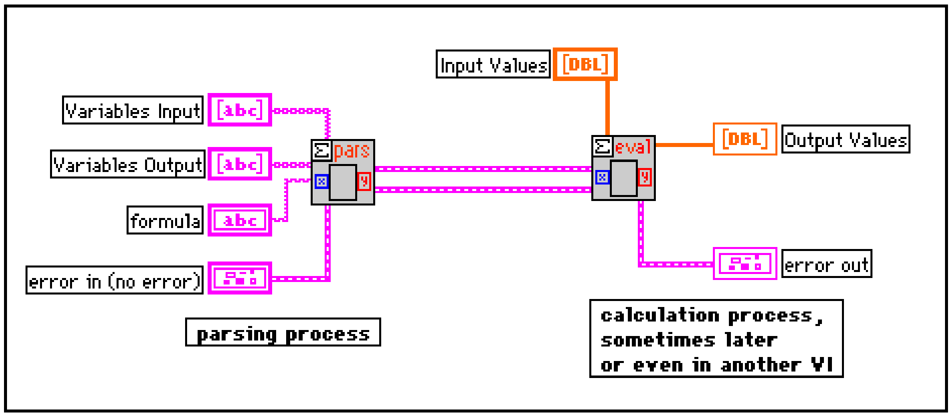
Direct and Indirect Forms

Because there are basically two steps involved in this process (parsing and then evaluation), there are two forms of parser VIs—the *direct* form and the *indirect* form. In the direct form, both the parsing and evaluation are done in the same VI. In the indirect form, the parsing and evaluation are done in separate VIs.

As an example, the direct version of the **Eval Formula Node** VI can be represented as in the following block diagram.



On the other hand, the indirect forms split the VI in two subVIs, as shown in the following illustration. You can use the indirect form in larger applications, where a two-step process (parsing and then evaluating) is more efficient.



The following table summarizes the different types of parser VIs.

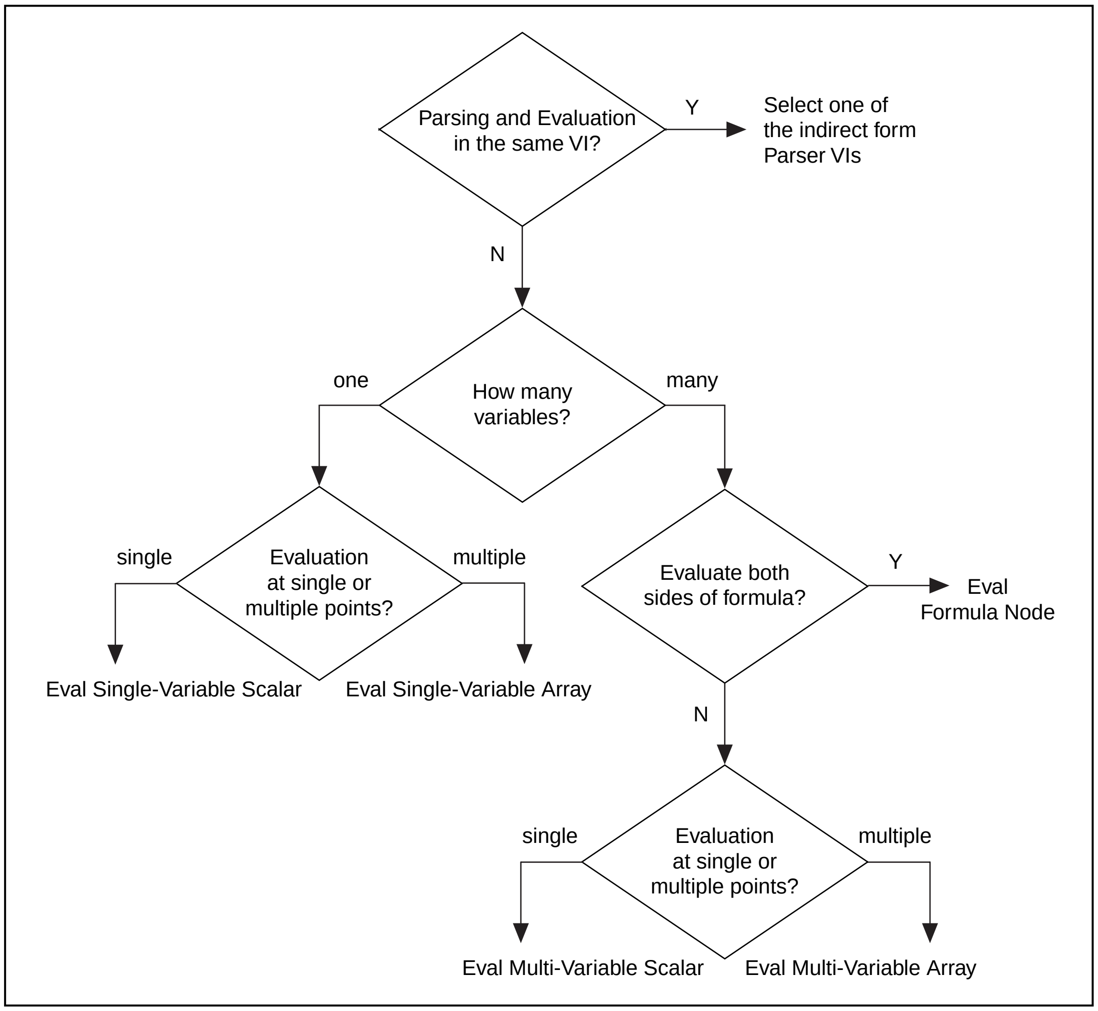
Direct Form (Parsing and evaluation done in one VI)			**Example**
No Variables	Eval Formula String	Evaluates RHS of a formula without variables.	sin(1.2) + 5
Single Variable	Eval Single-Variable Scalar	Evaluates RHS of a formula of one variable at one specified point.	cos(x) at x = 3.142
	Eval Single-Variable Array	Evaluates RHS of a formula of one variable at several specified points.	cos(x) at x = 0, 0.1, 0.2, ...
Many Variables	Eval Formula Node	Evaluates both sides of formula(s) with several variables.	x = a + b at a = 1, b = 2
	Eval Multi-Variable Scalar	Evaluates RHS of a formula with several variables at one specified point.	sin(x) + cos(y) + z at x = 1.0, y = 2.5, z = 3.1
	Eval Multi-Variable Array	Evaluates RHS of a formula with several variables at several specified points.	sin(x) + cos(y) + z at x = 1, y = 2, z = 3 and x = 3, y = 10, z = 1 and x = 0.1, y = -2, z = 0.0
Indirect Form (Parsing and evaluation in separate VIs)			
Many Variables (The combination evaluates the RHS of a formula of several variables at the specified point.)	Parse Formula String	Parses the RHS of a formula to determine input variables and the operations performed on them.	x = sin(z) + 7 * y
	Eval Parsed Formula String	Evaluates the parsed formula with the specified values for the input variables.	

Many Variables (The combination evaluates both sides of formula(s) of several variables at the specified point.)	Parse Formula Node	Parses both sides of the formula(s) to determine the input and output variables and the operations to be performed on them.	x = a + b
			y = a * b

	Eval Parsed Formula Node	Evaluates the parsed formula(s) with the specified values for the input variables.	
Others			
	Substitute Variables	Substitutes specified formulas for variables in the main formula.	

Note: ***The front panel of each parser VI has an example that shows how to enter values in the control inputs.***

The direct form of parser VIs are more widely used than the indirect form. The following flowchart guides you through the selection of a specific parser VI of the direct form.


Parsing and Evaluation in the same VI?
Y
Select one of the indirect form Parser VIs
N
How many variables?
one
many
Evaluation at single or multiple points?
single
multiple
Eval Single-Variable Scalar
Eval Single-Variable Array
Evaluate both sides of formula?
Y
Eval Formula Node
N
Evaluation at single or multiple points?
single
multiple
Eval Multi-Variable Scalar
Eval Multi-Variable Array

Comparison with Formula Node

A parser VI scans an input string and interprets this string as a collection of formulas. Then, the parser VI replaces the formulas with numeric calculations and outputs the results.The parser VI routines deal only with real numbers. There are some differences between the parser in the G Math Toolkit and the Formula Node found in the original LabVIEW (or BridgeVIEW) package. The following table outlines these differences.

Meaning	**Formula Node**	**Parser VI Routines (G Math Toolkit)**
Variables	No restrictions	Only *a*, *a*0, ..., *a*9, ... *z*, *z*0, ..., *z*9, are valid

Meaning	Formula Node	Parser VI Routines (G Math Toolkit)
Binary functions	Max, min, mod, rem	Not available
More complex math functions	Not available	Gamma, ci, si, spike, step, square
Assignment	=	Not available
Logical, conditional, inequality, equality	?:, ||, &&, !=, ==, <, >, <=, >=	Not available
π	pi	pi(1)= π,pi(2)= 2π

The precedence of operators is the same for the G Math Toolkit parser VIs as those of the formula nodes in LabVIEW and BridgeVIEW.

Error Structure

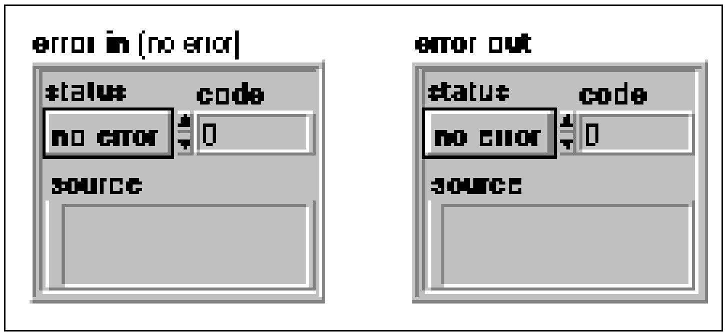
The parser VIs use the following error handling structure. This structure consists of a Boolean *status* button, a signed 32-bit integer numeric *code* indicator, and a string *source* indicator. These error handler components are explained below:

status is TRUE if an error occurred. If *status* is TRUE, this VI does not perform any operations.

code is the error code number identifying the error.

source explains the error in more detail.

The default status of the **error in** structure is FALSE (no error), indicated by an error code of 0.



With this structure, an application can programmatically decide the accuracy of formulas and control the data flow in case of errors. The application uses the *source* field of the error handling structure as a storage for a wrong formula input. This field displays limited error descriptions if an error is detected in your program. See the *Error Codes* appendix for the error codes and the messages of the parser VI routines.

Functions Available for Use with Parser VIs

Most of the functions that you can use in the formula node can also be used in the parser VIs. However, there are some differences. For a complete list of functions that you can use with the parser VIs, refer to the appendix.

The parser VIs are extremely powerful and can be used in a wide variety of applications. The first few exercises in this lesson cover building an arbitrary waveform generator that uses several of the parser routines and to understand their functionality.

Exercise 11-1

Objective: To build an arbitrary waveform generator using the Eval Single-Variable Array VI.

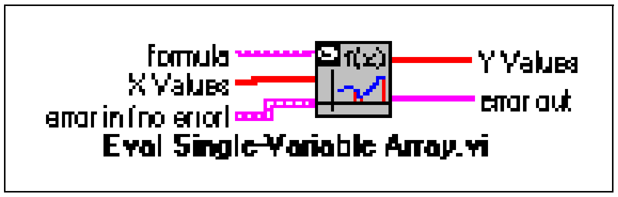
In this exercise, you will build an arbitrary waveform generator using the parser VIs. You will enter the formula for the waveform on the front panel and see the output waveform on a graph indicator.

Selection of Direct or Indirect Form Parser VI

First, you must decide whether to use the direct or the indirect form of the parser VIs. Because you are interested only in displaying the result of the evaluation on a graph, and are not concerned about the results of parsing, you will use the direct form of the parser VIs.

Selection of Parser VI

Looking at the table on page 11-4, depending on the number of variables in your formula, you need to choose the appropriate VI from several available choices of VIs of the direct form. Because you will plot a 1D function, you need a VI that can handle functions of at least one variable. And because you want to evaluate that function at more than one point, choose the **Eval Single-Variable Array** VI (**G Math » Parser** subpalette). The inputs and outputs of this VI are shown below.

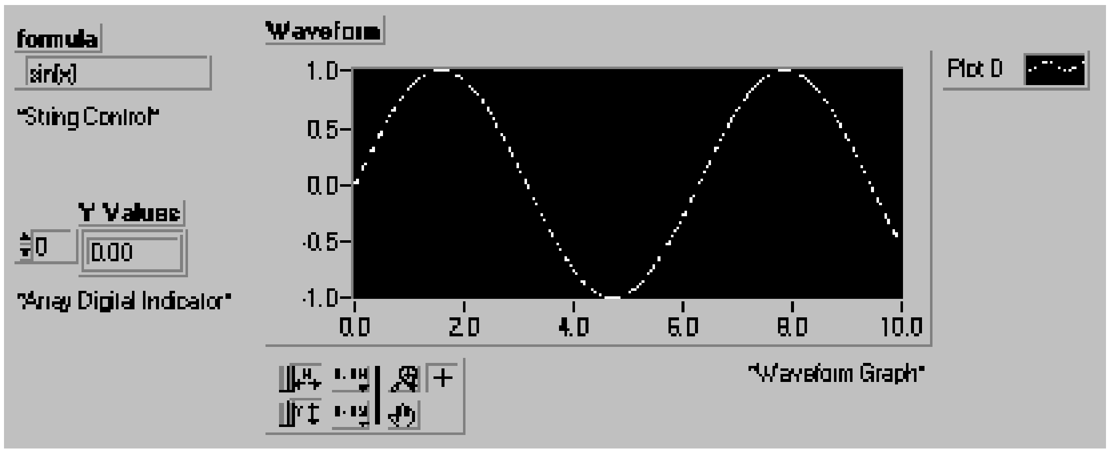


formula: a control for specifying your mathematical formula consisting of one variable.
X Values: values of the variable at which the formula is evaluated.
Y Values: numeric results of the evaluation of the formula in **formula** at the values specified in **X Values**.

You need to specify at least the inputs **formula** and **X Values**. For the **formula** input, you will enter a control on the front panel. On this control, you can type in your input formula. For the **X Values**, you will build a For Loop on the block diagram that will generate 1000 points at which to evaluate the equation you will type in the formula control. The corresponding front panel and block diagram are shown below. Note that the For Loop generates **X Values** from 0 to 10 in increments of 0.01.

1. Build the VI front panel and block diagram shown below.

Front Panel

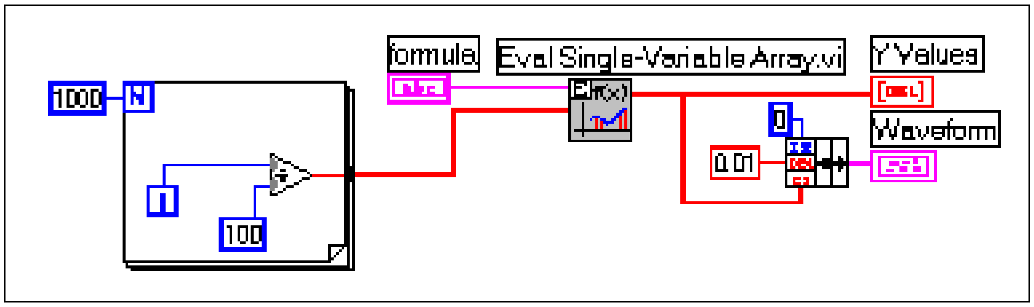


The formula string control is where you enter your formula.

The Y Values indicator shows the result of the calculation of the formula at specified discrete points.

The Waveform graph shows you a plot of the function you typed in the formula control.

Block Diagram



Eval Single-Variable Array VI (**G Math » Parser** subpalette) evaluates a function of a single variable at multiple points.

The For Loop generates the discrete set of points at which the formula in the formula control will be evaluated. These points range from 0 to 9.99 in increments of 0.01.

2. After building the above VI, return to the front panel.
3. Type in the equation *sin(x)* in the formula control and run the VI. See the result displayed on the graph indicator.

Note that *sin(x)* has only one variable, called *x*. You could also call the variable *y* or *z* or anything else. The appendix lists the rules for variable names.

4. Change the equation to *sin(x) + cos(x/2)* and run the VI.
5. Change the equation to *step(y-1) + sinc(y-2) + sin(y*10)* and run the VI. Note that now you have changed the variable to *y*.
6. Type in any other equation in the formula control and run the VI. The appendix contains a list of available functions you can use in the equation.

Note: ***If you get an error, refer to the appendix for a list of error codes.***

7. Save the VI as **EvalSVA.vi** in the library `Lvspcex.llb`.
8. Close the VI.

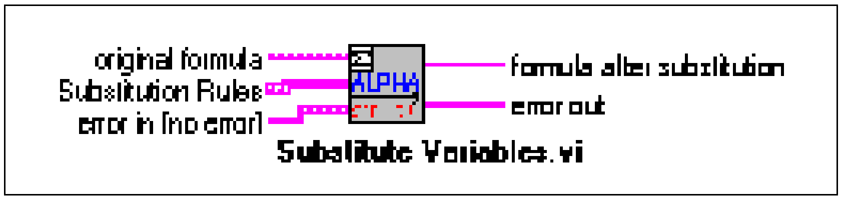
You have now seen how easy it is to build the waveform generator where the user can specify the formula for an arbitrary waveform on the front panel. In the next exercise, you will modify this waveform generator to have the functionality of easily specifying even more complex waveforms.

End of Exercise 11-1

Exercise 11-2

Objective: To generate even more complex waveforms by incorporating the Substitute Variables VI in the function generator.

Another useful VI in the parser library is the **Substitute Variables** VI (**G Math » Parser** subpalette). The connections to this VI are shown in the figure below. It is used to substitute formulas for parameters that are already defined in formulas.

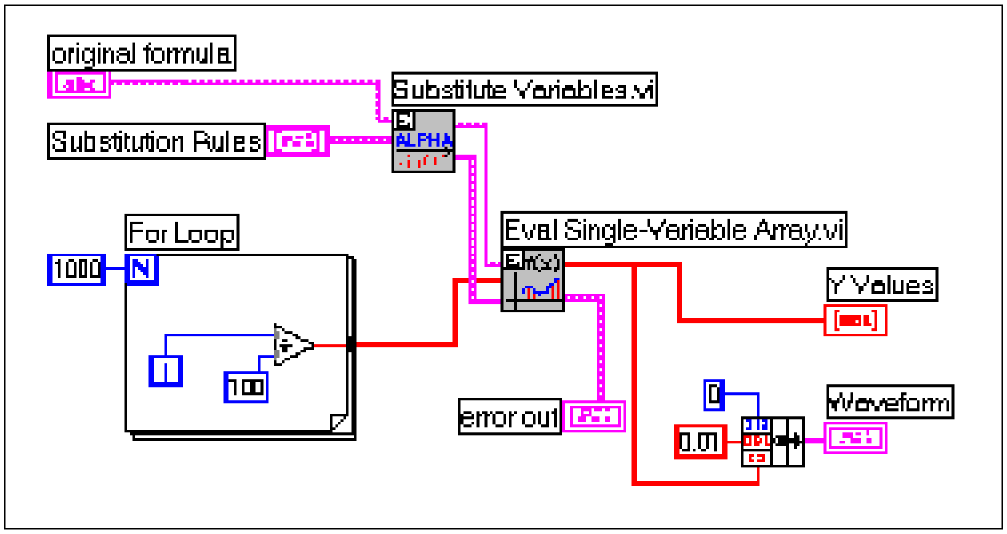


original formula: the main formula you type in.

Substitution Rules: specify the substitutions to be made for the parameters in **original formula**.

formula after substitution: the resulting formula after the parameter substitutions specified in **Substitution Rules**.

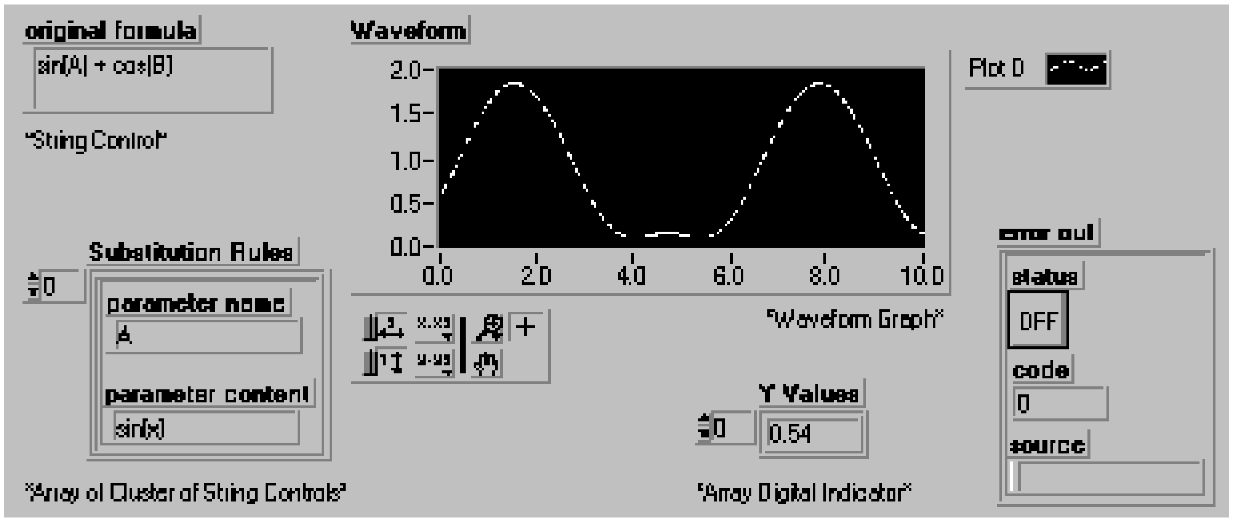
You will now modify the waveform generator you built in the previous exercise to use the **Substitute Variables** VI. Suppose you want to generate a "generic" waveform that is of the form *sin(A) + cos(B)*, where *A* and *B* can themselves be functions like sin(x), cos(x), square(x), sinc(x), ln(x), etc. But you want to use different functions for *A* and *B* each time you run the VI. This is accomplished as shown in the front panel and block diagrams below.



Substitute Variables VI (**G Math » Parser** subpalette) substitutes for parameters defined in formulas. The substitution is done according to the rules specified in the **Substitution Rules** cluster array.

Eval Single-Variable Array VI (**G Math » Parser** subpalette) evaluates a function of one variable at specified multiple points.

The For Loop generates a thousand points, ranging from 0 to 9.99 in steps of 0.01, at which to evaluate the user-specified formula.



The **original formula** string control is where you enter your formula of one variable. The formula may have one or more parameters.

Each element (cluster) in the **Substitution Rules** array specifies a parameter and its corresponding substitution.

Y Values is an array digital indicator that shows the result of calculating the formula at specified points.

1. Build the VI as shown in the figures above. The **formula after substitution** output of the **Substitute Variables** VI goes to the **formula** input of the **Eval Single-Variable Array** VI.

Hint: Build the block diagram first, pop up to create the controls and indicators, and then add the waveform graph.

2. In the **original formula** control, type in the formula *sin(A) + cos(B)*. This will be the "generic" formula. In the **Substitution Rules** control, you will provide the user with the capability of specifying the functions to be substituted for *A* and *B*.

3. In element 0 of the **Substitution Rules** control, type in

parameter name:	A
parameter content:	sin(x)

   In element 1, type in

parameter name:	B
parameter content:	cos(x)

   Note that this is the same as if you had typed *sin(sin(x)) + cos(cos(x))* in the formula control in Exercise 11-1.
4. Run the VI and see the waveform on the graph display.
5. Change the values in the **Substitution Rules** control to

element 0	**parameter name:**	A
	parameter content:	sin(x)
element 1	**parameter name:**	B
	parameter content:	square(x)

   Run the VI. The resulting equation is the same as if you had typed *sin(sin(x)) + cos(square(x))* in the formula control in Exercise 1-1.
6. Now change the control to

element 0	**parameter name:**	A
	parameter content:	step(x-5)
element 1	**parameter name:**	B
	parameter content:	square(x)

   Run the VI.

   With this simple VI that you have built, you now can generate any type of waveform that you so desire, with the functions given in the appendix.
7. Save the VI as **ESVA_SV.vi** in the library `Lvspcex.llb`.
8. Close the VI.

End of Exercise 11-2

C. Solving Differential Equations

You can use G Math to solve both ordinary or partial, linear or nonlinear, and either first or higher order differential equations. G Math has seven VIs for solving various types of differential equations. These VIs are listed below, classified according to the order of the differential equation they can solve.

Note: ***The order of a differential equation is the order of the highest derivative in the differential equation.***

The first five VIs can solve a set (one or more) of first-order differential equations. The next two VIs are for solving higher order differential equations. Note that you can convert a higher order differential equation into a set of first-order differential equations (later in this lesson, you will see an example of how to do this).

VIs for solving a set (one or more) of first-order differential equations:

ODE Cash Karp 5th order for solving differential equations using the Cash Karp method.

ODE Euler Method for solving differential equations using the Euler method.

ODE Runge Kutta 4th order for solving differential equations using the Runge Kutta method.

ODE Linear System Numeric for numerical solution of a linear system of differential equations.

ODE Linear System Symbolic for symbolic solution of a linear system of differential equations.

Out of these, the first three are for solving *nonhomogeneous* (right side ¦ 0) differential equations, whereas the last two are for *homogeneous* (right side = 0) differential equations.

VIs for solving higher order differential equations:

ODE Linear *n*th order Numeric for numeric solution of a linear system of nth order differential equations.

ODE Linear *n*th order Symbolic for symbolic solution of a linear system of nth order differential equations.

Both these VIs are for solving *homogeneous* differential equations.

Solving Nonhomogeneous Differential Equations

From the previous discussion, you see that there are three VIs available for solving nonhomogeneous differential equations. They are

- **ODE Cash Karp 5th order** VI
- **ODE Euler Method** VI
- **ODE Runge Kutta 4th order** VI

Each VI employs a different method for solving the differential equations. Each method uses a parameter known as the *step size* (denoted by *h*) that determines the spacing between points at which the solution is evaluated. This step size is a constant for the last two VIs (which employ the Euler and Runge-Kutta methods), whereas it is variable (it automatically adapts itself to the solution) for the first VI (which employs the Cash Karp method). The Euler method is the simplest method, but the Runge-Kutta method gives a more accurate solution.

The question arises as to which of these three VIs you should choose for your application. The general guidelines are:

- Select the **ODE Euler Method** VI for very simple ODEs.
- For all other cases, choose the **ODE Runge-Kutta 4th Order** VI or the **ODE Cash Karp 5th Order** VI.
  - If you need equidistant points (that is, constant *h*—for example, for robot control applications), choose the **ODE Runge-Kutta 4th Order** VI.
  - If you are interested in a global solution and fast computation, choose the **ODE Cash Karp 5th Order** VI.

A General Class of Second-Order Differential Equations

As an example, a general class of second-order differential equations is described by the following initial value problem

$$a\frac{d^2y}{dt} + b\frac{dy}{dt} + cy = g(t) \qquad (1)$$

with $y(0) = y_0$ and $\frac{dy(0)}{dt} = \frac{dy_0}{dt}$

where *y(0)* is the value of *y* at *t* = 0, *dy(0)/dt* is the value of *dy/dt* at *y*=0, and g(t) is known as the *forcing function*. *y(0)* and *dy(0)/d*t are known as the *initial conditions* (ICs). Note that because the right side of equation (1) is not equal to zero, it is a *nonhomogeneous* differential equation. In the absence of a forcing function (*g(t)* = 0), it would be a *homogeneous* differential equation.

Some of the practical applications of this class of differential equations are for modeling:

1. The motion of a mass on a vibrating spring

   $$\frac{md^2y}{dt^2} + \frac{cdy}{dt} + ky = F(t)$$

   where *m* is the mass, *c* is the damping coefficient, *k* is the spring constant, and *F(t)* is the applied force.

2. Flow of an electrical current in a series circuit

   $$L\frac{d^2Q}{dt^2} + R\frac{dQ}{dt} + \frac{Q}{C} = \frac{dE(t)}{dt}$$

   where *R*, *L* and *C* are the resistance, inductance, and capacitance, respectively, in the circuit, *Q* is the charge flowing in the series circuit, and E(t) is the applied voltage.

3. The motion of an oscillating pendulum

   $$\frac{d^2\theta}{dt^2} + \frac{c}{ml}\frac{d\theta}{dt} + \frac{g}{l}\sin(\theta) = 0$$

   where *m* is the mass of the pendulum, *l* is the length of the rod, *g* is the acceleration due to gravity, and θ is the angle between the rod and a vertical line passing through the point where the rod is fixed (that is, the equilibrium position).

To solve equations of the type of eqn (1) using the **ODE Euler Method** VI, the **ODE Runge-Kutta 4th Order** VI, or the **ODE Cash Karp 5th Order** VI, you first need to convert them into a set of first-order differential equations. This is achieved by making the substitution $x_1 = y$ and $x_2 = dy/dt$. Substituting these in equation (1), you get

$$a\frac{dx_2}{dx} + bx_2 + cx_1 = g(t)$$

$$\frac{dx_2}{dt} = \frac{g(t) - cx_1 - bx_2}{a}$$

Thus, now you have converted the second-order differential equation given by (1) to the following equivalent set of first order differential equations

$$\frac{dx_1}{dt} = x_2 \qquad (2)$$

$$\frac{dx_2}{dt} = \frac{g(t) - cx_1 - bx_2}{a} \qquad (3)$$

You will use G Math to solve this set of equations in the next exercise.

Exercise 11-3

Objective: To build a general G Math VI for applications whose solutions are second-order differential equations.

In this exercise, you will build a VI that will solve the general second-order differential equation of the type given in equation (1). This equation is also given below:

$$a\frac{d^2y}{dt^2} + \frac{bdy}{dt} + cy = g(t)$$

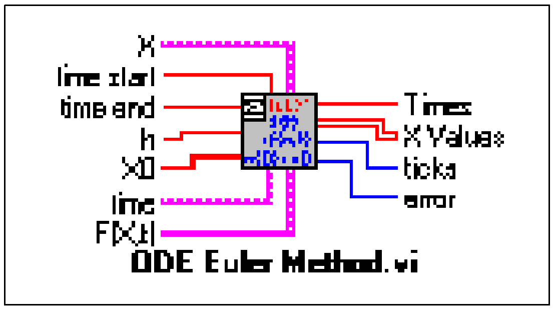
with $y(0) = y_0$ and $\frac{dy(0)}{dt} = \frac{dy_0}{dt}$

You will see how incorporation of the **Substitute Variables** VI enables you to solve problems related to a wide variety of applications.

Because you want to solve the above equation, which is a nonhomogeneous equation, you are faced with three choices of VIs:

- **ODE Cash Karp 5th Order**
- **ODE Euler Method**
- **ODE Runge Kutta 4th Order**

Because this is a simple second-order differential equation, you can actually select any of these VIs. Choose the **ODE Euler Method** VI, which has the following inputs and outputs:



X: an array of strings listing the dependent variables

time start: the point in time at which to start the calculations

time end: the point in time at which to end the calculations

h: time increment (step rate) at which to perform the calculations

X0: the initial conditions (I.C.). There is a one-to-one relationship between the components of **X** and that of **X0**. That is, the first value in **X0** is the I.C. of the first variable listed in **X**, the second value in **X0** is the I.C. of the second variable listed in **X**, and so on.

time: a string that defines what the independent variable is, usually time (t).

F(X,t): the right sides of the set of first-order differential equations. There is a one-to-one relationship between the elements of **F(X,t)** and **X**. This will be explained in more detail later.

☞ **Note:** ***F(X,t) requires its input as a set of first-order differential equations. Therefore, to solve equation (1) with this VI, you need to use the equivalent form given by equations (2) and (3).***

ticks: is the time in milliseconds required for the calculations

Times: the time instants at which the solution of the differential equations are evaluated. For the **ODE Euler Method** VI and the **ODE Runge Kutta 4th Order** VI, these instants begin at **time start** with increments of **h**, until **time end**.

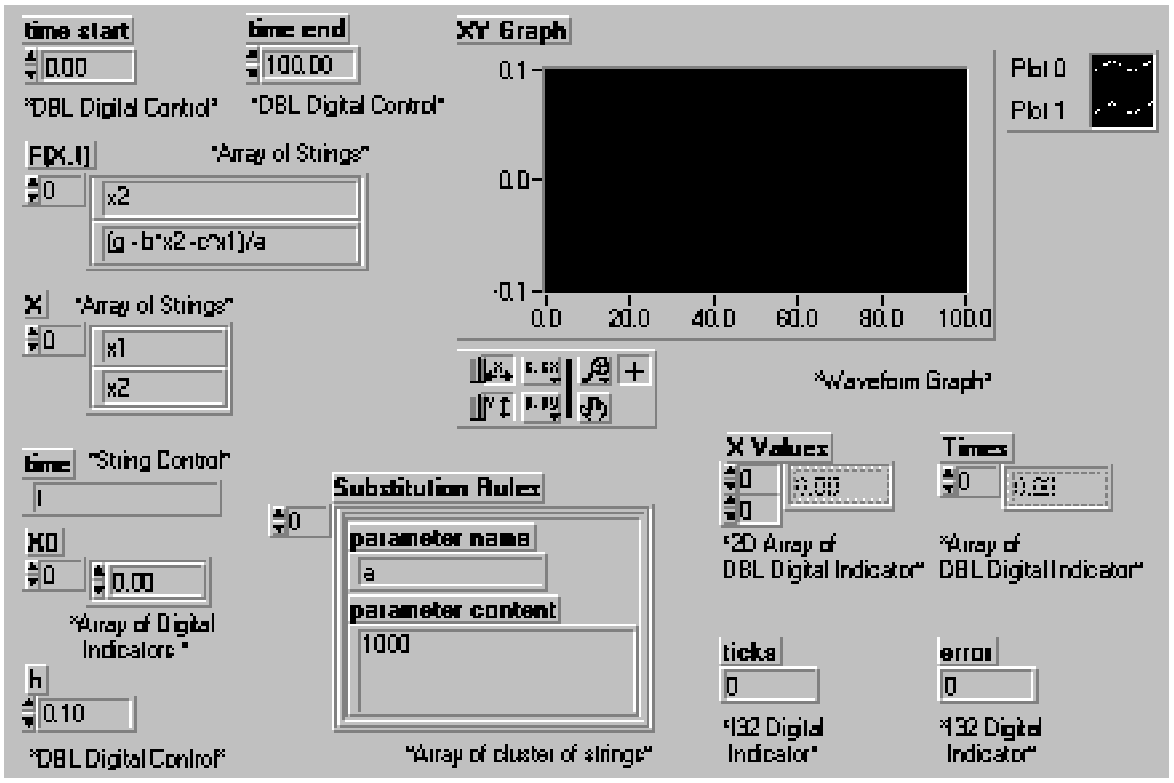
XValues: contains the solution for each of the dependent variables of the differential equations at each time instant in **Times**. The first column is the solution for the first variable in **X**, the second column is the solution for the second variable in **X**, and so on.

error: contains an error code in case of any error. Refer to the appendix for the list of errors and the corresponding error codes.

