LabVIEW™

Analysis Concepts

NATIONAL INSTRUMENTS™

April 2003 Edition
Part Number 370192B-01

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 61 2 9672 8846, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599,
Canada (Calgary) 403 274 9391, Canada (Montreal) 514 288 5722, Canada (Ottawa) 613 233 5949,
Canada (Québec) 514 694 8521, Canada (Toronto) 905 785 0085, Canada (Vancouver) 514 685 7530,
China 86 21 6555 7838, Czech Republic 420 2 2423 5774, Denmark 45 45 76 26 00,
Finland 385 0 9 725 725 11, France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, Greece 30 2 10 42 96 427,
Hong Kong 2645 3186, India 91 80 51190000, Israel 972 0 3 6393737, Italy 39 02 413091,
Japan 81 3 5472 2970, Korea 82 02 3451 3400, Malaysia 603 9059 6711, Mexico 001 800 010 0793,
Netherlands 31 0 348 433 466, New Zealand 64 09 914 0488, Norway 47 0 32 27 73 00,
Poland 48 0 22 3390 150, Portugal 351 210 311 210, Russia 7 095 238 7139, Singapore 65 6 226 5886,
Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 0 8 587 895 00,
Switzerland 41 56 200 51 51, Taiwan 886 2 2528 7227, United Kingdom 44 0 1635 523545

For further support information, see the *Technical Support and Professional Services* appendix. To comment on the documentation, send email to techpubs@ni.com.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or `ni.com/patents`.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

Chapter 3
Signal Generation

Chapter 4
Digital Filtering

Chapter 5
Smoothing Windows

Chapter 6
DC/RMS Measurements

Chapter 7
Distortion Measurements

Chapter 8
Limit Testing

Chapter 9
Curve Fitting

Chapter 10
Linear Algebra

Chapter 11
Probability and Statistics

Chapter 12
Point-By-Point Analysis

Appendix A
Technical Support and Professional Services

About This Manual

This manual provides information about analysis and mathematical concepts in LabVIEW.

Conventions

The following conventions appear in this manual:

»	The **»** symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.

Related Documentation

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

- *LabVIEW Measurements Manual*
- *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**
- *LabVIEW User Manual*

- *Getting Started with LabVIEW*
- *On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform* (Proceedings of the IEEE, Volume 66, No. 1, January 1978)

1

Introduction to Measurement Analysis in LabVIEW

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 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 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 also are converted to digital form before they are processed with a computer.

This chapter provides a background in basic digital signal processing and an introduction to the LabVIEW Measurement Analysis VIs.

The Importance of Data Analysis

The importance of integrating analysis libraries into engineering stations is that the raw data, as shown Figure 1-1, does not always immediately convey useful information. Often, you must transform the signal, remove noise disturbances, correct for data corrupted by faulty equipment, or compensate for environmental effects, such as temperature and humidity.

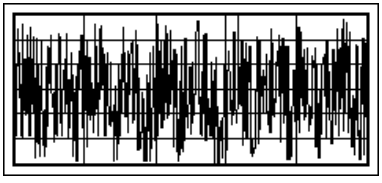


Figure 1-1. Raw Data

By analyzing and processing the digital data, you can extract the useful information from the noise and present it in a form more comprehensible than the raw data, as shown in Figure 1-2.

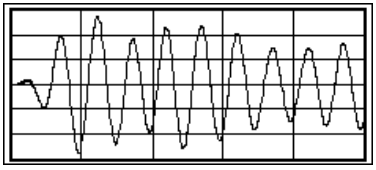


Figure 1-2. Processed Data

The LabVIEW block diagram programming approach and the extensive set of LabVIEW Measurement Analysis VIs simplify the development of analysis applications.

Data Sampling

Sampling Signals

To use digital signal processing techniques, you must first convert an analog signal into its digital representation. In practice, the conversion 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 sampling 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, $2\Delta t$, $3\Delta t$, and so on, is known as a sample. Thus, $x(0)$, $x(\Delta t)$, $x(2\Delta t)$, …, are all samples. The signal $x(t)$ can thus be represented by the following discrete set of samples.

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

Figure 1-3 shows an analog signal and its corresponding sampled version. The sampling interval is Δt. Notice that the samples are defined at discrete points in time.

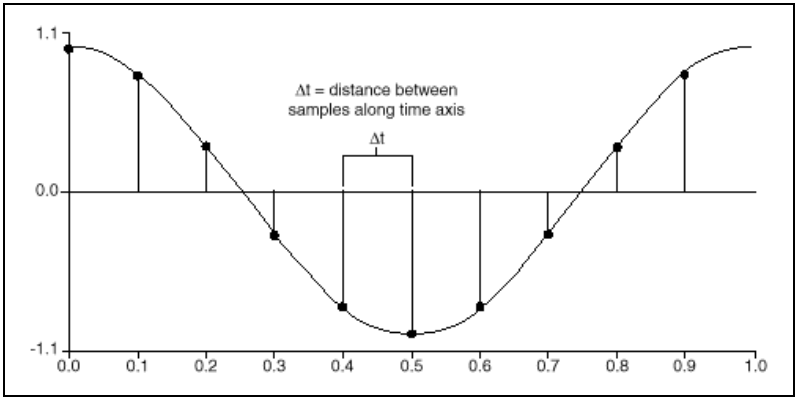


Figure 1-3. Analog Signal and Corresponding Sampled Version

The following notation represents the individual samples.

$$x[i] = x(i\Delta t)$$

for

$$i = 0, 1, 2, \ldots$$

If N samples are obtained from the signal $x(t)$, then you can represent $x(t)$ by the following sequence.

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

The preceding sequence representing $x(t)$ is known as the digital representation, or the sampled version, of $x(t)$. Notice that the sequence $X = \{x[i]\}$ is indexed on the integer variable i and does not contain any information about the sampling rate. So knowing only the values of the samples contained in X gives you no information about the sampling rate.

Sampling Considerations

A/D converters (ADCs) are an integral part of National Instruments DAQ boards. One of the most important parameters of an analog input system is the rate at which the DAQ device samples an incoming signal. The sampling rate determines how often an A/D conversion takes place. A fast sampling rate acquires more points in a given time and can, therefore, often form a better representation of the original signal than a slow sampling rate. Sampling too slowly might result in a poor representation of your analog signal. Figure 1-4 shows an adequately sampled signal, as well as the effects of undersampling. Undersampling

causes the signal to appear as if it has a different frequency than it actually does. The misrepresentation of a signal by the signal appearing to have a different frequency than it actually does is called an alias.

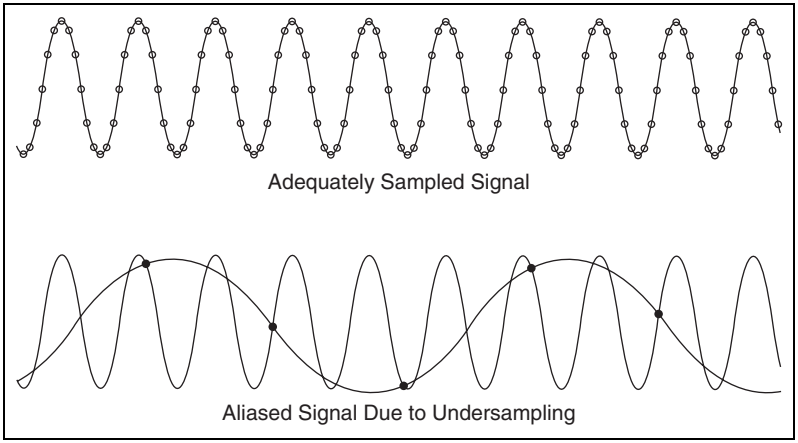


Figure 1-4. Aliasing Effects of an Improper Sampling Rate

According to Shannon's 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 you can accurately represent without aliasing is known as the Nyquist frequency. The Nyquist frequency equals one-half the sampling frequency. Signals with frequency components above the Nyquist frequency 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 1-5 and 1-6 illustrate the aliasing phenomenon. For example, assume the sampling frequency f_s is 100 Hz. Also, assume the input signal contains the following frequencies:

- F1 = 25 Hz
- F2 = 70 Hz
- F3 = 160 Hz
- F4 = 510 Hz

The frequencies contained in the input signal are shown in Figure 1-5.

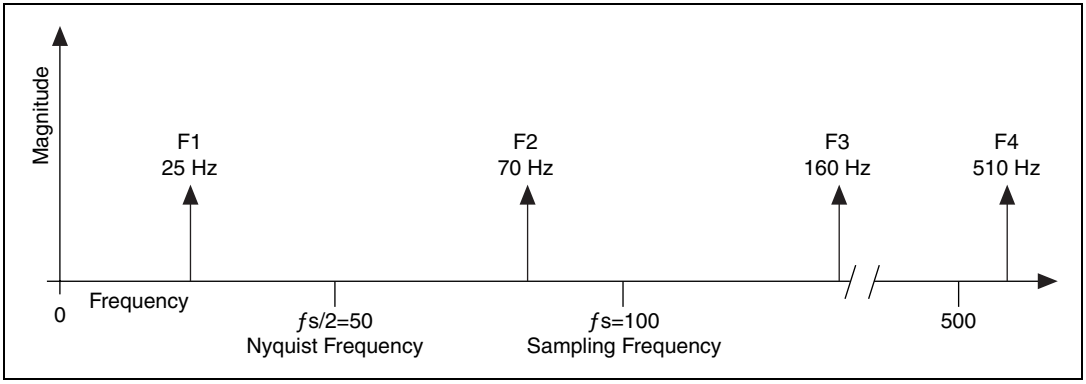


Figure 1-5. Actual Signal Frequency Components

In Figure 1-6, notice that frequencies below the Nyquist frequency of $f_s/2$ = 50 Hz are sampled correctly. Frequencies above the Nyquist frequency appear as aliases. For example, F1 appears at the correct frequency, but F2, F3, and F4 have aliases at 30 Hz, 40 Hz, and 10 Hz, respectively.

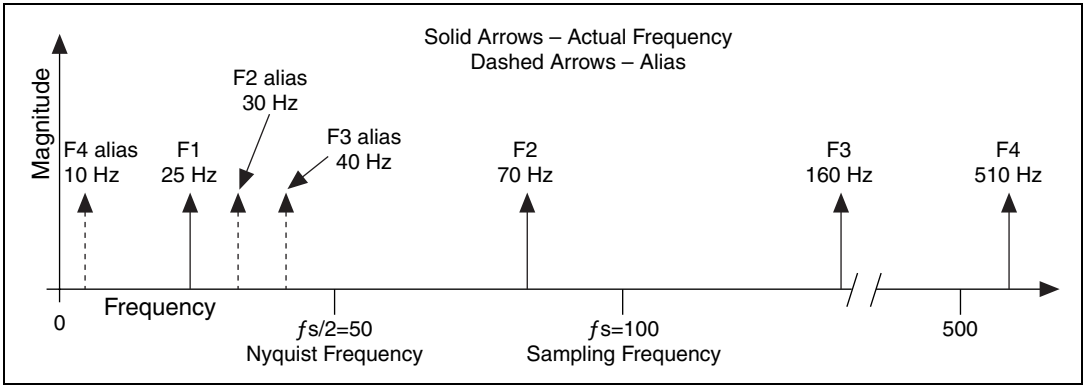


Figure 1-6. Signal Frequency Components and Aliases

The alias frequency equals the absolute value of the difference between the closest integer multiple of the sampling frequency and the input frequency, as shown in the following equation.

$$AF = |CIMSF - IF|$$

where *AF* is the alias frequency, *CIMSF* is the closest integer multiple of the sampling frequency, and *IF* is the input frequency. For example, you

can compute the alias frequencies for F2, F3, and F4 as shown in the following equations.

$$\text{Alias F2} = |100 - 70| = 30 \text{ Hz}$$
$$\text{Alias F3} = |(2)100 - 160| = 40 \text{ Hz}$$
$$\text{Alias F4} = |(5)100 - 510| = 10 \text{ Hz}$$

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 device. 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 1-7 shows the effects of various sampling rates. In case A, the sine wave of frequency f is sampled at the same frequency f_s (samples/sec) = f (cycles/sec), or at 1 sample per cycle. The reconstructed waveform appears as an alias at DC. As you increase the sampling to 7 samples/4 cycles, as in case b, the waveform increases in frequency, but aliases to a frequency less than the original signal (3 cycles instead of 4). The sampling rate in case B is $f_s = 7/4\ f$. If you increase the sampling rate to $f_s = 2f$, the digitized waveform has the correct frequency (same number of cycles), and can be reconstructed as the original sinusoidal wave, as shown in case C. For time-domain processing, it may be important to increase your sampling rate so that the samples more closely represent the original signal. By increasing the sampling rate to well above f, say to $f_s = 10f$, or 10 samples/cycle, you can accurately reproduce the waveform, as shown in case D.

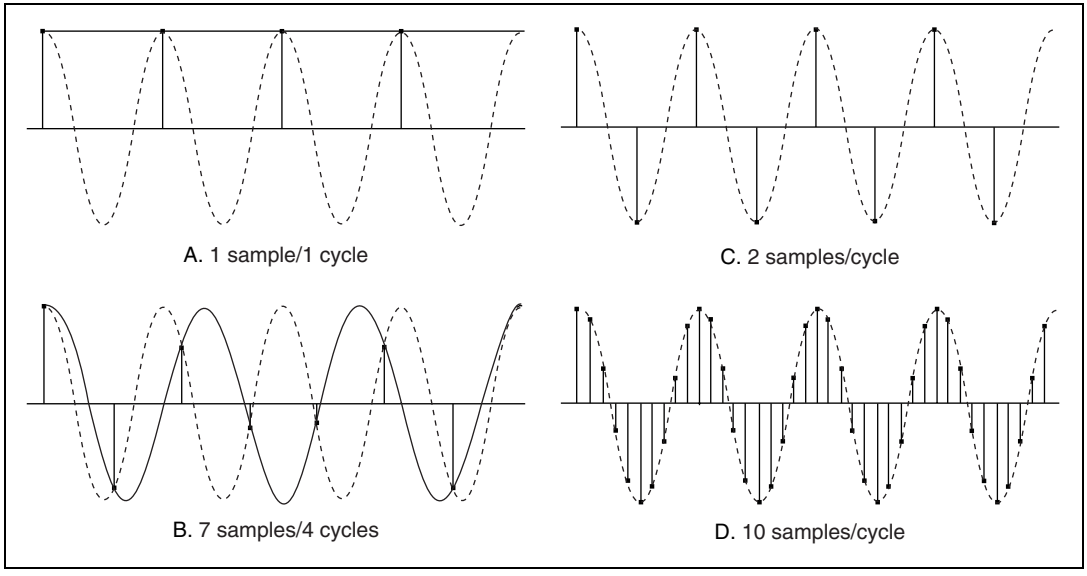


Figure 1-7. Effects of Sampling at Different Rates

Why Do You Need Anti-Aliasing 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 you 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 anti-alias filter because by attenuating the higher frequencies (greater than Nyquist), it prevents the aliasing components from being sampled. Because at this stage (before the sampler and the ADC) we are still in the analog world, the anti-aliasing filter is an analog filter.

An ideal anti-alias filter, shown in Figure 1-8 (a), passes all the desired input frequencies, the frequencies below f_1, and cuts off all the undesired frequencies, the frequencies above f_1. However, an ideal anti-alias filter is not physically realizable.

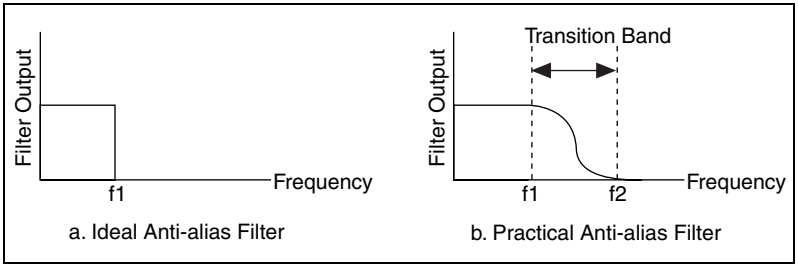


Figure 1-8. Ideal versus Practical Anti-Alias Filter

In practice, filters look as shown in Figure 1-8(b). Practical anti-alias filters 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 $< f_1$, the signals in the transition band could still cause aliasing. Therefore, in practice, you should use a sampling frequency greater than two times the highest frequency in the transition band. Because this sampling frequency turns out to be more than two times

the maximum input frequency (f_1), you might see that the sampling rate is more than twice the maximum input frequency.

Why Use Decibels?

On some instruments, you can display 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. The following example illustrates why this transformation is necessary.

Suppose that you want to display a signal with very large as well as very small amplitudes. Let us assume you have a display height of 10 cm and are utilizing 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 be barely visible on the display.

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

$$1\ \text{dB} = 10 \log_{10} (\text{Power Ratio}) = 20 \log_{10} (\text{Voltage Ratio})$$

Table 1-1 shows the relationship between the decibel and the power and voltage ratios.

Table 1-1. Decibels and Power and Voltage Ratio Relationship

dB	Power Ratio	Voltage Ratio
+40	10,000	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

Table 1-1. Decibels and Power and Voltage Ratio Relationship (Continued)

dB	Power Ratio	Voltage Ratio
–20	1/100	1/10
–40	1/10,000	1/100

Thus, you see that the dB scale is useful in compressing a wide range of amplitudes into a small set of numbers.

2

Frequency Analysis

This chapter describes the fundamentals of the discrete Fourier transform (DFT) and the fast Fourier transform (FFT) and how they are used in spectral analysis. Use the NI Example Finder, available by selecting **Help»Find Examples**, to find examples using the digital signal processing VIs and the measurement analysis VIs.

Frequency Domain versus Time Domain

The time-domain representation gives the amplitudes of the signal at the instants of time during which it was sampled. However, in many cases you want to know the frequency content of a signal rather than the amplitudes of the individual samples.

Fourier's theorem states that any waveform in the time domain can be represented by the weighted sum of sines and cosines. The same waveform can then be represented in the frequency domain as a pair of amplitude and phase values at each component frequency.

You can generate any waveform by adding up sine waves, each with a particular amplitude and phase. Figure 2-1 shows the original waveform, labeled *sum*, and its component frequencies. The fundamental frequency is shown at the frequency $f0$, the second harmonic at frequency $2f0$, and the third harmonic at frequency $3f0$.

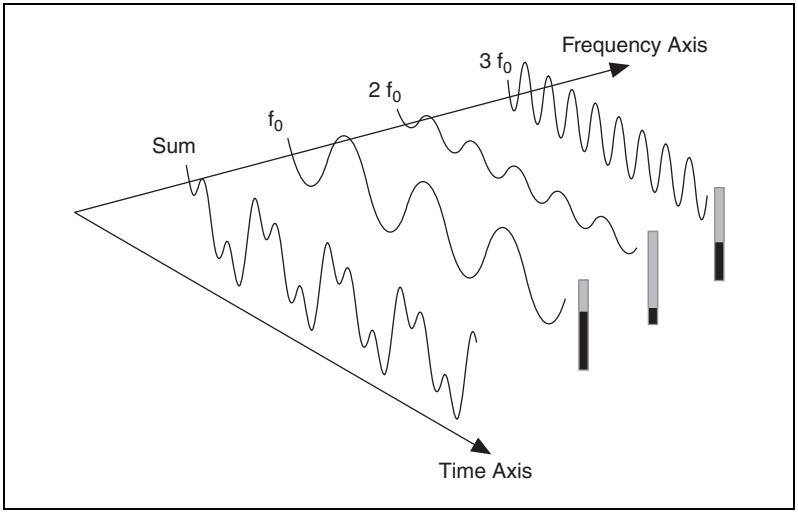


Figure 2-1. Signal Formed by Adding Three Frequency Components

In the frequency domain, you can conceptually separate the sine waves that add to form the complex time-domain signal. Figure 2-1 shows single frequency components, which spread out in the time domain, as distinct impulses in the frequency domain. The amplitude of each frequency line is the amplitude of that frequency component's time waveform. 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 samples of a signal obtained from a DAQ device constitute the time-domain representation of the signal. Some measurements, such as harmonic distortion, are very difficult to quantify by inspecting the time waveform on an oscilloscope. When the same signal is displayed in the frequency domain by an FFT Analyzer, also known as a Dynamic Signal Analyzer, you easily can measure the harmonic frequencies and amplitudes.

Discrete Fourier Transform (DFT)

The algorithm used to transform samples of the data from the time domain into the frequency domain is known as the discrete Fourier transform (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. Figure 2-2 illustrates using the DFT to transform data from the time domain into the frequency domain.

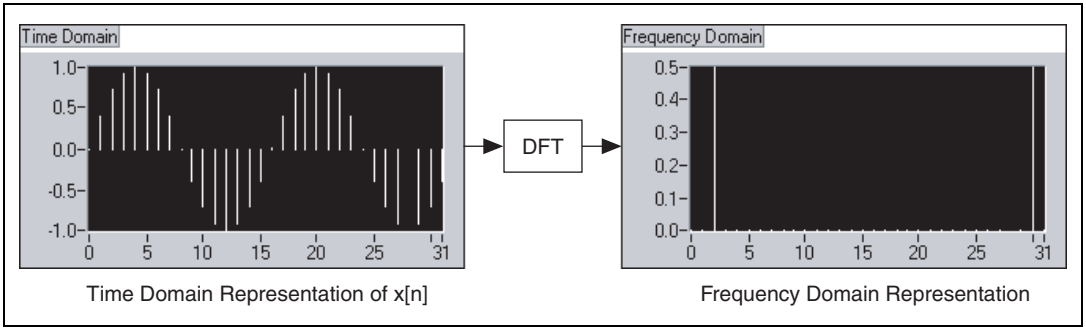


Figure 2-2. Discrete Fourier Transform

Suppose you have obtained *N* samples of a signal from a DAQ device. 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.

Relationship between *N* Samples in the Frequency and Time Domains

If a signal is sampled at a given sampling rate, the time interval between the samples, or sampling interval, is given by the following equation.

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

where Δt is the sampling interval and f_s is the sampling rate in samples per second (S/s).

The DFT is given by the following equation and results in $X[k]$, the frequency-domain representation of the sample signal.

$$X[k] = \sum_{i=0}^{N-1} x[i] e^{-j2\pi ik/N} \qquad \text{for } k = 0,1,2, \ldots, N-1, \qquad (2\text{-}1)$$

where $x[i]$ is the time-domain representation of the sample signal and N is the total number of samples. Notice 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, or frequency resolution, between the components of X in the frequency domain, which is given by the following equation.

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

where Δf is the frequency resolution, f_s is the sampling rate, N is the number of samples, Δt is the sampling interval, and $N\Delta t$ is the total acquisition time.

To improve the frequency resolution, that is, a smaller Δf, you must either increase N and keep f_s constant or decrease f_s and keep N constant. Both of the approaches used to improve the frequency resolution are equivalent to increasing $N\Delta t$, which is the time duration of the acquired samples.

DFT Calculation Example

This section provides an example of using Equation 2-1 to calculate the DFT for a DC signal. This example uses the following assumptions:

- $X[0]$ corresponds to the DC component, or the average value, of the signal.
- The DC signal has a constant amplitude of +1 V.
- The number of samples is four samples.
- Each of the samples has a value +1, as shown in Figure 2-3.
- The resulting time sequence for the four samples is given by the following equation.

$$x[0] = x[1] = x[3] = x[4] = 1$$

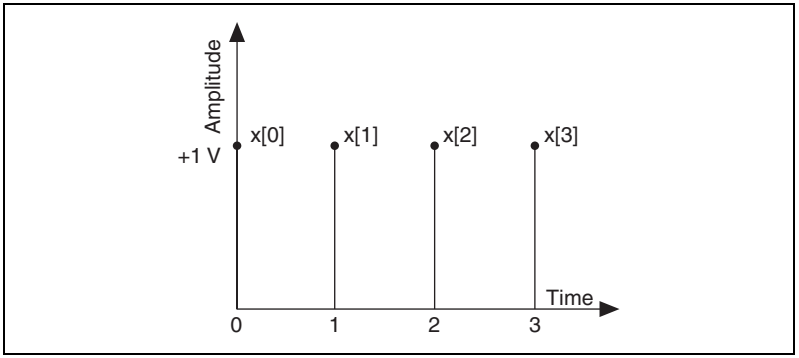


Figure 2-3. DFT Samples

The DFT calculation makes use of Euler's identity, which is given by the following equation.

$$\exp(-i\theta) = \cos(\theta) - j\sin(\theta)$$

Using Equation 2-1 to calculate the DFT of the sequence shown in Figure 2-3 and making use of Euler's identity, you get the following equations.

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

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

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

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

where $X[0]$ is the DC component and N is the number of samples.

Therefore, except for the DC component, all other values for the sequence shown in Figure 2-3 are zero, which is as expected. However, the calculated value of $X[0]$ depends on the value of N. Because in this example $N = 4$, $X[0] = 4$. If $N = 10$, the calculation results in $X[0] = 10$. This dependency of $X[\ ]$ on N also occurs for the other frequency components. Thus, you usually divide the DFT output by N 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 2-1, 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. In other words, every frequency component has a magnitude and phase.

Normally the magnitude of the spectrum is displayed. The magnitude is the square root of the sum of the squares of the real and imaginary parts.

The phase is relative to the start of the time record or relative to a single-cycle cosine wave starting at the beginning of the time record. Single-channel phase measurements are stable only if the input signal is triggered. Dual-channel phase measurements compute phase differences between channels so that if the channels are simultaneously sampled, triggering usually is not necessary.

The phase is the arctangent of the ratio of the imaginary and real parts and is usually between π and $-\pi$ radians, or 180 and –180 degrees.

For real signals ($x[i]$ real), such as those obtained from the output of one channel of a DAQ device, the DFT is symmetric with properties given by the following equations.

$$|X[k]| = |X[N-k]|$$

$$\text{phase}\,(X[k]) = -\text{phase}(X[N-k])$$

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 symmetric about the y-axis, whereas an odd symmetric signal is symmetric about the origin. Figure 2-4 illustrates even and odd symmetry.

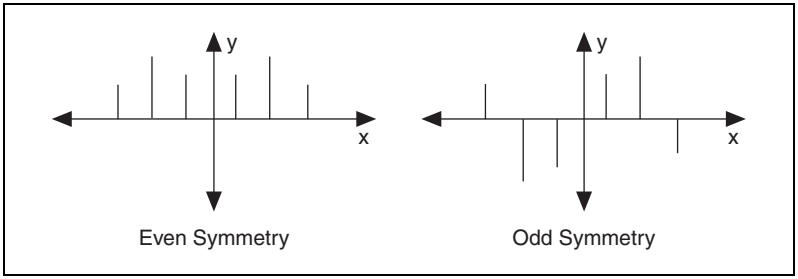


Figure 2-4. Signal Symmetry about the y-axis

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 because you can obtain the other half from this repetition. If the input signal is complex, the DFT is asymmetrical, and you cannot use only half of the samples to obtain the other half.

Frequency Spacing between DFT Samples

If the sampling interval is Δt seconds, and the first data sample ($k = 0$) is at 0 seconds, then the k^{th} data sample, where $k > 0$ and an integer, is at $k\Delta t$ seconds. Similarly, if the frequency resolution is Δf Hz, then the k^{th} sample of the DFT occurs at a frequency of $k\Delta f$ Hz. However, this is valid for only up to the first half of the frequency components. 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. For example, let $N = 8$ and p represent the index of the Nyquist frequency $p = N/2 = 4$. Table 2-1 shows the Δf to which each format element of the complex output sequence X corresponds.

Table 2-1. $X[p]$ for $N = 8$

$X[p]$	Δf
$X[0]$	DC
$X[1]$	Δf
$X[2]$	$2\Delta f$
$X[3]$	$3\Delta f$
$X[4]$	$4\Delta f$ (Nyquist frequency)

Table 2-1. $X[p]$ for $N = 8$ (Continued)

$X[p]$	Δf
$X[5]$	$-3\Delta f$
$X[6]$	$-2\Delta f$
$X[7]$	$-\Delta f$

The negative entries in the second column beyond the Nyquist frequency represent negative frequencies, that is, those elements with an index value > p.

For $N = 8$, $X[1]$ and $X[7]$ have the same magnitude; $X[2]$ and $X[6]$ have the same magnitude; and $X[3]$ and $X[5]$ have the same magnitude. The difference is that $X[1]$, $X[2]$, and $X[3]$ correspond to positive frequency components, while $X[5]$, $X[6]$, and $X[7]$ correspond to negative frequency components. $X[4]$ is at the Nyquist frequency.

Figure 2-5 illustrates the complex output sequence X for $N = 8$.

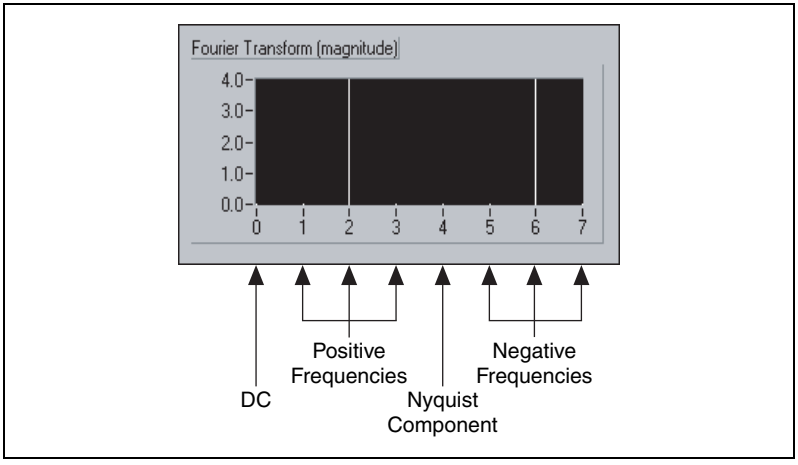


Figure 2-5. Complex Output Sequence X for $N = 8$

A representation where you see both the positive and negative frequencies is known as the two-sided transform.

When N is odd, there is no component at the Nyquist frequency. Table 2-2 lists the values of Δf for $X[p]$ when $N = 7$ and $p = (N-1)/2 = (7-1)/2 = 3$.

Table 2-2. $X[p]$ for $N = 7$

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

For $N = 7$, $X[1]$ 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, $X[1]$, $X[2]$, and $X[3]$ correspond to positive frequencies, while $X[4]$, $X[5]$, and $X[6]$ correspond to negative frequencies. Because N is odd, there is no component at the Nyquist frequency.

Figure 2-6 illustrates the complex output sequence $X[p]$ for $N = 7$.

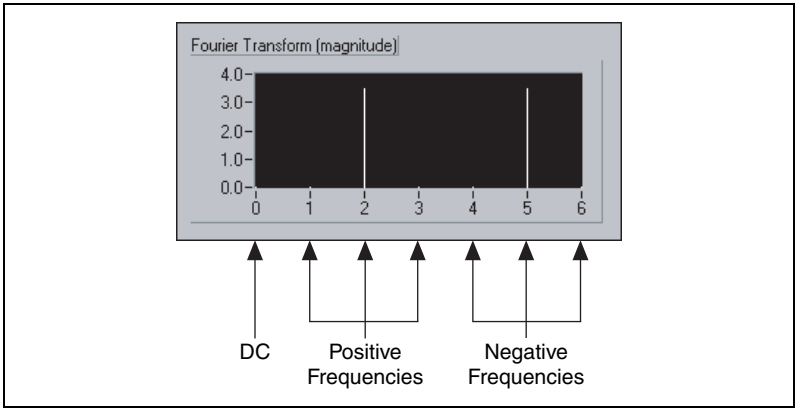


Figure 2-6. Complex Output Sequence $X[p]$ for $N = 7$

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

FFT Fundamentals

Direct implementation of the DFT on N data samples requires approximately N^2 complex operations and is a time-consuming process. The FFT is a fast algorithm for calculating the DFT when N is a power of two.

Computing Frequency Components

Each frequency component is the result of a dot product of the time domain signal with the complex exponential at that frequency and is given by the following equation.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\left(\frac{2\pi nk}{N}\right)} = \sum_{n=0}^{N-1} x(n)\left[\cos\left(\frac{2\pi nk}{N}\right) - j\sin\left(\frac{2\pi nk}{N}\right)\right]$$

The DC component is the dot product of $x(n)$ with $[\cos(0) - j\sin(0)]$, or with 1.0.

The first bin, or frequency component, is the dot product of $x(n)$ with $\cos(2\pi n/N) - j\sin(2\pi n/N)$. Here, $\cos(2\pi n/N)$ is a single cycle of the cosine wave, and $\sin(2\pi n/N)$ is a single cycle of a sine wave.

In general, bin k is the dot product of $x(n)$ with k cycles of the cosine wave for the real part of $X(k)$ and the sine wave for the imaginary part of $X(k)$.

The use of the FFT for frequency analysis implies two important relationships.

The first relationship links the highest frequency that can be analyzed to the sampling frequency and is given by the following equation.

$$F_{max} = \frac{f_s}{2},$$

where F_{max} is the highest frequency that can be analyzed and f_s is the sampling frequency. Refer to the *Aliasing* section of this chapter for more information about F_{max}.

The second relationship links the frequency resolution to the total acquisition time, which is related to the sampling frequency and the block size of the FFT and is given by the following equation.

$$\Delta f = \frac{1}{T} = \frac{f_s}{N},$$

where Δf is the frequency resolution, T is the acquisition time, f_s is the sampling frequency, and N is the block size of the FFT.

Fast FFT Sizes

When the size of the input sequence is a power of two, $N = 2^m$ for $m = 1, 2, 3, \ldots$, you can implement the computation of the DFT with approximately $N \log_2(N)$ operations, which makes the calculation of the DFT much faster. DSP literature refers to the algorithms for faster DFT calculation as fast Fourier transforms (FFTs). Examples of sequence sizes where you can use this algorithm are 512, 1,024, and 2,048.

The advantages of the FFT include speed and memory efficiency because the VI can compute the FFT in place. That is, no additional memory buffers are needed to compute the output. Conversely, the DFT can efficiently process any size sequence but is slower than the FFT and uses more memory because it must allocate additional buffers for storing intermediate results during processing.

In addition, another optimized algorithm is used for short DFTs of lengths 2, 3, 4, 5, 8, and 10. As a result, when the size of the sequence is not a power of two, but can be factored as

$$N = 2^m 3^k 5^j \qquad \text{for } m, k, j = 0, 1, 2, 3, \ldots,$$

the DFT can be computed with speeds comparable to the radix-2 FFT but requires more memory. You can use the optimized algorithm for sequence sizes such as 640, 480, 1,000, and 2,000.

When the sequence size cannot be factored into sizes that are in the set of short DFTs, a Chirp-Z implementation of the DFT is used. The Chirp-Z implementation is much faster than the direct evaluation of the DFT expression. The Chirp-Z algorithm uses more memory than the prime-factor algorithms because it must allocate additional buffers for storing intermediate results during processing.

Zero Padding

Zero padding is a technique employed to make the size of the input sequence equal to a power of two. In zero padding, you add zeros to the end of the input sequence so that the total number of samples is equal to the next higher power of two. 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, or 2^4, which is a power of two. Figure 2-7 illustrates padding 10 samples of a signal with zeros to make the total number of samples equal 16.