1. Build the VI whose front panel and block diagram are shown below.

Hint: An easy way to build the VI is to build the block diagram first, pop up on the terminals of the **ODE Euler Method** VI**,** and select **Create Control** for the inputs and **Create Indicator** for the outputs. The controls and indicators automatically appear in the correct form on the front panel. Finally, you can add the Waveform graph.

Front Panel



time start: 0.00

time end: 100.00

F(X,t): 0 — x2; (g - b*x2 - c*x1)/a

X: 0 — x1; x2

time: t

X0: 0 — 0.00

h: 0.10

Substitution Rules: 0 — parameter name: a; parameter content: 1000

X Values: 0, 0 — 0.00

Times: 0 — 0.00

ticks: 0

error: 0

The **time start** and the **time end** controls contain the starting and ending time instants.

The differential equations are typed in the **F(x,t)** control.

The dependent variables are entered into the **X** control.

The independent variable is entered into the **time** control.

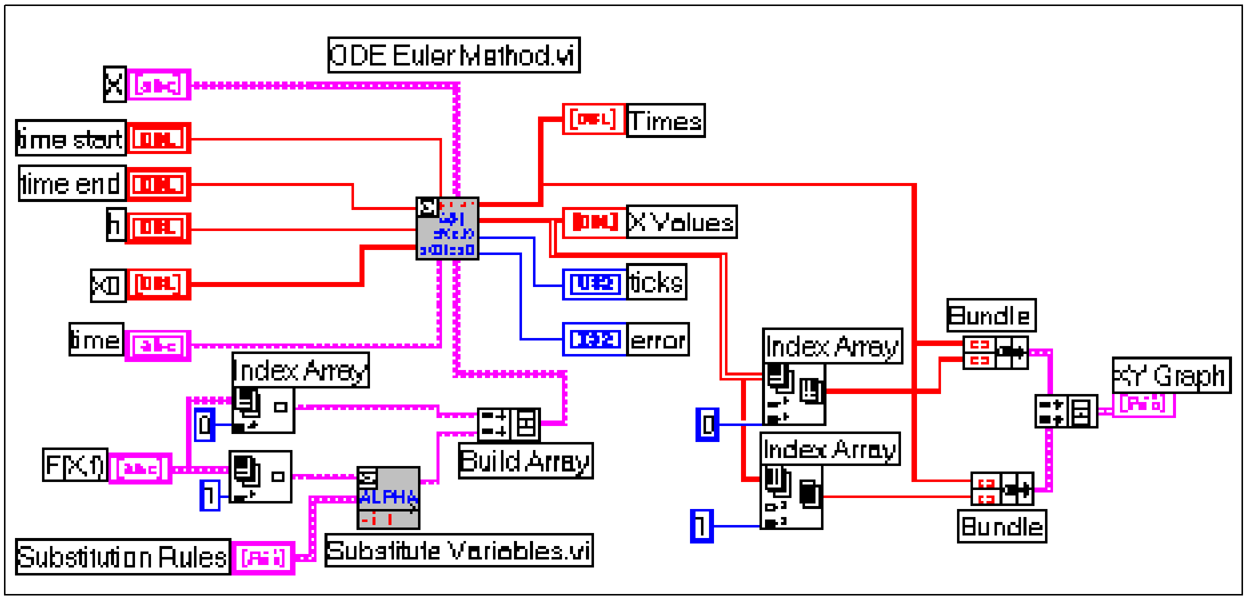
X0 contains the initial values (initial conditions) of the dependent variables.

h contains the step size.

Substitution Rules contains the parameters and the corresponding substitution.

Note: ***There is a one-to-one correspondence between the elements of X, X0, and F(x,t). The first element in X0 corresponds to the initial value of the first element in X, the second element in X0 corresponds to the initial value of the second element in X, and so on. Also, the first element in F(x,t) corresponds to the derivative of the first element in X, the second element in F(x,t) corresponds to the derivative of the first element in X, and so on.***

Block Diagram



ODE Euler Method VI (**G Math » Ordinary Differential Equations**) solves a set of differential equations using the Euler method.

Substitute Variables VI (**G Math » Parser**) substitutes the specified values for the corresponding parameters.

2. Enter the right side of equations (2) and (3), respectively, in the first two elements of **F(X,t)**. Thus, you will enter x_2 and $(g - b*x_2 - c*x_1)/a$.

   Equation (2) in the second element of **F(X,t)** contains the variables *a*, *b*, *c*, and *g*. The values of these variables will change depending on the application. Hence, in the block diagram, you use the **Substitute Variables** VI to assign different values to these variables. You can enter these values in the **Substitution Rules** control on the front panel.

3. **X** contains the names of the dependent variables, which in this case are entered as x_1 and x_2.

4. **time** contains the name of the independent variable, which in this case is *t*.

   X0 will contain the initial conditions for x_1 and x_2. The first value entered in **X0** will correspond to the first variable entered in **X**. The second value entered in **X0** will correspond to the second variable entered in **X**. The values of the ICs will vary depending on the application.

5. Save the VI as **ODE_2nd.vi** in the library `Lvspcex.llb`.

   Now you have a VI ready for solving any second-order nonhomogeneous differential equation. All that remains is to enter the corresponding values of *a, b, c,* and *g* in the **Substitution Rules** control, and the initial conditions of the variables **X0** control. Then you can specify the time interval (between **time start** and **time end**, in increments of **h**) for which you want the solution and run the VI. Out pops the answer. You will continue this procedure in the next exercise.

End of Exercise 11-3

Exercise 11-4

Objective: To design the suspension system of an automobile.

You will now use the VI you built in the previous exercise to solve a typical mechanical modeling problem. In particular, you will design the suspension system of an automobile to have a suitable response when the automobile goes over a pothole, a speed bump, and so on.

You can model the suspension system by the nonhomogeneous differential equation given in equation (1) (reproduced below),

$$a\frac{d^2y}{dt^2} + \frac{bdy}{dt} + cy = g(t)$$

with $y(0) = y_0$ and $\frac{dy(0)}{dt} = \frac{dy_0}{dt}$

where

y(t) is the position of the automobile. *y(t)* = 0 indicates that the vehicle is in a balanced or a stable position.

dy/dt is the up and down speed of the automobile.

c is the coil constant of the spring used in the suspension system. It is proportional to the displacement of the spring from its equilibrium position. A larger value of *c* indicates a strong spring coil and thus a tough suspension system. *c* is known as the *spring constant.*

b is a resistance coefficient that is proportional to the up and down speed of the automobile. A larger value of *b* indicates a stronger resistance to the up and down movement of the automobile, and thus a smoother suspension system. *b* is known as the *shock constant.*

a is the weight of the vehicle.

g(t) is the external force that causes the suspension system of the automobile to deviate from its equilibrium position. Some possible causes may include a speed bump, a pothole, or a stone in the middle of the road.

1. Using the VI developed in the previous exercise, enter the following values in the controls on the front panel:

   time start: 0

   time end: 100

   h: 0.1

   Substitution Rules

parameter name:	a	b	c	g
parameter content:	1000	125	1000	0
X0:	0	0		

The initial conditions of 0 specified in X0 indicate that the vehicle is initially in a stable equilibrium position. *g* = 0, means that there is no disturbing force.

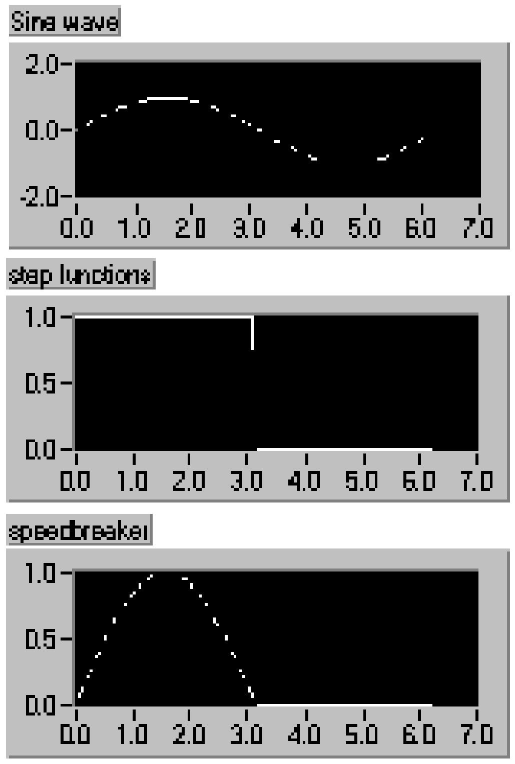
2. Run the VI and note the waveforms on the XY graph. What do you see? Explain the results.
3. Now assume that after driving for 10 seconds, the driver of the automobile drove on to the sidewalk. This action can be modeled by substituting a step function for *g*. In the **Substitution Rules** control, change the value of *g* from **0** to **100*step(t-10)** and run the VI.

   Notice that until t = 10, the values of *x* and *dx/dt* are equal to zero because no force (*g*) has been applied until then. At *t* = 0, a step function of magnitude 100 is applied and both the position and the up and down speed of the automobile change. However, as time passes, the oscillations subside and both values tend to settle down to an equilibrium position. You can verify this by entering 400 in the time end control and running the VI.
4. A speed bump can be approximately modeled by the following value for g:

   *10*sin(t)*(step(t) - step(t-3.142))*

   The multiplying constant (in this case 10) controls the height of the speed bump, while the remaining term models one "bump."

   How the above formula corresponds to a speed bump is shown in the figure below. The upper plot is a plot of one cycle of sin(t). The middle plot is a plot of step(t) - step(t-3.142), whereas the lower plot is the multiplication of the top two plots resulting in sin(t)*(step(t) - step((t-3.142)).



Enter the above value of *g* and run the VI.

What do you see? Explain the results.

5. A pothole can be modeled in a similar fashion, but with the opposite sign: *-10*sin(t)*(step(t) - step(t-3.142))*

   Enter the above value of g and run the VI. What change do you notice as compared with the results of step (4)?

6. Experiment with the value of *c* so that the oscillations reduce to zero in less than 80 seconds.

   Hint: reduce the value of *c* to a lower value (for example, 250)

7. Keeping the value of *c* at 1000, change the value of *b* so that the oscillations reduce to zero in less than 80 seconds.

   Hint: increase the value of *b* to a higher value (for example, 175)

Thus, you see that by proper selection of *b* and/or *c*, you can design the suspension system of an automobile to behave in a certain way (that is, to have a certain response). The selection of *b* and/or *c* could correspond to a particular choice of material to be used in building the suspension system.

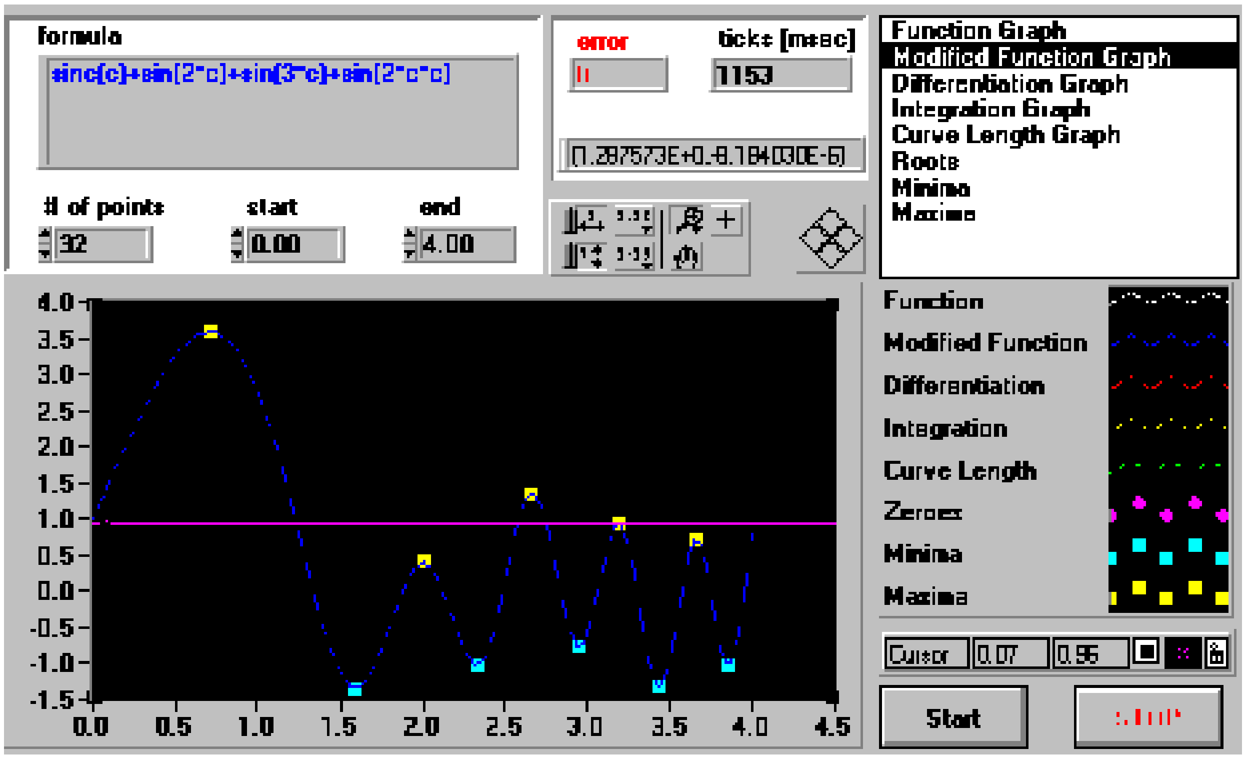
8. Close the VI when you are finished.

End of Exercise 11-4

Exercise 11-5

Objective: To see some of the capabilities of the G Math Toolkit for 1D functions.

1. Launch the **1D Explorer Example** VI from the `Labview/Examples/Gmath/Math.llb` folder. This VI shows a good overview of some of the mathematical functionalities of G Math.



2. The default formula in the formula control is *sinc(c)+sin(2*c)+sin(3*c)+sin(2*c*c)*. Choose the following options by holding down <Shift> and clicking the left mouse button on the following selections
   - Modified Function Graph
   - Integration Graph
   - Roots
   - Maxima
3. Run the VI and click the **Start** button on the bottom right of the panel. Plots of the above selections are displayed. Note that you can enter the start and end points directly on the front panel through the start and end controls. Also, note that LabVIEW returned the graphs quickly because LabVIEW runs compiled. (Other math packages are not compiled.)

4. Line up the cursor on a maxima and see its value.
5. Enter a new formula *sin(exp(x))* in the formula control and click on the **Start** button.
6. Zoom in on a maxima using the graph zoom option.
7. Experiment by typing in other formulas in the formula control.
8. When you finish, you can stop the VI by clicking on the **STOP** button at the bottom right of the panel.

End of Exercise 11-5

Exercise 11-6 (Optional)

Objective: To simulate the tank flow problem.

This exercise is a practical application that combines entering formulas on the front panel, solving an ordinary differential equation, and visualizing a process to simulate a tank inflow and outflow process.

Consider a cylindrical tank of constant cross section A cm^2. Water is pumped into the tank from the top at a constant rate $f_i(t)$ cm^3/s. Water flows out of the bottom of the tank by a valve of area $a(t)$ cm^2. Note that both the input and output flow rates are functions of time. You will observe how the height of the water, $h(t)$, in the tank varies with time.

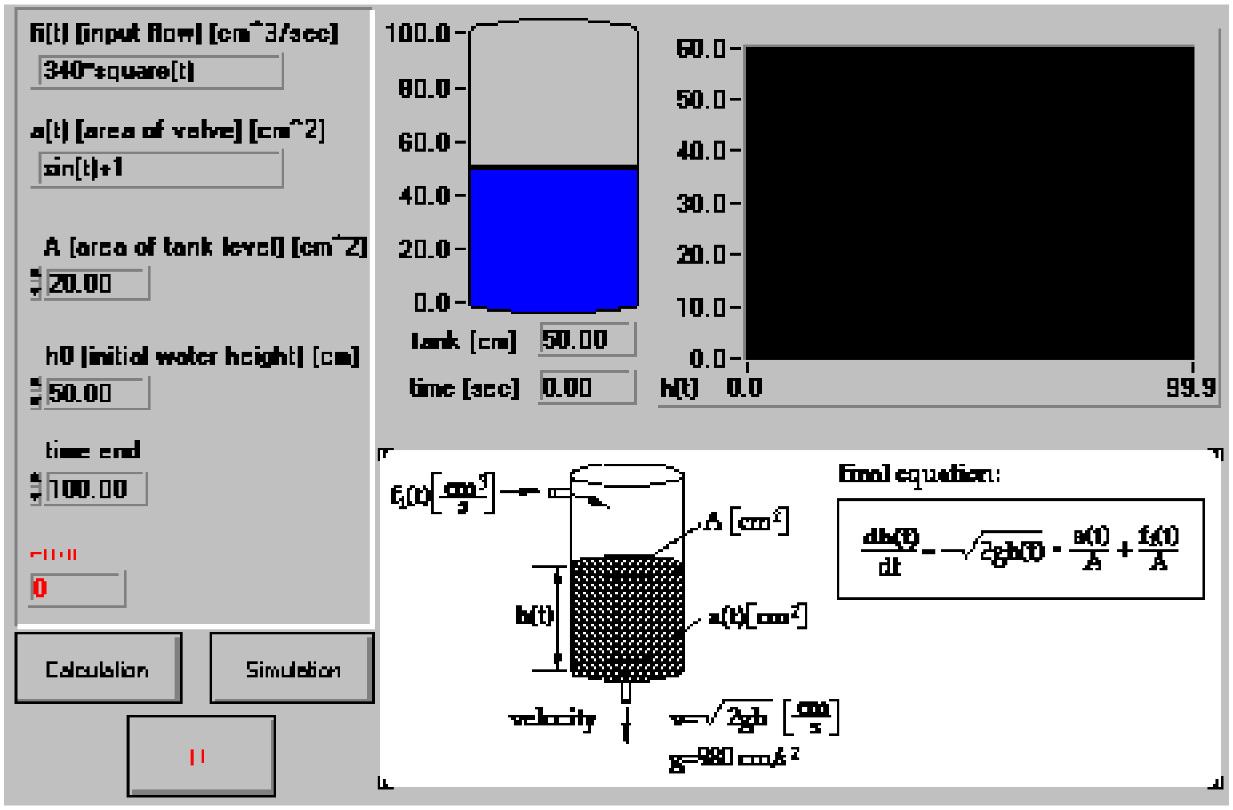
The solution to this problem is a first-order differential equation given by

$$\frac{dh(t)}{dt} = -\sqrt{2gh(t)} - \frac{a(t)}{A} + \frac{f_i(t)}{A}$$

where g is the acceleration due to gravity, equal to 980 cm/s^2.

The input flow rate to the tank, as well as the area of the outflow valve, can be modeled as equations on the front panel. Select the following equations for these values:

Input flow, $f_i(t)$:	340*square(t)
Area of valve, $a(t)$, for the outflow:	sin(t) + 1



fi[t] [input flow] [cm^3/sec]: 340*square[t]

a[t] [area of valve] [cm^2]: sin[t]+1

A [area of tank level] [cm^2]: 20.00

h0 [initial water height] [cm]: 50.00

time end: 100.00

error: 0

tank [cm]: 50.00

time [sec]: 0.00

$$\frac{dh(t)}{dt} = -\sqrt{2g\,h(t)}\cdot\frac{a(t)}{A} + \frac{f_i(t)}{A}$$

1. Open the **Process Control Explorer Example** VI, in the `Labview/Examples/Gmath/Misc.llb` folder. The explanation of the front panel is as already described above.

Note: ***In this example, all variables in the differential equation can be controlled from the front panel.***

2. Run the VI and select the **Calculation** button. This button will flash while the calculation is in progress.
3. Once the calculation is over, select the **Simulation** button. The simulated system is then graphed.
4. Change the input flow rate and area of tank to:

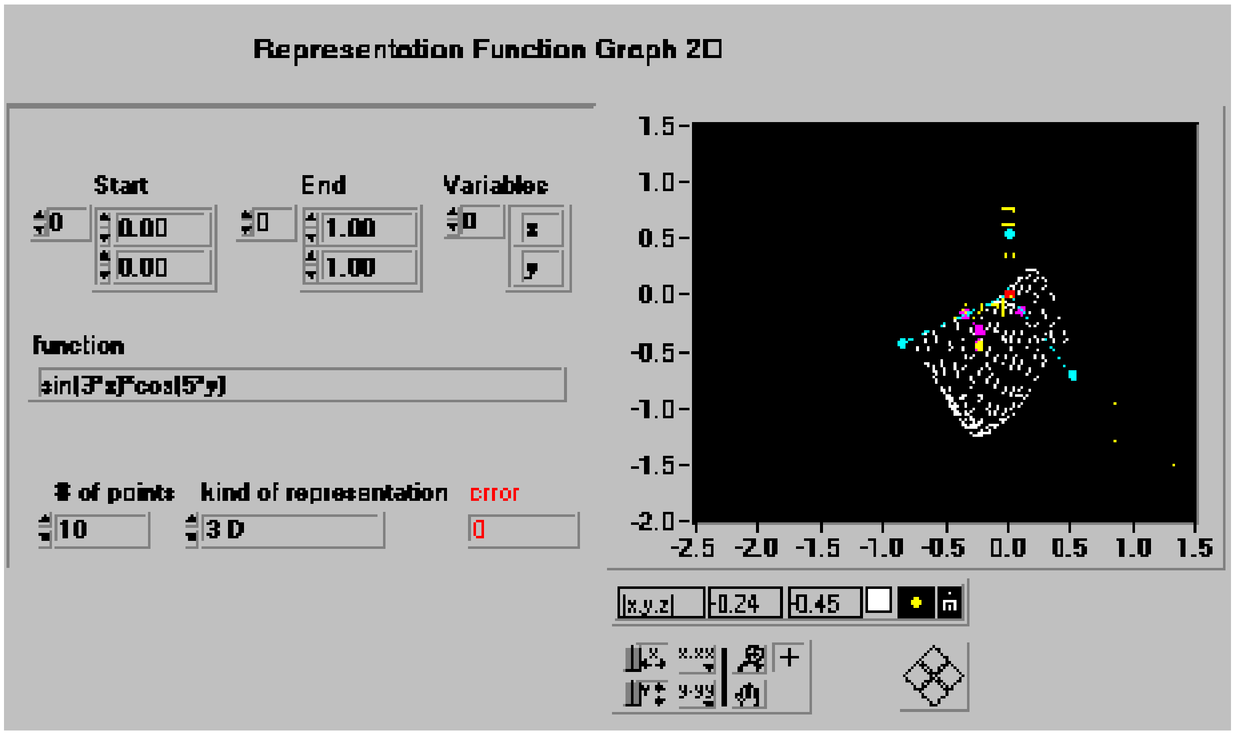
   fi(t), input flow rate: 340*sin(t)

   a(t), area of valve: 0.01
5. Select the **Calculation** button and then the **Simulation** button. Notice that the tank is now gradually emptying.
6. Select the **STOP** button.

End of Exercise 11-6

Exercise 11-7 (Optional)

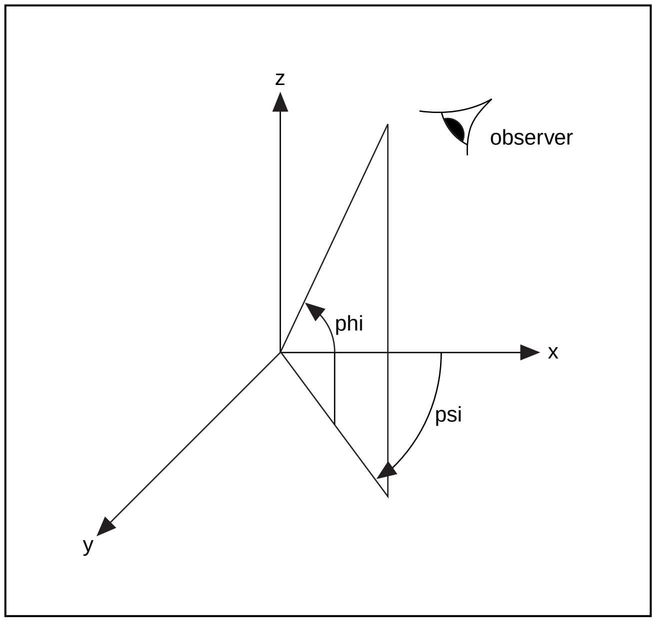
Objective: To see an example of the G Math Toolkit for data visualization.



1. In the `Labview/Examples/Gmath/Graphics.llb` folder, open and run the **Representation Function Graph 2D** VI.

Note: ***The default function value is sin(3*x)*cos(5*y). This is a function of two variables, x and y. The range of values to be plotted is indicated on the Start and End controls on the front panel. The default values are 0.0 and 1.0 for both the x and y variables.***

When the interactive subVI comes up, you can look at the plot from different viewpoints by controlling the values of *psi*, *phi*, and *r* with the help of the slider controls. *Psi* and *phi* control the angles with respect to the *xz* and *xy* planes, respectively, where *r* controls the distance from the origin. How these controls correspond to different viewpoints is shown in the figure below:



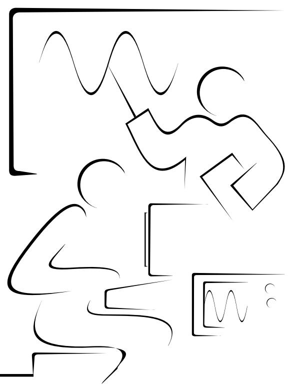
2. Change the *psi* value on the slider control. Note what happens.
3. Change the *phi* value on the slider control. Note what happens.
4. To return to the main VI, select the stop button on the front panel.
5. When you are done, close the VI.

End of Exercise 11-7

Summary

In this lesson, you saw the basic components of the G Math Toolkit and how to use it for solving problems in engineering, math, and education. In particular, you built a simple arbitrary waveform generator using the VIs in the parser library, and you also designed the suspension system of an automobile by making use of the differential equation VIs. Finally, you experimented with other VIs that showed some of the more advanced capabilities of G Math.

Notes



Lesson 12
Third-Octave Analyzers

Introduction

A third-octave analyzer measures the spectral energy contained in a specific set of third-octave bands and provides a standardized narrowband spectral analysis. It consists of a set of bandpass filters connected in parallel. It is widely used in the field of acoustics and audio signal processing to measure sound or acoustic intensity in various frequency bands. Its applications lie in the fields of vibration tests of machines, architectural acoustics, power measurements, etc. In this lesson, you will learn about the specifics of the bandpass filters used in third-octave analyzers, their applications, and how to use the Third-Octave Analyzer Toolkit.

You Will Learn:

A. About filter banks.

B. About how an octave analyzer is a special type of filter bank.

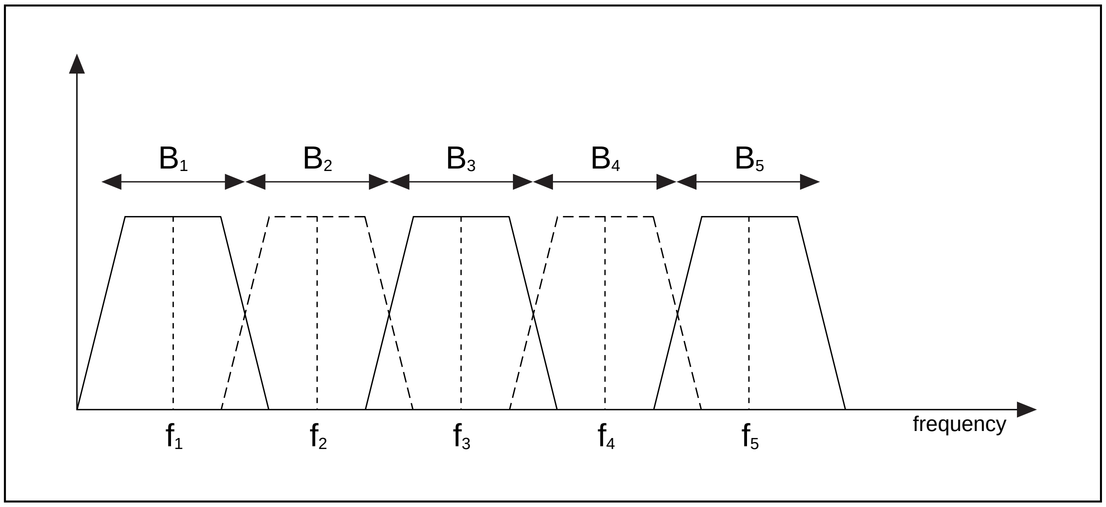
C. About third-octave analyzers.

D. About the applications of octave and third-octave analyzers.

E. About the features of the Third-Octave Analyzer Toolkit.

A. Filter Banks

A filter bank is a group of bandpass filters connected in parallel. Each filter is tuned to a different frequency range. A simple filter bank consisting of five filters is shown in the figure below. The center frequencies of each bandpass filter are denoted by f_1, f_2, f_3, f_4, and f_5, whereas their bandwidths are denoted by B_1, B_2, B_3, B_4, and B_5, respectively. There is a special name given to the center frequencies. For the mth filter, let the center frequency be f_m. Then f_m is known as the *center frequency*, the *midband frequency*, or the *geometric mean frequency*.[1] Note that in the figure, the bandwidths of all the filters are the same. That is, $B_1 = B_2 = B_3 = B_4 = B_5$.



One of the properties of bandpass filters is their *quality factor*, Q. The quality factor is a measure of how selective the filter is in passing frequencies around the center frequency and attenuating unwanted frequencies. It is defined as the ratio of the center frequency of the filter to its bandwidth.

$$Q = f_m/B_m$$

Thus, for a fixed center frequency, the larger the bandwidth, the smaller the quality factor, and vice versa. In the figure above, because the bandwidths of all the filters are the same, but the center frequencies are different, the quality factor is different for each filter.

1. Note that the actual term used depends on the relationship between f_m and the upper and lower passband frequency (f_U and f_L, respectively). This will be discussed in more detail later.

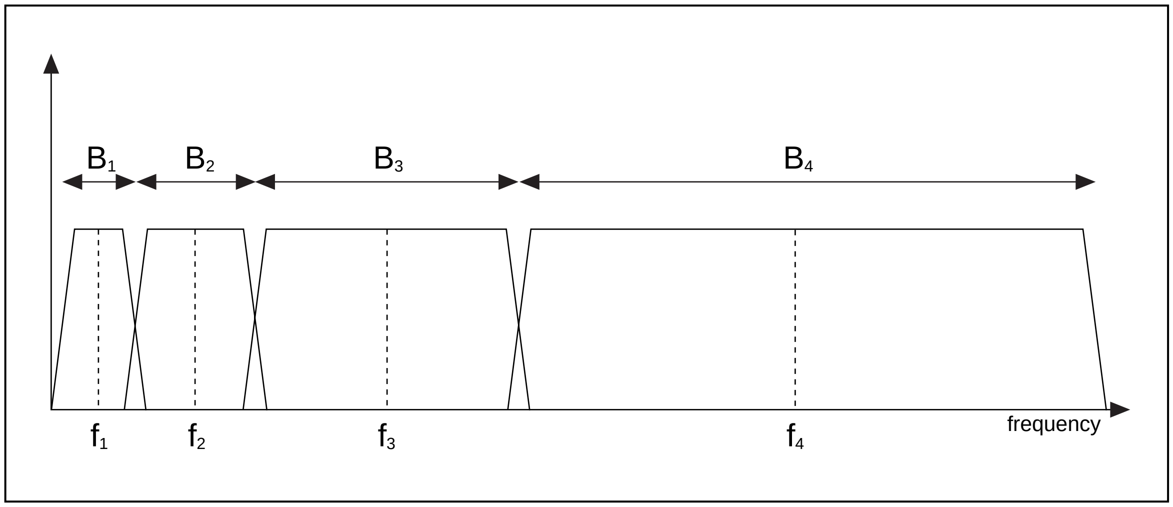
A practical application of a bandpass filter is in a radio or television receiver. Radio and television channels all occupy different frequency ranges. The antenna picks up signals of all the frequencies of all the channels. However, inside the radio or television receiver, the signal is passed through a bandpass filter used to select a particular channel (the one that you want to hear or see) and reject the other channels. When you change the tuning knob on your radio, or the channel selector knob on your TV, you are actually changing the center frequency of this bandpass filter. It is desirable for the bandpass filter to pass the selected channel and reject the unwanted channels. Hence, the bandpass filter needs to have a high Q so that you do not hear (or see) two or more channels at the same time.

B. Octave Analyzers

Now consider the filter bank shown in the figure below. It consists of four bandpass filters each with a different center frequency *and* different bandwidths. The center frequencies are denoted by f_1, f_2, f_3, and f_4, whereas the bandwidths are denoted by B_1, B_2, B_3, and B_4. The filters with higher center frequencies have larger bandwidths, and filters with lower center frequencies have lower bandwidths.

The special feature of this filter bank is that the center frequency of any of the filters is exactly twice (one octave higher) that of the center frequency of the filter just below it and exactly 1/2 (one octave lower) that of the center frequency of the filter just above it.

That is, $\frac{f_m}{f_{m-1}} = 2$ and $\frac{f_m}{f_{m+1}} = \frac{1}{2}$



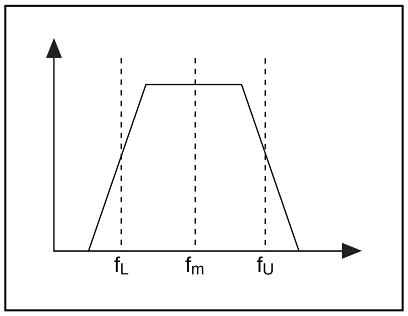
This same relationship holds true for the bandwidths of any two adjacent filters—the bandwidth of any filter is exactly twice that of the bandwidth of the filter just below it and exactly 1/2 that of the bandwidth of the filter just above it.

That is, $\frac{B_m}{B_{m-1}} = 2$ and $\frac{B_m}{B_{m+1}} = \frac{1}{2}$

Note that the quality factor = Q = f_m/B_m is now the same for all filters, because both f_m and the bandwidth of adjacent filters increase (or decrease) by the same amount. Therefore, these bandpass filters are known as *constant Q* bandpass filters.

Such a filter bank, where the ratios of center frequencies and bandwidths of adjacent filters are powers of 2 (2^1 = one octave), is known as an *octave analyzer*. Both the center frequencies and bandwidths increase logarithmically, and all the bandpass filters have the same value of Q. The bandwidth of each filter covers a frequency range of one octave.

For each bandpass filter, let f_m denote the center frequency, and f_U and f_L denote the upper and lower passband frequencies, respectively. This is shown in the figure below.



The relationship between them is that f_m is the geometric mean of f_U and f_L

$$f_m = \sqrt{f_u f_L}$$

and hence f_m is also known as the geometric mean frequency.

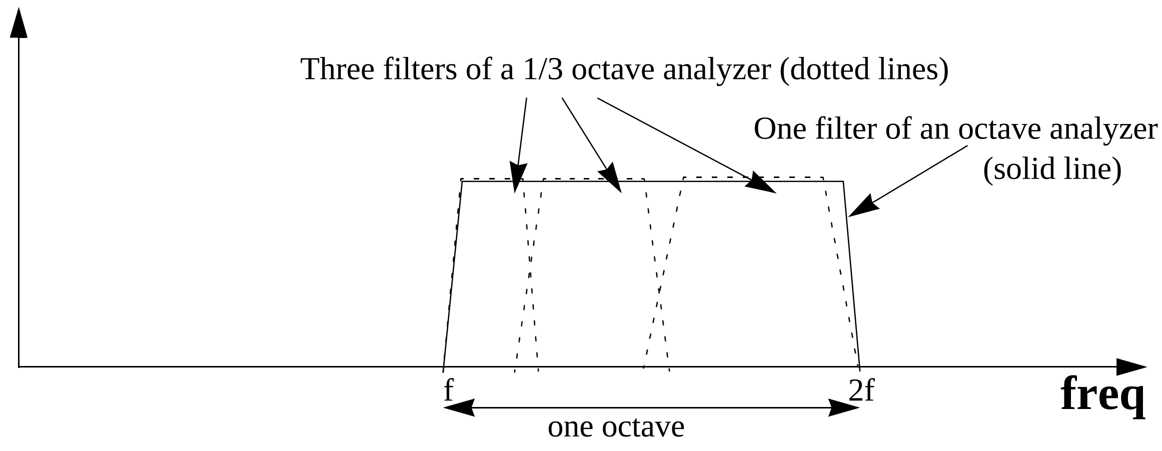
C. 1/3-Octave Analyzer or Third-Octave Analyzer

Now you can probably make a good guess as to what is a 1/3-octave analyzer, also known as a third-octave analyzer (TOA). It is very similar to an octave analyzer but with a different ratio of center frequencies and bandwidths of adjacent filters. It consists of a set of bandpass filters connected in parallel, where the ratio of center frequencies and bandwidths of adjacent filters is $2^{1/3}$.

$$\frac{f_m}{f_{m-1}} = 2^{1/3} \quad \text{and} \quad \frac{f_m}{f_{m+1}} = 2^{-1/3}$$

$$\frac{B_m}{B_{m-1}} = 2^{1/3} \quad \text{and} \quad \frac{B_m}{B_{m+1}} = 2^{-1/3}$$

Because the ratio is now $2^{1/3}$ (approximately = 1.26) it requires three of these filters to cover a frequency range of one octave. The net effect of this is that three filters of a 1/3-octave analyzer cover the same frequency range as one filter of an octave analyzer. This is shown in the figure below for one filter of the octave analyzer.


Three filters of a 1/3 octave analyzer (dotted lines)
One filter of an octave analyzer (solid line)
f
2f
freq
one octave

Note: ***In general, you can have a 1/n-octave analyzer where the ratio of center frequencies and bandwidths of adjacent bandpass filters is $2^{1/n}$. In practice, the most popular ones are 2^1, $2^{1/3}$, and $2^{1/12}$. These correspond to ratios with numeric values of 2, 1.26, and 1.06, respectively.***

For 1/3-octave analyzers also, there is $f_m = \sqrt{f_U f_L}$ where f_m is called the geometric mean frequency.

The filters in both octave and third-octave analyzers are also known as *constant percentage bandwidth* filters. That is, the bandwidth of the filters are a constant percentage of the center frequency. For an octave filter, the ratio of $f_2/f_1 = 2$, and its bandwidth is $1/\sqrt{2}$ = 71% of the center frequency. For a 1/3-octave filter, the ratio $f_2/f_1 = \sqrt[3]{2}$, and its bandwidth is $2^{1/6} - 2^{-1/6}$ = 23% of the center frequency.

D. Applications

The applications of 1/*n*-octave analyzers are mainly in the fields of acoustics and audio signal processing. Basically, you want to measure the sound or acoustic intensity in various frequency bands. A specific example is in the aviation industry, where measurement of noise emissions from an aircraft are necessary for it to get airworthiness certification. Federal aviation regulations require the use of real-time 1/3-octave analyzers for data analysis. Manufacturers of appliances and office equipment also tend to specify sound power levels in their product documentation so that customers can compare their products and choose the quieter one. By measuring sound power levels in a room or other enclosure such as a concert hall, acoustic engineers determine the proper material and design for the enclosure.

Other applications are in

- Vibration tests in aircraft and submarines
- Sound power determination
- Acoustic intensity measurement
- Audio equalization
- Architectural acoustics
- Appliance testing

The information obtained from a TOA can be used to reduce vibrations in aircraft and submarines, to reduce noise in aircraft, submarines, and household appliances, and to make sound (for example, music) more pleasant to hear (for example, equalizers and architectural acoustics).

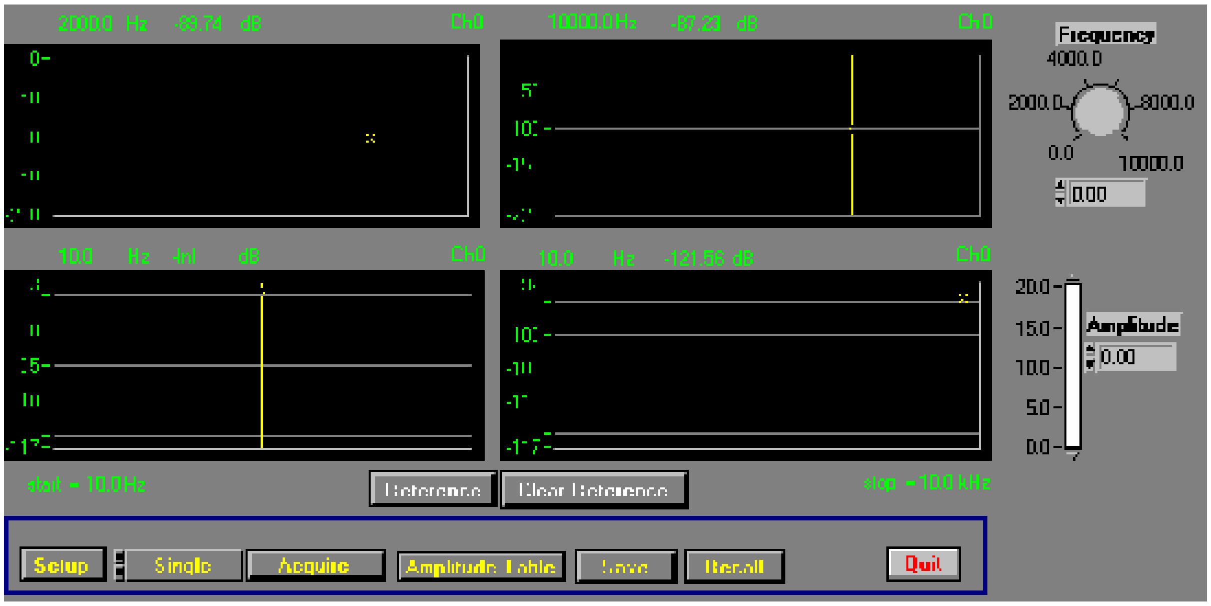
E. The Third-Octave Analyzer Toolkit

You saw that the applications of 1/n-octave analyzers are in those fields where there is a need to measure sound power in different frequency bands. To compare different measurements with each other, measurements are performed according to certain standards. The American National Standards Institute (ANSI) has specified the standards for the design of octave and third-octave filters. This information is found in the ANSI S1.11 (1986) *specifications for octave-band and fractional-octave-band analog and digital filters*. The National Instruments Third-Octave Analyzer Toolkit conforms to these specifications. The center frequencies of the BPFs of the third-octave analyzer, as given in the ANSI S1.11 (1986) specifications, are shown in the following table. Also shown is the frequency range corresponding to the passband of the filters. There are a total of 30 specified center frequencies spanning a range from 25 Hz – 20 KHz. However, in the National Instruments 1/3-octave analyzer, an additional BPF with a center frequency of 10 Hz has been added to the lower frequency range, giving a total of 31 BPFs.

Center freq. (Hz)	Passband (Hz)	Center freq. (Hz)	Passband (Hz)	Center freq. (Hz)	Passband (Hz)
25	22.4 - 28.2	400	355 - 447	6300	5620 - 7080
31.5	28.2 - 35.5	500	447 - 562	8000	7080 - 8910
40	33.5 - 44.7	630	562 - 708	10000	8910 - 11200
50	44.7 - 56.2	800	708 - 891	12500	11200 - 14100
63	56.2 - 70.8	1000	891 - 1120	16000	14100 - 17800
80	70.8 - 89.1	1250	1120 - 1410	20000	17800 - 22400
100	89.1 - 112	1600	1410 - 1780		
125	112 - 141	2000	1780 - 2240		
160	141 - 178	2500	2240 - 2820		
200	178 - 224	3150	2820 - 3550		
250	224 - 282	4000	3550 - 4470		
315	282 - 355	5000	4470 - 5620		

The Third-Octave Analyzer Toolkit can be used as a stand-alone application, an add-on toolkit for LabVIEW or BridgeVIEW, or with LabWindows/CVI. It can analyze the frequency of the signal in 31 frequency bands.

The figure below shows the front panel of the TOA stand-alone application. It can analyze up to four channels of data. Depending on the number of channels you choose, you could get either one, two, three, or four graphs on the front panel. In this particular figure, you see four graphs. The x-axis of each graph is the frequency scale and the y-axis of each graph is the power amplitude (in decibel) scale. Each graph also has a cursor that you can move to see the power value in each frequency band. This cursor is shown by the vertical line with an “x” through it. The value of the center frequency (in hertz) in the frequency band and the corresponding power value (in decibel) is displayed at the top of each graph.



On the front panel above, **Acquire** retrieves and analyzes a block of data. The analyzer does not start acquiring data until you click on this button.

The data can be acquired in *single* or *continuous* mode. The mode can be chosen by clicking on the up or down arrows on the control to the left of the **Acquire** button. When you select *single*, every time that you click the **Acquire** button, the analyzer acquires and analyzes a new block of data and displays the result on the corresponding graph. When you choose *continuous*, the analyzer starts to acquire and analyze data when you click the **Acquire** button. After it has displayed the data on the corresponding graph, it acquires the next block of data and analyzes and displays it. This continues until you click on the **Stop Acquire** button.

(The **Acquire** button becomes the **Stop Acquire** button during an acquisition.)

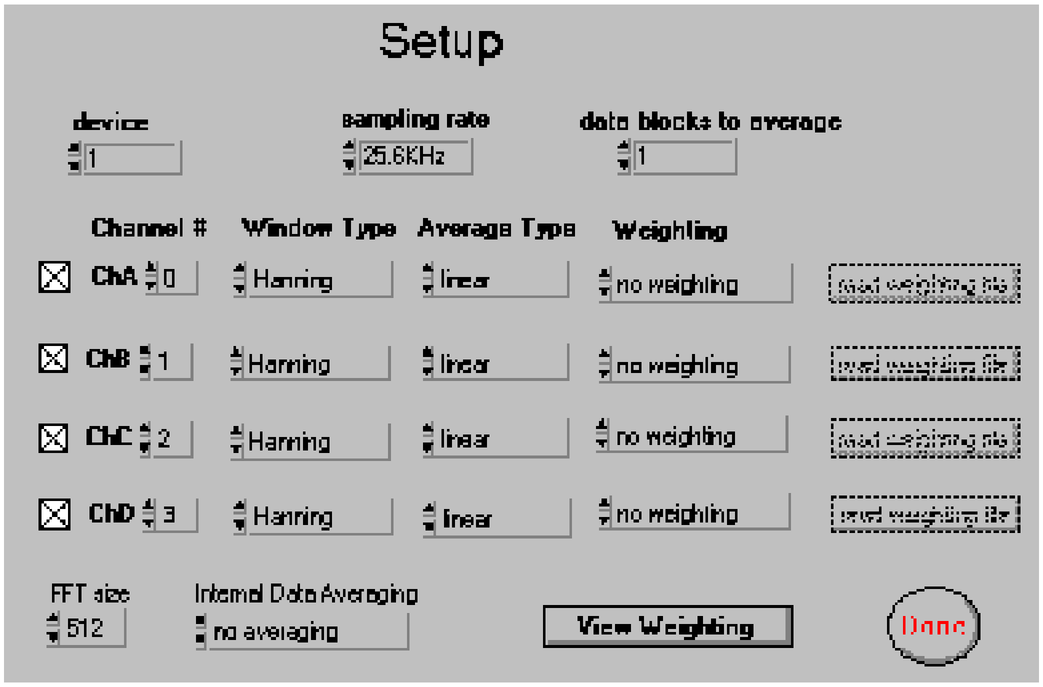
Amplitude Table shows a table with all 31 bands of output power values for each channel.

Save saves the 31 bands of power values in a spreadsheet format where each column represents 31 bands of the power value for each channel.

This button saves the power amplitude as well as some status information, such as channel numbers, window type, average type, and weighting value. We will discuss this status information later.

Quit stops the analyzer.

Setup opens the **Setup** panel shown below. The setup panel selects the parameters that you want to customize for your application. It appears when you first run the TOA. However, you can always access it at any time by clicking on the **Setup** button.



Setup

device: 1
sampling rate: 25.6KHz
data blocks to average: 1

	Channel #	Window Type	Average Type	Weighting	
☒	ChA 0	Hanning	linear	no weighting	read weighting file
☒	ChB 1	Hanning	linear	no weighting	read weighting file
☒	ChC 2	Hanning	linear	no weighting	read weighting file
☒	ChD 3	Hanning	linear	no weighting	read weighting file

FFT size: 512
Internal Data Averaging: no averaging

device specifies the identification number for your DAQ board.

In Windows, you can assign this number to your device through software. For the Macintosh, the device parameter indicates the physical slot number into which you insert your device. You can find this number by opening NI-DAQ in the **control panels** folder.

sampling rate determines the rate at which the signal is sampled, and thus the frequency range being analyzed (this is because as you have already seen, the sampling frequency must be at least twice the maximum frequency of the signal). Three choices of sampling rates are available: 12.8 KHz, 25.6 KHz, and 51.2 KHz. The corresponding frequency ranges being analyzed are: 5 Hz - 5 KHz, 10 Hz - 10 KHz, and 20 Hz - 20 KHz. These sampling rates were chosen because they are software selectable with the National Instruments dynamic signal acquisition boards (AT-A2150, NB-A2150F, and AT-DSP2200), which you can use with the TOA. The sample rate of 51.2 KHz conforms to the "Extended" range ANSI specifications.

Channel # indicates the channels from which to acquire the data and display it. Up to four channels can be chosen (and the channel number can also be the same in every control). Each channel has a box with a cross in it. If you do not need to use a channel, click inside the box until the cross disappears. This will disable all the parameters associated with that particular channel.

Window Type selects one of four commonly used windows (Rectangular, Hamming, Hanning, and Blackman) for each channel. The **Window Type** parameter defaults to the Hanning window.

Next, you will complete an exercise which uses the third-octave analyzer and highlights more of its features.

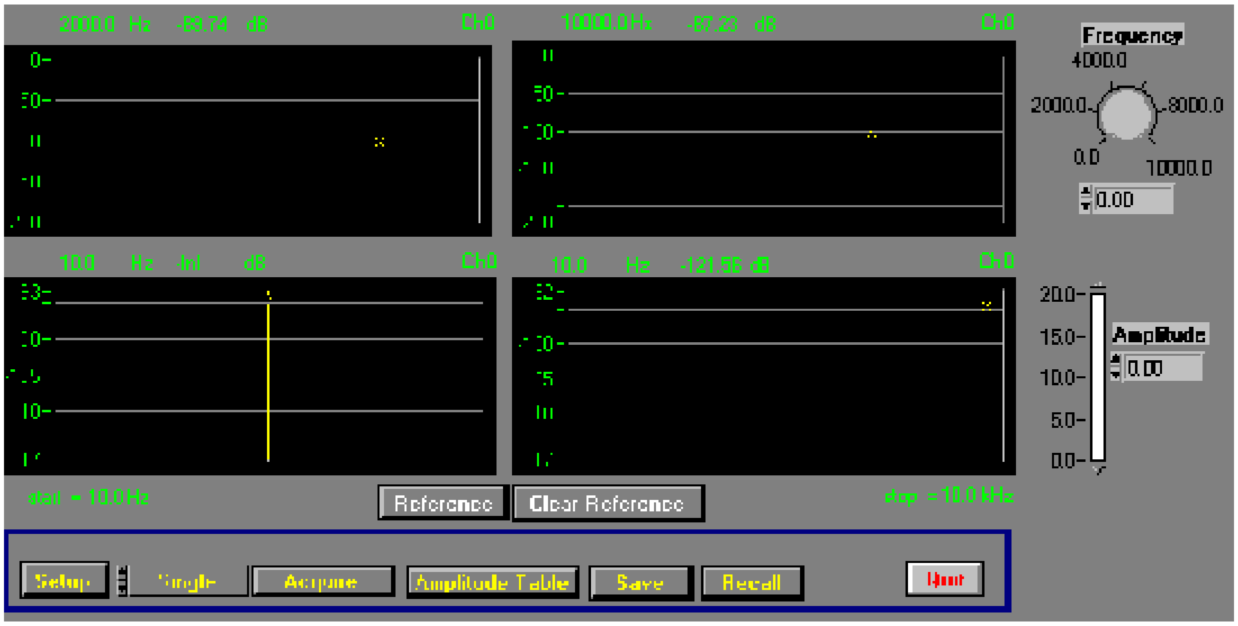
Exercise 12-1

Objective: To analyze a signal consisting of a sine wave with harmonics and added white noise.

Note: ***This exercise requires a modified library of the actual third-octave analyzer toolkit. If the file TOA\Octave.llb does not exist on your hard drive, you cannot do this exercise. In that case, you may read this exercise to become familiar with the front panel controls and indicators, and then move on to the next section.***

1. Open the **Third-Octave Analyzer** VI from the folder `Toa\Octave.llb`. You will see the following front panel:

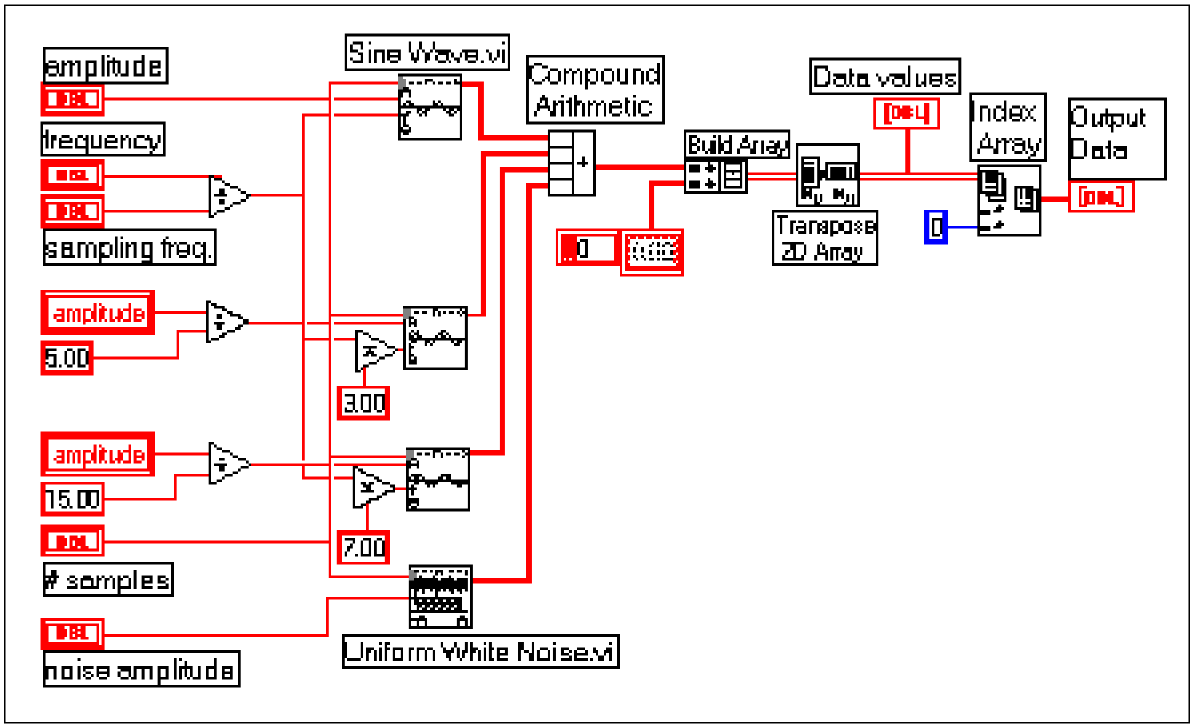
Front Panel



You see four graphs on the screen. This is because you can analyze signals coming from up to four channels. The number of graphs displayed on the screen depends on the number of channels you are analyzing.

The two controls (**Frequency** and **Amplitude**) on the right side of the front panel have been added only for the purpose of this exercise. They do not exist in the actual Third-Octave Analyzer Toolkit. They adjust the amplitude and frequency of a sine wave (with harmonics) that has been corrupted by white noise. On the front panel of the TOA, use the **Frequency** dial to control the frequency of the fundamental component of the sine wave between 0 - 10 KHz. Use the **Amplitude** slider control to adjust the amplitude of the fundamental between 0 - 20.

Although you do not need to be concerned about this for now, for those who are interested, the block diagram of the VI (**Signal.vi**) used to generate this simulated signal is shown below:

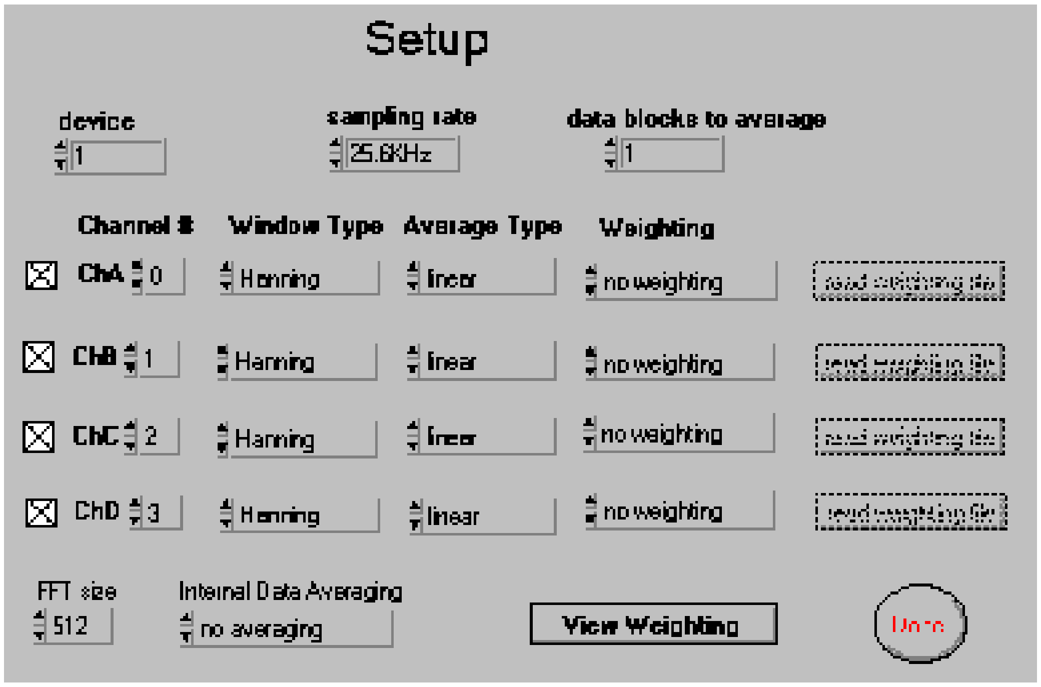


The above VI also adds to the fundamental its 3rd and 7th harmonics of amplitudes 1/5th and 1/15th, respectively. The addition of harmonics is simulated by the lower two **Sine Wave** VIs.

Setup Window

2. With the **Amplitude** and **Frequency** controls both set to 0.00, run the VI.

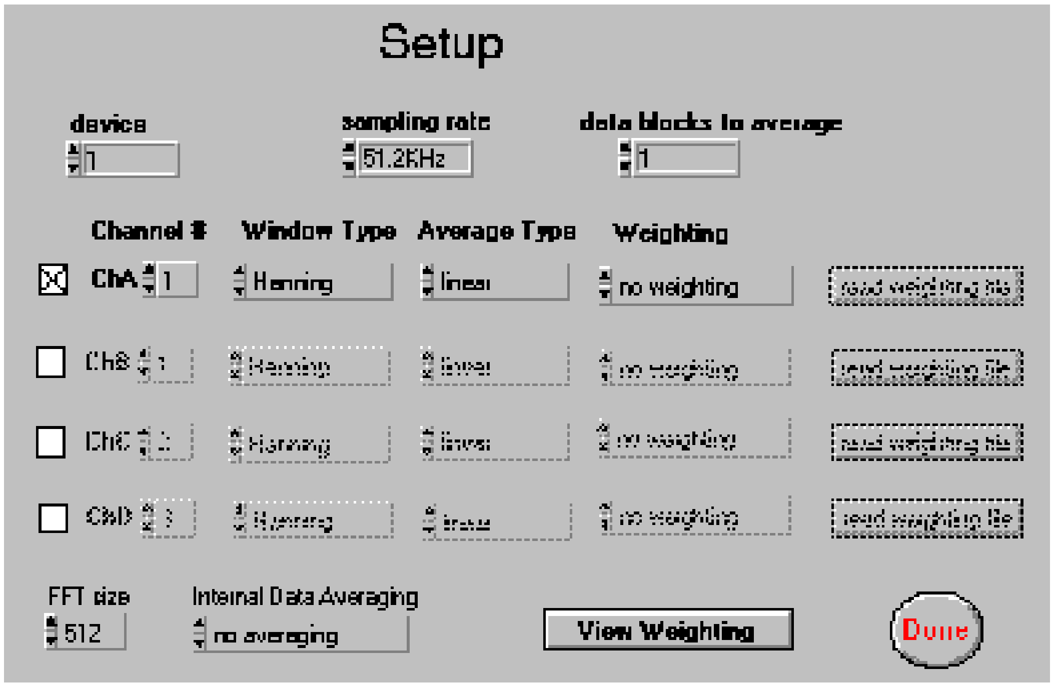
A new window called **Setup** appears.



The **Setup** window gives you control over the number of channels to sample, the sampling rate, FFT size, and much more. For now, you will leave the following controls at their default values:

device	1
sampling rate	25.6 K Hz
data blocks to average	1
FFT size	512
Internal Data Averaging	no averaging

3. In the **Setup** window, click on the white box to the left of the labels **ChB**, **ChC**, and **ChD**. When you click on the white boxes, the crosses in them disappear, and the selections for those channels are grayed out.



Setup

device: 1

sampling rate: 51.2KHz

data blocks to average: 1

	Channel #	Window Type	Average Type	Weighting	
[✓ ChA]	1	Hanning	linear	no weighting	read weighting file
ChB	1	Hanning	linear	no weighting	read weighting file
ChC	2	Hanning	linear	no weighting	read weighting file
ChD	3	Hanning	linear	no weighting	read weighting file

FFT size: 512

Internal Data Averaging: no averaging

4. Now, because only the white box to the left of **ChA** has a cross in it, you will use only **ChA** to acquire the signal.

   The device control corresponds to the number of the DAQ board in your computer. You can obtain the correct device number for your DAQ board by using the NI-DAQ Configuration Utility. For the purpose of this exercise, leave device set to 1.

   The **Channel #** control determines the particular channel on the DAQ board from which you will acquire the sampled signal. (A DAQ board usually has more than one channel—typically 8 or 16.) Leave this set to 0.

   The **Window Type** control determines the type of window you want to apply to the data samples.

5. Click on the **Done** button. The **Setup** window will disappear, and you will return to the main panel of the TOA. You can bring up the **Setup** window at any time to change the configuration parameters by clicking on the **Setup** button on the bottom left of the main panel of the TOA. Note that only one graph is displayed, because you had selected only 1 channel in the **Setup** window.

Single and Continuous Acquisition

6. The **Single** button (just next to the **Setup** button) is a menu ring control that determines how the samples of the data are collected. When set to Single mode, only one block of data is acquired when

the **Acquire** button is pressed. However, when set to Continuous mode, blocks of data are acquired one after the other when the **Acquire** button is pressed. While in the continuous mode of operation, the **Acquire** button changes to the **Stop Acquire** button. To stop the continuous data acquisition, you must press the **Stop Acquire** button.

7. With the menu ring control on Single, click the **Acquire** button.

   After a short time interval during which the data is obtained and the calculations for the power output are being done, a histogram appears on the graph. This is the histogram of white noise only (there is no signal because the amplitude of the sine wave has been set to 0.0 on the slider control). Each rectangle of the histogram gives the power in the frequency band of the corresponding BPF. Note that the power increases as you go toward higher frequencies. Why do you think that is so?

   Hint: White noise has a flat frequency spectrum (that is, the same average power at all frequencies). The filters used in the TOA have increasing bandwidths as the center frequency increases.

8. Click on the **Acquire** button again. Once again, after the data is obtained and the calculations are over, the display changes. The data is obtained, the calculations done, and the display updated each time you click on the Acquire button. However, if you do not click the **Acquire** button, nothing happens. This is the operation of the Single acquisition mode.

9. Click on the arrows of the **Single** button and change it to Continuous mode. Click on the **Acquire** button to start the acquisition.

   Now the operation of obtaining data, calculating the power outputs of the BPFs, and updating the data is done continuously. Note that the **Acquire** button has changed to the **Stop Acquire** button. You must click on the **Stop Acquire** button if you want to stop the continuous data acquisition.

Changing the FFT Size

10. You will notice a time delay between successive updating of the display. This is because the default size of the FFT has been set to 512 data points. It takes a certain amount of time to acquire the data points, perform the necessary calculations, and display the results. To decrease this time, you can choose a smaller FFT size.

    Click on the **Setup** button. The **Setup** window comes up. Set FFT size to 256 and click on the **Done** button. Now notice that the display is much faster.

Adding a Signal to the White Noise

11. Set the **Frequency** control to 100 Hz and the **Amplitude** control to 10. The display will show you three peaks. The highest peak is from the output of the BPF that covers the frequency range in which the fundamental sine wave of frequency 100 Hz falls. The other two peaks are from the outputs of the BPFs whose frequency bands include the 3rd and 7th harmonics (that is, 300 Hz and 700 Hz).

Cursor Control

12. Move the cursor around the display. On the top left, you can see the power output (in dB) and the corresponding center frequency (in Hz) of the BPF corresponding to the rectangle on which you place the cursor.

The Amplitude Table

13. Another way to see the power in each of the different frequency bands is to click on the **Amplitude Table** button. A new window comes up with a table showing the center frequencies and the corresponding power output (in dB) of each of the BPFs. To close this window, click on the **Ok** button.

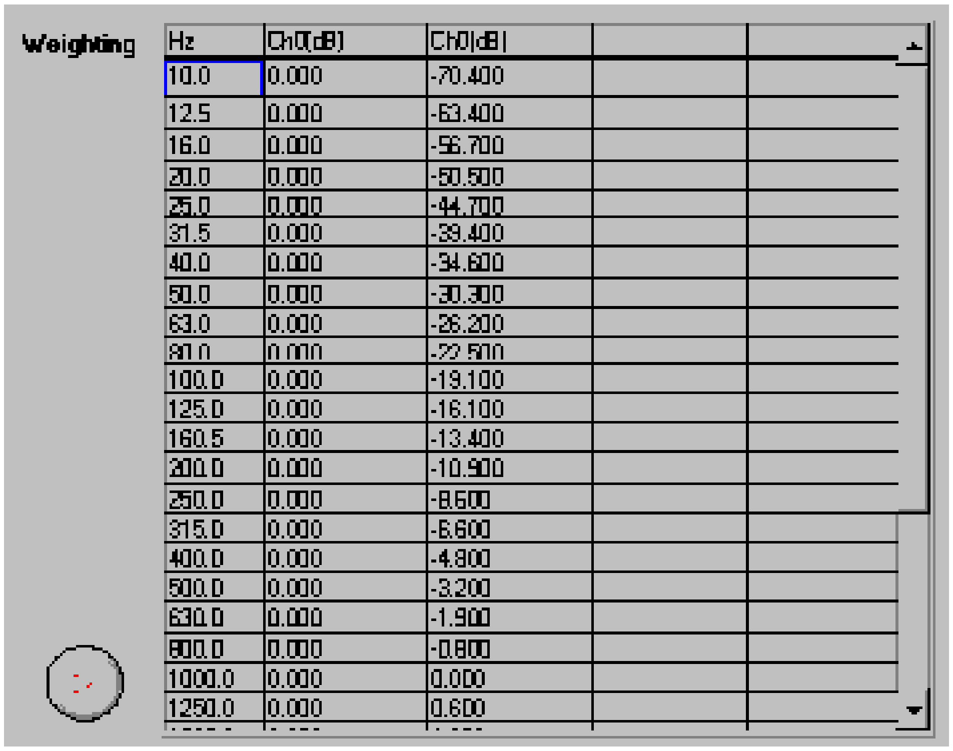
Multiple Channels and Weighting

14. You will now acquire and display data from more than one channel.

    Click on the **Setup** button to bring up the **Setup** window. In the **Setup** window, click on the white box to the left of the label **ChB** and change its **Channel #** to 0 (the same as for **ChA**). Change the **Weighting** of **ChB** to A-weighting.

    You have just sampled the same signal on both channels A and B. However, the signal obtained on **ChA** is displayed with no weighting, whereas the signal obtained on **ChB** is displayed with A-weighting. Weighting is done because the human ear responds differently to different frequencies. By using A-weighting, the analyzer mimics the human hearing response to audio signals. Internally, the analyzer accomplishes this by adding a specified weighting value (in dB) to the calculated power before displaying it. Other weighting selections that are available are none (no weighting) and custom (user defined).