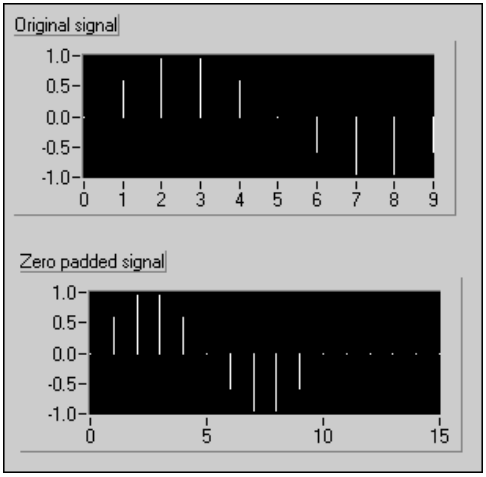


Figure 2-7. Zero Padding

The addition of zeros to the end of the time-domain waveform does not affect the spectrum of the signal. 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 improving the frequency resolution by increasing the number of samples, N. Recall that $\Delta f = f_s/N$. By increasing N, you decrease Δf, which improves the frequency resolution.

FFT VIs

The **Frequency Domain** palette contains two VIs that compute the FFT of a signal, the Real FFT VI and the Complex FFT VI.

The difference between the two VIs is that the Real FFT VI computes the FFT of a real-valued signal, whereas the Complex FFT VI 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. Therefore, you can use the Real FFT VI for most applications. Of course, you also could use the Complex FFT VI by setting the imaginary part of the signal to zero. An example of an application where you use the Complex FFT VI is when the signal consists of both a real and an imaginary component. A signal consisting of a real and an imaginary component 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 in Figure 2-8.

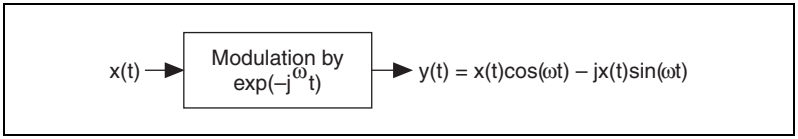


Figure 2-8. Modulation by a Complex Exponential

Power Spectrum

As discussed in the *Magnitude and Phase Information* section, the DFT or FFT of a real signal is a complex number, having a real and an imaginary part. You can obtain the power in each frequency component represented by the DFT or FFT by squaring the magnitude of that frequency component. Thus, the power in the k^{th} frequency component, that is, the k^{th} element of the DFT or FFT, is given by the following equation.

$$\text{power} = |X[k]|^2,$$

where $|X[k]|$ is the magnitude of the frequency component. Refer to the *Magnitude and Phase Information* section of this chapter for information about computing the magnitude of the frequency components.

The plot showing the power in each of the frequency components is known as the power spectrum. Because the DFT or 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 $-k\Delta f$, excluding DC and Nyquist components. The total power in the DC component is $|X[0]|^2$. The total power in the Nyquist component is $|X[N/2]|^2$.

Loss of Phase Information

Because the power is obtained by squaring the magnitude of the DFT or FFT, the power spectrum is always real. The disadvantage of obtaining the power by squaring the magnitude of the DFT or FFT is that the phase information is lost. If you want phase information, you must use the DFT or FFT, which gives you a complex output.

You can use the power spectrum in applications where phase information is not necessary, such as calculating 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.

Using the Power Spectrum VI

You can use the Power Spectrum VI to calculate the power spectrum of the time domain data samples. Just like the DFT and 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 given by the following equation.

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

Frequency Response Function

When analyzing two simultaneously sampled channels, you usually want to know the differences between the two channels rather than the properties of each.

In a typical dual-channel analyzer, as shown in Figure 2-9, the instantaneous spectrum is computed using a window function and the FFT for each channel. The averaged FFT spectrum, auto power spectrum, and cross power spectrum are computed and used in estimating the frequency response function. You also can use the coherence function to check the validity of the frequency response function.

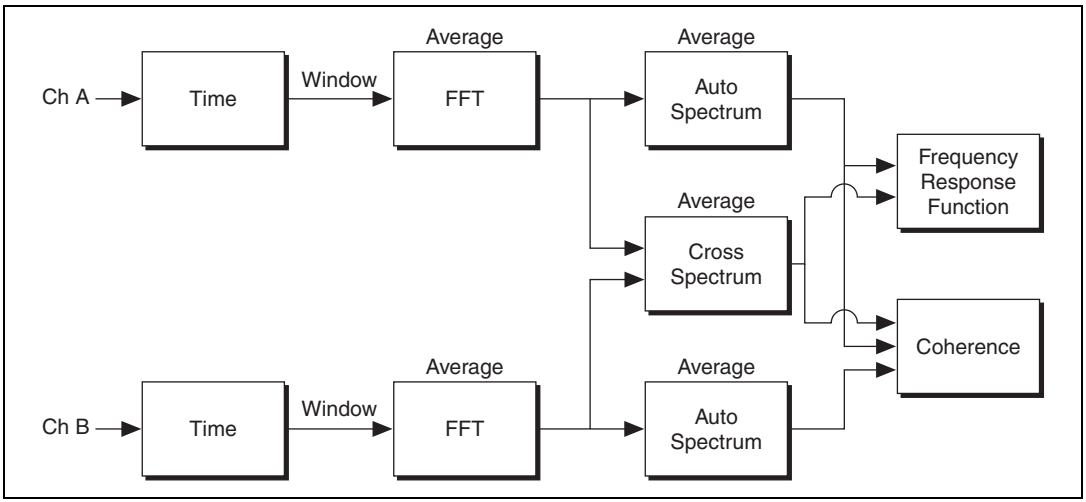


Figure 2-9. Dual-Channel Frequency Analysis

The frequency response of a system is described by the magnitude, |H|, and phase, ∠H, at each frequency. The gain of the system is the same as its magnitude and is the ratio of the output magnitude to the input magnitude at each frequency. The phase of the system is the difference of the output phase and input phase at each frequency.

Aliasing

According to Shannon's sampling theorem, the highest frequency you can analyze in an input sequence is the Nyquist frequency. The Nyquist frequency is given by the following equation.

$$f_N = \frac{f_s}{2},$$

where f_N is the Nyquist frequency and f_s is the sampling frequency.

Any analog frequency greater than f_N after sampling appears as a frequency between 0 and f_N. Such a frequency is known as an alias frequency. In the digital, or sampled, domain, there is no way to distinguish alias frequencies from the frequencies that actually lie between 0 and f_N. Therefore, alias frequencies need to be removed from the analog signal before sampling by the A/D converter.

In order to remove alias frequencies present at frequencies higher than the Nyquist frequency, you must use an analog lowpass filter. The anti-aliasing analog lowpass filter should exhibit a flat passband frequency response with a good high-frequency alias rejection and a fast roll-off in the transition band.

Refer to Chapter 1, *Introduction to Measurement Analysis in LabVIEW*, for more information about aliasing.

Windowing

In practical applications, you obtain only a finite number of samples of the signal. The FFT assumes that this time record repeats. If you have an integral number of cycles in your time record, the repetition is smooth at the boundaries. However, in practical applications, you usually have a nonintegral number of cycles. In the case of a nonintegral number of cycles, the repetition results in discontinuities at the boundaries. These artificial discontinuities were not originally present in your signal and result in a smearing or leakage of energy from your actual frequency to all other frequencies. This phenomenon is known as spectral leakage. The amount of leakage depends on the amplitude of the discontinuity, with a larger amplitude causing more leakage.

A signal that is exactly periodic in the time record is composed of sine waves with exact integral cycles within the time record. Such a perfectly periodic signal has a spectrum with energy contained in exact frequency bins.

A signal that is not periodic in the time record has a spectrum with energy split or spread across multiple frequency bins. The FFT spectrum models the time domain as if the time record repeated itself forever. It assumes that the analyzed record is just one period of an infinitely-repeating periodic signal.

Because the amount of leakage is dependent on the amplitude of the discontinuity at the boundaries, you can use windowing to reduce the size of the discontinuity and hence reduce spectral leakage. Windowing consists of multiplying the time-domain signal by another time-domain waveform, known as a window, whose amplitude tapers gradually and smoothly towards zero at edges. The result is a windowed signal with very small or no discontinuities, and therefore reduced spectral leakage. You can choose from among many different types of windows. The one you choose depends on your application and some prior knowledge of the signal you are analyzing.

Refer to Chapter 5, *Smoothing Windows*, for more information abut windowing.

Averaging to Improve the Measurement

Averaging successive measurements usually improves measurement accuracy. Averaging is usually performed on measurement results or on individual spectra, but not directly on the time record.

You can choose from among the following common averaging modes:

- RMS averaging
- Vector averaging
- Peak hold

RMS Averaging

RMS averaging reduces signal fluctuations but not the noise floor. The noise floor is not reduced because RMS averaging averages the energy, or power, of the signal. RMS averaging also causes averaged RMS quantities of single-channel measurements to have zero phase. RMS averaging for dual-channel measurements preserves important phase information. RMS averaged measurements are computed according to the following equations.

FFT spectrum $\sqrt{\langle X^* \bullet X \rangle}$

power spectrum $\langle X^* \bullet X \rangle$

cross spectrum $\langle X^* \bullet Y \rangle$

frequency response $H1 = \frac{\langle X^* \bullet Y \rangle}{\langle X^* \bullet X \rangle}$

$$H2 = \langle \frac{Y^* \bullet Y}{Y^* \bullet X} \rangle$$

$$H3 = \frac{(H1 + H2)}{2}$$

where X is the complex FFT of signal x (stimulus),

Y is the complex FFT of signal y (response),

X^* is the complex conjugate of X,

Y^* is the complex conjugate of Y,

$\langle X \rangle$ is the average of X, real and imaginary parts being averaged separately.

Vector Averaging

Vector averaging eliminates noise from synchronous signals. Vector averaging computes the average of complex quantities directly. The real part is averaged separately from the imaginary part. Averaging the real part separately from the imaginary part can reduce the noise floor for random signals because random signals are not phase-coherent from one time record to the next. The real and imaginary parts are averaged separately, reducing noise but usually requiring a trigger.

FFT spectrum	$\langle X \rangle$
power spectrum	$\langle X^* \rangle \bullet \langle X \rangle$
cross spectrum	$\langle X^* \rangle \bullet \langle Y \rangle$
frequency response	$\frac{\langle Y \rangle}{\langle X \rangle}$ $(H1 = H2 = H3)$

where X is the complex FFT of signal x (stimulus),

Y is the complex FFT of signal y (response),

X^* is the complex conjugate of X,

$\langle X \rangle$ is the average of X, real and imaginary parts being averaged separately.

Peak Hold

Peak hold averaging retains the peak levels of the averaged quantities. Peak hold is performed at each frequency line separately, retaining peak levels from one FFT record to the next.

FFT spectrum	$MAX\sqrt{(X^* \bullet X)}$
power spectrum	$MAX(X^* \bullet X)$

where X is the complex FFT of signal x (stimulus), and

X^* is the complex conjugate of X.

Weighting

When performing RMS or vector averaging, you can weight each new spectral record using either linear or exponential weighting.

Linear weighting combines N spectral records with equal weighting. When the number of averages is completed, the analyzer stops averaging and presents the averaged results.

Exponential weighting emphasizes new spectral data more than old and is a continuous process.

Weighting is applied according to the following equation.

$$Y_i = \frac{N-1}{N} Y_{i-1} + \frac{1}{N} X_i ,$$

where X_i is the result of the analysis performed on the i^{th} block,

Y_i is the result of the averaging process from X_1 to X_i,

$N = i$ for linear weighting,

N is a constant for exponential weighting ($N = 1$ for $i = 1$).

Summary

Frequency analysis is a general-purpose tool used for a wide variety of applications dealing with dynamic signals, including electrical and mechanical engineering, sound and vibration measurements, production testing, and biomedical applications.

You can convert the time-domain representation, or sample values, of a signal into the frequency-domain representation by means of an algorithm known as the discrete Fourier transform (DFT). To have fast calculation of the DFT, an algorithm known as the fast Fourier transform (FFT) is used. You can use the FFT when the number of signal samples is a power of two.

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.

The output of the conventional DFT or FFT is two-sided because it contains information about both the positive and the negative frequencies. You can

convert the two-sided output of the conventional DFT or FFT into a one-sided DFT or FFT by using only half the DFT or FFT output points. The frequency spacing between the samples of the DFT or FFT is given by the following equation.

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

You can calculate the power spectrum from the DFT or 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 units of the output of the Power Spectrum VI are in V^2_{rms}. However, the power spectrum does not provide any phase information.

3

Signal Generation

The generation of signals is an important part of any test or measurement system. Common test signals include the sine wave, the square wave, the triangle wave, the sawtooth wave, several types of noise waveforms, and multitone signals consisting of a superposition of sine waves. This chapter discusses some of the fundamentals of signal generation.

The following applications are examples of uses for signal generation:

- Simulate signals to test your algorithm when real-world signals are not available, for example, when you do not have a DAQ device for obtaining real-world signals, or when access to real-world signals is not possible.
- Generate signals to apply to a digital-to-analog (D/A) converter.

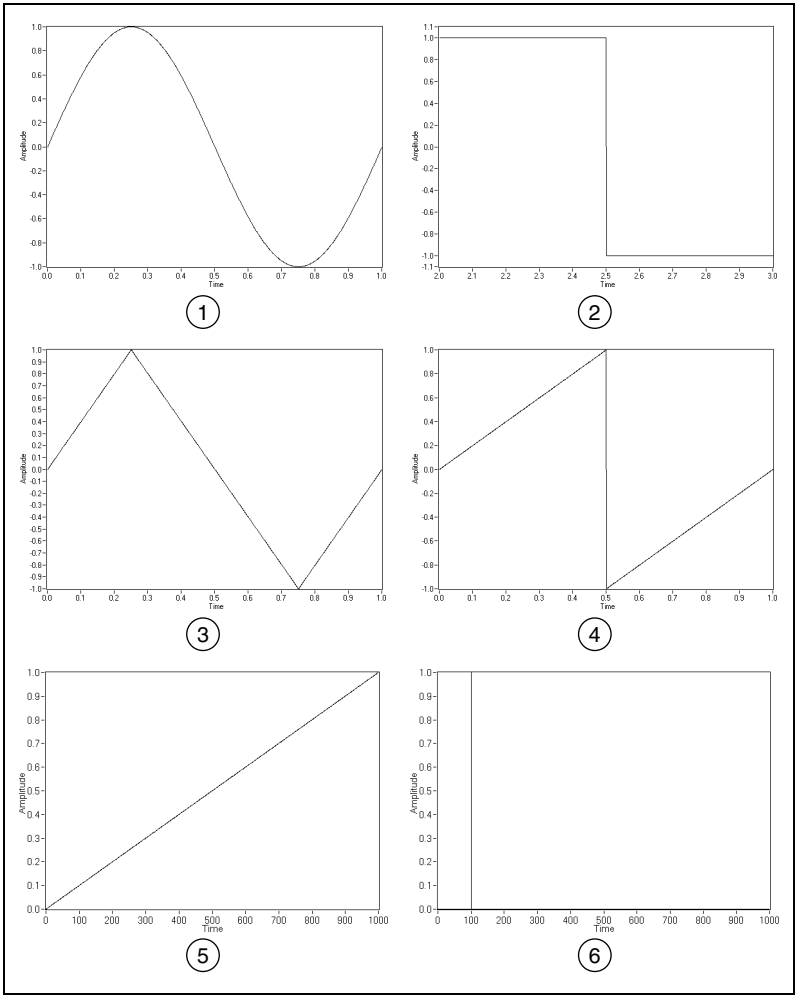
Common Test Signals

The most common signal for audio testing is the sine wave. A single sine wave is often used to determine the amount of harmonic distortion introduced by a system. Multiple sine waves are widely used to measure the intermodulation distortion or to determine the frequency response. The following table lists the signals used for some typical measurements.

Table 3-1. Typical Measurements and Signals

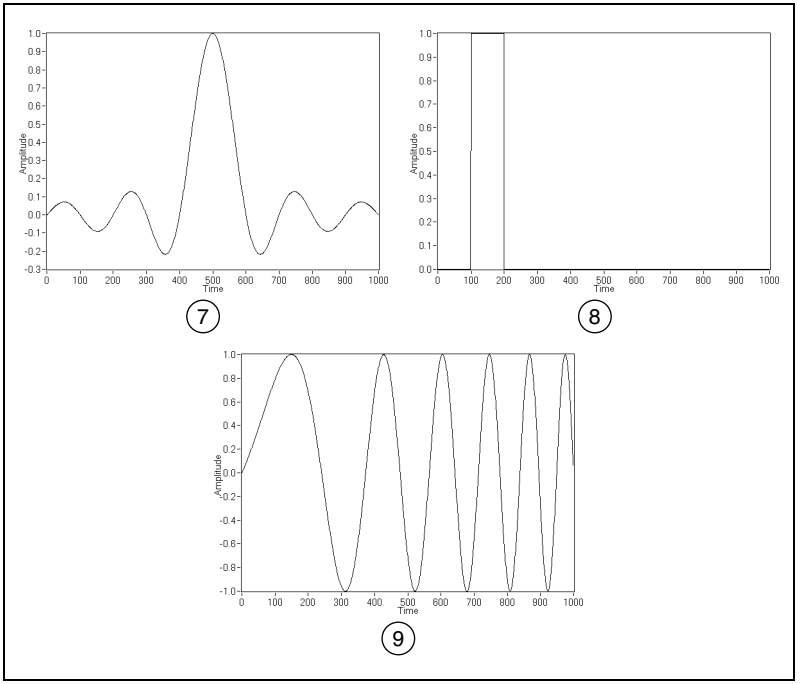
Measurement	Signal
Total Harmonic Distortion	Sine wave
Intermodulation Distortion	Multitone (two sine waves)
Frequency Response	Multitone (many sine waves, Impulse, Chirp)
Interpolation	Sinc
Rise Time, Fall Time, Overshoot, Undershoot	Pulse
Jitter	Square wave

These signals form the basis for many tests and are used to measure the response of a system to a particular stimulus. Some of the common test signals available in most signal generators are shown in Figure 3-1 and Figure 3-2.