15. In the **Setup** window, click on the **View Weighting** bar to see a table showing the weighting values for each frequency band for each channel. Click on the **OK** button to close the window showing the table.



Weighting

Hz	Ch0[dB]	Ch0|dB|		
10.0	0.000	-70.400		
12.5	0.000	-63.400		
16.0	0.000	-56.700		
20.0	0.000	-50.500		
25.0	0.000	-44.700		
31.5	0.000	-39.400		
40.0	0.000	-34.600		
50.0	0.000	-30.300		
63.0	0.000	-26.200		
80.0	0.000	-22.500		
100.0	0.000	-19.100		
125.0	0.000	-16.100		
160.5	0.000	-13.400		
200.0	0.000	-10.900		
250.0	0.000	-8.600		
315.0	0.000	-6.600		
400.0	0.000	-4.800		
500.0	0.000	-3.200		
630.0	0.000	-1.900		
800.0	0.000	-0.800		
1000.0	0.000	0.000		
1250.0	0.000	0.600		

16. In the **Setup** window, click on the Done button.
17. Now you are back to the main front panel of the TOA. The top graph shows the display from **ChA** (with no weighting) and the bottom graph shows the display from **ChB** (with A-weighting). You can thus compare the outputs from different channels.
18. Click on the **Stop Acquire** button to stop the data acquisition.
19. Click on the **Quit** button to stop the VI.
20. Close the VI. Do not save any changes.

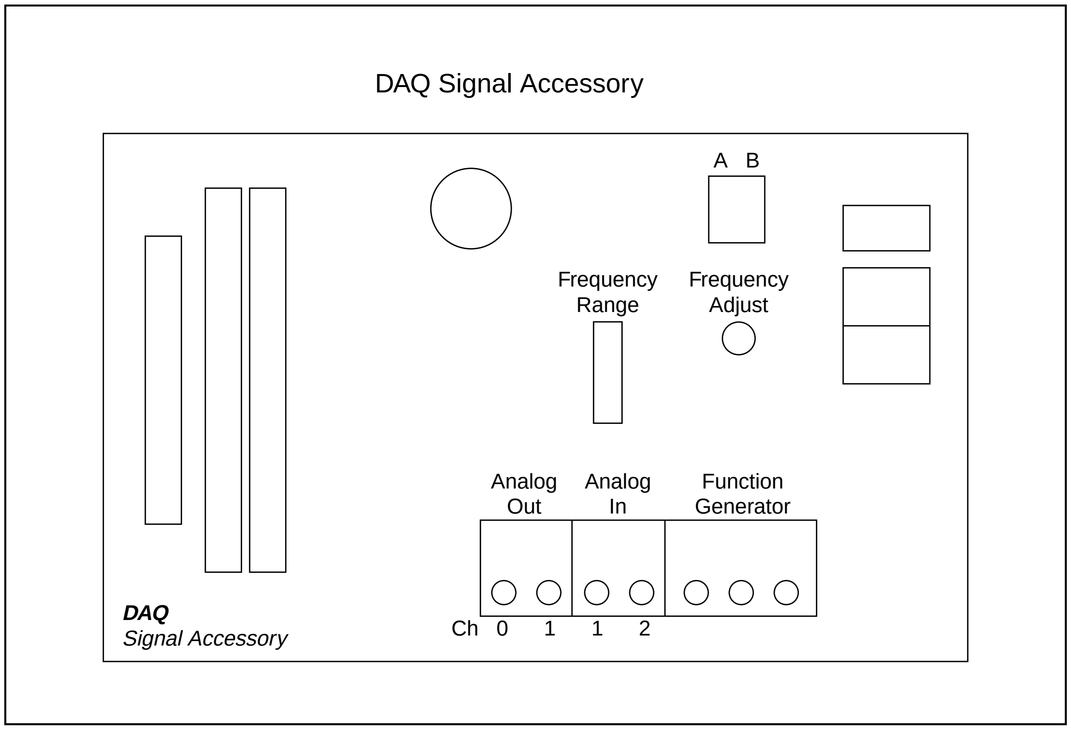
End of Exercise 12-1

F. Calibration Using the Third-Octave Analyzer

You can also use the TOA in applications involving calibration. For this purpose, the first step is to apply to the TOA a reference signal from a highly accurate source. The result obtained is then saved (by clicking on the **Save** button) as a reference for future use. This previously recorded signal can be loaded later for use as a reference by clicking on the **Reference** button. You can compare this reference signal with the current signal you are obtaining, because the analyzer plots both signals on the same graph.

The following three exercises outline the steps on how to use the TOA for calibration. In the next exercise, you will generate a reference signal. In the exercise following that, you will save the results of third-octave analysis on this reference signal. In the final exercise, you will compare these results with those obtained from another source.

The reference signal is obtained from the function generator on a DAQ Signal Accessory. Some of the connections on the box are as shown below.



You will use this box to generate a sinusoidal signal of frequency 200 Hz and peak amplitude of 1 V. This will be your reference signal.

Exercise 12-2

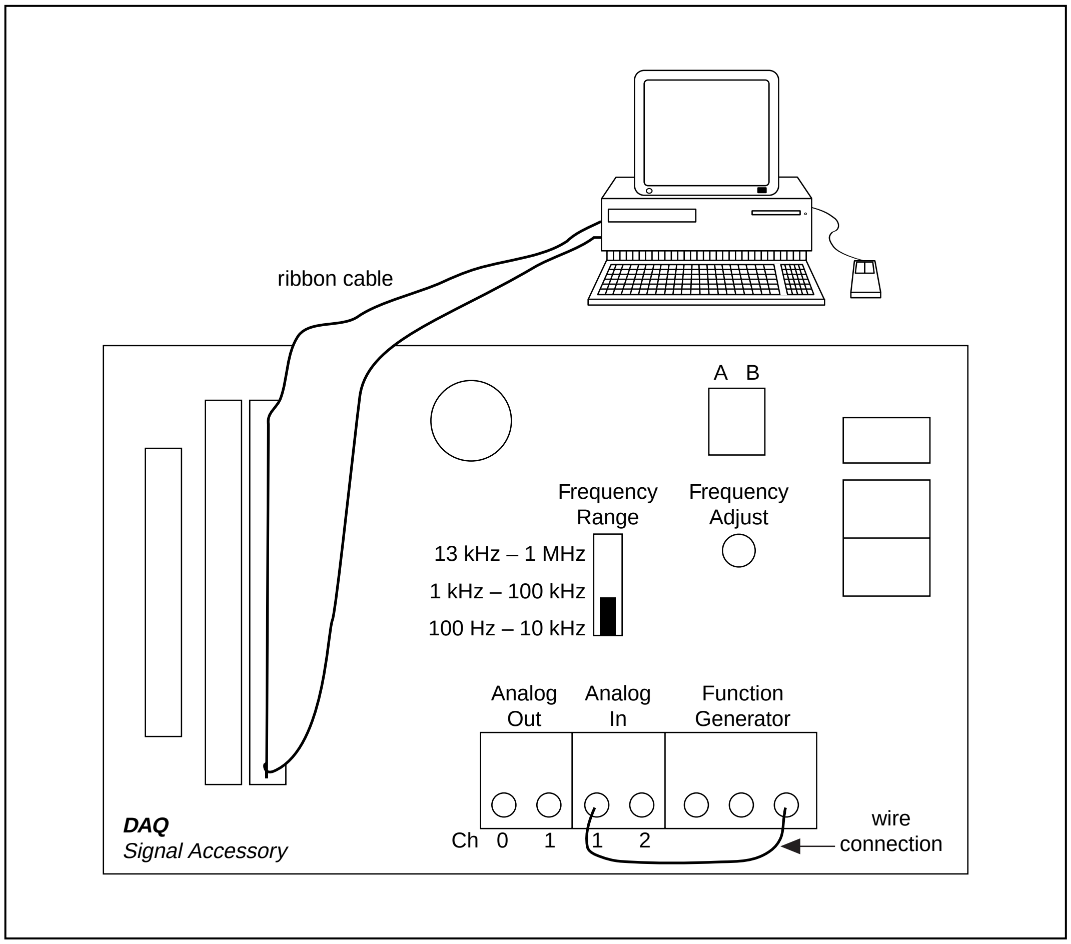
Objective: To generate a sinusoidal reference signal of frequency 200 Hz and peak amplitude of 1.0 V.

1. Make the following connections between the DAQ Signal Accessory and data acquisition board on your computer.

   Connect the sine wave output of the Function Generator to Analog In channel 1. Thus the sine wave will be available at channel # 1 of the DAQ board.

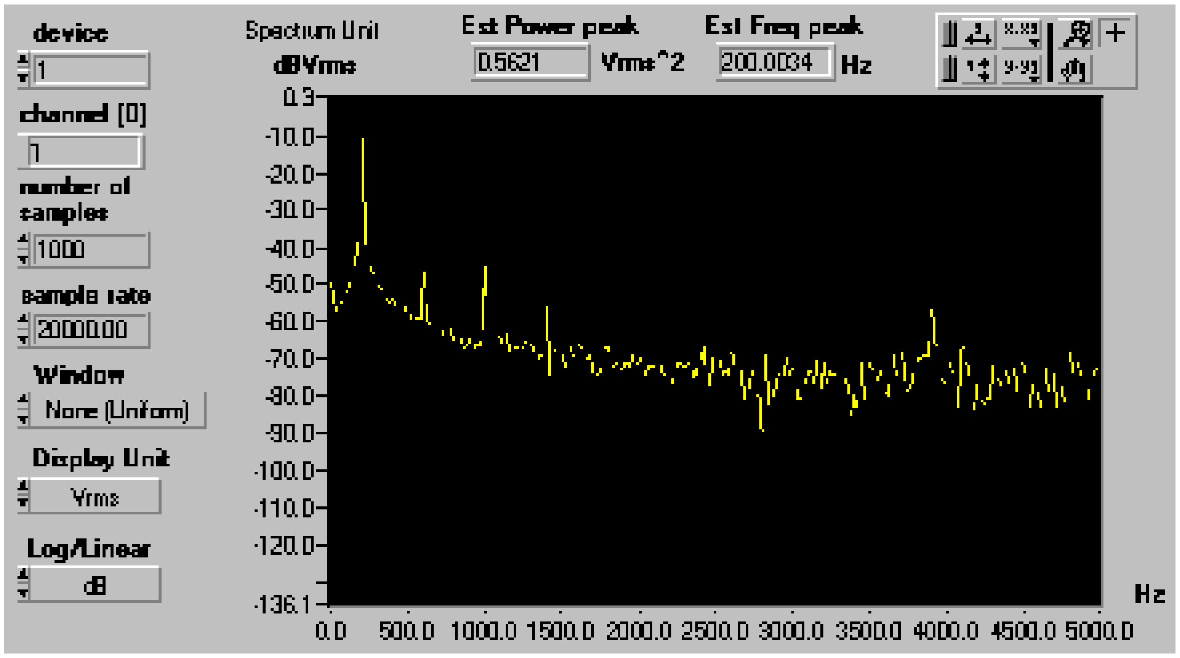
   Set the Frequency Range slider control to 100Hz-10kHz.

   Connect the DAQ Signal Accessory cable connection through a ribbon cable to the DAQ board on your computer.



2. Bring up LabVIEW (or BridgeVIEW) and load the **Examples » Analysis » measure » daqmeas.llb » Simple Spectrum Analyzer** VI. You will use this VI to measure the exact amplitude and frequency

of the sinusoidal signal from the Function Generator output of the DAQ Signal Accessory.



device: 1
channel [0]: 1
number of samples: 1000
sample rate: 20000.00
Window: None (Uniform)
Display Unit: Vrms
Log/Linear: dB
Spectrum Unit: dBVrms
Est Power peak: 0.5621 Vrms^2
Est Freq peak: 200.0034 Hz

3. Make the following changes to the default settings on the **Simple Spectrum Analyzer** VI:

   Enter a value of 1 in the **channel(0)** string control.

   Change number of samples to 1000.

   Select None (Uniform) window in the **Window** control.

   Leave all the other settings at their default values.

Note: ***With a sampling rate of 20000 and number of samples selected as 1000, the frequency spacing is 20000/1000 = 20 Hz. The FFT lines will thus fall on frequency values of 0, 20, 40, 60, and so on. The signal of 200 Hz will therefore fall exactly on the eleventh FFT line.***

4. Run the VI. In the **Est Power Peak** and the **Est Frequency Peak** indicators on the top, you will see the estimate of the power and frequency of the highest peak in the spectrum.
5. Adjust the **Frequency Adjust** control on the DAQ Signal Accessory to get an **Est Frequency Peak** of 200 Hz (or as close to it as possible). (You will need to run the VI each time after you change the **Frequency Adjust** control.)
6. Once you obtain a frequency of 200 Hz, note the value in the **Est Power Peak** indicator. This value is in units of V^2_{rms}. You can

convert this to a peak voltage (V_{peak}) reading by using the following formula:

$$V_{peak} = \sqrt{2}\sqrt{V_{rms}^2}$$

You should obtain a value near 1.0 V.

7. Make a note of the frequency and the corresponding voltage V_{peak}. Be sure not to change the settings on the DAQ Signal Accessory.
8. When you are done, close the VI. Do not save any changes.

End of Exercise 12-2

You will use the sinusoidal signal of the frequency and amplitude that you have noted in the previous exercise, as the reference signal against which we can compare other signals. This reference signal has a frequency of 200 Hz and a peak amplitude of 1.0 V.

☞ **Note:** ***The method you have used in the previous exercise to generate the reference signal is just to illustrate the principles involved in calibration of the TOA. In practice, the reference signal should be obtained from a more accurate and reliable source.***

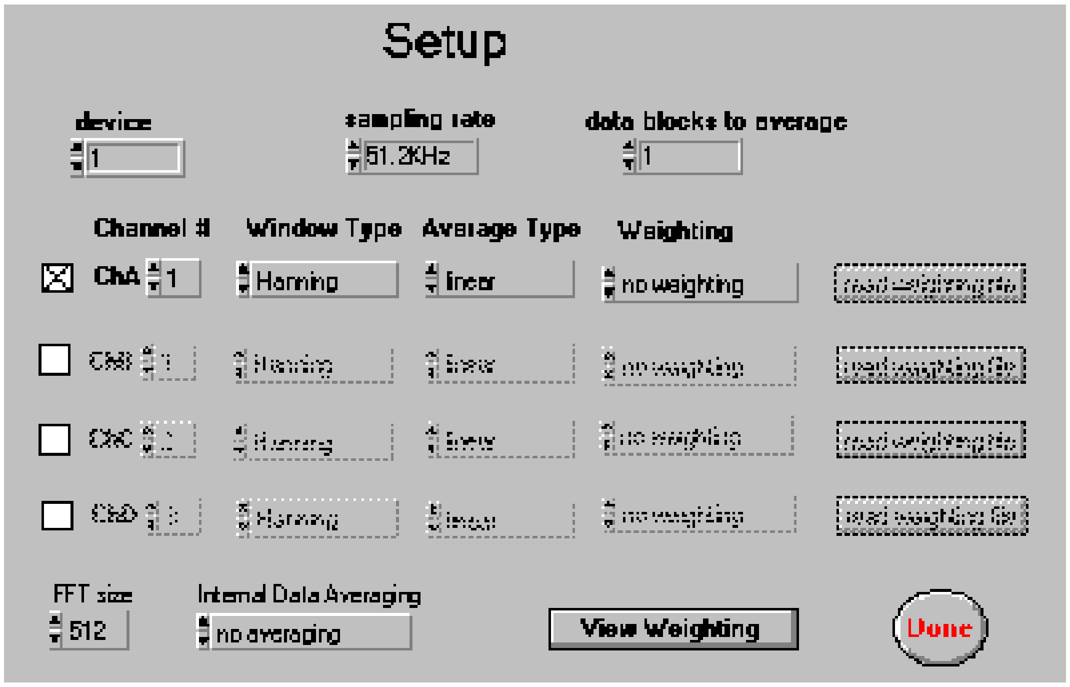
Exercise 12-3

Objective: To save the results obtained from the TOA for a standard reference signal.

In this exercise, you will use the same connections (between the DAQ Signal Accessory box and your computer) as those you had before in the previous exercise.

1. Run the **Third-Octave Analyzer** from the **Start » Programs** menu.
2. In the **Setup** panel, make sure only **ChA** is selected. This is done by clicking in the white box to the left of **ChB**, **ChC**, and **ChD**, so that the white box does not have a cross in it, and also that the selections for these channels are grayed out.
3. Change the **Channel #** setting to 1. Leave the other settings at their default values as shown:

device:	1	**sampling rate**:	51.2KHz
data blocks to average:	1	**Window Type**:	Hanning
Average Type:	linear	**Weighting**:	no weighting
FFT size:	512	**Internal Data Averaging**:	no averaging



Setup

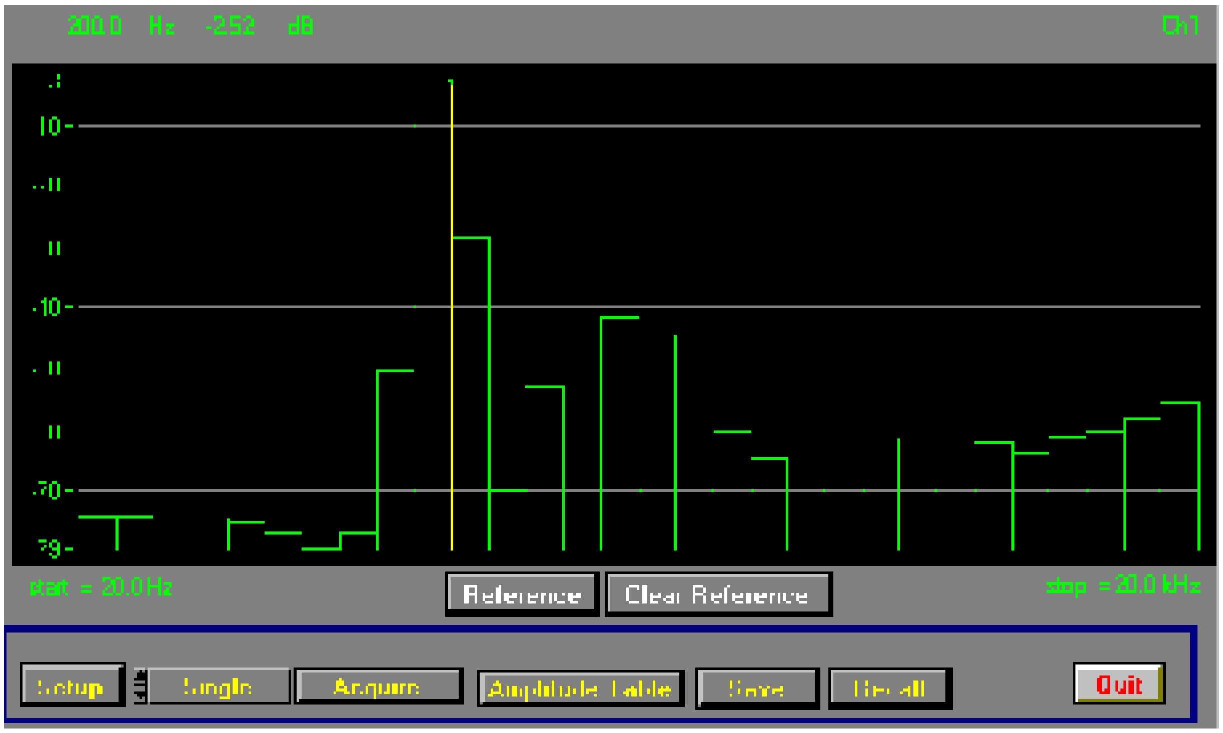
device 1 · sampling rate 51.2KHz · data blocks to average 1

	Channel #	Window Type	Average Type	Weighting	
☒ ChA	1	Hanning	linear	no weighting	read weighting file
☐ ChB	[illegible]	Hanning	linear	no weighting	read weighting file
☐ ChC	[illegible]	Hanning	linear	no weighting	read weighting file
☐ ChD	[illegible]	Hanning	linear	no weighting	read weighting file

FFT size 512 · Internal Data Averaging no averaging

4. When you are finished, click on the **Done** button at the bottom right.
5. On the TOA front panel, press the **Acquire** button. The VI has now obtained samples of the sinusoidal signal of frequency 200 Hz and

peak amplitude of 1 V and is showing the results of third-octave analysis on this signal.



Move the cursor so that it aligns with the band corresponding to the largest peak in the display. The frequency reading on the top of the display should read 200 Hz. The amplitude reading should be near -3.0 dB (which corresponds to a peak amplitude of 1.0 V).

You will now save these results as a reference that can be compared with other signals.

6. Click on the **Save** button. When prompted for the filename, type in `standard.ref`. You could use any filename, but it is recommended that you use one with a `.ref` extension if you plan to use it as a reference file.

   The file `standard.ref` now contains the results of third-octave analysis on the reference signal.

7. Exit the VI by clicking on the **Quit** button at the bottom right.

End of Exercise 12-3

Exercise 12-4

Objective: To compare the results of third-octave analysis on a given signal with that obtained from a standard reference signal.

In the previous exercise, you had saved the results of third-octave analysis on our standard reference signal. In this exercise, you will compare these results with those obtained from another source.

Exchange your DAQ Signal Accessory with someone else in the class and make the connections as you had done in the previous two exercises. Assume that your previous DAQ Signal Accessory was well calibrated and of high enough quality to generate the standard reference signal. You will now compare those results with those obtained using the other DAQ Signal Accessory.

☞ **Note:** ***Do not change the settings on the other DAQ Signal Accessory.***

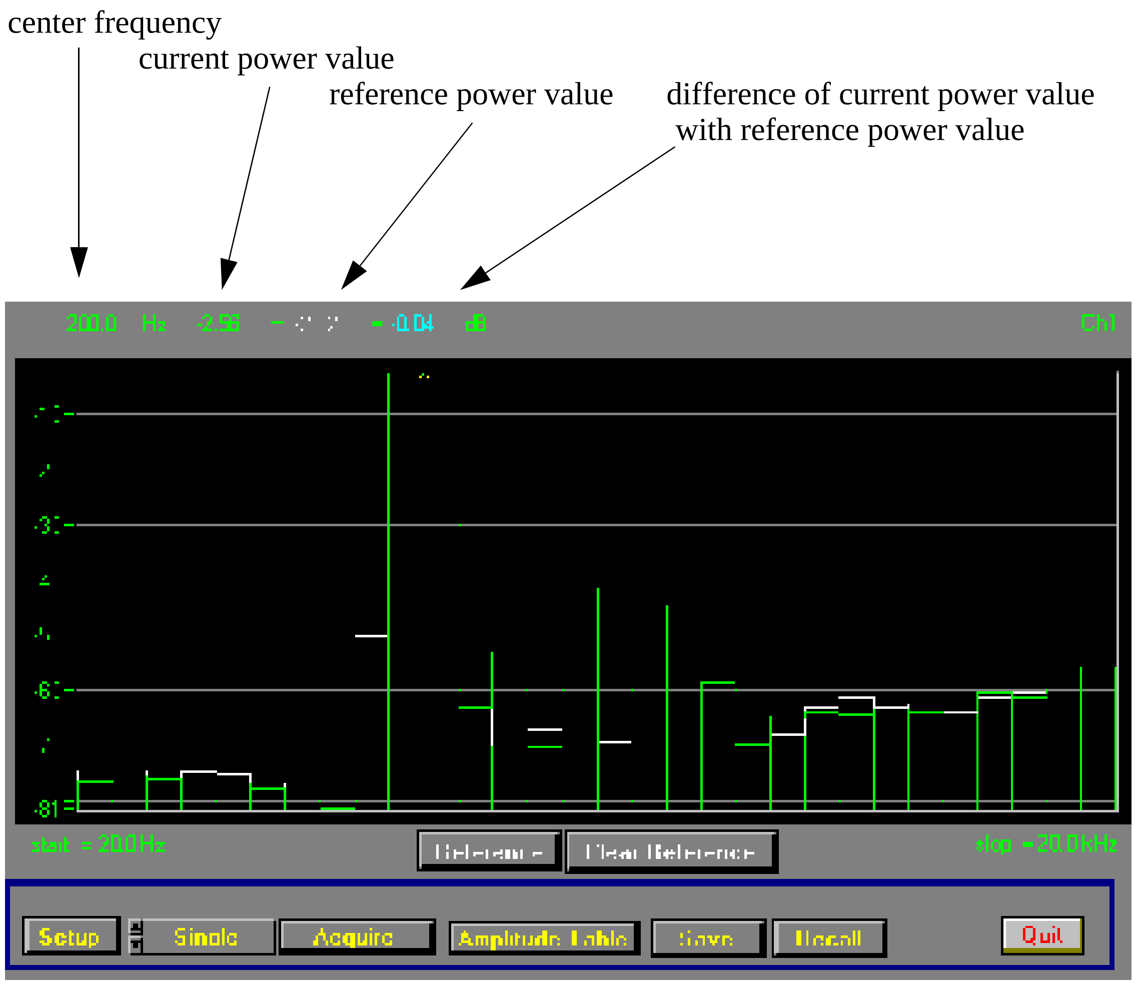
1. Open the **Third-Octave Analyzer** from the **Start » Programs** menu.
2. In the **Setup** panel, make the selection as you had done in the previous exercise. That is, select only **ChA**, with a corresponding **Channel #** of *1*. The other controls have the following values:

device:	1	**sampling rate**:	51.2KHz
data blocks to average:	1	**Window Type**:	Hanning
Average Type:	linear	**Weighting**:	no weighting
FFT size:	512	**Internal Data Averaging**:	no averaging

3. After verifying the above settings, click on the **Done** button.
4. On the front panel, click on the **Reference** button. A new window comes up from which you can select the file `standard.ref` that you had saved in the previous exercise.

   This file contains the results of third-octave analysis on a reference signal of 200 Hz having a peak amplitude of 1 V. The results stored in the file will be plotted in white.
5. Click on the **Acquire** button to obtain a single block of data. The results will be overlaid on the same plot as that of the reference signal.

6. Move the cursor around the plot. For the band on which the cursor is placed, the top left of the display shows you the center frequency, the dB levels of the two plots, and the difference in dB levels of the two plots. The figure below shows the display for the band with the 200 Hz center frequency.



7. To calibrate the new DAQ Signal Accessory that you obtained, you need to adjust the amplitude and frequency on the new box so that the difference in power reading in the band with a center frequency of 200 Hz is zero. That will mean that both the DAQ Signal Accessory boxes are producing signals with exactly the same frequency and amplitude.
8. When you finish, exit the VI by clicking on the **Quit** button.

End of Exercise 12-4

Summary

In this lesson, you learned the basic theory behind the operation of octave analyzers. You have seen that they consist of a set of bandpass filters. The ratio of the center frequencies and bandwidths of adjacent bandpass filters is equal to 2 for an octave analyzer and $2^{1/3}$ (approximately 1.26) for a TOA. That means that as you go higher in frequency, the center frequency and bandwidth of one BPF is twice that of the adjacent lower frequency BPF for the octave analyzer (or 1.26 times that of the adjacent lower frequency BPF for a third-octave analyzer).

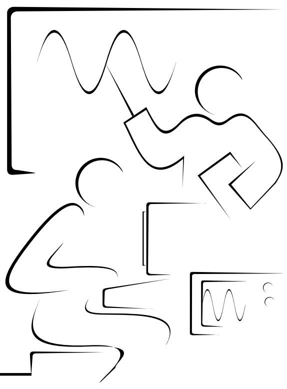
You used the TOA on a signal consisting of a sine wave (and its harmonics) with added white noise. Specifically, you learned:

- How to select the number of channels.
- The difference between single and continuous modes of operation.
- How to change the FFT size.
- About weighting the power output of the BPFs.
- How to see the power output and center frequency values by using the cursor or the Amplitude Table control.

Finally, you saw how to use the TOA to compare the power levels between two signals, one of which is a reference signal.

Notes

Lesson 13
Joint Time-Frequency Analysis



Introduction

In this lesson, you will learn the basic theory behind joint time-frequency analysis (JTFA) and how to use the JTFA Toolkit in a variety of applications.

You Will Learn:

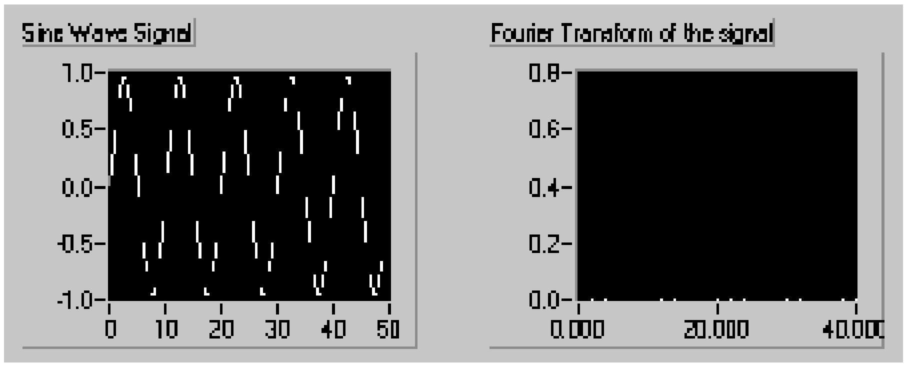
A. Why you need joint time-frequency analysis.

B. About joint time-frequency analysis.

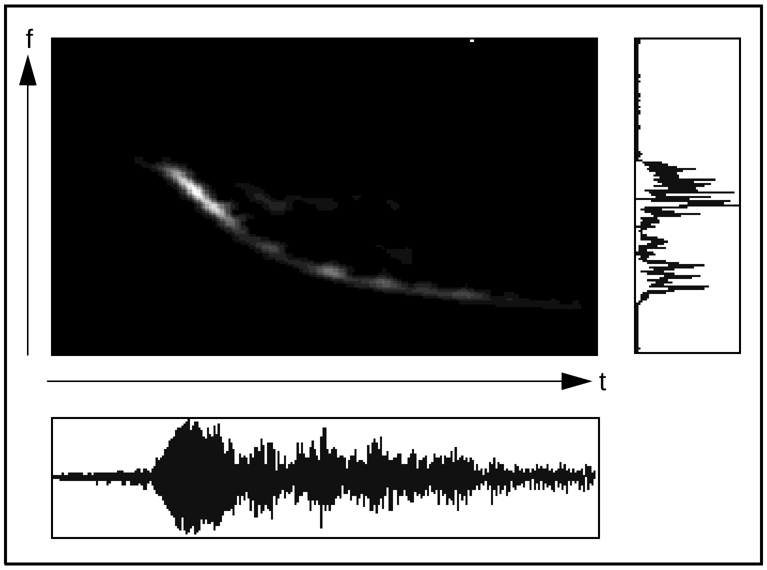
C. About the JTFA Toolkit and its applications.

A. Why Do You Need Joint Time-Frequency Analysis?

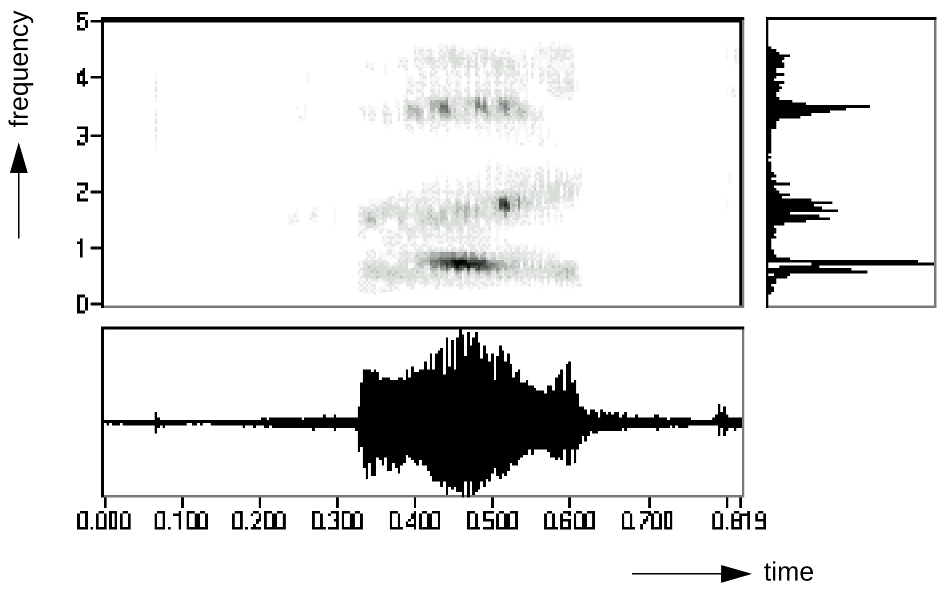
A signal can be represented in a number of different ways. You want the representation that most accentuates the qualities of the signal in which you are interested. You may be very familiar with the representation of a signal as a function of time. This representation shows how the signal magnitude changes over time. Alternately, you can also represent the signal as a function of frequency by performing a Fourier transform. The figure below shows the time representation of a sine wave and its Fourier transform.



Traditionally, signals are studied as a function of time or frequency, but not both. However, a number of signals encountered in real-world applications have time-dependent frequency representations. One such example is signals representing the tones of music that vary with time. Therefore, in many practical applications, it is very useful to characterize the signal in time and frequency domains simultaneously.



The figure above shows the time-frequency plot of a bird sound. The plot on the right is the standard Fourier spectrum. From this spectrum alone, you cannot tell how the frequencies have changed over time. Beneath the time-frequency plot is a plot of the time waveform of the bird sound. This shows you only how the sound level changes as a function of time. The advantage of having a time frequency plot is that not only can you tell what the range of frequencies were, but also how these frequencies changed as a function of time. For this bird sound, you can see that at the beginning, the bird was making a sound at higher frequency, which then changed as a function of time, as indicated on the graph. Furthermore, in the time-frequency plot, not only can you see how the frequency changed in time, but you can see the intensity of the frequency, indicated by the relative brightness levels of the plot.



The figure above shows graphs of a speech signal. The plot at the bottom of the figure is the time waveform of such a signal. This plot shows how the magnitude of this signal changes as a function of time. The plot in the upper right corner is the frequency representation of the same signal. This representation reveals four prominent frequency tones in the spectrum. However, from the spectrum alone, you cannot tell how these frequencies evolve over time. This is where joint time-frequency analysis comes into the picture. The 2-D contour plot in the figure above is the result of such an analysis. This plot is the time-dependent spectrum, a function of both time and frequency, and reveals how different frequency tones evolved as a function of time. From this, not only can you see how the frequency changed, but you can also see the intensity of the frequencies, as shown by the relative brightness levels of the plot. Consequently, by using JTFA, you can better understand the mechanism of human speech.

From these two examples, you have seen that time-frequency analysis offers a better understanding of the nature of the signal. Several other applications in this lesson illustrate this point.

B. Joint Time-Frequency Analysis

This section briefly introduces different techniques for joint time-frequency analysis and outlines the algorithms associated with each technique. Refer to *Joint Time-Frequency Analysis: Methods and Applications* by Shie Qian and Dapang Chen, Prentice Hall, 1996 for a detailed discussion of these algorithms.

There are two types of joint time-frequency representations, namely the linear and the bilinear (quadratic). All linear transformations are achieved by comparing the signal to be analyzed with a set of prudently selected elementary functions, known as the analysis function. As an example, consider the Fourier transform $F(\omega)$ of a signal *f(t)*.

$$F(\omega) \propto \int f(t)e^{-j\omega t}$$

In this case, $e^{-j\omega t}$ is the analysis function. The elementary functions selected in the inverse transformation are called synthesis functions. For example, in the inverse Fourier transform,

$$f(t) \propto \int F(\omega)e^{j\omega t}$$

$e^{j\omega t}$ is the synthesis function. The short-time Fourier transform (STFT) and the wavelet transform are two widely used methods to obtain a linear representation. Wavelets are discussed in detail in the next lesson.

Short-Time Fourier Transform (STFT)

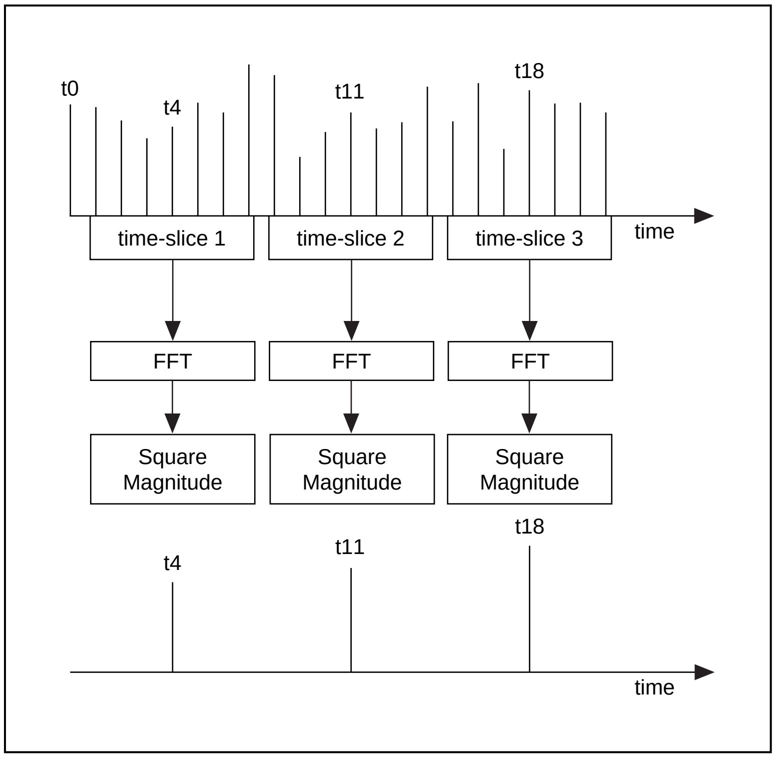
Similar to the example seen above, for the continuous-time STFT, the analysis function and the synthesis function have the same form. In this case, the inverse problem—that is, recovering the original time functions—is very simple.

For digital signal processing applications, it is necessary to extend the STFT framework to discrete-time signals. For a given discrete

signal $s[i]$, the following equation defines the STFT for $0 \le k \le \frac{L}{2}$:

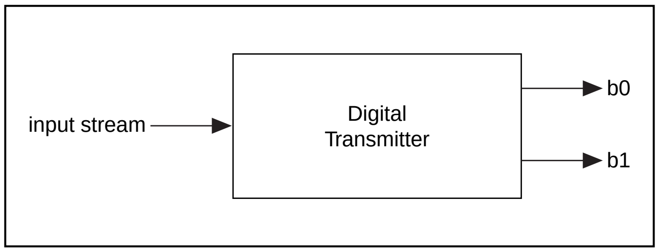
$$STFT[i, k] = \left| \sum_{m=-\frac{L}{2}}^{\frac{L}{2}-1} s[i-m]g[m]W_L^{-mk} \right|$$

where $g[m]$ is an analysis signal.



The STFT is also called a sliding-window fast Fourier transform (FFT) because a window function breaks the signal into several time slices. The FFT computes the frequency spectrum for each slice of windowed data, and then you take the square magnitude of each FFT. The result of this operation for each time slice is associated with the time index in the middle of that particular slice of windowed data. For example, the result of the operation for time-slice1 is associated with the time index t4, and that for time-slice3 is associated with the index t18. To establish a complete three-dimensional spectrogram, you can slide the window to the right one or more points at a time and compute a new spectrogram.

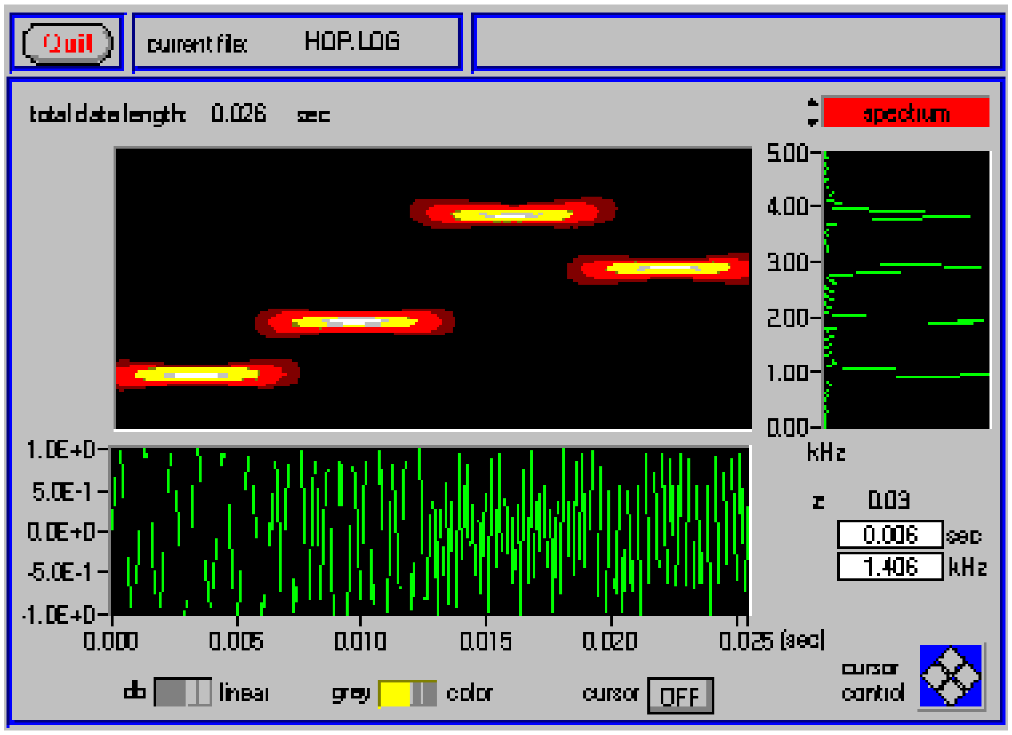
Consider one classical example. A signal known as the frequency hopper signal is widely used in digital communication:



Assume that two bits are being transmitted simultaneously over two transmission lines. Let the frequency corresponding to the combination

00 be f1, the frequency corresponding to 01 be f2, to 10 be f3, and to 11 be f3. Then, depending on which combination is being transmitted, the signal hops between these four frequencies, hence the name frequency hopper.

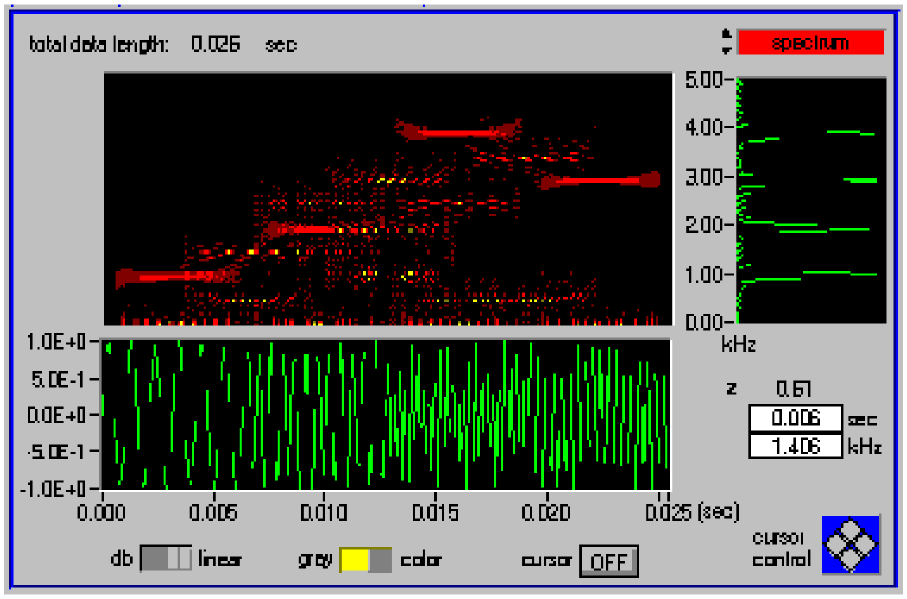
The figure below shows the time waveform of a frequency hopper signal, the corresponding spectrum as calculated by FFT, and the STFT distribution of this signal. From the time waveform, you cannot see any frequency information. From the spectrum, you can see that there are four distinct frequency components but cannot tell when those components occurred in time. The STFT representation provides both the time and frequency information in the same plot. From this spectrogram, not only can you distinctly see the four frequency levels, but you can also see when these frequencies changed in time.



Wigner-Ville Distribution

There are different ways to obtain bilinear[1] joint time-frequency representations. The first is the Wigner-Ville distribution. This technique is very simple and it better characterizes the signal's time-dependent spectra than the STFT spectrogram. Also, this distribution possesses many properties useful for signal analysis.

1. Called bilinear because the signal *x(t)* appears twice in the equations for each representation. For more information, refer to *Joint Time-Frequency Analysis: Methods and Applications* by Shie Qian and Dapang Chen, Prentice Hall, NJ, 1996.



The figure above shows the time waveform of a frequency hopper signal, the corresponding spectrum as calculated by FFT, and the Wigner-Ville distribution of this signal. From the time waveform, you cannot see the frequency information. From the spectrum, you can see four distinct frequency components but cannot tell when those components occurred in time. The Wigner-Ville distribution provides both the time and frequency information in the same plot. However, as you can see from the plot, the main deficiency of this distribution is the cross-term interference. This cross-term interference occurs at frequencies midway between the two main frequencies. If there are four main frequencies (as in the example above), there are six terms due to cross interference. However, a good characteristic of these cross-term interferences is that they highly oscillate between positive and negative values. These positive and negative parts cancel out each other on averaging. To do so, you can apply a low-pass filter to retain the low frequency components and remove the high frequency parts. Because the discarded high-frequency parts have small averages, the lowpass filtered Wigner-Ville distribution presumably preserves the useful properties with reduced cross-term interference.

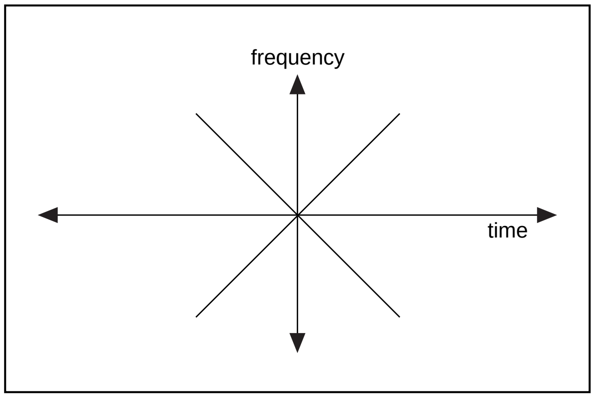
There are two types of lowpass filters, namely the linear type and the nonlinear type. There are different types of linear filters. It is interesting to note that all these linear representations can be written in a general form that was introduced by L. Cohen[1].

1. Interested readers should refer to "Generalized Phase-Space Distribution Functions," L. Cohen, in the *Journal of Mathematical Physics*, vol. 7, pp. 781-806, 1966.

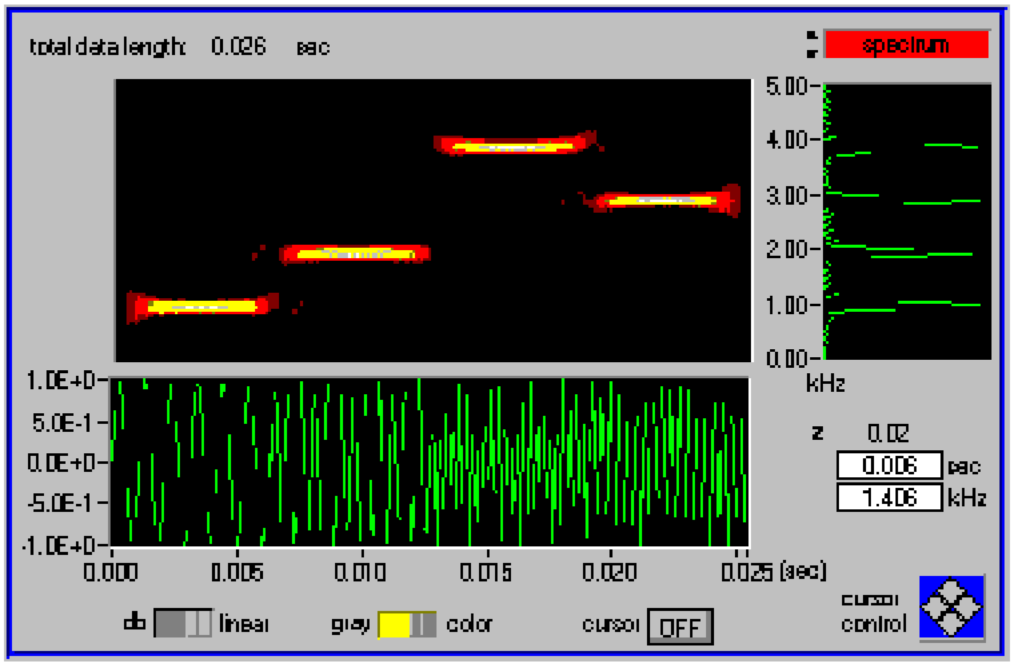
Cohen’s Class

Linear filters fall into a general category known as Cohen’s class of filters. With no filtering, the Cohen’s class distribution is the same as the Wigner-Ville Distribution. In the JTFA Toolkit, two of the most commonly used linear filters are implemented below. They are the cone-shaped distribution and the Choi-Williams distribution.

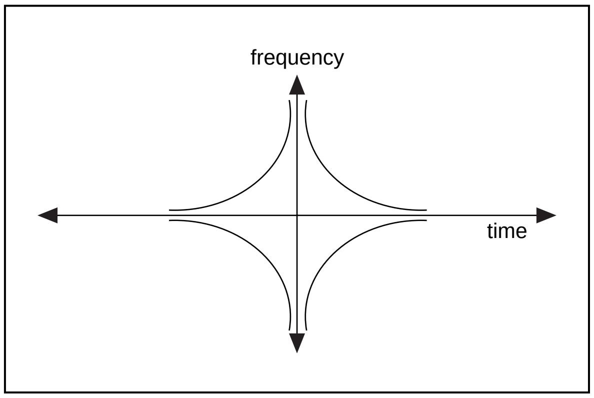
Cone-Shaped Distribution



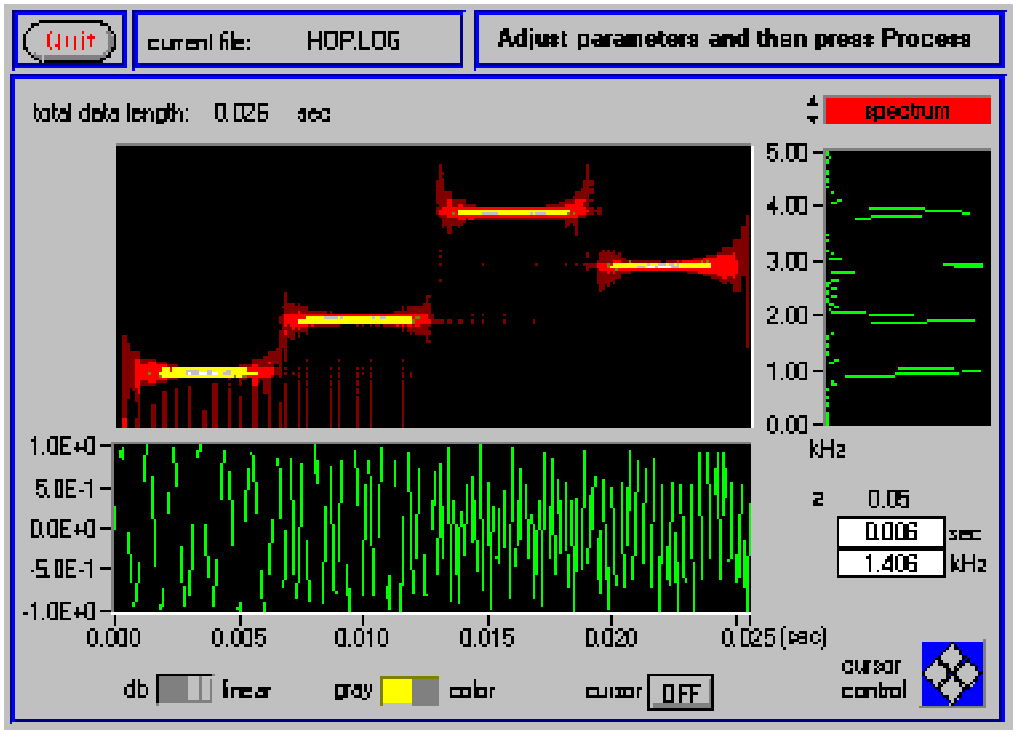
When the lowpass filter has a cone shape as shown above, the Cohen’s class distribution is known as the cone-shaped distribution. The figure below shows the cone-shaped distribution of the frequency hopper signal.



Choi-Williams Distribution



When the lowpass filter has an exponential shape, as shown above, the Cohen's class distribution is known as the Choi-Williams distribution (CWD). The figure below shows the Choi-Williams distribution of the frequency hopper signal.



current file: HOP.LOG

Adjust parameters and then press Process

total data length: 0.026 sec

α 0.05

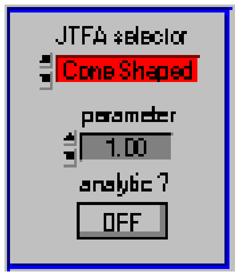
0.006 sec

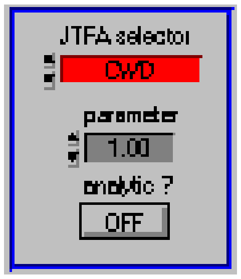
1.406 kHz

db linear

gray color

cursor OFF



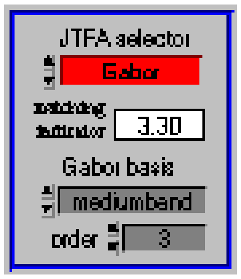


Both these linear filters give reduced cross-term interference at the expense of poor resolution. When you select one of these two algorithms using the JTFA selector control on the toolkit front panel (discussed in detail in the next section), you are asked to set the value of a parameter. This parameter allows you to balance the cross-term interference and resolution. The bigger this parameter is, the less the interference; however , the lower interference is at the cost of smeared resolution. Start with a value of 0.5 for the parameter. The desired value for this parameter is application dependent; however, it is always desirable to choose this value so that less interference occurs.

Gabor Spectrogram

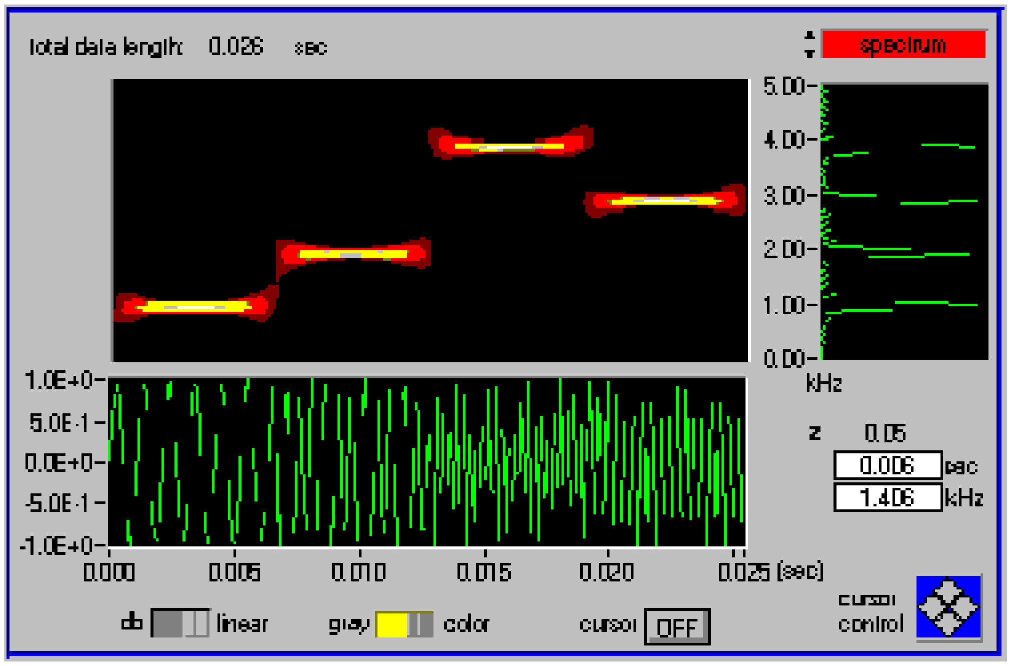
This section discusses the nonlinear type of low-pass filters described by the time-frequency distribution series, also known as the Gabor Spectrogram[1]. The advantage of the nonlinear type of filters is that it gives both good resolution and small cross-term interference. The Gabor spectrogram decomposes the Wigner-Ville distribution as DC plus a group of oscillated time-frequency functions.

1. The Gabor spectrogram was invented by researchers at National Instruments and is a National Instruments patent. It has won several technological awards. For more information on how to construct such a filter, refer to S.Qian and D.Chen, “Discrete Gabor Transform,” *IEEE Transactions on Signal Processing*, vol. 41, no. 7, July 1993, pp. 2429-2439



JTFA selector
Gabor
matching indicator 3.30
Gabor basis
mediumband
order 3

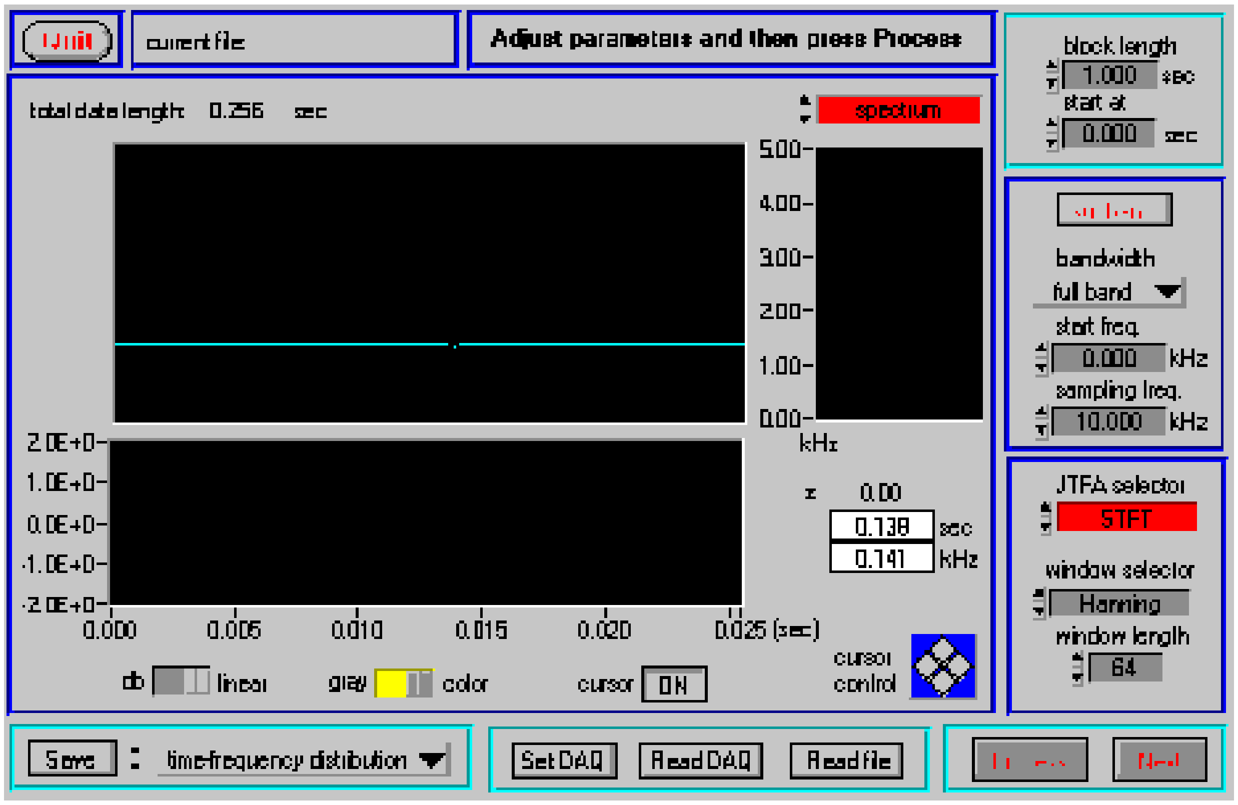
When you select the Gabor spectrogram using the JTFA selector on the toolkit front panel, you are asked to choose the order of the filter. The lower order Gabor spectrogram has less cross-term interference but lower resolution. The higher order Gabor spectrogram has better resolution but more cross-term interference. For order = 0, the Gabor spectrogram is similar to the STFT when using Gaussian window function. As the order gets larger, the Gabor spectrogram converges to the Wigner-Ville distribution. However, it is computationally more expensive. The best choice is usually order three to four. In this case, the Gabor spectrogram not only has better resolution than the STFT, but also possesses much less cross-term interference than the Choi-Williams and Wigner-Ville distributions. You can choose the Gabor basis to be wideband, mediumband or narrowband. The matching indicator tells you how closely the basis function approximates your input signal. The larger the value, the closer the match. The figure on the following page shows the third-order Gabor Spectrogram of the Frequency Hopper signal.



To summarize, you have examined different algorithms commonly used for obtaining a joint time-frequency representation. There has been minimal mathematical description of these algorithms. However, if you have understood the discussion above, you are ready to use the JTFA toolkit for different practical applications, as you will see in the next section.

C. The JTFA Toolkit and Its Applications

With the JTFA Toolkit, you can use any of the aforementioned algorithms to analyze stored data files and view the resulting spectrogram on an intensity plot. It is a stand-alone application that you can use in real-world applications such as radar and economic data analysis, where you need to analyze a signal in the time and frequency domain simultaneously. The list of such applications is endless. This section discusses simple applications. These applications may seem rather specific, but the ideas and methodologies behind them are of general interest to all JTFA users. The figure below shows the front panel of the JTFA analyzer.



Acquiring Data

When you start the JTFA application, it prompts you to adjust the parameters and then press **Process**. You first read in the signal to be analyzed. This signal can be stored in an existing file, which must be either a datalog format file (an internal file format used by LabVIEW) or an ASCII text file. If your file is text, JTFA converts it to datalog format when you click on the **Read file** button. You must convert the file to datalog, but you can save the converted file under a new name to preserve your original file. After conversion, the JTFA analyzer opens your file as a datalog file. On the front panel, you see two more buttons labeled Set DAQ and Read DAQ. If your computer is equipped with a National Instruments DAQ board, click on the **Read DAQ** button; the JTFA analyzer starts collecting data from the DAQ board and stores it a specified data file. When you first click on the **Read DAQ** button, it prompts you to set the DAQ parameters. The application then uses these parameters until you exit. You can use the **Set DAQ** button to change the DAQ parameters. After you select the file to analyze, the toolkit reads a block of data or frame of samples from the file.

Analyzing Data

You can select one of the different algorithms discussed in the previous section to analyze your data using the **JTFA selector** control. Each control has a different set of parameters. The default is set to the short-time Fourier transform (STFT) algorithm. In this case, the parameters to be selected are the window selector control and the window length control. The other five choices are the Gabor Spectrogram, the Choi-Williams distribution, the cone-shaped distribution, the Wigner-Ville distribution, and the Adaptive Spectrogram[1]. The application then analyzes the samples and displays the resultant spectrogram using an intensity plot. It also displays the traditional spectrum to the right and the time-waveform below the two-dimensional spectrogram.

The parameters **start at** and **block length** control the time range of the analyzed signal. While the former determines the start time, the latter determines the length of one frame of the analyzed signal. If the length of the signal is less than **start at + block length**, the JTFA analyzer processes whatever is available. Because of memory limitations, if the length of the signal is too long, JTFA displays the longest time duration allowed in a status indicator dialog box and stops processing.

The parameters **start freq.** and **bandwidth** control the frequency range of the analyzed signal. **start freq** determines the lower boundary of the frequency. The selection of the bandwidth is limited to $2^{-k} * Nyquist$ frequency, $0 \le k \le 5$. If **start freq. + bandwidth** is greater than the Nyquist frequency, the JTFA ignores the setting of **start freq.** and automatically sets the lower boundary of the frequency to zero (default value).

By clicking on the subband button, you can select the preemphasis control. You can use the preemphasis filter to reduce the influence of the DC component and enhance the high-frequency component. The degree of preemphasis is controlled by the preemphasis parameter. When this value is equal to zero (default), there is no preemphasis. When this value is equal to one, JTFA completely removes the DC component, and the

1. For a detailed discussion on the Adaptive Spectrogram, refer to *Joint Time-Frequency Analysis: Methods and Applications*, by Shie Qian and Dapang Chen, Prentice Hall, NJ, 1996.

frequency components in the vicinity of the Nyquist frequency are approximately doubled.

On a final note, if you change the **JTFA selector**, **subband**, or any of the parameters, you must recompute the spectrogram to see the effect of the changes. To recompute the spectrogram, click on the **Process** button. The JTFA computes the spectrogram using the values in the controls at the time you click on the **Process** button.

Post Analysis of Data

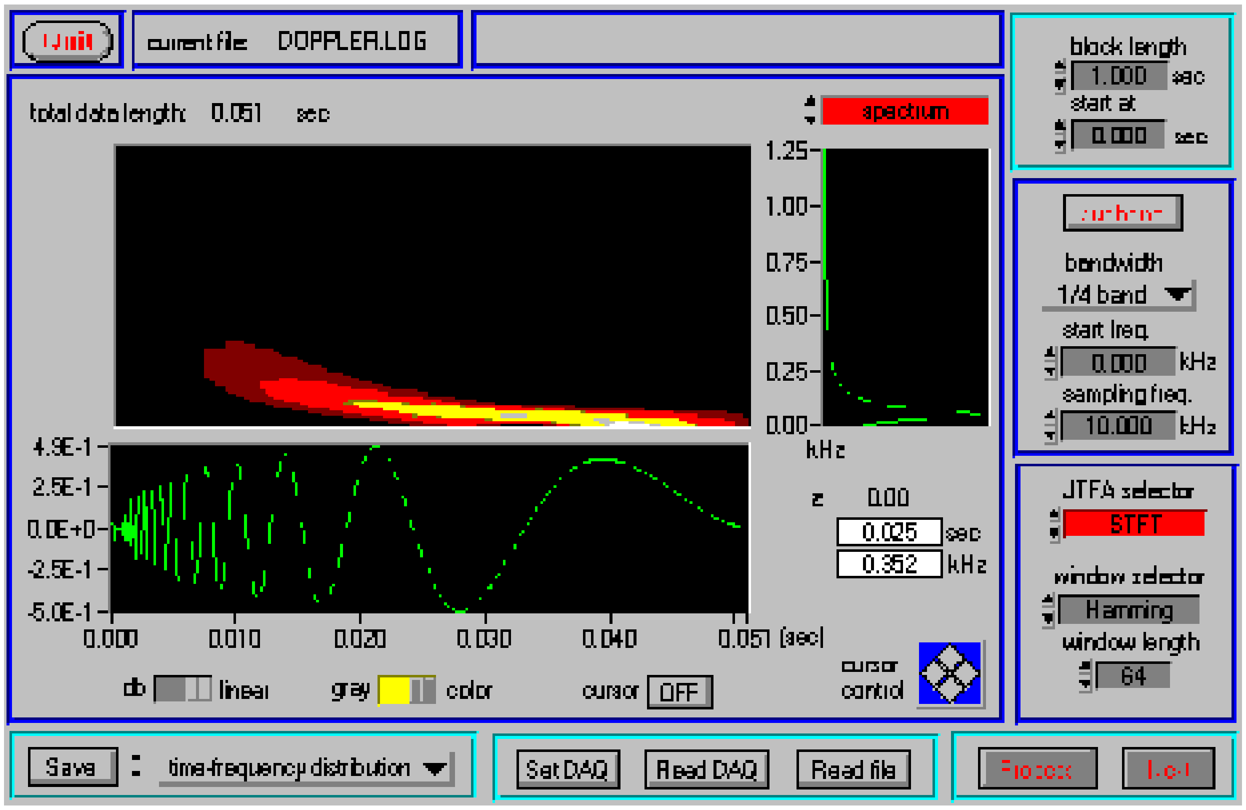
After you have analyzed the data using the JTFA toolkit, you might want to save the data plots to a file for later use. You can save the time waveform, spectrum, and spectrogram as spreadsheet files. Select the plots that you want to save (using the selector to the right of the Save button) and then click on the **Save** button. You can then process the next frame of samples by clicking on the **Next** button. After you finish analyzing one file, you may close the current file and open a new one. To open a new file, click on the **Read** file button and indicate the file that you want to open. The program closes the original file automatically.

The following exercises will help you further understand how to use the JTFA Toolkit.