1	Sine Wave	4	Sawtooth Wave
2	Square Wave	5	Ramp
3	Triangle Wave	6	Impulse

Figure 3-1. Common Test Signals



7 Sinc 8 Pulse 9 Chirp

Figure 3-2. More Common Test Signals

The most useful way to view these signals is in terms of their frequency content. For example, a sine wave has a single frequency component. A square wave consists of the superposition of many sine waves at odd harmonics of the fundamental frequency. The amplitude of each harmonic is inversely proportional to its frequency. Similarly, the triangle and sawtooth waves also have harmonic components that are multiples of the fundamental frequency. An impulse contains all frequencies that can be represented for a given sampling rate and number of samples. Chirp patterns have discrete frequencies that lie within a certain range. The discrete frequencies of chirp patterns depend on the sampling rate, the start and end frequencies, and the number of samples.

Multitone Generation

The common test signals, except for the sine wave, do not allow full control over their spectral content. For example, the harmonic components of a square wave are fixed in frequency, phase, and amplitude relative to the fundamental. On the other hand, you can generate multitone signals with a specific amplitude and phase for each individual frequency component.

A multitone signal is the superposition of several sine waves or tones, each with a distinct amplitude, phase, and frequency. A multitone signal is typically created so that an integer number of cycles of each individual tone are contained in the signal. If an FFT of the entire multitone signal is computed, then each of the tones falls exactly onto a single frequency bin, which means no spectral spread or leakage occurs.

Multitone signals are a part of many test specifications and allow the fast and efficient stimulus of a system across an arbitrary band of frequencies. Multitone test signals are used to determine the frequency response of a device, and with appropriate selection of frequencies, also can be used to measure such quantities as intermodulation distortion.

Crest Factor

The relative phases of the constituent tones with respect to each other determines the crest factor of a multitone signal with specified amplitude. The crest factor is defined as the ratio of the peak magnitude to the RMS value of the signal. For example, a sine wave has a crest factor of 1.414:1.

For the same maximum amplitude, a multitone signal with a large crest factor contains less energy than one with a smaller crest factor. Another way to express this is to say that a large crest factor means that the amplitude of a given component sine tone is lower than the same sine tone in a multitone signal with a smaller crest factor. A higher crest factor results in individual sine tones with lower signal-to-noise ratios. Therefore, proper selection of phases is critical to generating a useful multitone signal.

To avoid clipping, the maximum value of the multitone signal should not exceed the maximum capability of the hardware that generates the signal which means a limit is placed on the maximum amplitude of the signal. You can generate a multitone signal with a specific amplitude by different combinations of the phase relationships and amplitudes of the constituent sine tones. A good approach to generating a signal is to choose amplitudes and phases that result in a lower crest factor.

Phase Generation

The following schemes are used to generate tone phases of multitone signals:

- Varying the phase difference between adjacent frequency tones linearly from 0 to 360 degrees
- Varying the tone phases randomly

Varying the phase difference between adjacent frequency tones linearly from 0 to 360 degrees allows the creation of multitone signals with very low crest factors. However, the resulting multitone signals possess the following potentially undesirable characteristics:

- The multitone signal is very sensitive to phase distortion. If in the course of generating the multitone signal the hardware or signal path induces non-linear phase distortion, the crest factor can vary considerably.
- The multitone signal might display some repetitive time-domain characteristics that possibly are undesirable, as shown in the multitone signal in Figure 3-3.

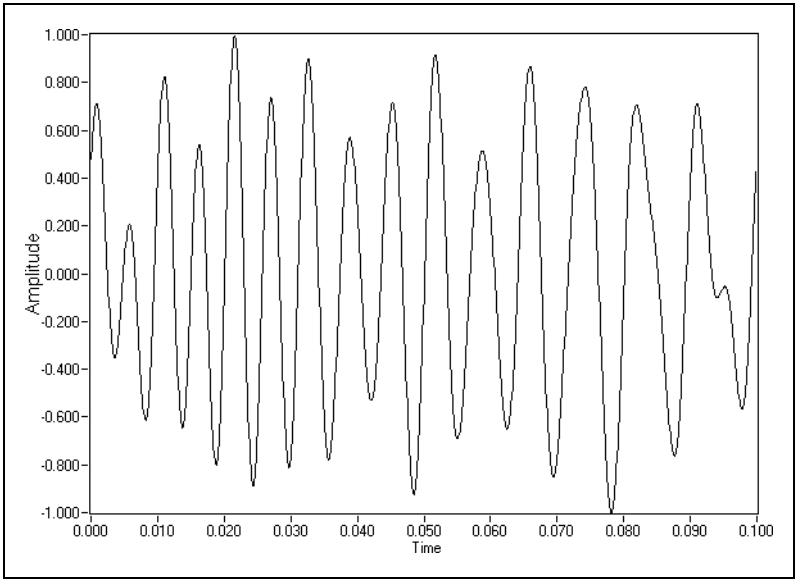


Figure 3-3. Multitone Signal with Linearly Varying Phase Difference between Adjacent Tones

Notice that the signal in Figure 3-3 resembles a chirp signal in that its frequency appears to decrease from left to right. The apparent decrease in

frequency from left to right is characteristic of multitone signals generated by linearly varying the phase difference between adjacent frequency tones. Having a signal that is more noise-like than the signal in Figure 3-3 is often more desirable.

Varying the tone phases randomly results in a multitone signal whose amplitudes are nearly Gaussian in distribution as the number of tones increases. Figure 3-4 illustrates a signal created by varying the tone phases randomly.

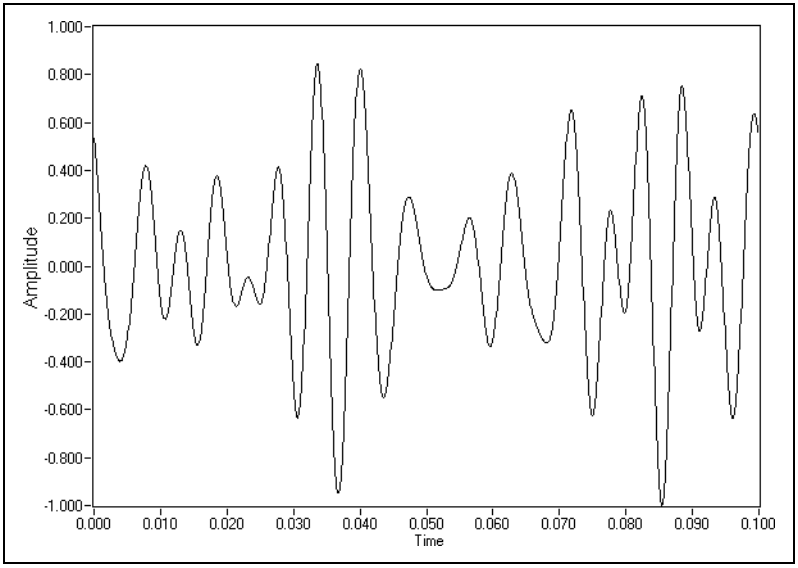


Figure 3-4. Multitone Signal with Random Phase Difference between Adjacent Tones

In addition to being more noise-like, the signal in Figure 3-4 is also much less sensitive to phase distortion. Multitone signals with the sort of phase relationship shown in Figure 3-4 generally achieve a crest factor between 10 and 11 dB.

Swept Sine versus Multitone

To characterize a system, you often must measure the response of the system at many different frequencies. You can use the following methods to measure the response of a system at many different frequencies:

- Swept sine continuously and smoothly changes the frequency of a sine wave across a range of frequencies.
- Stepped sine provides a single sine tone of fixed frequency as the stimulus for a certain time and then increments the frequency by a discrete amount. The process is continued until all the frequencies of interest have been reached.
- Multitone provides a signal composed of multiple sine tones.

A multitone signal has significant advantages over the swept sine and stepped sine approaches. For a given range of frequencies, the multitone approach can be much faster than the equivalent swept sine measurement, due mainly to settling time issues. For each sine tone in a stepped sine measurement, you must wait for the settling time of the system to end before starting the measurement.

The settling time issue for a swept sine can be even more complex. If the system has low-frequency poles and/or zeroes or high Q-resonances, the system might take a relatively long time to settle. For a multitone signal, you must wait only once for the settling time. A multitone signal containing one period of the lowest frequency, actually one period of the highest frequency resolution, is enough for the settling time. Once the response to the multitone signal is acquired, the processing can be very fast. You can use a single FFT to measure many frequency points, amplitude and phase, simultaneously.

The swept sine approach is more appropriate than the multitone approach in certain situations. Each measured tone within a multitone signal is more sensitive to noise because the energy of each tone is lower than that in a single pure tone. For example, consider a single sine tone of amplitude 10 V peak and frequency 100 Hz. A multitone signal containing 10 tones, including the 100 Hz tone, might have a maximum amplitude of 10 V. However, the 100 Hz tone component has an amplitude somewhat less than 10 V. The lower amplitude of the 100 Hz tone component is due to the way that all the sine tones sum. Assuming the same level of noise, the signal-to-noise ratio (SNR) of the 100 Hz component is better for the case of the swept sine approach. In the multitone approach, you can mitigate the reduced SNR by adjusting the amplitudes and phases of the tones, applying higher energy where needed, and applying lower energy at less critical frequencies.

When viewing the response of a system to a multitone stimulus, any energy between FFT bins is due to noise or unit-under-test (UUT) induced distortion. The frequency resolution of the FFT is limited by your measurement time. If you only want to measure your system at 1.000 kHz and 1.001 kHz, using two independent sine tones is the best approach. Using two independent sine tones, you can perform the measurement in a few milliseconds, while a multitone measurement requires at least one second. The difference in measurement speed is because you must wait long enough to obtain the required number of samples to achieve a frequency resolution of 1 Hz. Some applications, such as finding the resonant frequency of a crystal, combine a multitone measurement for coarse measurement and a narrow-range sweep for fine measurement.

Noise Generation

You can use noise signals to perform frequency response measurements, or to simulate certain processes. Several types of noise are typically used, namely Uniform White Noise, Gaussian White Noise, and Periodic Random Noise.

The term white in the definition of noise refers to the frequency domain characteristic of noise. Ideal white noise has equal power per unit bandwidth, resulting in a flat power spectral density across the frequency range of interest. Thus, the power in the frequency range from 100 Hz to 110 Hz is the same as the power in the frequency range from 1,000 Hz to 1,010 Hz. In practical measurements, achieving the flat power spectral density requires an infinite number of samples. Thus, when making measurements of white noise, the power spectra are usually averaged, with more number of averages resulting in a flatter power spectrum.

The terms uniform and Gaussian refer to the probability density function (PDF) of the amplitudes of the time-domain samples of the noise. For uniform white noise, the PDF of the amplitudes of the time domain samples is uniform within the specified maximum and minimum levels. Another way to state this is to say that all amplitude values between some limits are equally likely or probable. Thermal noise produced in active components tends to be uniform white in distribution. Figure 3-5 shows the distribution of the samples of uniform white noise.

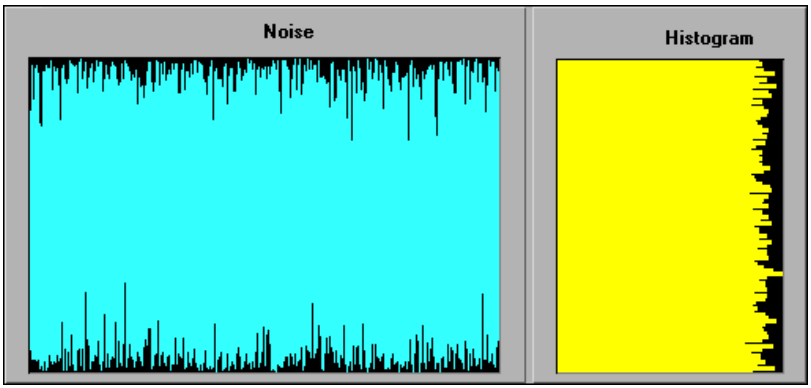


Figure 3-5. Uniform White Noise

For Gaussian white noise, the PDF of the amplitudes of the time domain samples is Gaussian. If uniform white noise is passed through a linear system, the resulting output is Gaussian white noise. Figure 3-6 shows the distribution of the samples of Gaussian white noise.

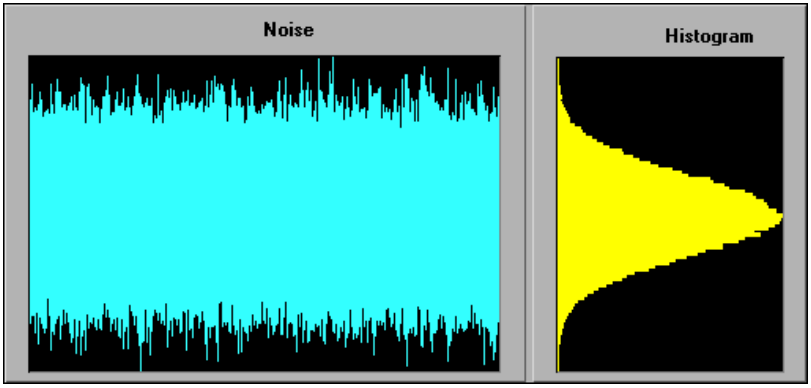


Figure 3-6. Gaussian White Noise

Periodic random noise (PRN) is a summation of sinusoidal signals with the same amplitudes but with random phases. PRN consists of all sine waves with frequencies that can be represented with an integral number of cycles in the requested number of samples. Since PRN contains only integral-cycle sinusoids, you do not need to window PRN before performing spectral analysis because PRN is self-windowing and therefore has no spectral leakage.

PRN does not have energy at all frequencies, as white noise does, but only at discrete frequencies which correspond to harmonics of a fundamental

frequency. The fundamental frequency is equal to the sampling frequency divided by the number of samples. However, the level of noise at each of the discrete frequencies is the same.

You can use PRN to compute the frequency response of a linear system with one time record instead of averaging the frequency response over several time records, as you must for nonperiodic random noise sources. Figure 3-7 shows the spectrum of periodic random noise and the averaged spectra of white noise.

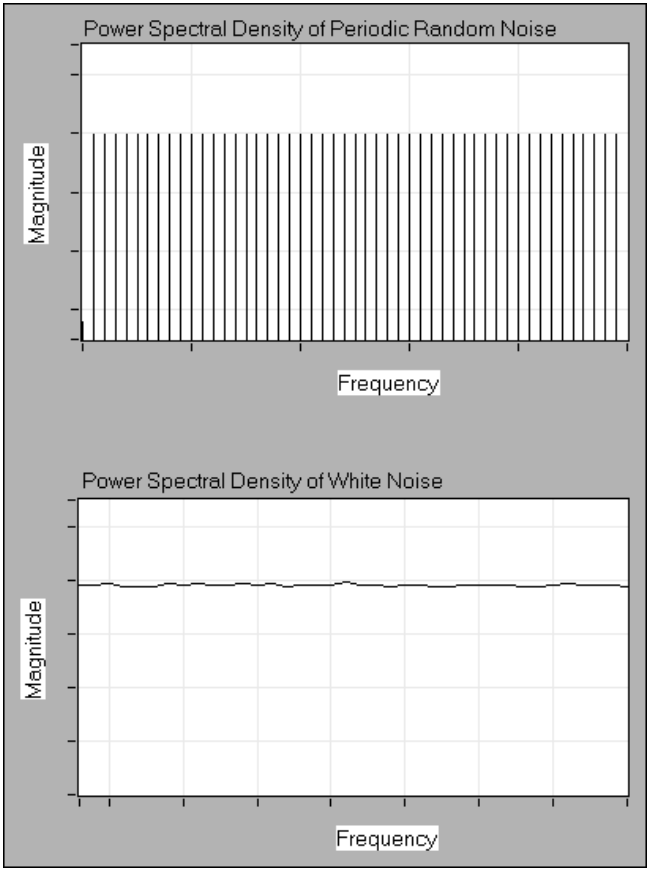


Figure 3-7. Spectral Representation of Periodic Random Noise and Averaged White Noise

Normalized Frequency

In the analog world, a signal frequency is measured in hertz (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, as shown by the following equation.

$$digital\ frequency = \frac{analog\ frequency}{sampling\ frequency}$$

The *digital frequency* is known as the normalized frequency and is measured in cycles per 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. The normalized frequency ranges from 0.0 to 1.0, which corresponds to a real frequency range of 0 to the sampling frequency f_s. The normalized frequency also wraps around 1.0, so that a normalized frequency of 1.1 is equivalent to 0.1. For example, a signal sampled at the Nyquist rate of $f_s/2$ means it is sampled twice per cycle, that is, two samples/cycle. This sampling rate corresponds 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 the signal is sampled in one cycle, that is, the number of samples per 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 per sample. You must use normalized units of cycles per sample with the following signal generation 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 per sample by dividing cycles by the number of samples generated.

You need only divide the frequency in cycles by the number of samples. For example, a 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 per second. If you need to convert from Hz to cycles per sample, divide your frequency in Hz by the sampling rate given in samples per second, as shown in the following equation.

$$\frac{cycles\ per\ second}{samples\ per\ second} = \frac{cycles}{sample}$$

For example, you divide a frequency of 60 Hz by a sampling rate of 1,000 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 also can use the signal generation VIs in conjunction with National Instruments hardware to generate analog output signals.

Wave and Pattern VIs

The names of most of the signal generation VIs contain the word *wave* or *pattern*. A basic difference exists between the operation of the two different types of VIs. The difference has to do with whether the VI can keep track of the phase of the signal it generates each time the VI 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. The wave VIs 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 specifies whether the phase of the first sample generated when the wave VI is called is the phase specified in the **phase in** control or 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**. A FALSE value of **reset phase** sets the initial phase to the value of **phase out** when the VI last executed.

All of the wave VIs are reentrant, which means they can keep track of phase internally. The Wave VIs accept frequency in normalized units of cycles per 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.

4

Digital Filtering

This chapter introduces the concept of filtering, compares analog and digital filters, discusses Finite Infinite Response (FIR) and Infinite Impulse Response (IIR) filters, and helps you determine how to choose the most appropriate digital filter.

What Is Filtering?

Filtering is the process by which the frequency content of a signal is altered. The implicit assumption is that the signal content of interest is separable from the raw signal. Classical linear filtering assumes that the signal content of interest is distinct from the remainder of the signal in the frequency domain (Fourier Transform). Filtering is one of the most commonly used signal processing techniques. For example, consider 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 filtering the audio signal. Removing noise and performing decimation (lowpass filtering the signal and reducing the sample rate) are other filtering applications.

Advantages of Digital Filtering Over Analog Filtering

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 have an infinite number of values. Analog filter design is about 50 years older than digital filter design. 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, such as audio, telecommunications, geophysics, and medical monitoring.

Some advantages of digital filters over analog filters include the following:

- Digital filters are software programmable and therefore are easy to build and test.
- Digital filters require only the arithmetic operations of multiplication and addition/subtraction and therefore are easier to implement.
- Digital filters do not drift with temperature or humidity or require precision components.
- Digital filters have a superior performance-to-cost ratio.
- Digital filters do not suffer from manufacturing variations or aging.

Common Digital Filters

Digital filters can be classified in many ways. The traditional approach is to first classify a filter based on the values upon which it operates. The simplest filters are those that operate on input values only. These filters are called Moving Average (MA) filters or Finite Impulse Response (FIR) filters. These filters perform a convolution of the filter coefficients with a sequence of input values, producing an equally numbered sequence of output values. The term FIR is used because if a single impulse is present at the input of the filter and all subsequent inputs are zero, then the output of the filter becomes zero after some finite time, equal to the number of filter coefficients.

If a filter operates on current and previous input values and current and previous output values, then the filter is termed Infinite Impulse Response (IIR) or Auto Regressive Moving Average (ARMA). The impulse response of such a filter is infinite in the sense that the response of the filter to an impulse never goes to zero.

Each type of filter has advantages and disadvantages. Filter design, as with all other engineering practices, involves tradeoffs. FIR filters are simple, and can be designed to provide a linear phase response or constant group delay. IIR filters can achieve the same level of attenuation as FIR filters with far fewer coefficients. This means that the IIR filter can be significantly faster and more efficient.

Ideal Filters

Filters alter or remove unwanted frequencies. Depending on the frequency range that they either pass or attenuate (reject), 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 in Figure 4-1.

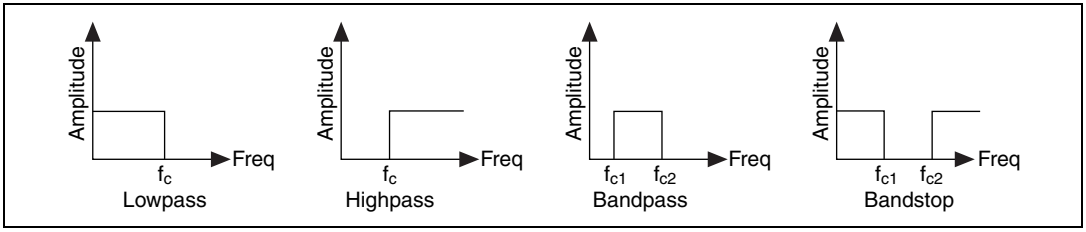


Figure 4-1. Ideal Frequency Response

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 cut-off 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 attenuated. The passband and the stopband for the different types of filters are shown in Figure 4-2.

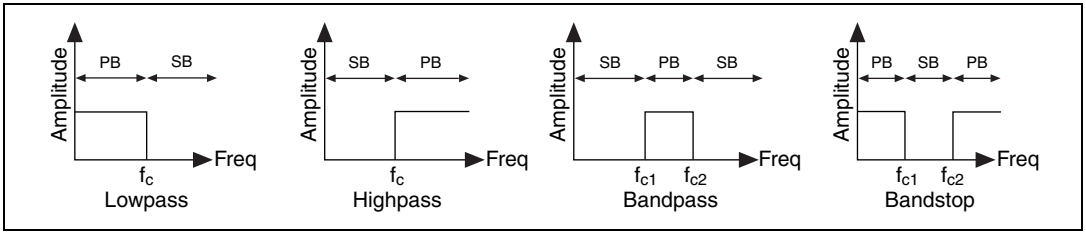


Figure 4-2. Passband and Stopband

Notice that while the lowpass and highpass filters have one passband and one stopband, the bandpass filter has one passband and two stopbands, and the bandstop filter has two passbands and one stopband.

Practical (Nonideal) Filters

Ideally, a filter should have a unit gain (0 dB) in the passband, and a gain of zero (–infinity 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 one (0 dB) in the passband to zero (–infinity dB) in the stopband.

Transition Band

The following diagrams show the passband, the stopband, and the transition region (TR) for the different types of nonideal filters. In each plot in Figure 4-3, the x-axis represents frequency, and the y-axis represents the magnitude of the filter in dB. Notice that the passband is now the region where the frequency range within which the gain of the filter varies from 0 dB to –3 dB.

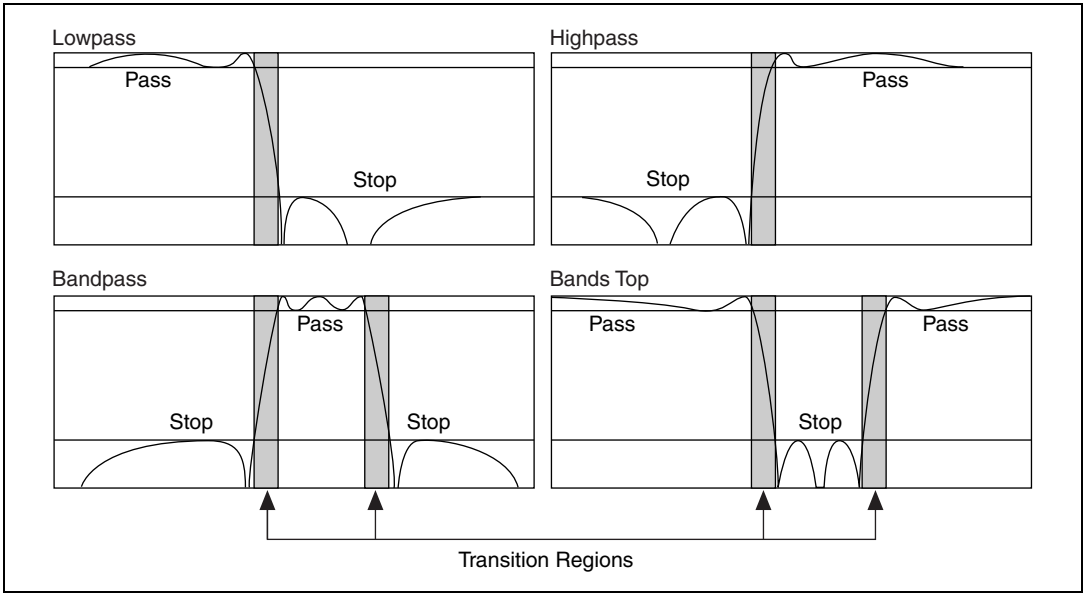


Figure 4-3. Nonideal Filters

Passband Ripple and Stopband Attenuation

In many applications, you can 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 (dB), defined by the equation:

$$\mathrm{dB} = 20 \log\left(\frac{A_o(f)}{A_i(f)}\right)$$

where log denotes the base 10 logarithm, and $A_i(f)$ and $A_o(f)$ are the amplitudes at a particular frequency f before and after the filtering, respectively.

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

$$-0.02 = 20 \log\left(\frac{A_o(f)}{A_i(f)}\right)$$

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

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

You can view practical filter design as approximating the ideal desired magnitude response subject to certain constraints. The ideal passband and stopband are flat and constant. Practical filter passbands and stopbands may have ripples. Ideal filters have no transition region. Practical filters have transition regions. Practical filter design allows tradeoffs between these different components (passband ripple, stopband ripple, stopband attenuation, transition region width) subject to the filter structure (FIR or IIR) and the design algorithm.

FIR Filters

FIR filters have several different design methods. FIR filters have ripple in the magnitude response, so the design problem can be restated as how you can design a filter that has a magnitude response as close to the ideal as possible and distributes the ripple in a desired fashion. For example, a lowpass filter has an ideal characteristic magnitude response. A particular application may allow some ripple in the passband and more ripple in the stopband. The filter design algorithm should balance the relative ripple requirements while producing the sharpest transition region.

The simplest approach is the Windowed FIR design. The Windowed FIR design takes the inverse FFT of the desired magnitude response and applies a time domain window to the result. The advantages of this method are conceptual simplicity and ease of implementation. The disadvantages are the inefficiency and difficulty in specification. For a given number of taps, the Windowed FIR design does not distribute ripple equally and has a wider transition band than other designs. It also is difficult to specify a cut-off frequency that has a particular attenuation. To design a Windowed FIR filter, you must specify the ideal cut-off frequency, the sampling frequency, the number of taps, and the window type.

The other main FIR design approach uses the Parks-McClellan algorithm, also known as Remez Exchange. This is an iterative algorithm that produces filters with a magnitude response for which the weighted ripple is evenly distributed over the passband and stopband and that have a sharp transition region. The advantage of this approach is the optimal response of the designed filter. The disadvantages are the complexity and length of time required to design. Park-McClellan design time is much longer than the Windowed approach. A specialization of the Parks-McClellan approach is equiripple FIR design. The only difference between them is the equiripple design weights the passband and stopband ripple equally. To design an FIR filter using the equiripple approach, you must specify the cut-off frequency, the number of taps, the filter type, and pass and stop frequencies. The cut-off frequency for equiripple designs specifies the edge of the passband and/or the stopband. Equiripple filters have a ripple in the passband that causes the magnitude response in the passband to be greater than or equal to 1. Similarly, the magnitude response in the stopband is always less than or equal to the stopband attenuation. For example, if you specify a lowpass filter, the passband cut-off frequency is the highest or largest frequency for which the passband conditions hold true. Similarly, the stopband cut-off is the lowest frequency for which the stopband conditions are met. Both design approaches deliver FIR filters with a linear phase characteristic.

When you use conventional techniques to design FIR filters with especially narrow bandwidths, the resulting filter lengths can be very long. FIR filters with long filter lengths often require lengthy design and implementation times and are more susceptible to numerical inaccuracy. In some cases, conventional filter design techniques, such as the Parks-McClellan algorithm, might fail the design altogether.

IIR Filters

IIR filters are filters that may or may not have ripple in the passband and/or the stopband. Digital IIR filter design derives from the classical analog designs. These designs are Butterworth, Chebyshev, Chebyshev II, Elliptic, and Bessel.

Butterworth Filters

A smooth response at all frequencies and a monotonic decrease from the specified cut-off frequencies characterize the frequency response of Butterworth filters. Butterworth filters are maximally flat, the ideal response of unity in the passband and zero in the stopband. The half power frequency or the 3 dB down frequency corresponds to the specified cut-off frequencies.

Figure 4-4 shows the response of a lowpass Butterworth filter. The advantage of Butterworth filters is a smooth, monotonically decreasing frequency response. After you set the cut-off frequency, LabVIEW sets the steepness of the transition proportional to the filter order. Higher-order Butterworth filters approach the ideal lowpass filter response.

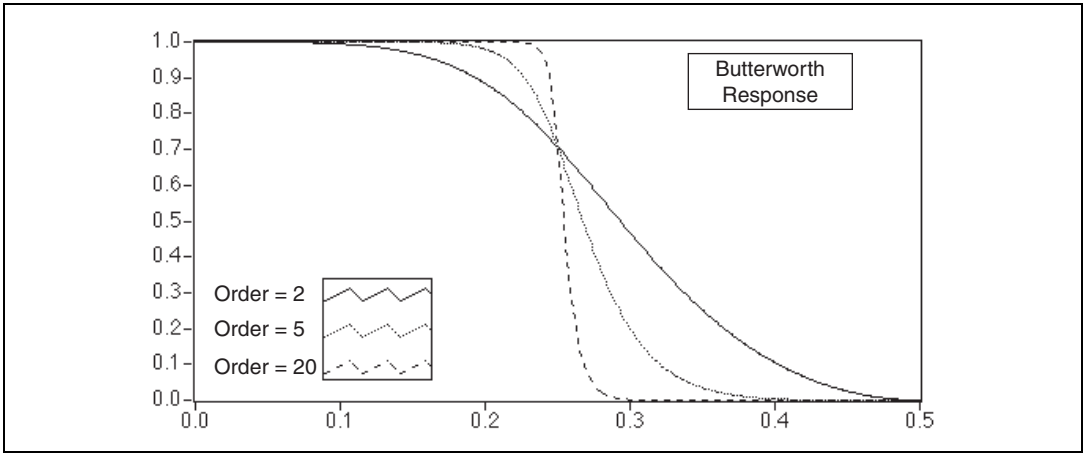


Figure 4-4. Butterworth Filter Response

Chebyshev Filters

Butterworth filters do not always provide 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).

Chebyshev filters minimize peak error in the passband by accounting for the maximum absolute value of the difference between the ideal filter and the filter response you want (the maximum tolerable error in the passband). The frequency response characteristics of Chebyshev filters have an equiripple magnitude response in the passband, monotonically decreasing magnitude response in the stopband, and a sharper rolloff than Butterworth filters.

Figure 4-5 shows the response of a lowpass Chebyshev filter. Notice that the equiripple response in the passband is constrained by the maximum tolerable ripple error and that the sharp rolloff appears in the stopband. The advantage of Chebyshev filters over Butterworth filters is that Chebyshev filters have a sharper transition between the passband and the stopband with a lower-order filter. This produces smaller absolute errors and higher execution speeds.

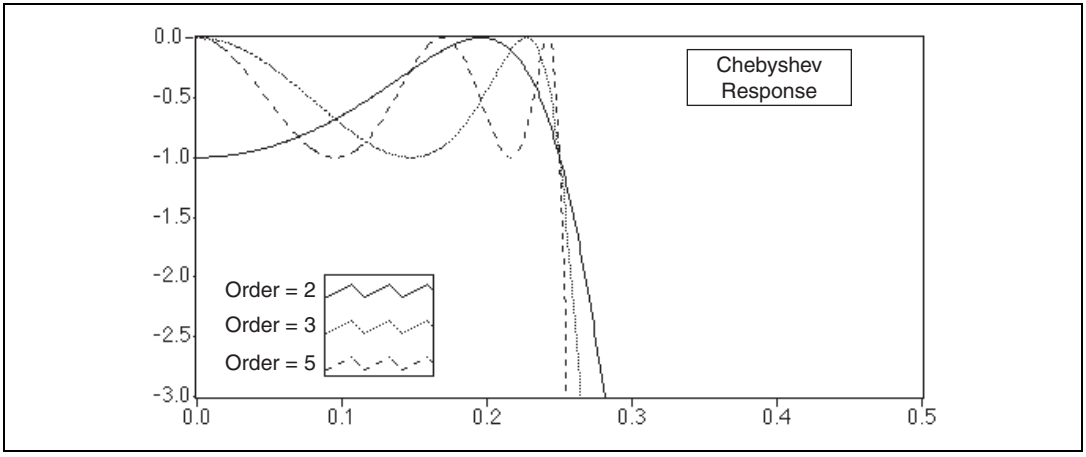


Figure 4-5. Chebyshev Filter Response

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 distribute the error over the stopband (as opposed to the passband), and Chebyshev II filters are maximally flat in the passband (as opposed to the stopband).

Chebyshev II filters minimize peak error in the stopband by accounting for the maximum absolute value of the difference between the ideal filter and the filter response you want. 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.

Figure 4-6 plots the response of a lowpass Chebyshev II filter. Notice that the equiripple response in the stopband is constrained by the maximum tolerable error and that the smooth monotonic rolloff appears in the stopband. 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 distribute the error in the stopband instead of the passband.

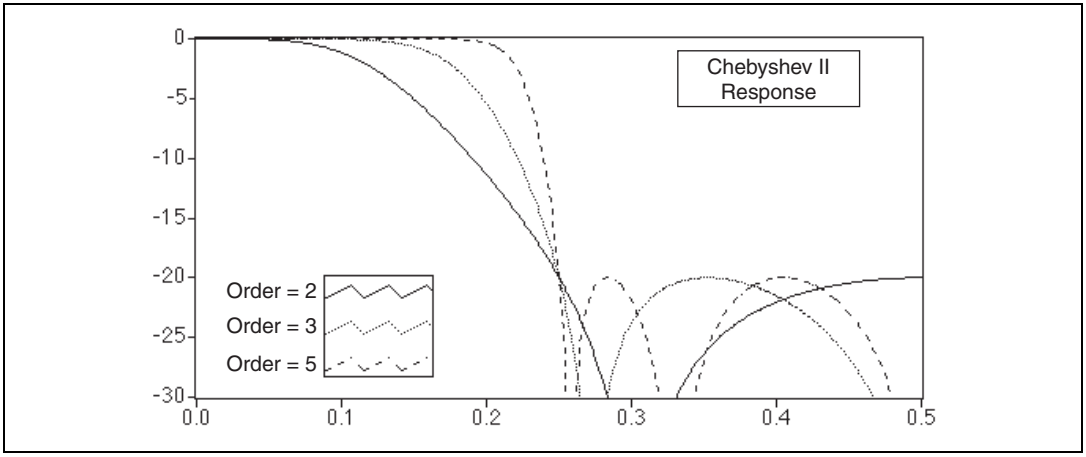


Figure 4-6. Chebyshev II Filter Response

Elliptic (or Cauer) Filters

Elliptic filters minimize the peak error by distributing it over the passband and the stopband. Equiripples in the passband and the stopband characterize the magnitude response of elliptic filters. 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 widely used.