Exercise 13-1

Objective: To use the JTFA Toolkit to analyze a doppler signal

In this exercise, you will familiarize yourself with different controls in the JTFA Toolkit. You will analyze a doppler signal and see the advantages of a JTFA spectrogram over a normal frequency spectrum.



current file DOPPLER.LOG

total data length: 0.051 sec

block length 1.000 sec

start at 0.000 sec

bandwidth 1/4 band

start freq. 0.000 kHz

sampling freq. 10.000 kHz

JTFA selector STFT

window selector Hamming

window length 64

z 0.00

0.025 sec

0.352 kHz

cursor OFF

1. Open the front panel of the JTFA Toolkit as shown above.
2. You will analyze data in the file `c:\jtfa\data\doppler.log`. Click on the **Read file** button and choose this data file. The name of the data file can now be seen in the current file box.
3. Adjust some of the parameters. Change the **bandwidth** to 1/4 band. You will experiment with all the different algorithms that can be selected using the **JTFA selector**. First, select the STFT algorithm. You will use the Hamming window and window length = 64. Use the default values for all the other parameters.
4. After setting all parameters, click on the **Process** button.

Note: ***Remember that the application will not start computing until you click on the Process button.***

5. After the computation is complete, three waveforms will appear on the JTFA window. You will now analyze these waveforms. The time

waveform of the doppler signal can be seen in the bottom left part of the figure. The standard Fourier spectrum can be seen on the top right part. From this plot, you can see the range of frequencies present in the spectrum. But the plot does not tell how these frequencies evolved as a function of time. The larger plot in the upper left corner is the time-dependent spectrum, a function of both time and frequency. From it, not only can you see how the frequency changed with time, but you also can see the intensity of the frequencies as shown by the relative brightness levels of the plot.

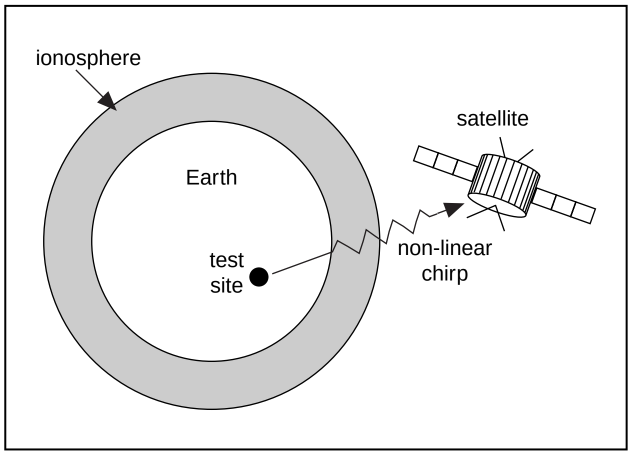
6. You can now select different algorithms in the **JTFA selector** control and see how the time-frequency spectrogram looks in each case.

End of Exercise 13-1

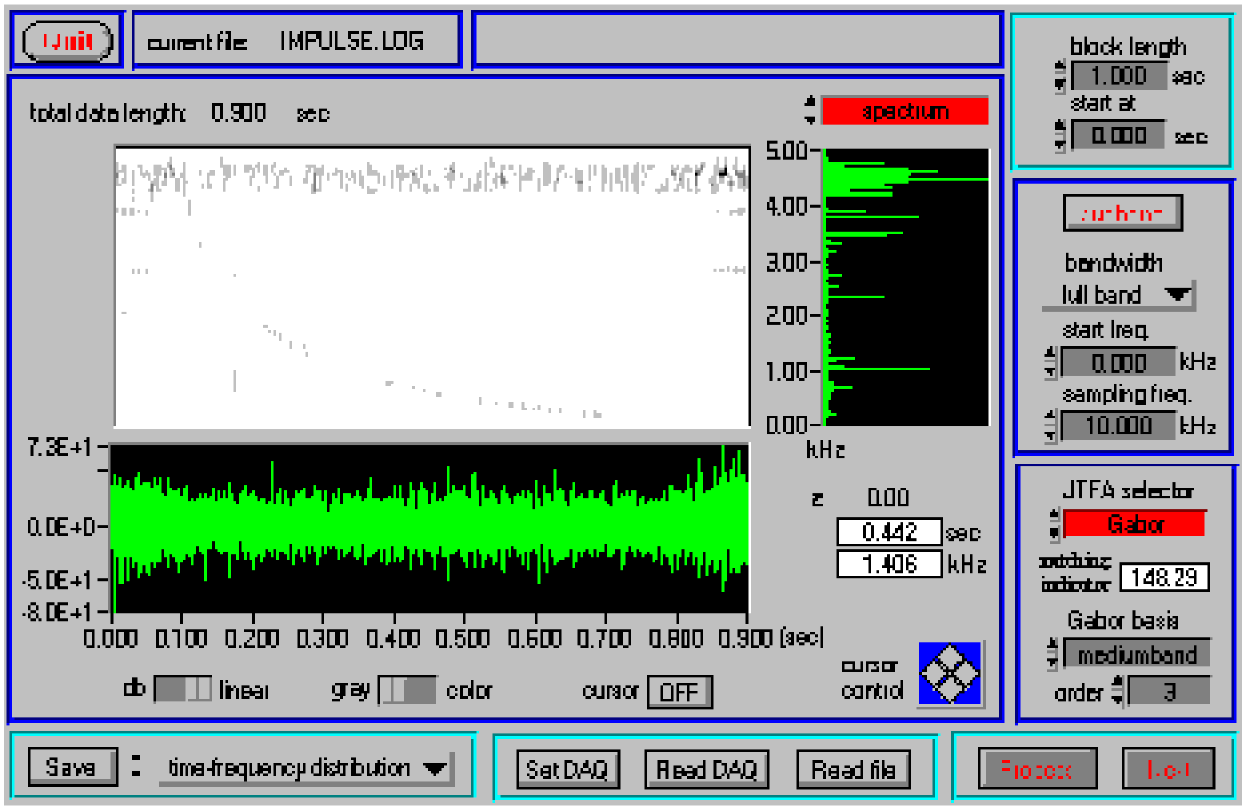
Exercise 13-2

Objective: To study the application of detecting impulse signals in low-orbit satellites using JTFA.

In this exercise, you will examine a very interesting application for joint time-frequency analysis. This application involves the detection of impulse signals by low-orbit satellites.

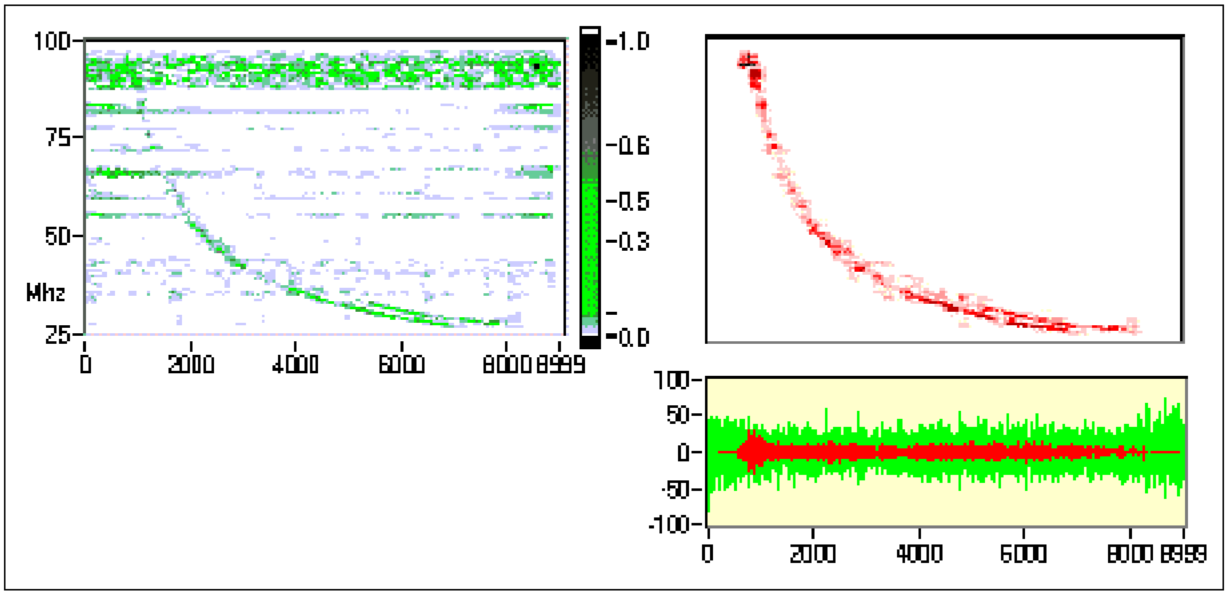


Consider an impulse signal generated at some location on Earth. Such an impulse may be caused by nuclear weapons testing, and hence the detection and estimation of this signal is an important national security issue. But the detection of this signal is not very easy. As this signal passes through the dispersive media surrounding the Earth (for example, the ionosphere), the signal becomes a nonlinear chirp signal. Furthermore, it is severely corrupted by random noise from the ionosphere, and hence the detection of this signal via standard Fourier transform techniques is not possible. However, as you will see in this example, JTFA helps to detect this signal properly.



1. Open the front panel of the JTFA Toolkit as shown above, if it is not already open from the previous exercise.
2. You will analyze data in the file `c:\jtfa\data\impulse.log`. Click on the **Read file** button and choose this data file. The name of the data file can now be seen in the **current file** box.
3. Adjust some of the parameters. Change the panel color to gray using the front panel switch. You will experiment with all the available algorithms using the **JTFA selector**. First, select the *Gabor spectrogram*. Use the default values for all the other parameters.
4. After setting all parameters, click on the **Process** button. Remember that the application will not start computing until you have done this.
5. After the computation is complete, three waveforms will appear on the JTFA front panel. The time waveform of the ionized impulse signal can be seen in the bottom left part of the figure. As explained earlier, after passing through dispersive media, such as the ionosphere, the impulse signal becomes a nonlinear chirp signal with added random noise. Because of the low signal-to-noise ratio, you can hardly see this chirp signal in the time waveform. It is completely hidden by the noise signal. The standard Fourier spectrum can be seen on the top right part. From this plot, you can see the range of frequencies that are present in the spectrum. The

larger plot in the upper left corner is the time-dependent spectrum, a function of both time and frequency. From this plot, you can immediately identify the presence of the chirp-type signal arching across the joint time-frequency domain. Using such a representation, you can do extensive postprocessing using the Analysis Library VIs and mask the desired signal, as shown in figure below. You can then apply the inverse transformation to recover the original time waveform. The figure below compares the noisy signal and the reconstructed signal.



6. You can now select different algorithms and see how the time-frequency spectrogram looks in each case.
7. Exit the JTFA analyzer by pressing the **Quit** button.

End of Exercise 13-2

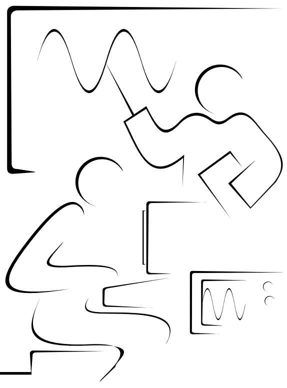
Summary:

- The need to analyze a signal in time and frequency domains simultaneously has led to the popularity of joint time-frequency analysis.
- The JTFA toolkit is an excellent tool to learn this technique and use it in many real-world applications.
- Short time-frequency transform, Gabor spectrogram, Wigner-Ville distribution, Choi-Williams distribution, Cone-Shaped distribution, and adaptive spectrogram are some of the algorithms used for joint time-frequency analysis. The choice of algorithm depends on your application.
- Some of the interesting applications for JTFA are in radar, medical imaging, and economic data analysis. In Exercise 13-2, you and scientists at the Los Alamos National Laboratory made substantial progress in detecting a radio frequency (RF) nonlinear chirp-type signal in a noisy environment.

Notes

Notes

Lesson 14
Wavelet and Filter Banks Designer



Introduction

Wavelets are being applied in a diverse range of applications for the analysis of nonstationary signals. These applications include removing noise from signals, detecting abrupt discontinuities, and compressing large amounts of data. The design of wavelets is closely related to the design of filters. The Wavelet and Filter Banks Designer provides you with the flexibility to interactively design the filters, and hence wavelets, for your specific application.

You Will Learn:

A. About signal representation through transforms: the Fourier transform and the short-time Fourier transform (STFT).

B. About the wavelet transform.

C. About the applications where wavelet analysis is useful.

D. About the relationship between wavelets and filter banks.

E. How to use the Wavelet and Filter Banks Designer (WFBD) Toolkit on both a 1D signal (an electrocardiogram) and a 2D signal (an image of a fingerprint).

A. Signal Representation

Before learning about the wavelet transform and its applications, you first need to understand some basic concepts from linear algebra. In particular, you need to be familiar with inner products and basis vectors.

Inner Product

The inner product of two vectors is a mathematical operation that determines the similarity of these vectors with respect to each other. For example, given the vectors $x = (x_1, x_2)$, and $y = (y_1, y_2)$, their inner product is

$$<x,y> = x_1 * y_1 + x_2 * y_2$$

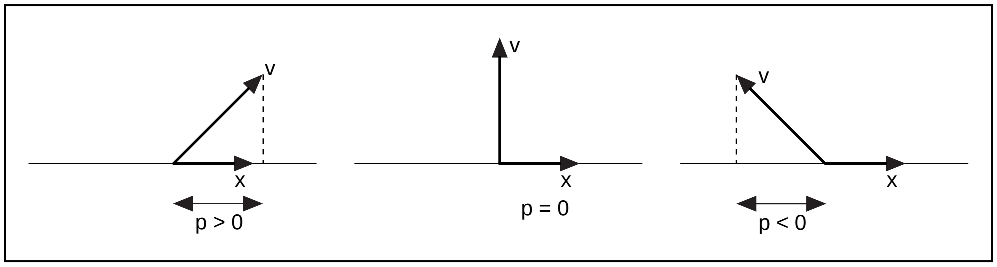
The inner product determines the similarity by calculating the *projection* (or *component*) of one vector in the direction of the other. For example, if you have two vectors, say *v* and *x*, the projection (denoted by *p*) of *v* on *x* is given by

$$p = \frac{<v,x>}{\|x\|}$$

where $||x||$ is the magnitude of the vector *x*. If *x* is a unit vector, $||x|| = 1$, and the projection of *v* on *x* is simply

$$p = <v,x>$$

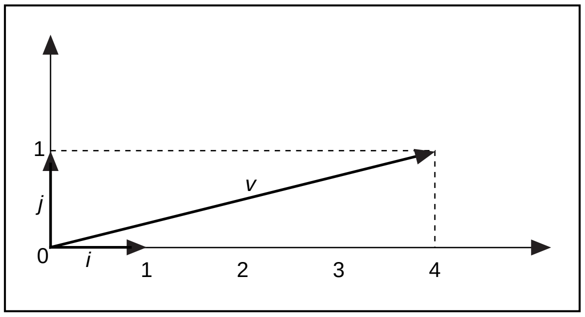
The magnitude of the projection $|p|$ is the length of the orthogonal projection of *v* on a straight line having the same direction as the unit vector *x*. The projection *p* itself could be either positive, negative, or zero. This is illustrated in the figure below:



The larger the value of $|p|$, the more similar the vectors are to one another, and vice versa. When the inner product is equal to zero, the two vectors are perpendicular to each other. Another way to say this is that the two vectors are *orthogonal* when their inner product is equal to zero.

Basis Vectors

Consider a vector in 2D. Any such vector can be written as the weighted sum of two other unit vectors, $i = (1,0)$ and $j = (0,1)$. For example, the vector *v* = (4,1) can be written as *v* = 4**i* + 1**j*. This is shown graphically in the figure below:



The vectors *i* and *j* are known as the *basis vectors* in 2D. Every other vector in 2D can be written as a combination of these two basis vectors. An important property of basis vectors is that they are perpendicular (orthogonal) to each other. That corresponds to their inner product being equal to zero. Also, their magnitudes are equal to one. For the basis vectors *i* and *j* in 2D,

$<i, j> = < (1,0), (0,1) > = 1*0 + 0*1 = 0$

In other words, this means that they are linearly independent (see Lesson 8, *Linear Algebra*) and you cannot express one of them in terms of the other.

For the vector *v* = (4,1), you can take its inner product with respect to the unit vectors *i* and *j*:

$<v,i> = 4*1 + 1*0 = 4$ and

$<v,j> = 4*0 + 1*1 = 1$

This gives the projection of *v* on the unit vectors. Because <*v*,*i*> is greater than <*v*,*j*>, you can say that the vector *v* is more similar to *i* than to *j*. This is readily apparent from the figure above, where *v* is more horizontal than vertical. (Note that because *i* and *j* are unit vectors, their magnitude is equal to 1.0. Thus, in the above equations, we did not specifically normalize by the magnitude.)

If you know the values of the inner product of *v* with the basis vectors *i* and *j*, you can obtain *v* from the basis vectors by multiplying them with the values of the corresponding inner products and adding up the results. For the example given,

$4*i + 1*j = 4*(1,0) + 1*(0,1) = (4,0) + (0,1) = (4,1) = v.$

The important points to remember from the above discussion are:

- Basis vectors are chosen to be orthogonal to each other.

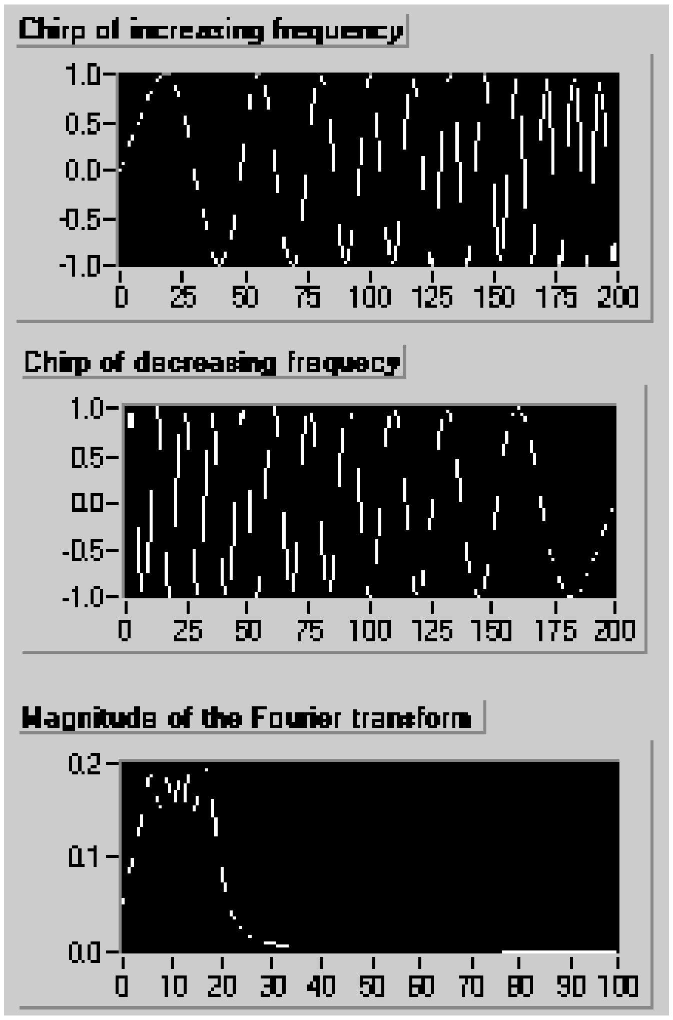
- Any vector *v* can be broken down as the sum of weighted basis vectors. This is achieved by taking the inner product of the vector with each of the basis vectors.
- The larger the value of the inner product, the greater the similarity of the given vector with respect to that particular basis vector.
- You can reconstruct the vector *v* by knowing the values of its inner product with each of the basis vectors. The reconstruction is achieved by multiplying the inner product with the corresponding basis vector and adding the results.

Fourier Transform

Just as you can break up any vector in 2D in terms of a set of basis vectors *i* and *j*, you can also break up a time signal *s(t)* in terms of a set of basis functions. A well known popular example of basis functions in signal analysis are the sines and cosines of the Fourier transform. These sines and cosines are *harmonically* related to each other, which means that their frequencies are all integer multiples of a fundamental frequency. In addition, the sines and cosines are *orthogonal* (if you take their inner product, the result is zero).

When you take the Fourier transform of a time signal *s(t)*, you are actually taking the inner product of the signal with each of the basis functions. So you actually are seeing the similarity of the signal to each of the sines and cosines of different frequencies. If you get a large value of the inner product with a basis function at a particular frequency, it means that quite a bit of that frequency is present in the signal. If the inner product happens to be zero, it means that frequency is not present in *s(t)*.

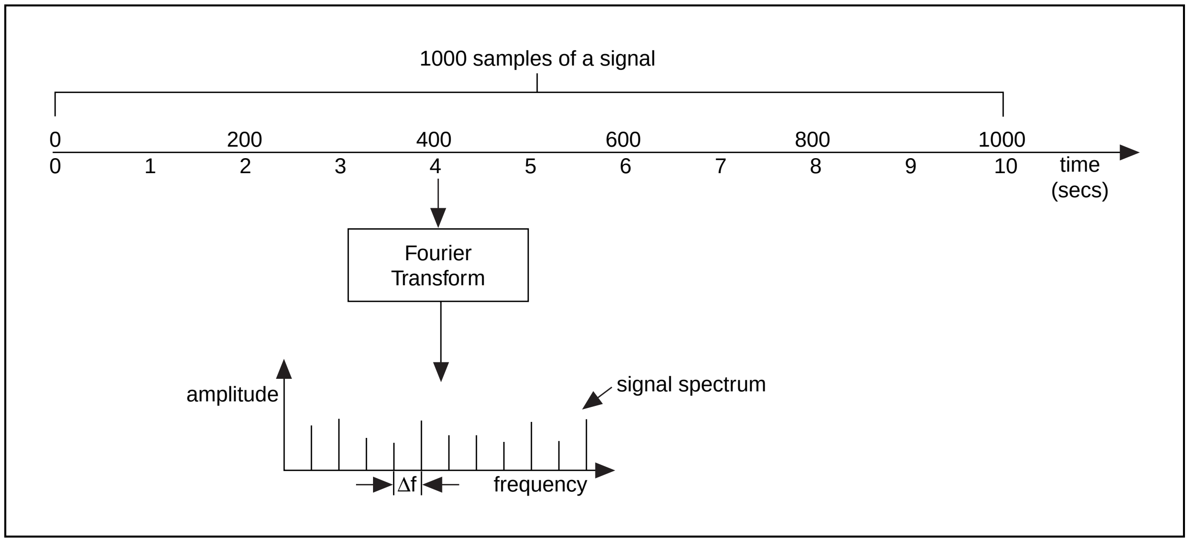
Although the Fourier transform is a useful tool in obtaining information about the frequency content of a signal, its disadvantage is that it is unable to tell us when in time a particular frequency occurs. For example, consider the two chirp signals shown in the top two plots of the figure on the next page. The frequency of the top signal is increasing, whereas that of the bottom signal is decreasing. However, both have exactly the same magnitude of the Fourier transform, as shown in the lowermost figure.



Looking at the Fourier transform of a signal, you cannot tell how its frequency changes with time. All that you observe is the frequency content of the sampled signal in the given time interval. The drawback of this form of traditional Fourier transform analysis is that it is only useful for the analysis of *stationary* signals (that is, signals whose frequency content does not vary with time).

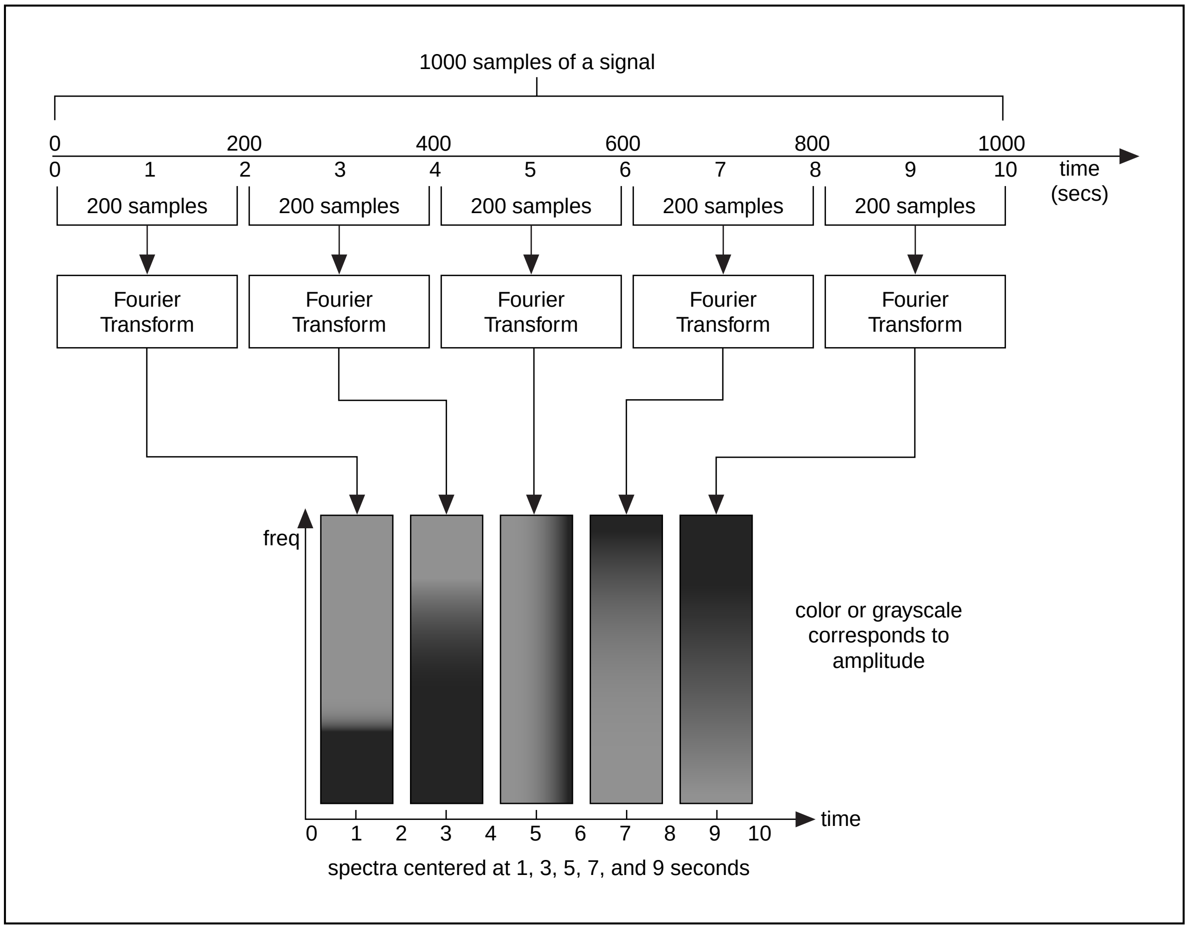
Windowed Fourier Transform or Short-Time Fourier Transform (STFT)

One method of obtaining time information (in addition to the frequency information) with the Fourier transform is to break up the signal into small time intervals before taking the Fourier transform. For example, suppose you have 1000 samples of a signal collected over a period of 10 seconds (so the sampling frequency is 100 Hz). You could take the Fourier transform of the entire 1000 samples, as shown below:



The result is a two-dimensional spectrum where the *x*-axis is the frequency and the *y*-axis is the amplitude. The frequency resolution in this case is $\Delta f = f_s/1000$, where fs is the sampling frequency. For the same sampling rate, to increase the frequency resolution (that is, reduce Δf), you need to increase the number of samples to more than 1000.

However, you can also divide the signal into five equal time intervals of two seconds each, and analyze the frequency content of each interval separately. In each of these time intervals, you will have 200 samples of the signal. You can then take the Fourier transform of successive 2-second time intervals, as shown below.



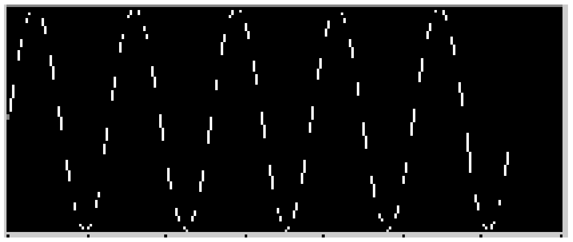
The result is a 3D spectrum, where the *x*-axis represents time, the *y*-axis represents frequency, and the *z*-axis (out of the page) represents the amplitude of the frequencies during the time for which the transform was taken. The *z*-axis can be in color or gray scale to represent the amplitude values. This method of dividing the signal into smaller time intervals and then taking the Fourier transform of successive intervals is known as the short-time Fourier transform (STFT).

Note that each spectrum is obtained over a time interval of 2 seconds. So, you know that any frequency component indicated in the spectrum occurred somewhere within those 2 seconds. Thus this method provides you with better time resolution than the first case, where you had an interval of 10 seconds, somewhere during which a frequency shown in the spectrum could have occurred. However, note that the frequency resolution is now $\Delta f = f_s/200$, and is now five times larger than in the first case.

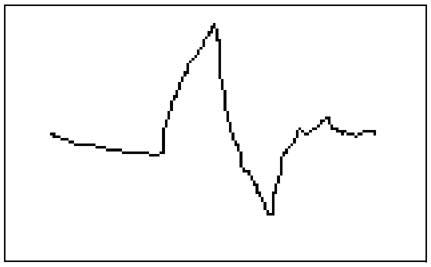
Good time resolution is necessary for the rapidly changing (high-frequency) parts of a signal, whereas good frequency resolution is necessary for the smoothly varying (low frequency) parts of a signal. To improve the frequency resolution, you can (for a given sampling frequency) increase the size of the time interval during which you obtain the samples, but then the time resolution is affected. If you improve the time resolution by decreasing the time interval, the frequency resolution is affected, because now you have fewer samples, so Δf is larger. Hence, there is a trade-off between the time resolution and the frequency resolution when using the STFT.

B. The Wavelet Transform

The wavelet transform is another form of analysis where you can measure the similarity of a signal to a set of basis functions. For the Fourier transform, these basis functions were limited to being sines and cosines of different frequencies. Also, they existed for all time. However, for the wavelet transform, there are many possibilities of different shapes of basis functions. In addition, the functions are time limited (they are nonzero only within a finite time interval) and are usually irregular, with an average value of zero. These basis functions used in the wavelet transform are called *wavelets*. An example of a basis function for the Fourier transform and the wavelet transform are as shown:

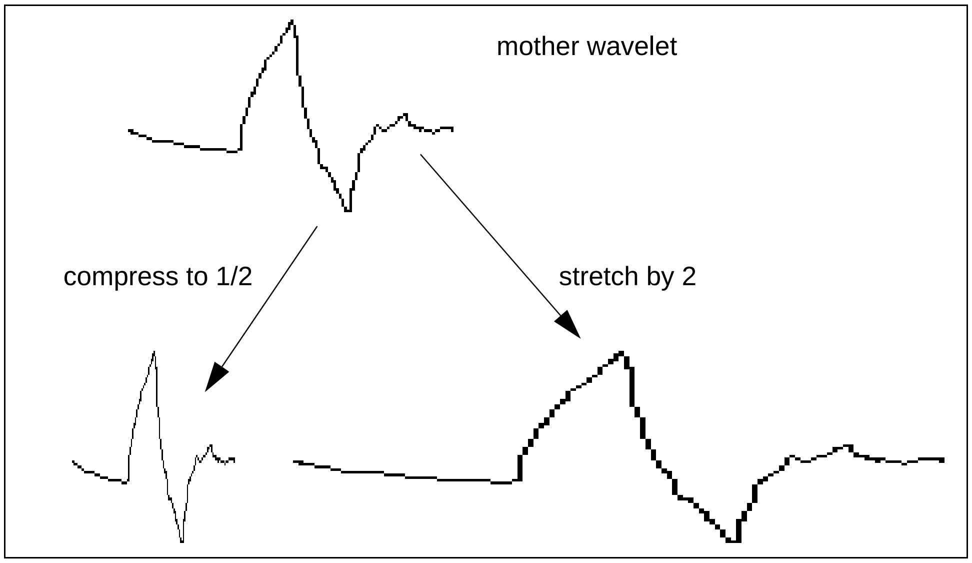


Basis function for Fourier transform

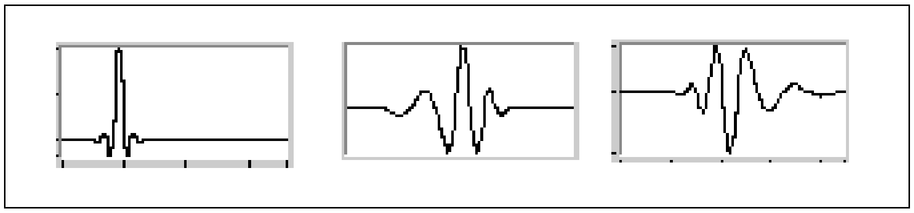


Basis function for wavelet transform

Using the wavelet basis functions, you can obtain both good time resolution and good frequency resolution. You first select (or design) a prototype wavelet known as a *mother wavelet*. This wavelet is then either “stretched” or “compressed” in time to obtain other wavelets for the basis. The wavelet obtained by “stretching” has a longer time duration than the mother wavelet and is good for extracting the low-frequency information of the signal. The wavelet obtained by “compressing” has a shorter time duration than the mother wavelet and is good for extracting the high-frequency information of the signal. Thus, it is useful for temporal analysis. The scale change (compressing or stretching) is done in powers of two. The figure below shows a mother wavelet and two other wavelets obtained from it:



An important step in wavelet analysis is the choice of wavelets for the basis functions. As discussed, you want to see the similarity of the signal under consideration to the basis functions. So depending on the signal (that is, application), a particular shape of a wavelet may be a better representation for the signal than another shape. Unlike the Fourier transform, where the shapes of the basis functions are restricted to being sines and cosines, in the wavelet transform you can have a large number of possibilities. However, some restrictions do apply (such as the basis functions being orthogonal to each other), but these are handled automatically by the Wavelet and Filter Banks Designer (WFBD) toolkit, and you, as the user, need not worry about them. The figure below shows examples of some other wavelets.



There are several wavelet packages available on the market today. Whereas other wavelet packages have a fixed selection of wavelets from which you can choose, the National instruments WFBD lets you design your own wavelets to best suit your particular application. By experimenting with the parameters on the Design Panel of the WFBD, you can design different wavelets. Unlike the Fourier transform, wavelets are specifically suited for the analysis of *nonstationary* signals (signals whose frequency contents vary with time).

The National Instruments WFBD is an interactive design package. It enables you to design wavelets for both 1D signals and 2D images. The signal or image can be loaded from a file or acquired by using DAQ or IMAQ hardware.

Comparison of the Wavelet Transform and Fourier Transform

There are several important differences between the wavelet and Fourier transforms. The following table summarizes these differences.

	Fourier Transform	**Wavelet Transform**
Basis functions	Sines and cosines	Wavelets
Assumption	Basis functions exist for all time	Basis functions are time limited
Type of signal	Stationary	Nonstationary
Analysis	Same resolution at all frequencies	Multiresolution

C. Applications of Wavelets

The wavelet transform is useful for the analysis of nonstationary signals, whose frequency contents vary with time. Examples of nonstationary signals are biomedical (for example, electrocardiogram), music, turbulence, seismic, sound, and vibration data that change slowly or abruptly. For signals like these, information about when the changes occur can often be very important.

You have seen that you can use the wavelet transform to have both good time and good frequency resolution simultaneously. Wavelets with a short time duration are good for extracting high-frequency information (such as discontinuities and abrupt changes) from a signal, whereas wavelets with a longer time duration are useful for low-frequency analysis. Thus, it is possible to see both the *forest* and the *trees*. Wavelets have been used in diverse fields covering a wide variety of applications. Some of the practical uses of wavelets are:

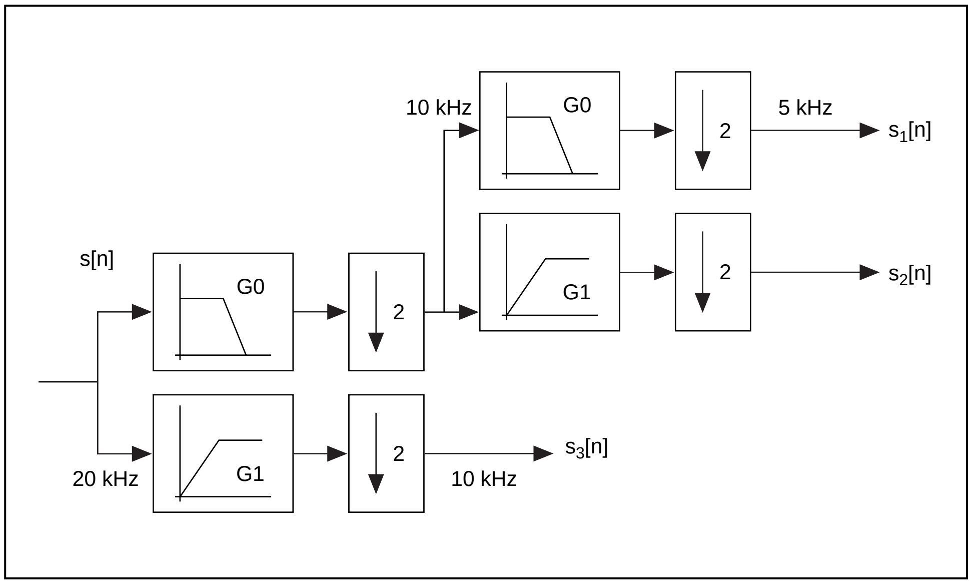
- Removing noise from a signal.
- Feature extraction for use in pattern recognition and classification.
- Detection of discontinuities.
- Data compression (for example, images) that can be used to speed image processing, implement faster modems, speed transfers on the Internet, video conferencing, satellite image transmission, and telecommunications.

Wavelets vs. Joint Time-Frequency Analysis

Both the WFBD and the JTFA toolkit are useful in analyzing nonstationary signals. But when would you choose one over the other? The answer lies in what you hope to achieve. The general rule is that if you are interested in applications that require reconstruction, you should use wavelet analysis. Examples of such applications are noise removal and data compression. In noise removal, you want to reconstruct the original signal from a signal containing both additive noise and the original signal. In data compression, you want to reconstruct the original signal from a compressed version of the signal. However, if you are more interested in obtaining insight into the frequency contents of a signal, or in understanding the physical nature of the process that generated the signal, the JTFA toolkit may be better for such cases.

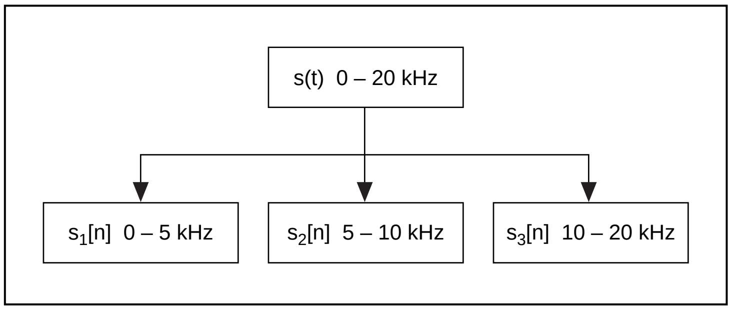
D. Wavelets and Filter Banks

It turns out that the design of wavelets is closely related to the design of filter banks. But what is a filter bank? It consists of a set of filters that breaks a signal into different frequency ranges. Consider two filters: a lowpass filter, G0, and a highpass filter, G1. You will use these filters to split a signal into different frequency bands, as shown in the figure below:

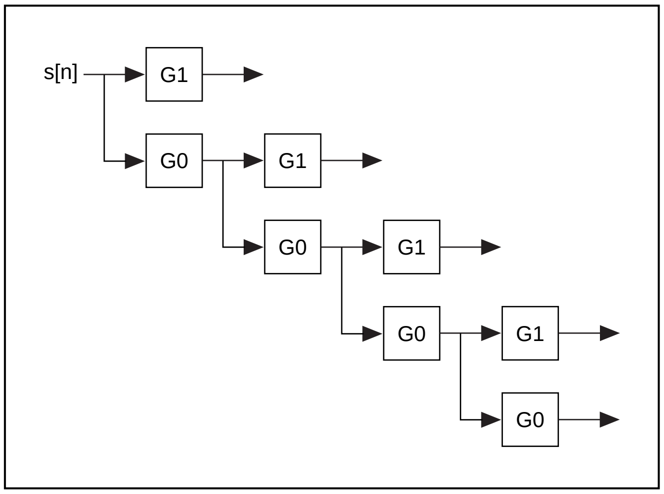


The input signal, $s[n]$, consists of frequencies in the range of 0 - 20 KHz. It is first lowpass filtered by G0 and highpass filtered by G1 around the frequency of 10 KHz. The output of the highpass filter is $s_3[n]$ and occupies a frequency range of 10 KHz - 20 KHz. The output of the lowpass filter is in the range 0 - 10 kHz and is once again filtered, this time around 5 KHz, by both a highpass filter and a lowpass filter, to give the signals $s_2[n]$ and $s_3[n]$. $S_2[n]$ occupies a frequency range of 5 KHz - 10 KHz, and $s_1[n]$ occupies a range of 0 - 5 KHz. In the figure, the downward pointing arrow with a 2 next to it shows the process of downsampling (decimation) by a factor of two. Because the frequency span of the filtered signal has been reduced by half, you can safely decimate and use only half the number of samples.

So, now the original signal $s[n]$ has been broken down into three signals, each occupying different frequency ranges shown below:



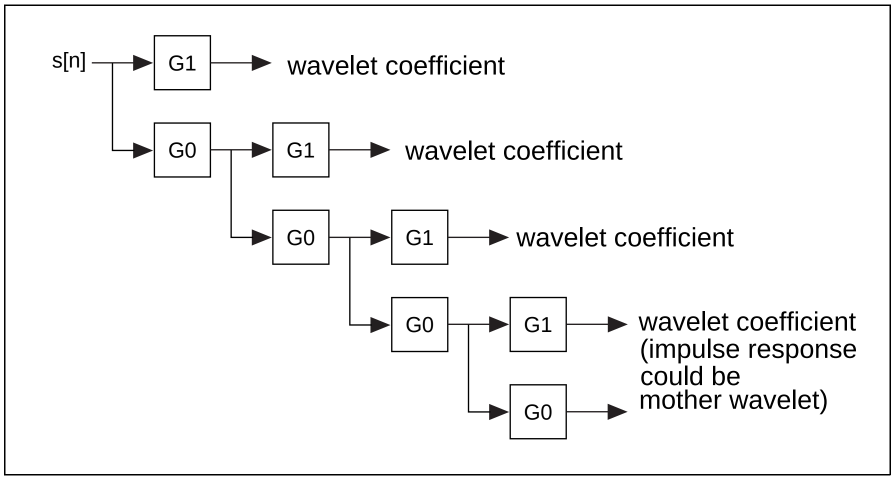
In practice, you do not need to stop after only two stages of filtering. You could continue to split the lowpass filtered signal into even more frequency bands, as shown below:



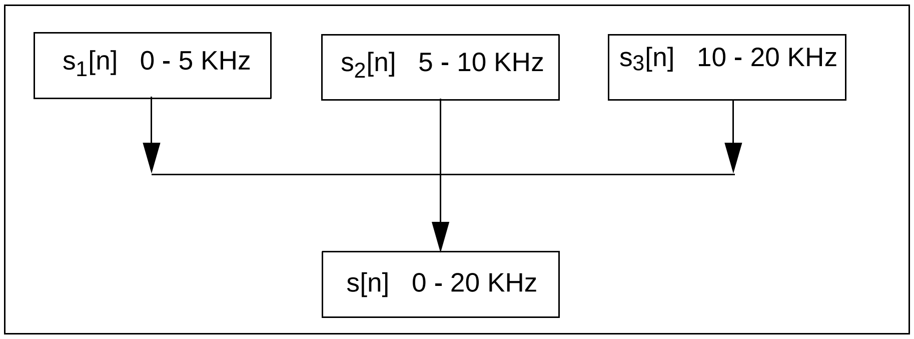
Note: ***Although the process of decimation has not been explicitly shown, it is assumed.***

The design of the filters is such that the outputs of each highpass filter are approximations of the wavelet transform. The wavelet obtained from the highpass filter corresponding to the lowest frequency band has the longest time duration and could be the *mother wavelet*. It is thus able to extract low-frequency information from the signal. As you go toward the higher frequencies, the impulse response (that is, wavelets) from the highpass filters are the compressed versions of the mother wavelet and have correspondingly shorter time durations. Thus, they are more suitable for extracting the high-frequency information (for example, discontinuities) from the signal. All these wavelets together form the set of basis functions to represent a signal.

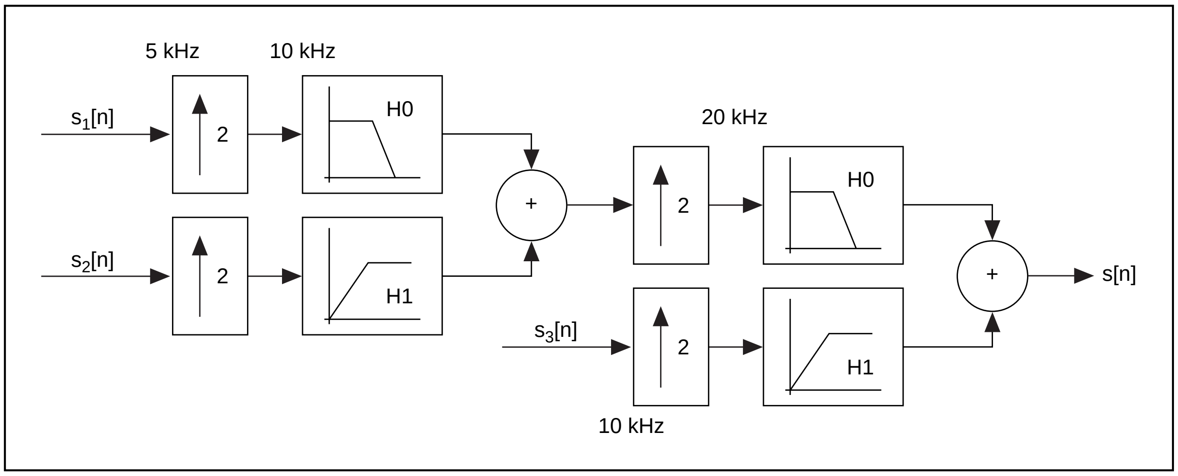
The process of highpass filtering is equivalent to taking the inner product of the signal with respect to the wavelet basis functions. The output of each highpass filter is the value of the inner product of the signal with one of the wavelets. This value is the wavelet coefficient and indicates the similarity of the signal with respect to the corresponding wavelet.



If the filters that make up the filter bank satisfy certain conditions, it is possible to reconstruct the original signal s[n] from $s_1[n]$, $s_2[n]$, and $s_3[n]$.



This can be achieved by using another set of lowpass filters, H0, and highpass filter, H1, and adding their outputs together, as shown below:



This process of filtering is now equivalent to reconstructing the original signal from the wavelet basis functions. (The upward pointing arrow with a 2 next to it shows the process of upsampling by a factor of two.)

A special characteristic about these filter banks as compared to the traditional filter banks is that the outputs of these filter banks can be used to reconstruct the original signal s(t). The filter banks are said to have *perfect reconstruction*. From the traditional filter banks, it is not always possible to reconstruct the original signal. For reconstruction, the filters being used need to satisfy certain conditions, and thus a special relationship exists between the filters G0, G1, H0, and H1. Because of this relationship, it becomes necessary only to design the lowpass filters, G0 and H0. Knowing G0 and H0, the highpass filters G1 and H1 are automatically designed by the software.

In the next section, you will see how designing wavelets with the help of the WFBD involves designing the lowpass filters G0 and H0.

E. Using the Wavelet and Filter Banks Designer

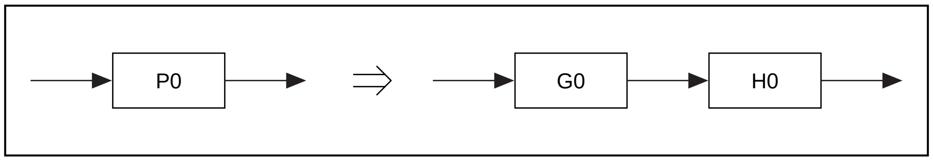
Using the WFBD package, there are three important steps involved in the design of wavelets. Because of the relationship between wavelets and filter banks, these steps basically boil down to selecting and designing the filters for the filter banks.

Selecting the Type of Filter Bank

The first step consists of selecting a particular type of filter bank. There are two types of filter banks that you can choose from, *orthogonal* and *biorthogonal*. The orthogonal filter bank is a special case of a biorthogonal filter bank. The main difference between the orthogonal and biorthogonal filter banks is that the biorthogonal filters can be linear phase, but the orthogonal filters cannot. So, if your application requires linear phase, then you should not use the orthogonal filter banks. (Linear phase is important for images, because the eye is very sensitive to changes in phase.)

Selecting the Type of Filter

The next step deals with designing the lowpass filters G0 and H0, and has to do with designing a filter P0 which is the cascade of the two filters G0 and H0. The filter P0 is illustrated as shown:



The final step involves splitting the filter P0 into G0 and H0. These three steps are outlined below:

Step 1: Select the type of filter bank. The available choices are:

- Orthogonal
- Biorthogonal

Step 2: Select the type of filter for P0. The available choices are:

- Maxflat
- Positive equiripple
- General equiripple

Any of the three types of filters mentioned in step 2 can be used with biorthogonal filter banks. However, with orthogonal filter banks, you can have only maxflat or positive equiripple filters, but not general equiripple.

There is no hard and fast rule, or even general guidelines, as to which type of filter you should choose for any specific application. You will need to experiment with the different choices to select a suitable one.

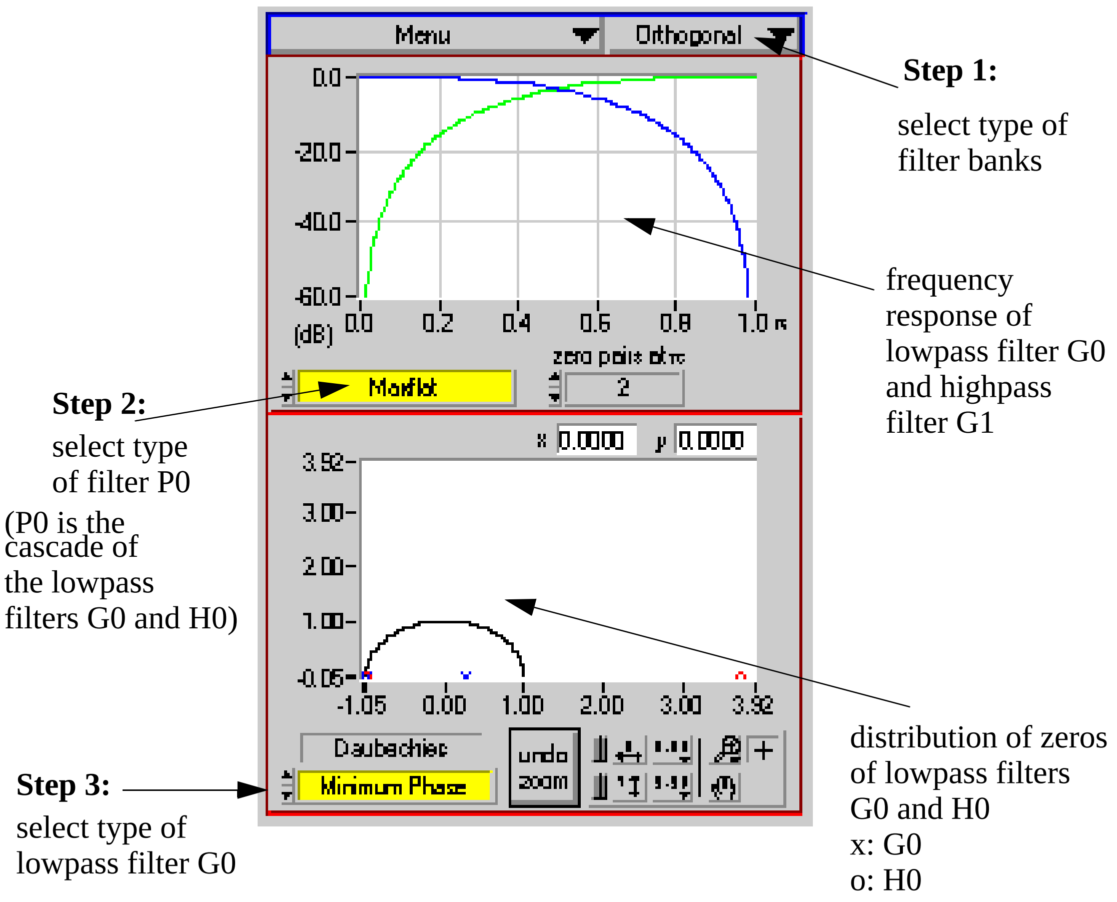
Step 3: Select the type of lowpass filter G0 and separate P0 into G0 and H0. This step consists of assigning the zeros of P0 to the lowpass filters G0 and H0. The user decides which zeros to assign to G0 and which to assign to H0.

Which Type of Filter or Filter Bank Is a Good Choice?

Different selections in steps 1, 2, and 3 will result in different wavelets. Although some guidelines have been given as to which choice is preferable over the other, in general there is no hard and fast rule for a particular choice of filter or filter bank. The choice of a suitable wavelet for a particular application is usually done by a trial and error process. Thus, you will need to experiment with the different combinations of available choices to obtain the best wavelet for your particular application.

The Design Panel

When you bring up the WFBD application from the **Programs** menu, you get the following front panel.


Menu
Orthogonal
Step 1:
select type of filter banks
0.0
-20.0
-40.0
-60.0
(dB)
0.0 0.2 0.4 0.6 0.8 1.0 π
frequency response of lowpass filter G0 and highpass filter G1
zero pairs at π
2
Maxflat
Step 2:
select type of filter P0
(P0 is the cascade of the lowpass filters G0 and H0)
x 0.0000 y 0.0000
3.92
3.00
2.00
1.00
-0.05
-1.05 0.00 1.00 2.00 3.00 3.92
distribution of zeros of lowpass filters G0 and H0
x: G0
o: H0
Daubechies
Minimum Phase
undo zoom
Step 3:
select type of lowpass filter G0

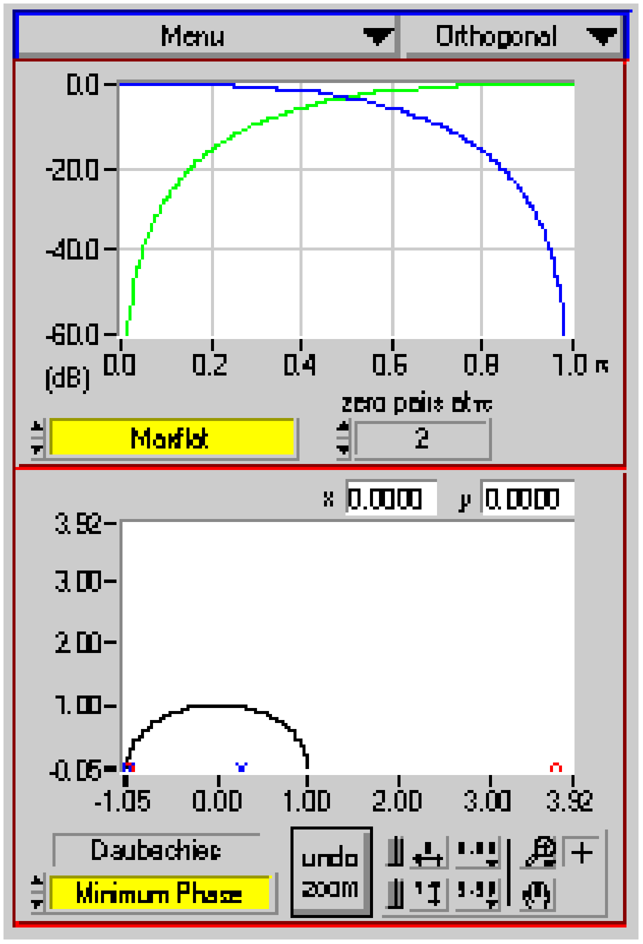
An explanation of the plots and controls is provided in the above figure. The figure also shows you the sequence of steps that you need to follow while designing your wavelet.

In the following exercises, you will see how this panel is used for analyzing an ECG signal (a 1D signal) and for data compression of fingerprints (a 2D image).

Exercise 14-1

Objective: To design a wavelet that captures the important features of an ECG signal, but with less than half the number of data points.

1. Launch the WFBD from the **Programs** menu.
2. The Design Panel appears. It gives you the power of designing arbitrary wavelets for your particular application. The Design Panel is shown below:



You have seen that the wavelet transform can be implemented as a bank of filters that decompose a signal into multiple frequency bands. It separates and retains the signal features in one or a few of these subbands. Thus, one of the biggest advantages of using the wavelet transform is that signal features can be easily extracted. Because of their ability to effectively extract signal features, wavelet transforms find many applications in the fields of data compression, pattern recognition, edge detection, echo detection, speech recognition, and texture analysis.

The key to creating a successful wavelet application is to select an appropriate wavelet, which is equivalent to selecting a good set of filters in the filter bank. You saw that there are three major steps involved in the design of the filters and filter banks. These steps are repeated below.

Step 1: Select the type of filter bank. The available choices are orthogonal or biorthogonal.

Step 2: Select the type of filter for P0. The available choices are maxflat, positive equiripple, or general equiripple. Remember that the general equiripple option is not available with orthogonal filter banks.

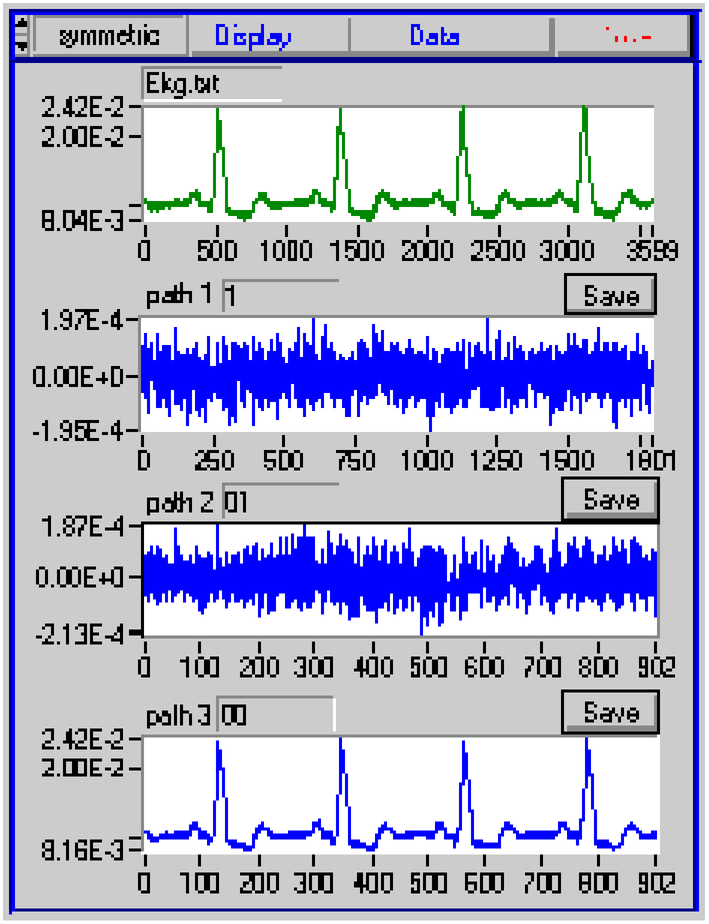
Step 3: Select the type of lowpass filter G0 and separate P0 into G0 and H0. The separation is achieved by deciding which zeros to assign to G0 and which to assign to H0.

The wavelet that you design will depend on the choices that you select in steps 1, 2, and 3. Thus, you can experiment with the different combinations to obtain the best wavelet for your particular application.

3. In the Design Panel, select **Menu » 1D Data Test**. A new window (1D_Test) will appear. In this window, you will load a 1D signal (for this example, you will use an ECG signal) having 3600 data points (samples) and see how it can be represented with a smaller number of samples.
4. In the new window, select **Data » Read from File » txt file**. Choose the file `Ecg.txt`.
5. You will now see four plots on the screen. In the topmost plot, you see the original ECG signal. It consists of 3600 data points (samples). You want to preserve the main features of this signal by using a minimum number of data points.

   Note that the original ECG signal has been decomposed into three parts by several stages of lowpass and highpass filtering. This is shown in the plots labeled **path1**, **path 2**, and **path 3**. A 0 indicates a stage of lowpass filtering, and a 1 indicates a stage of highpass filtering. Referring back to the explanation in section D, the figures in the plots labeled **path1**, **path 2**, and **path 3** correspond to $s_3[n]$, $s_2[n]$, and $s_1[n]$, respectively. The lowermost plot, obtained after two stages of lowpass filtering, contains the main features of the original ECG, but has only about 900 data points. Its size is therefore about one-fourth of that of the original data. The other two plots basically consist of only higher frequency noise and can be ignored.

Therefore, the plot in **path 3** can be used to represent the original ECG, but with fewer data points.



6. Switch to the Design Panel. Experiment with the controls on the design panel so as to change the wavelet being used, and note the corresponding changes in the plots on the 1D_Test window.
7. In particular, change the type of filter from maxflat to positive equiripple. Note that some features of the original signal appear in the “noise” channels also. Now the plot labeled **path 2** also contains some feature of the original ECG. Thus, this is not a good design for the wavelet, because now to represent the original signal, you cannot use only the 900 sample points of the lowermost plot, because it does not contain all the “information” of the original ECG signal. Some of this information is also present in the plot labeled **path 2**.
8. To see the wavelet, select **Menu » Wavelet and Filters** in the Design Panel. For more details, refer to the WFBD reference manual.

End of Exercise 14-1

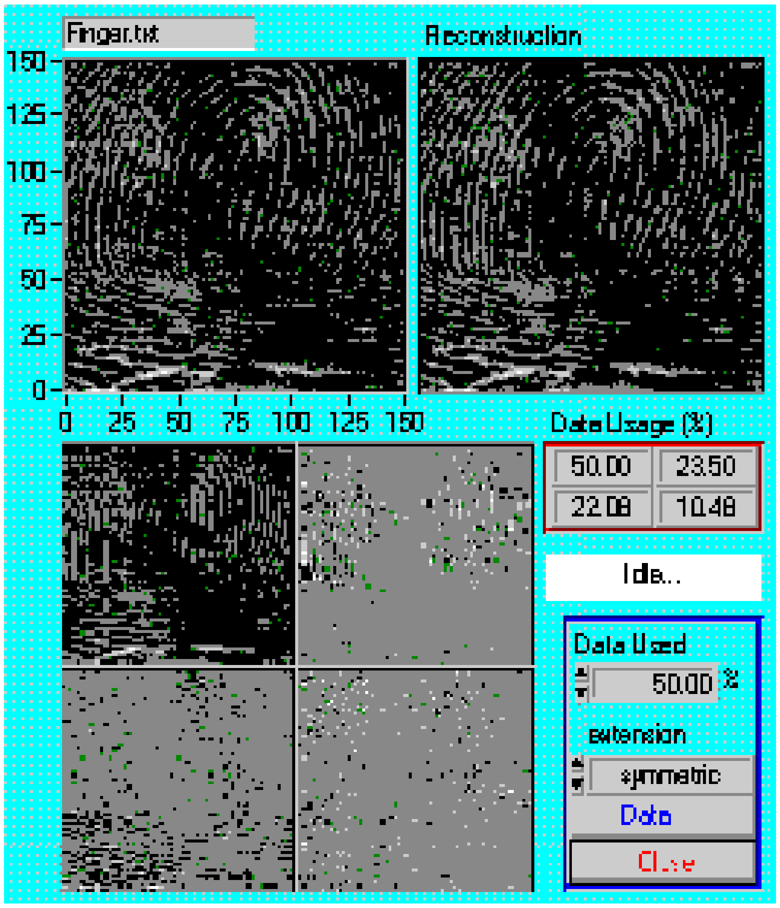
Exercise 14-2

Objective: To design a wavelet for reconstructing a fingerprint.

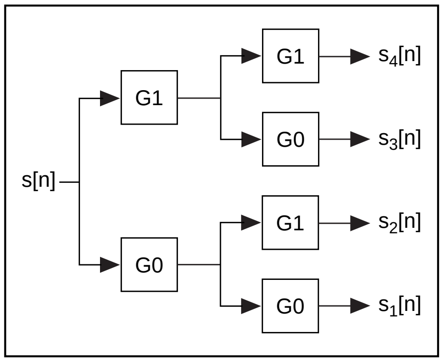
1. Open the WFBD Toolkit from the **Programs** menu, if it is not already open from the previous exercise.
2. In the Design Panel, select **MENU » 2D Data Test**. A new window will appear with the title 2D_Tes**t**. In this window, you can load two-dimensional signals (for example, images) on which to perform wavelet analysis.
3. From the control in the lower right corner of the 2D_Test window, select **Data » text** and choose the file `Finger.txt`.

   You will see two fingerprints on the upper half of the 2D_Test window. On the top left figure, you will see an original fingerprint consisting of 150x150 pixels. Law enforcement agencies have a need to keep millions of fingerprints on file. This can obviously occupy a lot of disk space. In addition, it would take an extremely long time to transmit so much data over the network. So, wavelets can be used to reduce the amount of data to represent the fingerprint. By an appropriate choice of wavelets, you can reduce the number of data points by over 50 percent.

   The figure on the top right shows the image that has been reconstructed by using the wavelet transform. The ratio of the number of data points used for reconstruction to the number of data points of the original image is shown in the Remaining Data control. In the example shown, only 50 percent of the total number of points (150x150) are used to reconstruct the original fingerprint. Note that there is hardly any difference between the original image and the reconstructed image.



The lowermost figures show the images obtained after the process of lowpass and highpass filtering (refer to section D). To recapitulate, if the input image is *s*[*n*], then the resulting images after filtering are $s_1[n]$, $s_2[n]$, $s_3[n]$, and $s_4[n]$, as shown below.



Thus, $s_1[n]$ is obtained after two stages of lowpass filtering, $s_4[n]$ is obtained after two stages of highpass filtering, and so on. If H denotes the result of highpass filtering, and L denotes a result of lowpass filtering, then their relationship with the images on the figure of the bottom half of the 2D_Test window is shown below:

LL $\boldsymbol{s_1[n]}$	LH $\boldsymbol{s_2[n]}$
HL $\boldsymbol{s_3[n]}$	HH $\boldsymbol{s_4[n]}$

4. Increase **Data Used** from 50 percent to 75 percent. Note that there is not much improvement in the reconstructed image.
5. Decrease **Data Used** to 25 percent. Is there a difference between the original and the reconstructed image? Experiment on the design panel to get a good reconstructed image with **Data Used** set at 25 percent.
6. When you are done, exit the WFBD by selecting **Menu » Quit** in the Design Panel.

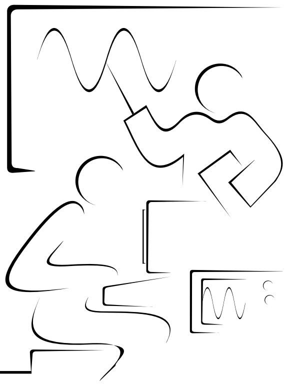
End of Exercise 14-2

Summary

You have seen that the basis functions for the wavelet transform are known as *wavelets* and are compressed and stretched versions of one another, with a finite time duration. Wavelets with a short time duration are good for high-frequency analysis, whereas those with a longer time duration are good for low-frequency analysis. Using wavelets, it is possible to simultaneously get good time as well as frequency resolution. Depending on your application, a particular wavelet may be more suitable than another wavelet. Hence, designing an appropriate wavelet is an important part of wavelet analysis. The design of wavelets is closely related to the design of filter banks. Using the Wavelet and Filter Banks Designer, you can interactively design the filters in the filter bank and simultaneously monitor the effect they have on the signal. Because of their ability for effective feature extraction, wavelets are widely used for data compression.

Notes

Notes



Appendix A Error Codes

Analysis Error Codes

Table A-1. Analysis Error Codes

Code	Name	Description
0	NoErr	No error; the call was successful.
–20001	OutOfMemErr	There is not enough memory left to perform the specified routine.
–20002	EqSamplesErr	The input sequences must be the same size.
–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.
–20005	SamplesGEOneErr	The number of samples must be greater than or equal to one.
–20006	SamplesGETwoErr	The number of samples must be greater than or equal to two.
–20007	SamplesGEThreeErr	The number of samples must be greater than or equal to three.
–20008	ArraySizeErr	The input arrays do not contain the correct number of data values for this VI.
–20009	PowerOfTwoErr	The size of the input array must be a power of two: size = 2^m, $0 < m < 23$.
–20010	MaxXformSizeErr	The maximum transform size has been exceeded.
–20011	DutyCycleErr	The duty cycle must meet the condition: $0 \le$ duty cycle ≤ 100.