Figure 4-7 plots the response of a lowpass elliptic filter. Notice that the ripple in both the passband and stopband is constrained by the same maximum tolerable error (as specified by ripple amount in decibels). Also, notice the sharp transition edge for even low-order elliptic filters.

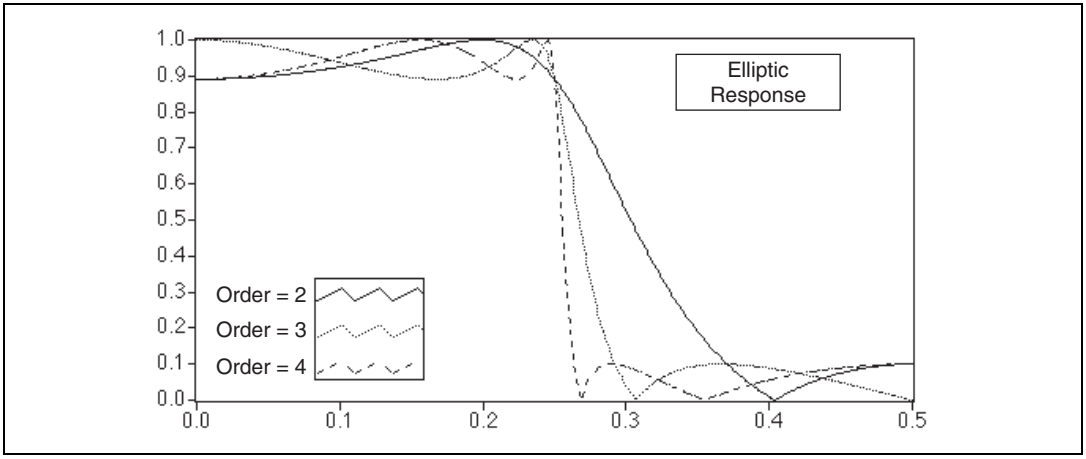


Figure 4-7. Elliptic Filter Response

Bessel Filters

You can use Bessel filters to reduce nonlinear phase distortion inherent in all IIR filters. In higher order filters and those with a steeper rolloff, this condition is more pronounced, especially in the transition regions of the filters. Bessel filters have maximally flat response in both magnitude and phase. Furthermore, the phase response in the passband of Bessel filters, which is the region of interest, is nearly linear. Like Butterworth filters, Bessel filters require high-order filters to minimize the error and, for this reason, are not widely used. You also can obtain linear phase response using FIR filter designs.

Figure 4-8 and Figure 4-9 plot the response of a lowpass Bessel filter. Notice that the response is smooth at all frequencies, as well as monotonically decreasing in both magnitude and phase. Also, notice that the phase in the passband is nearly linear.

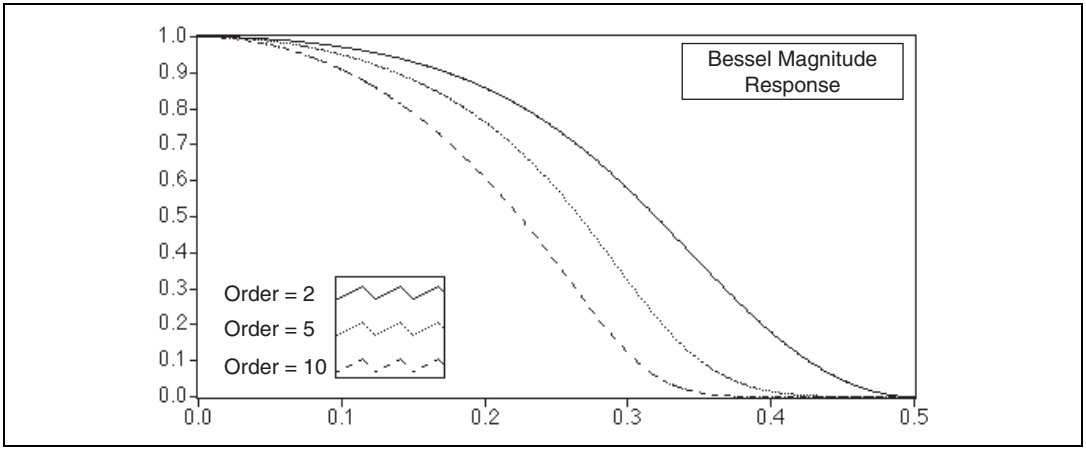


Figure 4-8. Bessel Magnitude Filter Response

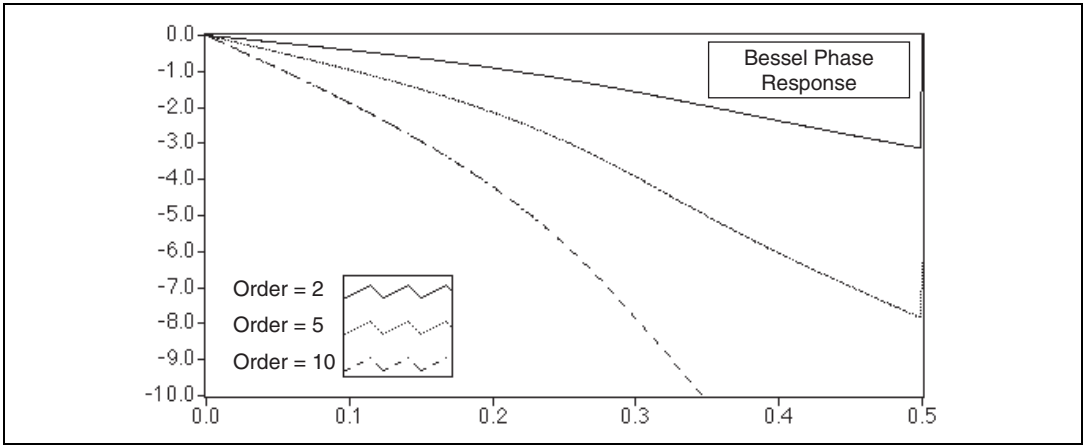


Figure 4-9. Bessel Phase Filter Response

Choosing and Designing a Digital Filter

Some of the factors affecting the choice of a suitable filter are whether you require linear phase, whether you can tolerate ripples, and whether you require a narrow transition band. Use Figure 4-10 as a guideline for selecting the correct filter. Keep in mind that in practice, you may need to experiment with several different options before finding the best one.

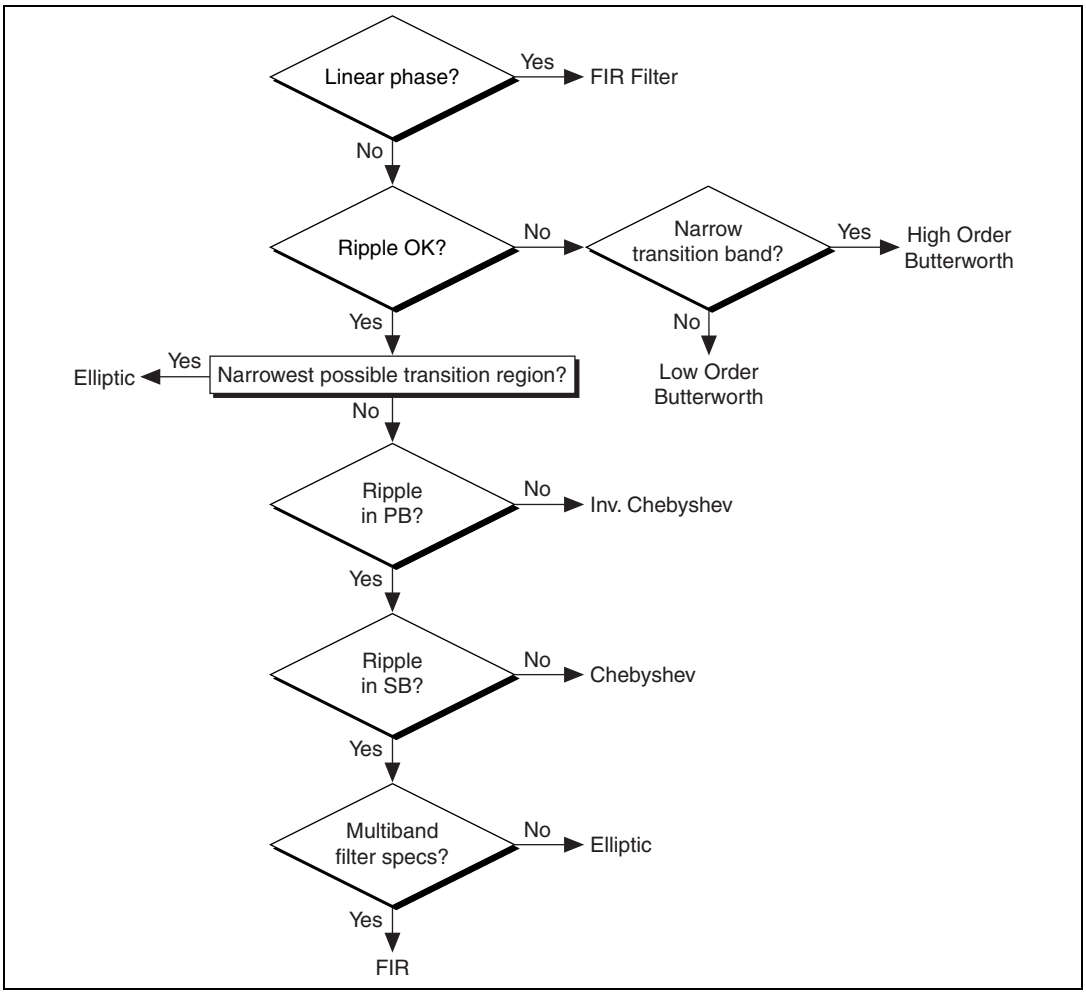


Figure 4-10. Filter Flowchart

After you choose the type of filter, you must specify the design parameters. The first filter design parameter to consider is sampling rate. The maximum frequency component of the signal of interest usually determines the sampling rate. A common rule of thumb is to choose a sampling rate that is 10 times the highest frequency component of the signal of interest. The possible tradeoff occurs when the cut-off frequency of the filter must be very close to either DC or the Nyquist frequency. At these points, a filter may converge more slowly. The solution is to increase the sampling rate if the cut-off is too close to Nyquist, or reduce the sampling rate if the cut-off is too close to DC. In practice, a particular sampling rate is chosen and adjusted only if there are problems.

5

Smoothing Windows

This chapter discusses spectral leakage, using windows to decrease spectral leakage, the different types of windows, choosing the correct window type, and the differences between windows used for spectral analysis and windows used for filter coefficient design.

Applying a window to a signal is known as windowing. You can use windowing in the following applications:

- Defining the duration of the observation
- Reducing spectral leakage
- Separating a small amplitude signal from a larger amplitude signal with frequencies very close to each other
- Designing FIR filter coefficients

The Windows VIs provide a simple method of improving the spectral characteristics of a sampled signal. Use the NI Example Finder, available by selecting **Help»Find Examples**, to find an example of how to use the Windows VIs.

Spectral Leakage and Smoothing Windows

Spectral leakage is a phenomenon in which the energy at one frequency appears to have leaked out into all other frequencies. In practical, signal-sampling applications, you can obtain only a finite record of the signal, even when you carefully observe the sampling theorem and sampling conditions. Unfortunately for the discrete-time system, the finite sampling record results in a truncated waveform with different spectral characteristics from the original continuous-time signal. These discontinuities, shown in Figure 5-1, produce leakage of spectral information, resulting in a discrete-time spectrum that is a smeared version of the original continuous-time spectrum.

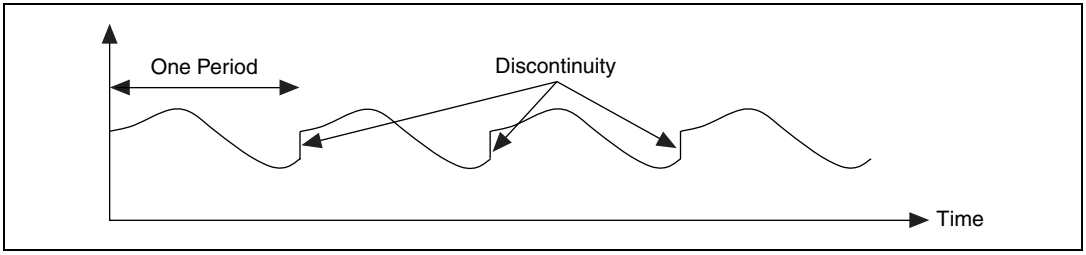


Figure 5-1. Periodic Waveform Created from Sampled Period

When you use the DFT/FFT to find the frequency content of a signal, the assumption is made that the data you have is a single period of a periodically repeating waveform, as shown in Figure 5-1. The first period shown is the one sampled. The waveform corresponding to the sampled period is then repeated in time to produce the periodic waveform.

Sampling an Integer Number of Cycles

Figure 5-2 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, this waveform is repeated. The periodic time waveform of the sine wave from Graph 1 is shown in Graph 2. Graph 3 shows the spectral representation of the waveform. 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 the waveform in Graph 2 does not have any discontinuities is because an integer number of cycles, in this case, 1, of the time waveform have been sampled.

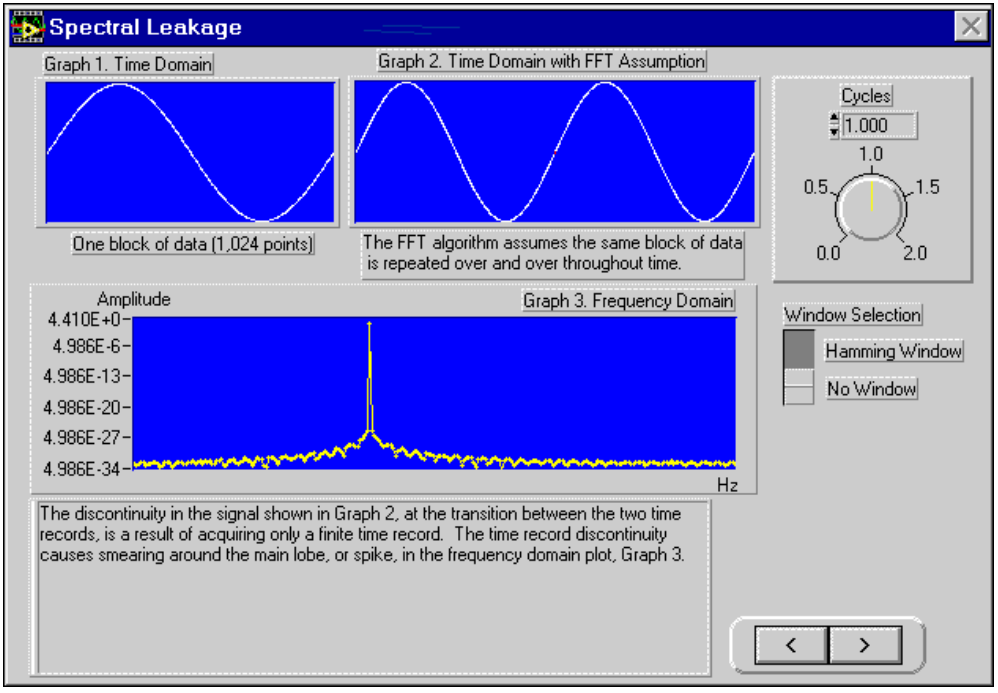


Figure 5-2. Sine Wave and Corresponding Fourier Transform

Sampling a Noninteger Number of Cycles

Because of the assumption of periodicity of the waveform, artificial discontinuities between successive periods occur when you sample a noninteger number of cycles. The artificial discontinuities appear as very high frequencies in the spectrum of the signal, frequencies that were not present in the original signal. These high frequencies could be much higher than the Nyquist frequency and, as you have seen before, are aliased somewhere between 0 and $f_s/2$. Therefore, the spectrum you get by using the DFT/FFT is not the actual spectrum of the original signal but is a smeared version. In the smeared version of the spectrum, the energy at one frequency appears to have leaked out into all the other frequencies. This phenomenon is known as spectral leakage.

Figure 5-3 shows the spectral representation when a noninteger number of cycles of the time waveform is sampled. Graph 1 consists of 1.25 cycles of the sine wave. When the sampled waveform is repeated periodically, the resulting waveform, as shown in Graph 2, consists of discontinuities. Graph 3 shows the spectral representation of the waveform. Notice how the energy is now spread, or smeared, over a wide range of frequencies.

The energy has leaked out of one of the FFT lines and smeared itself into all the other lines causing spectral leakage.

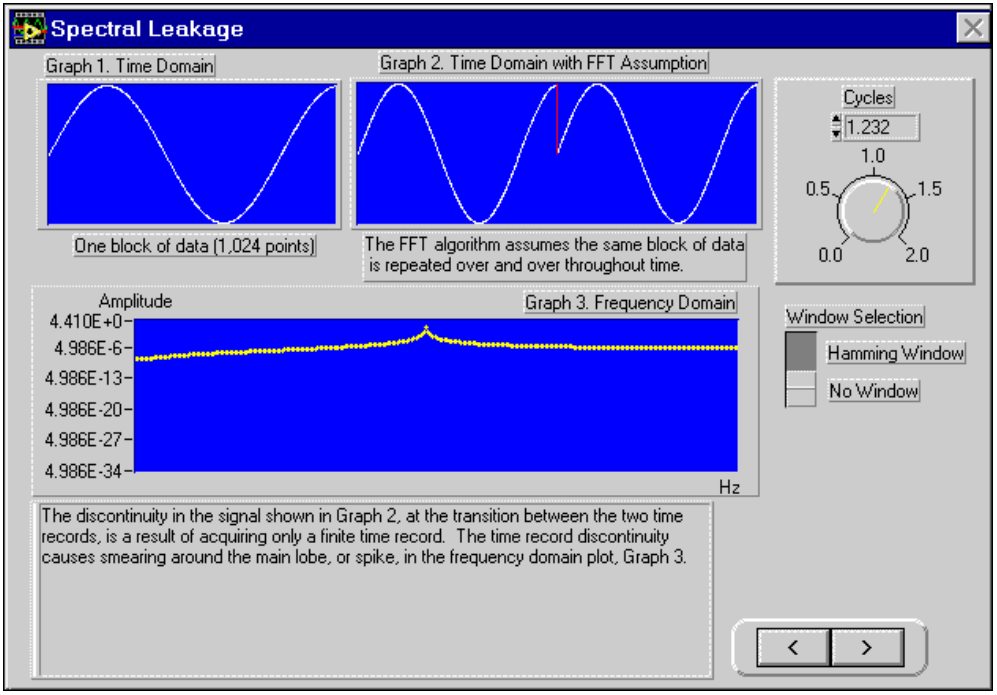


Figure 5-3. Spectral Representation When Sampling a Nonintegral Number of Samples

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 –infinity to +infinity. With an infinite time record, the FFT would calculate one single line at the correct frequency. However, waiting for infinite time is not possible in practice. To overcome the limitations of a finite time record, windowing is used to reduce the spectral leakage.