Table A-1. Analysis Error Codes (Continued)

Code	Name	Description
–20012	CyclesErr	The number of cycles must be greater than zero and less than or equal to the number of samples.
–20013	WidthLTSamplesErr	The width must meet the condition: $0 < \text{width} < \text{samples}$.
–20014	DelayWidthErr	The delay must meet the condition: $0 \le (\text{delay} + \text{width}) < \text{samples}$.
–20015	DtGEZeroErr	dt must be greater than or equal to zero.
–20016	DtGTZeroErr	dt must be greater than zero.
–20017	IndexLTSamplesErr	The index must meet the condition: $0 \le \text{index} < \text{samples}$.
–20018	IndexLengthErr	The index must meet the condition: $0 \le (\text{index} + \text{length}) < \text{samples}$.
–20019	UpperGELowerErr	The upper value must be greater than or equal to the lower value.
–20020	NyquistErr	The cutoff frequency, f_c, must meet the condition: $0 \le f_c \le \frac{f_s}{2}$
–20021	OrderGTZeroErr	The order must be greater than zero.
–20022	DecFactErr	The decimating factor must meet the condition: $0 < \text{decimating} \le \text{samples}$.
–20023	BandSpecErr	The band specifications must meet the condition: $0 \le f_{flow} \le f_{high} \le \frac{f_s}{2}$
–20024	RippleGTZeroErr	The ripple amplitude must be greater than zero.
–20025	AttenGTZeroErr	The attenuation must be greater than zero.
–20026	WidthGTZeroErr	The width must be greater than zero.
–20027	FinalGTZeroErr	The final value must be greater than zero.
–20028	AttenGTRippleErr	The attenuation must be greater than the ripple amplitude.
–20029	StepSizeErr	The step-size, μ, must meet the condition: $0 \le \mu \le 0.1$.

Table A-1. Analysis Error Codes (Continued)

Code	Name	Description
–20030	LeakErr	The leakage coefficient must meet the condition: $0 \leq \text{leak} \leq \mu$.
–20031	EqRplDesignErr	The filter cannot be designed with the specified input values.
–20032	RankErr	The rank of the filter must meet the condition: $1 \leq (2\ \text{rank} + 1) \leq \text{size}$.
–20033	EvenSizeErr	The number of coefficients must be odd for this filter.
–20034	OddSizeErr	The number of coefficients must be even for this filter.
–20035	StdDevErr	The standard deviation must be greater than zero for normalization.
–20036	MixedSignErr	The elements of the **Y Values** array must be nonzero and either all positive or all negative.
–20037	SizeGTOrderErr	The number of data points in the **Y Values** array must be greater than two.
–20038	IntervalsErr	The number of intervals must be greater than zero.
–20039	MatrixMulErr	The number of columns in the first matrix is not equal to the number of rows in the second matrix or vector.
–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.
–20042	LevelsErr	The number of levels is out of range.
–20043	FactorErr	The level of factors is out of range for some data.
–20044	ObservationsErr	Zero observations were made at some level of a factor.
–20045	DataErr	The total number of data points must be equal to the product of the levels for each factor and the observations per cell.

Table A-1. Analysis Error Codes (Continued)

Code	Name	Description
–20046	OverflowErr	There is an overflow in the calculated F-value.
–20047	BalanceErr	The data is unbalanced. All cells must contain the same number of observations.
–20048	ModelErr	The Random Effect model was requested when the Fixed Effect model was required.
–20049	DistinctErr	The x values must be distinct.
–20050	PoleErr	The interpolating function has a pole at the requested value.
–20051	ColumnErr	All values in the first column in the X matrix must be one.
–20052	FreedomErr	The degrees of freedom must be one or more.
–20053	ProbabilityErr	The probability must be between zero and one.
–20054	InvProbErr	The probability must be greater than or equal to zero and less than one.
–20055	CategoryErr	The number of categories or samples must be greater than one.
–20056	TableErr	The contingency table must not contain a negative number.
–20061	InvSelectionErr	One of the input selections is invalid.
–20062	MaxIterErr	The maximum iterations have been exceeded.
–20063	PolyErr	The polynomial coefficients are invalid.
–20064	InitStateErr	This VI has not been initialized correctly.
–20065	ZeroVectorErr	The vector cannot be zero.

G Math Toolkit Error Codes

Table A-2. G Math Toolkit Error Codes

Error Code Number	Error Code Description
0	No error
–23001	Syntax error of parser
–23002	Discrepancy between function, variables and coordinates
–23003	Number of contours out of range
–23004	Number of color palettes out of range
–23005	Negative distance
–23006	Not a valid path
–23007	Not a graphs file
–23008	Wrong input, Euler method
–23009	Wrong input, Runge Kutta method
–23010	Wrong input, Cash Karp method
–23011	Nonpositive step rate
–23012	Nonpositive accuracy
–23013	Matrix vector conflict
–23014	A and X0 have different dimensions
–23015	Empty X0
–23016	Singular eigenvector matrix
–23017	Multiple roots
–23018	Left point is a root
–23019	Right point is a root
–23020	Left point greater than right point
–23021	Both function values have the same sign
–23022	Nonpositive accuracy or nonpositive delta x(h)
–23023	Wrong dimension of start
–23024	No root found

Table A-2. G Math Toolkit Error Codes (Continued)

Error Code Number	Error Code Description
–23025	Nonvalid triplet (a,b,c)
–23026	No optimum found
–23027	Not exactly one variable
–23028	Wrong model equation
–23029	Levenberg Marquardt has failed
–23030	m >= n >= 0 is violated or the matrix of derivatives has the wrong dimension
–23031	No valid point
–23032	Maximum does not exist
–23033	Vectors have different dimensions or empty vectors
–23034	Ill conditioned system
–23035	Nonpositive number
–23036	Different parameters
–23037	Not exactly two functions
–23038	No variables in expression
–23039	Parameter problem
–23040	Derivative out of range
–23041	Not exactly two variables
–23042	Negative argument
–23043	Argument out of range (0,1]
–23044	Argument out of range [0,1]
–23045	n<k
–23046	Empty array
–23047	Argument out of range [0,100]
–23048	Invalid time increment
–23049	Invalid window length
–23050	Signal length not a multiple of number

Table A-2. G Math Toolkit Error Codes (Continued)

Error Code Number	Error Code Description
–23051	Signal length not a power of two
–23052	Signal length not a prime and ≥ 5
–23053	Signal length not a power of two and ≥ 4
–23054	Non–unique variables

Functions for Use with G Math Toolkit Parser VIs

Table A-3. Functions for Use with G Math Toolkit Parser VIs

Function	Corresponding G Math Function name	Description
abs(x)	Absolute Value	Returns the absolute value of *x*.
acos(x)	Inverse Cosine	Computes the inverse cosine of *x*.
acosh(x)	Inverse Hyperbolic Cosine	Computes the inverse hyperbolic cosine of *x* in radians.
asin(x)	Inverse Sine	Computes the inverse sine of *x* in radians.
asinh(x)	Inverse Hyperbolic Sine	computes the inverse hyperbolic sine of *x* in radians.
atan(x)	Inverse Tangent	Computes the inverse tangent of *x* in radians.
atanh(x)	Inverse Hyperbolic Tangent	Computes the inverse hyperbolic tangent of *x* in radians.
ci(x)	Cosine Integral	Computes the cosine integral of *x* where *x* is any real number.
ceil(x)	Round to +Infinity	Rounds *x* to the next higher integer (smallest integer ≥ *x*.)
cos(x)	Cosine	Computes the cosine of *x* in radians.
cosh(x)	Hyperbolic Cosine	Computes the hyperbolic cosine of *x* in radians.
cot(x)	Cotangent	Computes the cotangent of *x* in radians (1/tan(x)).
csc(x)	Cosecant	Computes the cosecant of *x* in radians (1/sin(x)).

Table A-3. Functions for Use with G Math Toolkit Parser VIs (Continued)

Function	Corresponding G Math Function name	Description
exp(x)	Exponential	Computes the value of *e* raised to the power *x*.
expm1(x)	Exponential(Arg)–1	Computes the value of *e* raised to the power of *x* – 1 ($e^x - 1$)
floor(x)	Round to –Infinity	Truncates x to the next lower integer (Largest integer $\leq x$)
gamma(x)	Gamma Function	$\Gamma(n+1) = n!$ for all natural numbers *n*.
getexp(x)	Mantissa and exponent	Returns the exponent of *x*.
getman(x)	Mantissa and exponent	Returns the mantissa of *x*.
int(x)	Round to nearest integer	Rounds its argument to the nearest even integer.
intrz	Round toward zero	Rounds *x* to the nearest integer between *x* and zero.
ln(x)	Natural Logarithm	Computes the natural logarithm of *x* (to the base *e*).
lnpl(x)	Natural Logarithm (Arg + 1)	Computes the natural logarithm of (*x* + 1).
log(x)	Logarithm Base 10	Computes the logarithm of *x* (to the base 10).
log2(x)	Logarithm Base 2	Computes the logarithm of *x* (to the base 2).
pi(x)	Represents the value π = 3.14159...	pi(*x*) = *x**π pi(1) = π pi(2.4) = 2.4*π
rand( )	Random Number (0–1)	Produces a floating-point number between 0 and 1.
sec(x)	Secant	Computes the secant of *x* (1/cos(*x*)).
si(x)	Sine Integral	Computes the sine integral of *x* where *x* is any real number.
sign(x)	Sign	Returns 1 if *x* is greater than 0. Returns 0 if *x* is equal to 0. Returns –1 if *x* is less than 0.

Table A-3. Functions for Use with G Math Toolkit Parser VIs (Continued)

Function	Corresponding G Math Function name	Description
sin(x)	Sine	Computes the sine of *x* in radians.
sinc(x)	Sinc	Computes the sine of *x* divided by *x* in radians (sin(*x*)/*x*).
sinh(x)	Hyperbolic Sine	Computes the hyperbolic sine of *x* in radians.
spike(x)	Spike function	spike(*x*) returns: 1 if $0 \le x \le 1$ 0 for any other value of *x*.
sqrt(x)	Square Root	Computes the square root of *x*.
square(x)	Square function	square (*x*) returns: 1 if $2n \le x \le (2n+1)$ 0 if $2n+1 \le x \le (2n+2)$ where *x* is any real number and *n* is any integer.
step(x)	Step function	step(*x*) returns: 0 if $x < 0$ 1 if any other condition obtains.
tan(x)	Tangent	Computes the tangent of *x* in radians.
tanh(x)	Hyperbolic Tangent	Computes the hyperbolic tangent of *x* in radians.

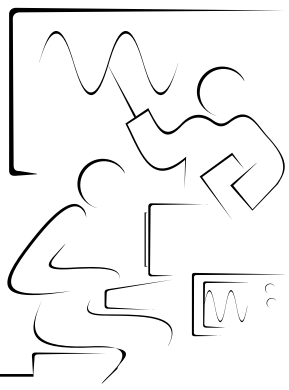
G Math Toolkit Parser Error Codes

Table A-4. G Math Toolkit Parser Error Codes

Error Code Number	Error Code Description	Error Example
0	No error	sin(x)
1	Bracket problem at the beginning	1+x)
2	Incomplete function expression	sin(x)+
3	Bracket problem	()
4	Bracket problem at the end	(1+x

Table A-4. G Math Toolkit Parser Error Codes (Continued)

5	Wrong decimal point	1,2 (US)
6	Wrong number format	1e–3 instead of 1E–3
7	Wrong function call	sin()
8	Not a valid function	sins(x)
9	Incomplete expression	x+
10	Wrong variable name	a11
11	Wrong letter	sin(X)
12	Too many decimal points	1.23.45
21	Contains more than one variable	1+x+y4
22	Inconsistency in variables or numbers	Depends on application
23	Contains variables	Depends on application
24	Variables output problem	Depends on application



Appendix B Frequently Asked Questions

Background

1. **Can the antialias filter be a hybrid filter?**

   When going from the analog to the digital domain, the antialias filter must be an analog filter. In general, when going from a domain P to a domain Q, the filter should be in domain P to avoid the problem of aliasing in the domain Q. Thus, when sampling to go from an analog to a digital signal, the filter should be an analog filter. However, when performing decimation to reduce the sampling rate, you are going from the digital domain (with a higher sampling rate) to the digital domain (with a lower sampling rate). In this case, the filter before the decimation is performed will be a digital filter.

Signal Generation

1. **Why are there Pattern VIs, when the Wave VIs will do all that the pattern VIs can and more?**

   The historical reason is that Pattern VIs were created first. Also, the Pattern VIs are meant to be used to generate a block of data at a time, whereas the Wave VIs can be called iteratively in a loop.

2. **Why do I need normalized frequency? Why not just use the common frequency unit of Hertz in the input controls and have the VI automatically convert it internally into normalized frequency if necessary?**

   Because some people think in terms of Hertz (cycles per second) and some think in terms of cycles. Normalized frequency handles both.

3. **What does it mean that the Wave VIs are "reentrant"?**

   You can have several of them on the same block diagram, and each one will have its own code. They will all work independently of each other.

Signal Processing

1. **What are the limits/problems of zero padding? Does it introduce harmonics? Does it affect the frequency content of a signal in any way?**

   Zero padding performs interpolation in the frequency domain. No, it does not introduce harmonics. In addition to improving (that is, lowering) the frequency resolution, faster algorithms (using the FFT instead of the DFT) are possible.

2. **What is the physical meaning of the cross power spectrum?**

   It is a measure of the similarity of two signals in the frequency domain. It is the frequency-domain equivalent of the cross-covariance function. It shows the joint presence of energy in the two signals.

Windows

1. **Why do different toolkits provide only some specific window choices? Why not all window choices?**

   One reason is that the windows chosen are the most common windows for that particular application. The other is that several of the windows give very similar results, so there is no point in including all of them.

Digital Filters

1. **What is meant by linear phase?**

   Linear phase in digital filters means that the phase distortion is nothing more than a digital delay. All input samples will be shifted by some constant number of samples, so this phase change can be easily “fixed” and/or modeled.

   Nonlinear phase means that the individual sine waves that make up the input signal get shifted in time by different amounts. This sort of phase change is very difficult to work around. Some signals (like the square wave) are very sensitive to this sort of phase distortion.

2. **What does it mean when the IIR filters “execute in place”?**

   In place means that the input array space (memory locations) are being reused as the output array space. In place usually implies lower demands on memory.

3. **Do the FIR filters have only zeros?**

   FIR filters do have zeros, but they also have poles at the origin. If you take the Z-transform of an FIR filter, and rewrite it as the product of factored terms, you will find that in addition to zeros, they also have poles at the origin.

Curve Fitting

1. **What is the limit in LabVIEW and BridgeVIEW on the number of parameters used for curve fitting?**

   There is no limit. However, keep in mind that the memory requirements, as well as the time required to find the solution, will increase with increasing number of parameters.

2. **Some curve fitting algorithms allow you to give weight to certain data points. Can I do that in the VIs from the advanced analysis library of LabVIEW and BridgeVIEW?**

   No, it is not possible.

3. **In performing a fit, is there any rule of thumb as to how many data points to use?**

   Normally, you need at least one more than the number of parameters for which you are trying to solve. But there is no such rule as to whether you should use at least five times more, 10 times more, etc. As an example, in performing a polynomial fit, the number of data points to be used to obtain a "good" fit may be correlated to how close the data is to the underlying polynomial (that is, how much noise). It depends.

4. **In the Levenberg-Marquardt VI, why is the derivative information needed?**

   The information is needed to calculate the Jacobian, which is needed in the algorithm to solve for the coefficients that you are trying to determine. See the *Analysis VI Reference Manual* for details.

Digital Filter Design Toolkit

1. **Is a demo version of the toolkit available?**

   Yes. In addition to being a separate toolkit, the DFD also ships with the Signal Processing Suite (SPS). The entire SPS is available in demo form also.

2. **Is there a UNIX version of the DFD Toolkit?**

   No, it has not been released on UNIX.

3. **In the Arbitrary FIR Design panel, if I select "import from file" and then "cancel" the operation, I get an error, "Error 43 occurred...". Why is that?**

   This is due to a bug that will be fixed in the next release.

Wavelets

1. **When I do the decimation (by 2), what options do I have? Can I throw away every even sample, every odd sample, the first half, the second half, etc?**

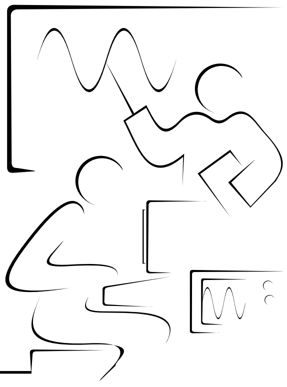
As long as you filter the data before performing the decimation, it does not matter whether you throw away every odd sample or every even sample. However, you must be consistent. You cannot throw away the odd samples at the beginning and then the even samples later, or vice versa. Similarly, you cannot throw away every seventh sample in the beginning, and then every fourth sample later. It is not proper to remove the first half samples, or the next half samples. Normally, just the odd or even samples are removed.

2. **When I do the upsampling (by 2), what options do I have?**

   In upsampling by 2, you need to insert zeros so as to double the number of samples. The zeros to be inserted could be in either the odd samples position or the even samples position. Depending on the manner in which downsampling was done, you would insert zeros in the same place. So, if you removed the odd-numbered samples during downsampling, you would insert them in the odd-numbered places during upsampling. However, if you removed the even-numbered samples during downsampling, you would insert them in the even-numbered places.

3. **In Wavelets » 1D Data Test » ekg.txt » save the 2nd plot, if I see the text file, all the numbers are zero because they were very small and were not saved with a sufficiently high precision. So, how can I change the format of saving the text file?**

   If you are using the wavelets executable, you cannot change the format in which the text file is being saved. However, if you are using the LabVIEW wavelet VI libraries, you can go into the block diagram to change the format.



Appendix C References

The following documents contain more detailed information about the topics discussed in this course.

National Instruments Manuals

- *Digital Filter Design Toolkit Reference Manual*
- *G Math Toolkit Reference Manual*
- *JTFA Reference Manual*
- *LabVIEW Analysis VI Reference Manual*
- *LabVIEW Data Acquisition Course Manual*
- *Third-Octave Analysis Toolkit Reference Manual*
- *Wavelet and Filter Banks Design Reference Manual*

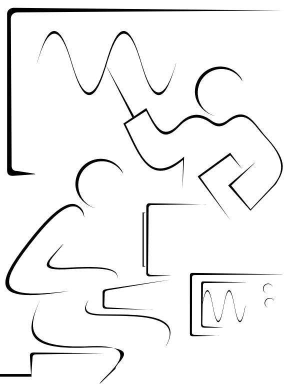
National Instruments Application Notes

- Application Note #023—*Digital Signal Processing Fundamentals*
- Application Note #041—*The Fundamentals of FFT-Based Signal Analysis and Measurement in LabVIEW and LabWindows*
- Application Note #040—*Fast Fourier Transforms and Power Spectra in LabVIEW*
- Application Note #091—*G Math—A New Paradigm for Mathematics*
- Application Note #097—*Designing Filters Using the Digital Filter Design Toolkit*

Other Documents

- *Advanced Engineering Mathematics*, Erwin Kreyszig, Fifth Edition, Wiley Eastern Limited, 1983
- *ANSI S1.11-1986 Specification for Octave-Band and Fractional-Octave-Band Analog and Digital Filters*

- *Designing Digital Filters*, Charles S. Williams, Prentice-Hall, 1986
- *Digital Filters*, R.W.Hamming, Third Edition, Prentice-Hall, 1989
- *Discrete-Time Signal Processing*, Alan V. Oppenheim and Ronald W. Schafer, Prentice Hall, 1989
- *The FFT: Fundamentals and Concepts*, Robert W. Ramirez, Prentice-Hall, 1985
- *Joint Time-Frequency Analysis: Methods and Applications*, Shie Qian and Dapang Chen, Prentice-Hall, 1996
- *Multirate Systems and Filter Banks*, P.P.Vaidyanathan, Prentice-Hall, 1993
- *Numerical Recipes*, Press and Flannery and Teukolsky and Vetterling, Cambridge University Press, 1986
- “On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform,” Frederic J. Harris, *Proceedings of the IEEE*, Vol. 676, No. 1, January 1978, pp. 51-84.
- “Some Windows with Very Good Sidelobe Behaviour,” Albert H. Nuttall, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-29, No. 1, February 1981, pp. 84-91.
- “Windows to FFT Analysis,” Svend Gade and Henrik Herlufsen, *Sound and Vibration*, March 1988, pp. 14-22.



Appendix D Glossary

Prefix	Meaning	Value
M-	mega-	10^6
K-	kilo-	10^3
m-	milli-	10^{-3}
µ-	micro-	10^{-6}
n-	nano-	10^{-9}

Numbers/Symbols

1D	One-dimensional.
2D	Two-dimensional.
×	Infinity.
¼	Pi.
ý	Delta. Difference. ý*x* denotes the value by which *x* changes from one index to the next.

A

AC	Alternating current.
ADC	Analog to digital convertor, the hardware that converts an analog signal into a digital signal.
analog signal	A signal whose values are continuous and are defined at each and every instant of time.

ANSI	American National Standards Institute.
antialias filter	An analog lowpass filter used to limit the frequency of the analog signal, before sampling, to less than the Nyquist frequency. This prevents aliasing in the digital domain.
array	Ordered, indexed set of data elements of the same type.

B

bandpass filter	A filter that passes signals within a certain band of frequencies.
bandstop filter	A filter that attenuates signals within a certain band of frequencies.
BPF	Bandpass filter.
BSF	Bandstop filter.
Butterworth filter	A filter characterized by a smooth response at all frequencies, and the absence of ripples in both the passband and the stopband. Due to the absence of ripples, it is also known as a maximally flat filter.

C

Chebyshev filter	A filter characterized by ripples in the passband, but a smooth response in the stopband.
Chebyshev II filter	*See* Inverse Chebyshev Filter.
coherent gain (CG)	The coherent gain is the zero frequency gain (or DC gain) of a window. The rectangular (or uniform) window has the highest CG.
complex conjugate transpose	A matrix operation consisting of taking the complex conjugate of each element of the matrix and then transposing the resulting matrix.
complex matrix	A matrix with at least one element that is a complex number.
condition number	A measure of how close the matrix is to being singular. The condition number of a square nonsingular matrix is defined as $cond(\mathbf{A}) = \|\mathbf{A}\|_p \cdot \|A^{-1}\|_p$ where p corresponds to the p^{th} norm of the matrix. The condition number can vary between 1 and infinity. A matrix with a large condition number is nearly singular, while a matrix with a condition number close to 1 is far from being singular.

curve fitting	Technique for extracting a set of curve parameters or coefficients from a data set to obtain a functional description of the data set.

D

DAQ	Data acquisition.
data acquisition	Process of acquiring data, typically from A/D or digital input plug-in boards.
dB	Decibels.
DC	Direct current.
decibel	A logarithmic scale used to compress large amplitudes and expand small amplitudes. It is given by one dB = 10 $\log_{10}$ (Power Ratio) = 20 $\log_{10}$ (Voltage Ratio)
DFT	Discrete Fourier transform, the algorithm used to transform samples of the data from the time domain into the frequency domain.
difference equations	Equations that describe the operation of a system (for example, a filter) in the discrete time domain.
digital signal	A signal that can take on only specific amplitude values that are defined at discrete points in time.
discrete-time signal	A signal whose values are continuous in amplitude, but which is defined only at discrete points in time.
DSP	Digital signal processing.

E

elliptic filter	A filter characterized by ripples in both the passband and the stopband.
equivalent noise bandwidth (ENBW)	The equivalent noise bandwidth of a window is the width of an ideal rectangular response that will pass the same amount of noise power as the frequency response of the window. The rectangular (or uniform) window has the smallest ENBW.

F

FFT	Fast Fourier transform. A fast method for calculating the discrete Fourier transform. It is used when the number of samples is a power of two.
filter bank	A set of filters connected in parallel. Each filter may be tuned to a different frequency range. The filters in the filter bank may or may not have the same bandwidth.
FIR filter	Finite impulse response filter. A type of filter whose output depends only on the current and past inputs. It is also known as a nonrecursive filter.
forward coefficients	Forward coefficients are those that multiply inputs.
frequency response	The Fourier transform of the impulse response. It consists of two parts, the magnitude response and the phase response. The magnitude response is a plot of the magnitude of the frequency response at different frequencies, whereas the phase response is a plot of the phase of the frequency response at different frequencies. (see Impulse Response)

G

Gaussian probability density function	A density function that is completely characterized by its mean and standard deviation and is given by $$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$ where μ is the mean and σ is the standard deviation. It is also known as the normal probability density function.

H

harmonic distortion	The distortion inherent in a nonlinear system that results in generation of frequencies at its output that are harmonics of the input frequency. The more the degree of nonlinearity of the system, the higher the frequencies of the harmonics.
Hermitian matrix	A complex matrix whose complex conjugate transpose is equal to the matrix itself. *See also* complex conjugate transpose.
highpass filter	A filter that passes frequencies above a certain cut-off frequency. It passes high frequencies, but attenuates low frequencies.

HPF	Highpass filter.
Hz	Hertz, or cycles per second.

I

IEEE	Institute of Electrical and Electronics Engineers.
IIR filter	Infinite impulse response filter. A type of filter whose output depends not only on the current and past inputs, but also on past outputs. It is also known as a recursive filter.
IMAQ	Image acquisition.
impulse	A signal that has a value of one at a particular time instant and zero everywhere else.
impulse response	The response of a system to an input that is an impulse.
inf	Digital display value for a floating-point representation of infinity.
in place	In place means that the input array space (memory locations) are being reused as the output array space. In place usually implies lower demands on memory.
inverse Chebyshev filter	A filter characterized by a smooth response in the passband, but with ripples in the stopband.
I/O	Input/output. The transfer of data to or from a computer system involving communications channels, operator input devices, and/or data acquisition and control interfaces.

J

joint time-frequency analysis (JTFA)	A method of analysis that simultaneously provides both time and frequency information. It shows how the frequency spectrum of a signal varies with time.

K

Levenberg-Marquardt	A general curve fitting algorithm used to estimate the coefficients of a curve to fit a set of samples. It can be used for both linear and nonlinear relationships, but is almost always used to fit a nonlinear curve. This is because the general linear fit method is better suited to linear curve fitting. The Levenberg-Marquardt method does not

	always guarantee a correct result, so it is absolutely necessary to verify the results.

L

linear phase	Linear phase in digital filters means that the phase distortion is nothing more than a digital delay. All input samples will be shifted by some constant number of samples, so this phase change can be easily "fixed" and/or modeled. Nonlinear phase means that the individual sine waves that make up the input signal get shifted in time by different amounts. This sort of phase change is very difficult to work around. Some signals (like the square wave) are very sensitive to this sort of phase distortion.
lower triangular matrix	A matrix whose elements above the main diagonal are all zero.
lowpass filter	A filter that passes frequencies below a certain cut-off frequency. It passes low frequencies, but attenuates high frequencies.
LPF	Lowpass filter.
LU decomposition	A method that factors a matrix as a product of an upper and a lower triangular matrix.

M

matrix	Two-dimensional array.
MB	Megabytes of memory.
mother wavelet	A prototype wavelet, which is compressed or expanded in time to derive other wavelets.
MSE	Mean squared error. The MSE is a relative measure of the residuals between the expected curve values and the actual observed values.

N

NaN	Digital display value for a floating-point representation of *not a number*, typically the result of an undefined operation, such as log(-1).
norm	The norm of a vector or matrix is a measure of the magnitude of the vector or matrix. There are different ways to compute the norm of a matrix. These include the *2-norm* (Euclidean norm), the *1-norm*, the *Frobenius norm* (F-norm), and the *Infinity norm* (inf-norm).

	Each norm has its own physical interpretation. The LabVIEW/BridgeVIEW **Matrix Norm** VI can be used to compute the norm of a matrix.
normal probability density function	*See* Gaussian probability density function.
nonlinear phase	*See* linear phase.
nonrecursive filter	*See* FIR filter.
nonsingular matrix	Matrix in which no row or column is a linear combination of any other row or column, respectively.
normalized frequency	A frequency (in Hertz) that is specified as a ratio with respect to the sampling frequency (in samples/second). Its units are in cycles/sample. However, if the frequency is given in terms of cycles, then it is divided by the number of samples to convert it to the normalized frequency.
Nyquist frequency (f_N)	Half the sampling frequency, $f_N = f_s/2$, where f_s is the sampling frequency.
Nyquist theorem	A theorem stating that to recover an analog signal from its samples, the sampling frequency should be at least twice the highest frequency in the signal.

O

observation matrix (H)	A matrix used as an input to the **General LS Linear Fit** VI. If there are *N* data points, and *k* coefficients (a_0, a_1, ...a_{k-1}) for which to solve, H will be an *N-by-k* matrix with *N* rows and *k* columns. Thus, the number of rows of H is equal to the number of data points, whereas the number of columns of H is equal to the number of coefficients for which we are trying to solve.
octave	A doubling in frequency.
one-sided transform	A representation consisting of only the positive frequency (and DC) components.

P

passband	The range of frequencies that are passed by a filter with a gain of almost one (0 dB).

passband ripple	The amount of variation of the passband gain from unity (0 dB). It is usually specified in dB. It is given by $\text{ripple (dB)} = 20\log_{10}\frac{A_o(f)}{A_i(f)}$ where $A_i(f)$ and $A_o(f)$ are the amplitudes of a particular frequency before and after the filtering, respectively.
pattern VIs	These are signal generation VIs that do not keep track of the phase of the signal that they generate each time they are called. The **Chirp Pattern** VI is the only pattern VI that requires its frequency input in terms of normalized frequency.
pole-zero plot	A plot showing the positions of the poles and zeros of a system. The pole-zero plot is useful in determining the stability of the system.
power spectrum	The power spectrum of a signal gives you the power in each of its frequency components. It can be calculated by squaring the magnitude of the Fourier transform (DFT or FFT) of the signal.

Q

quality factor (Q)	A measure of how selective a bandpass filter is in passing frequencies around the center frequency and attenuating unwanted frequencies. It is defined as the ratio of the center frequency of the filter to its bandwidth. $Q = f_m/B_m$ where f_m is the center frequency, and B_m is the bandwidth of the filter. Thus, for a fixed center frequency, the larger the bandwidth the smaller the quality factor, and vice versa.

R

rank	The *rank* of a matrix A, denoted by ρ(A), is the maximum number of linearly independent columns in A. The number of linearly independent columns of a matrix is equal to the number of independent rows. So, the rank can never be greater than the smaller dimension of the matrix. Consequently, if A is an $n \times m$ matrix, then $\rho(A) \le min(n, m)$ where *min* denotes the minimum of the two numbers. The rank of a square matrix pertains to the highest order nonsingular matrix that can be formed from it. So, the rank pertains to the highest order matrix that we can obtain whose determinant is not zero. A square

	matrix is said to have full rank if and only if its determinant is different from zero.
recursive filter	*See* IIR filter.
regression analysis	*See* curve fitting.
reverse coefficients	The reverse coefficients are those that multiply the outputs.
RHS	Right hand side.
ripple	A measure of the deviation of a filter from the ideal filter specifications.
RMS	Root mean square.

S

sampling frequency	The number of samples acquired per second. Its units are samples/second.
short-time Fourier transform (STFT)	The term for taking a Fourier transform of shorter time intervals of samples of a signal, rather on of the entire set of samples. Also known as the windowed Fourier transform.
singular value decomposition (SVD)	A method that decomposes a matrix into the product of three matrices $A = USV^T$, where *U* and *V* are orthogonal matrices, and *S* is a diagonal matrix. SVD is useful for solving analysis problems such as computing the rank, norm, condition number, and pseudoinverse of matrices.
spectral leakage	A phenomenon where it appears as if energy has leaked out from one frequency into another. It occurs because of the discontinuities introduced when the sampled waveform is repeated periodically in time. The larger the discontinuity, the more the leakage. Leakage can be reduced by reducing the amplitude of the discontinuities. The reduction is achieved by use of multiplying the time domain waveform by a window function. Note that if there are an integer number of cycles in the sampled waveform, there is no leakage.
step response	The response of a system to a step input.
stopband	The range of frequencies that are attenuated by the filter.

stopband attenuation	The amount of attenuation in the stopband of a filter, usually specified in dB. It is given by $A(\text{dB}) = 20\log_{10}\frac{A_o(f)}{A_i(f)}$ where $A_i(f)$ and $A_o(f)$ are the amplitudes of a particular frequency before and after the filtering, respectively.
SVD	Singular value decomposition.
symmetric matrix	A matrix whose transpose is equal to the matrix itself.

T

total harmonic distortion	A relative measure of the amplitudes of the fundamental to the amplitudes of the harmonics. If the amplitude of the fundamental is A_1, and the amplitudes of the harmonics are A_2 (2nd harmonic), A_3 (3rd harmonic), A_4 (4th harmonic), ...A_N (Nth harmonic), then the total harmonic distortion (THD) is given by $\text{THD} = \frac{\sqrt{A_1^2 + A_2^2 + A_3^2 + \ldots + A_N^2}}{A_1}$ When the THD is expressed as a percentage, it is known as the percentage total harmonic distortion (%THD) and is given by $\%\ \text{THD} = \frac{100 \times \sqrt{A_1^2 + A_2^2 + A_3^2 + \ldots + A_N^2}}{A_1}$
transient response	The transient that initially appears at the output of a filter when the VI is run. The time duration of the transient depends on the order of the filter. It can be eliminated by setting the init/cont control of the VI to TRUE.
transition region	The region between the passband and the stopband where the gain of the filter varies from one (0 dB) or almost one (in the passband) to a very small value (in the stopband).
transpose	A matrix operation that consists of interchanging the rows and columns of a matrix.
two-sided transform	A representation consisting of both the positive and negative (and DC) frequency components.

U

upper triangular matrix	A matrix whose elements below the main diagonal are all zero.

V

vector	One-dimensional array.

W

wavelets	The time-limited basis functions of the wavelet transform.
wavelet transform	The transform whose basis functions are time limited and are known as *wavelets*. Using the wavelet transform, it is possible to have both good time and good frequency resolution simultaneously. The selection of a suitable wavelet is an important consideration in the use of the wavelet transform.
wave VIs	Signal generation VIs that keep track of the phase of the signal that they generate each time they are called. They require their frequency input to be in terms of normalized frequency.
Wigner-Ville distribution	A method of joint time-frequency analysis. A drawback of this method is the presence of unwanted cross-terms between actual signal frequency components. These cross-terms can be removed by averaging.
window	A smoothing function applied to a time domain waveform, before it is transformed into the frequency domain, so as to minimize spectral leakage.
windowed Fourier transform	*See* short-time Fourier transform (STFT).

Z

zero padding	Addition of zeros to the end of a sequence so that the total number of samples is equal to the next higher power of two. When zero padding is applied to a set of samples in the time domain, faster computation is possible by using the FFT instead of the DFT. In addition, the frequency resolution (Δf) is improved (made smaller) because $\Delta f = f_s/N$, where f_s is the sampling frequency and N is the total number of samples.

Notes