Using Windows to Prevent Spectral Leakage

A simple way to improve the spectral characteristics of a sampled signal is to apply smoothing windows. When performing Fourier or spectral analysis on finite-length data, you can use windows to minimize the transition edges of your truncated waveforms, thus reducing spectral leakage. When used in this manner, smoothing windows act like predefined, narrowband, lowpass filters.

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. Windowing consists of multiplying the time record by a finite length window whose amplitude varies smoothly and gradually towards zero at the edges, as shown in Figure 5-4. In Figure 5-4, 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.

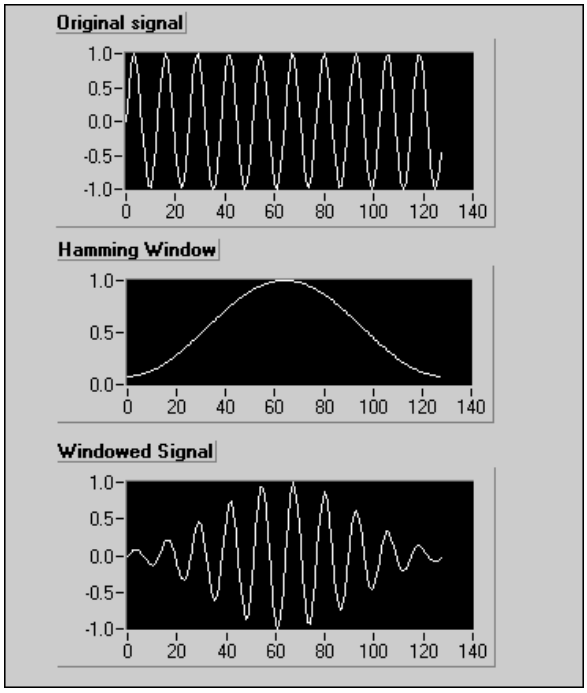


Figure 5-4. Time Signal Windowed Using a Hamming Window

Notice that if the time record contains an integral number of cycles, as shown in Figure 5-2, then 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.

Characteristics of Different Types of Window Functions

This section discusses the different types of windows available in LabVIEW. Depending on your application, you might find a particular window type more useful than the others.

Windowing a signal in the time domain is equivalent to multiplying the signal by the window function. Multiplication in the time domain is equivalent to convolution in the frequency domain. Therefore, 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. The length, or time interval, of a window is defined in terms of the number of samples.

Rectangular (None)

The rectangular window has a value of one over its length. The following equation defines the rectangular window.

$$w(n) = 1.0 \qquad \text{for } n = 0, 1, 2, \ldots, N - 1$$

where N is the length of the window and w is the window value.

Applying a rectangular window is equivalent to not using any window 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 in Figure 5-5.

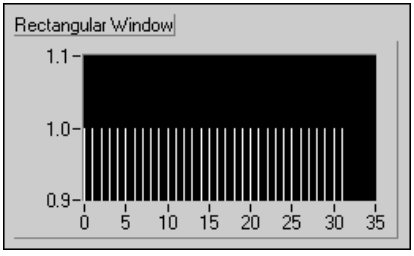


Figure 5-5. Rectangular Window

The rectangular window is useful for analyzing transients that have a duration shorter than that of the window. The rectangular window is also used in order tracking, where the effective sampling rate is proportional to

the speed of the shaft in rotating machines. In order tracking, the rectangular window detects the main mode of vibration of the machine and its harmonics.

Hanning

The Hanning window has a shape similar to that of half a cycle of a cosine wave. The following equation defines the Hanning window.

$$w(n) = 0.5 - 0.5\cos\frac{2\pi n}{N} \qquad \text{for } n = 0, 1, 2, \ldots, N-1$$

where N is the length of the window and w is the window value.

A Hanning window with $N = 32$ is shown in Figure 5-6.

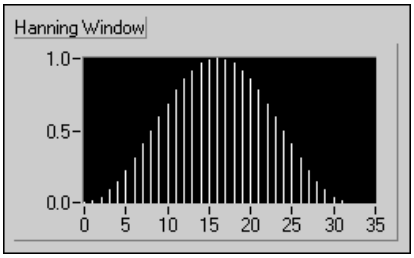


Figure 5-6. Hanning Window

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

Hamming

The Hamming window is a modified version of the Hanning window. The shape of the Hamming window is similar to that of a cosine wave. The following equation defines the Hamming window.

$$w(n) = 0.54 - 0.46\cos\frac{2\pi n}{N} \qquad \text{for } n = 0, 1, 2, \ldots, N-1$$

where N is the length of the window and w is the window value.

A Hamming window with $N = 32$ is shown in Figure 5-7.

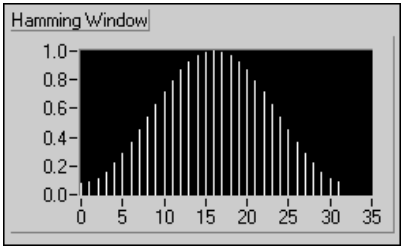


Figure 5-7. Hamming Window

The Hanning and Hamming windows are somewhat similar, as shown in Figures 5-6 and 5-7. However, notice that in the time domain, the Hamming window does not get as close to zero near the edges as does the Hanning window.

Kaiser-Bessel

The Kaiser-Bessel window is a flexible window whose shape you 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. Figure 5-8 shows the Kaiser-Bessel window for different values of beta.

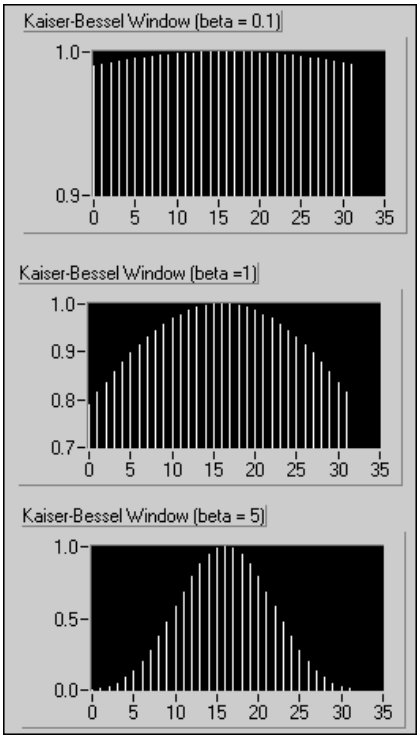


Figure 5-8. Kaiser-Bessel Window

Notice 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.

The Kaiser-Bessel window is good for detecting two signals of almost the same frequency but with significantly different amplitudes.

Triangle

The shape of the triangle window is that of a triangle. The following equation defines the triangle window.

$$w(n) = 1 - \left|\frac{2n - N}{N}\right| \qquad \text{for } n = 0, 1, 2, \ldots, N - 1$$

where N is the length of the window and w is the window value.

A triangle window for $N = 32$ is shown in Figure 5-9.

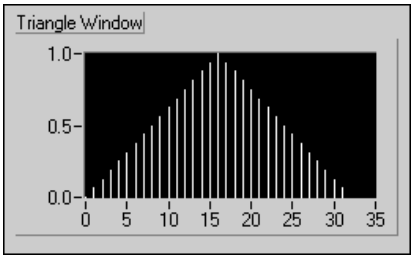


Figure 5-9. Triangle Window

Flat Top

The flat top window has the best amplitude accuracy of all the window functions at ±0.02 dB for signals exactly between integral cycles. However, the increased amplitude accuracy comes at the expense of frequency selectivity. The following equation defines the flat top window.

$$w(n) = \sum_{k=0}^{4} (-1)^k a_k \cos(k\omega)$$

where

$$\omega = \frac{2\pi n}{N}$$

$$a_0 = 0.215578948$$
$$a_1 = 0.416631580$$
$$a_2 = 0.277263158$$
$$a_3 = 0.083578947$$
$$a_4 = 0.006947368$$

A flat top window is shown in Figure 5-10.

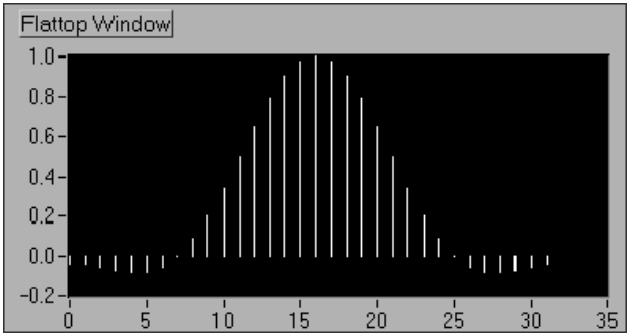


Figure 5-10. Flat Top Window

The flat top window is most useful in accurately measuring the amplitude of single frequency components with little nearby spectral energy in the signal.

Exponential

The shape of the exponential window is that of a decaying exponential. The following equation defines the exponential window.

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

where N is the length of the window, w is the window value, and f is the final value.

The initial value of the window is one and gradually decays toward zero. You can adjust the final value of the exponential window to between 0 and 1.

The exponential window for $N = 32$, with the final value specified as 0.1, is shown in Figure 5-11.

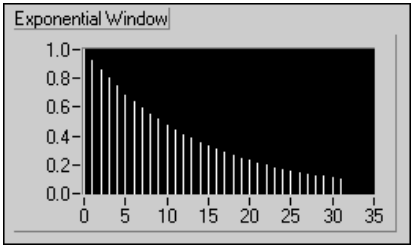


Figure 5-11. Exponential Window

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

Windows for Spectral Analysis versus Windows for Coefficient Design

Spectral Analysis and filter coefficient design place different requirements on a window. Spectral analysis requires a DFT-even window, while filter coefficient design requires a window symmetric about its midpoint. This section discusses the differences between windows for spectral analysis and windows for filter coefficient design.

Spectral Analysis

The Windows VIs in LabVIEW are designed for spectral analysis applications. In spectral analysis applications, the input signal is windowed by passing it through one of the Windows VIs. The windowed signal is then passed to a DFT-based VI for frequency-domain display and analysis.

The window functions designed for spectral analysis must be DFT-even, a term defined by Fredric J. Harris in his paper *On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform* (*Proceedings of the IEEE*, Volume 66, No. 1, January 1978). A window function is DFT-even if its dot product, or inner product, with integral cycles of sine sequences is identically zero. Another way to think of a DFT-even sequence is that its DFT has no imaginary component.

Figures 5-12 and 5-13 illustrate the Hanning window and one cycle of a sine pattern for a sample size of 8.

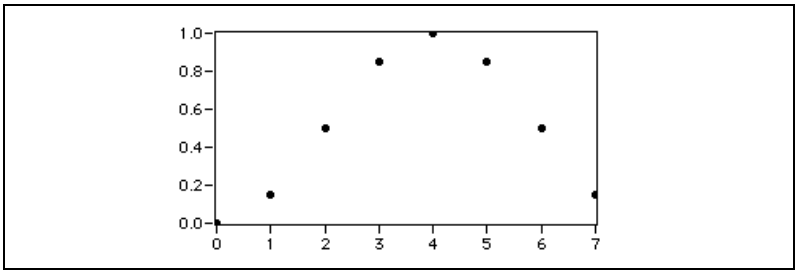


Figure 5-12. Hanning Window for Sample Size 8

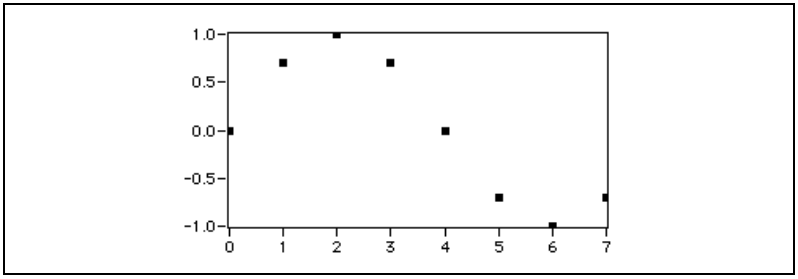


Figure 5-13. Sine Pattern for Sample Size 8

You can see that the DFT-even Hanning window is not symmetric about its midpoint and its last point is not equal to its first point, much like one complete cycle of a sine pattern.

The DFT considers input sequences to be periodic. In other words, the signal being analyzed is actually a concatenation of the input signal. Figure 5-14 shows three cycles of the sequences from Figures 5-12 and 5-13, demonstrating the smooth periodic extension of the DFT-even window and the single-cycle sine pattern.

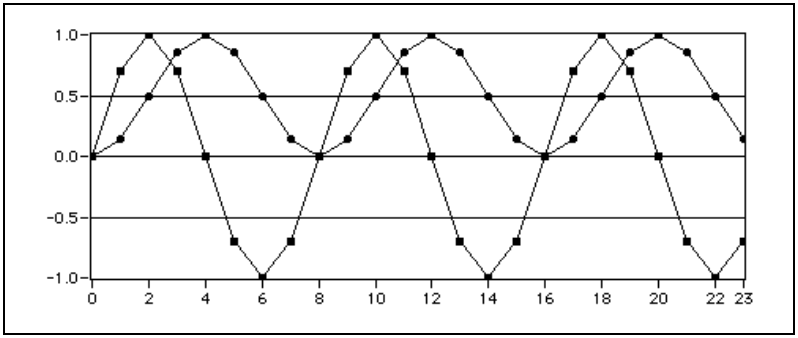


Figure 5-14. Periodic Extension

Windows for Coefficient Design

FIR filter design requires windows that are symmetric about their midpoint. Refer to Chapter 4, *Digital Filtering*, for more information about filtering.

The following equations of the Hanning window function illustrate the difference between the DFT-even window function for spectral analysis and the symmetrical window function for coefficient design.

The Hanning window function for spectral analysis is given by the following equation.

$$w[i] = 0.5\left(1 - \cos\left(\frac{2\pi i}{N}\right)\right) \qquad \text{for } i = 0, 1, 2, \ldots, N-1$$

where N is the length of the window and w is the window value.

The Hanning window function for symmetrical coefficient design is given by the following equation.

$$w[i] = 0.5\left(1 - \cos\left(\frac{2\pi i}{N-1}\right)\right) \qquad \text{for } i = 0, 1, 2, \ldots, N-1$$

where N is the length of the window and w is the window value.

The preceding two equations show that you can implement the symmetrical window functions by slightly modifying the use of the DFT-even window functions.

Choosing the Correct Window Type

The type of window you should use 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 you are analyzing. Table 5-1 shows the different types of signals and the appropriate windows that you can use with them.

Table 5-1. Signals and Windows

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
Spectral 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
Accurate single tone amplitude measurements	Flat top

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.

6

DC/RMS Measurements

Two of the most common measurements of a signal are its direct current (DC) and root mean square (RMS) levels. This chapter introduces measurement analysis techniques for making DC and RMS measurements of a signal.

What Is the DC Level of a Signal?

You can use DC measurements to define the value of a static or slowly varying signal. DC measurements can be both positive and negative. The DC value usually is constant within a specific time window. You can track and plot slowly moving values, such as temperature, as a function of time using a DC meter. In that case, the observation time that results in the measured value has to be short compared to the speed of change for the signal. Figure 6-1 illustrates an example DC level of a signal.

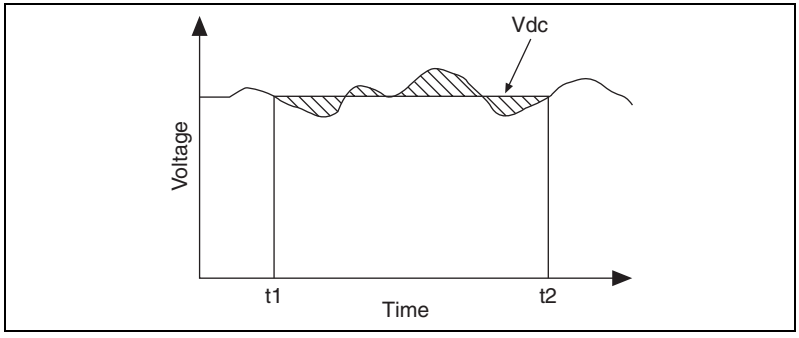


Figure 6-1. DC Level of a Signal

The DC level of a continuous signal $V(t)$ from time $t1$ to time $t2$ is given by the equation:

$$V_{dc} = \frac{1}{(t2 - t1)} \cdot \int_{t1}^{t2} V(t)dt$$

where $t2 - t1$ represents the integration time or measurement time.

For digitized signals, the discrete-time version of the previous equation is given by:

$$V_{dc} = \frac{1}{N} \cdot \sum_{i=1}^{N} V_i$$

For a sampled system, the DC value is defined as the mean value of the samples acquired in the specified measurement time window.

Between pure DC signals and fast-moving dynamic signals is a "gray zone" where signals become more complex, and measuring the DC level of these signals becomes quite challenging.

Real world signals often contain a significant amount of dynamic influence. Often, you do not want the dynamic part of the signal. The DC measurement identifies the static DC signal hidden in the dynamic signal, for example, the voltage generated by a thermocouple in an industrial environment, where external noise or hum from the main power can disturb the DC signal significantly.

What Is the RMS Level of a Signal?

The RMS level of a signal is the square root of the mean value of the squared signal. RMS measurements are always positive. Use RMS measurements when a representation of energy is needed. You usually acquire RMS measurements on dynamic signals (signals with relatively fast changes) like noise or periodic signals. Refer to the *LabVIEW Measurements Manual* for more information about when to use RMS measurements.

The RMS level of a continuous signal $V(t)$ from time $t1$ to time $t2$ is given by the equation:

$$V_{rms} = \sqrt{\frac{1}{(t2 - t1)} \cdot \int_{t1}^{t2} V^2(t)dt}$$

where $t2 - t1$ represents the integration time or measurement time.

The RMS level of a discrete signal V_i is given by the equation:

$$V_{rms} = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} V_i^2}$$

One difficulty is encountered when measuring the dynamic part of a signal using an instrument that does not offer an AC-coupling option. A true RMS measurement includes the DC part in the measurement, a measurement you might not want.

Averaging to Improve the Measurement

Instantaneous DC measurements of a noisy signal can vary randomly and significantly, as shown in Figure 6-2. You can measure a more accurate value by averaging out the noise that is superimposed on the desired DC level. In a continuous signal, the averaged value between two times, $t1$ and $t2$, is defined as the signal integration between $t1$ and $t2$, divided by the measurement time, $t2 - t1$, as shown in Figure 6-1. The area between the averaged value V_{dc} and the signal that is above V_{dc} is equal to the area between V_{dc} and the signal that is under V_{dc}. For a sampled signal, the average value is the sum of the voltage samples divided by the measurement time in samples, or the mean value of the measurement samples. Refer to the *LabVIEW Measurements Manual* for more information about averaging in LabVIEW.

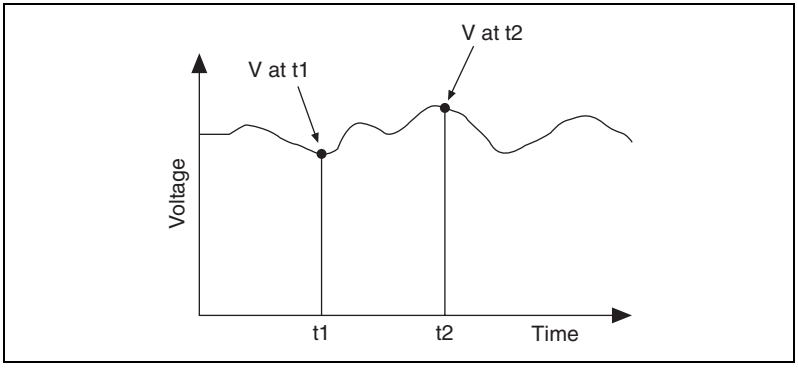


Figure 6-2. Instantaneous DC Measurements

An RMS measurement is an averaged quantity, because it is the average energy in the signal over a measurement period. You can improve the RMS measurement accuracy by using a longer averaging time, equivalent to the integration time or measurement time.

There are several different strategies to use for making DC and RMS measurements, each dependent on the type of error or noise sources. When choosing a strategy, you must decide if accuracy or speed of the measurement is more important.

Common Error Sources Affecting DC and RMS Measurements

Some common error sources for DC measurements are single frequency components (or tones), multiple tones, or random noise. These same error signals can interfere with RMS measurements, so in many cases the approach taken to improve RMS measurements is the same as for DC measurements.

DC Overlapped with Single Tone

Consider the case where the signal you measure is composed of a DC signal and a single sine tone. The average of a single period of the sine tone is ideally zero, because the positive half-period of the tone cancels the negative half-period.

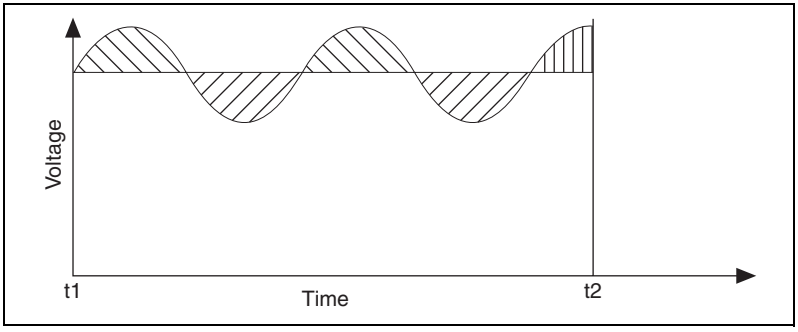


Figure 6-3. DC Signal Overlapped with Single Tone

Any remaining partial period, shown in Figure 6-3 with vertical hatching, introduces an error in the average value and, therefore, in the DC measurement. Increasing the averaging time reduces this error, because the integration is always divided by the measurement time $t2 - t1$. If you know

the period of the sine tone, you can take a more accurate measurement of the DC value by using a measurement period equal to an integer number of periods of the sine tone. The most severe error occurs when the measurement time is a half-period different from an integer number of periods of the sine tone, because this is the maximum area under or over the signal curve.

Defining the Equivalent Number of Digits

Defining the Equivalent Number of Digits (ENOD) makes it easier to relate a measurement error to a number of digits, similar to digits of precision. ENOD translates measurement accuracy into a number of digits.

$$\text{ENOD} = \log_{10}(\text{Relative Error})$$

A 1 percent error corresponds to 2 digits of accuracy, and a 1 part per million error corresponds to 6 digits of accuracy ($\log_{10}(0.000001) = 6$).

ENOD is only an approximation that tells you what order of magnitude of accuracy you can achieve under specific measurement conditions. This accuracy does not take into account any error introduced by the measurement instrument or data acquisition hardware itself. ENOD only gives you a tool for computing the reliability of a specific measurement technique.

Thus, the ENOD should at least match the accuracy of the measurement instrument or measurement requirements. For example, it is not necessary to use a measurement technique with an ENOD of 6 digits if your instrument has an accuracy of only 0.1 percent (3 digits). Similarly, you do not get the six digits of accuracy from your 6-digit accurate measurement instrument if your measurement technique is limited to an ENOD of only 3 digits.

DC Plus Sine Tone

Figure 6-4 shows that for a 1.0 VDC signal overlapped with a 0.5 V single sine tone, the worst ENOD increases with measurement time, x-axis shown in periods of the additive sine tone, at a rate of approximately 1 additional digit for 10 times more measurement time. To achieve 10 times more accuracy, you need to increase your measurement time by a factor of 10. In other words, accuracy and measurement time are related through a first-order function.

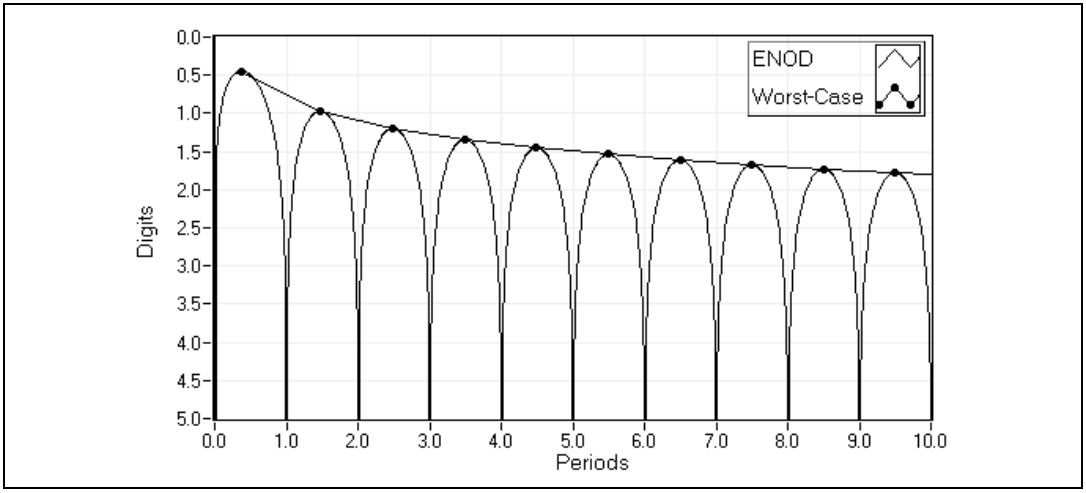


Figure 6-4. Digits versus Measurement Time for 1 VDC Signal with 0.5 Single Tone

Windowing to Improve DC Measurements

The worst ENOD for a DC signal plus a sine tone occurs when the measurement time is at half-periods of the sine tone. You can greatly reduce these errors due to non-integer number of cycles by using a weighting function before integrating to measure the desired DC value. The most common weighting or window function is the Hann window, commonly known as the Hanning window.

Figure 6-5 shows a dramatic increase in accuracy from the use of the Hann, also known as Hanning, window. The accuracy as a function of the number of sine tone periods is improved from a first-order function to a third-order function. In other words, you can achieve 1 additional digit of accuracy for every $10^{1/3} = 2.15$ times more measurement time using the Hann window instead of 1 digit for every 10 times more measurement time without using a window. As in the non-windowing case, the DC level is 1.0 V and the single tone peak amplitude is 0.5 V.

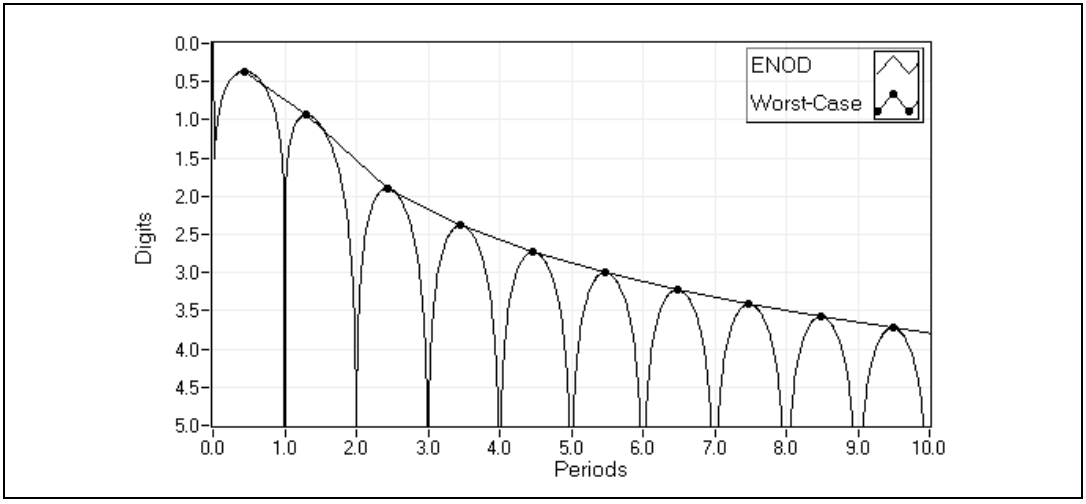


Figure 6-5. Digits versus Measurement Time for DC + Tone Using Hann Window

You can use other types of window functions to further reduce the necessary measurement time or greatly increase the resulting accuracy. Figure 6-6 shows that the Low Sidelobe (LSL) window can achieve more than six ENOD of worst accuracy when averaging your DC signal over only five periods of the sine tone (same test signal).

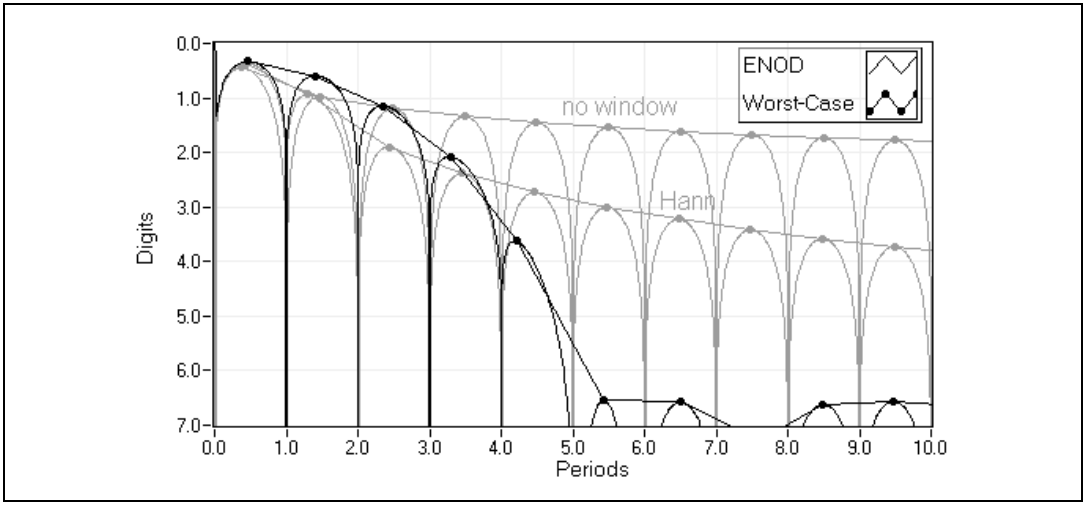


Figure 6-6. Digits versus Measurement Time for DC + Tone Using LSL Window

RMS Measurements Using Windows

Like DC measurements, the worst ENOD for measuring the RMS level of signals sometimes can be significantly improved by applying a window to the signal before RMS integration. For example, if you measure the RMS level of the DC signal plus a single sine tone, the most accurate measurements are made when the measurement time is an integer number of periods of the sine tone. Figure 6-7 shows that the worst ENOD varies with measurement time (in periods of the sine tone) for various window functions. Here, the test signal contains 0.707 VDC with 1.0 V peak sine tone.

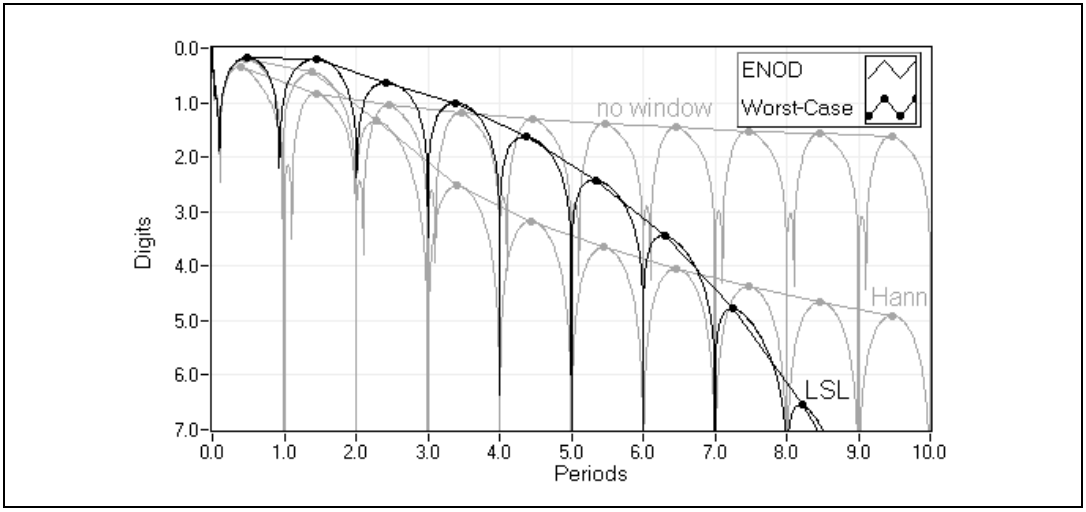


Figure 6-7. Digits versus Measurement Time for RMS Measurements

Applying the window to the signal increases RMS measurement accuracy significantly, but the improvement is not as large as in DC measurements. For this example, the LSL window achieves six digits of accuracy when the measurement time reaches eight periods of the sine tone.

Using Windows with Care

Window functions can be very useful to improve the speed of your measurement, but you must be careful. The Hann window is a general window recommended in most cases. Use more advanced windows like the LSL window only if you know enough about your signal that you are sure the window is not doing more damage than good. For example, you can significantly reduce RMS measurement accuracy if the signal you want to

measure is composed of many frequency components close to each other in the frequency domain.

You also must make sure that the window is scaled correctly or that you update scaling after applying the window. The most useful window functions are pre-scaled by their *coherent gain*—the mean value of the window function—so that the resulting mean value of the scaled window function is always 1.00. DC measurements do not need to be scaled when using a properly scaled window function. For RMS measurements, each window has a specific *equivalent noise bandwidth* that you must use to scale integrated RMS measurements. You must scale RMS measurements using windows by the reciprocal of the square root of the equivalent noise bandwidth.

Rules for Improving DC and RMS Measurements

Use the following guidelines when determining a strategy for improving your DC and RMS measurements:

- If your signal is overlapped with a single tone, longer integration times increase the accuracy of your measurement. If you know the exact frequency of the sine tone, use a measurement time that corresponds to an exact number of sine periods. If you do not know the frequency of the sine tone, apply a window, such as a Hann window, to significantly reduce the measurement time needed to achieve a specific accuracy.
- If your signal is overlapped with many independent tones, increasing measurement time increases the accuracy of the measurement. As in the single tone case, using a window significantly reduces the measurement time needed to achieve a specific accuracy.
- If your signal is overlapped with noise, do not use a window. In this case, you can increase the accuracy of your measurement by increasing the integration time or by pre-processing or conditioning your noisy signal with a lowpass (or bandstop) filter.

RMS Levels of Specific Tones

You can always improve the accuracy of an RMS measurement by choosing a specific measurement time (to contain an integer number of cycles of your sine tones) or by using a window function. The measurement of the RMS value is based only on the time domain knowledge of your signal. You can use advanced techniques when you are interested in a specific frequency or narrow frequency range.

You can use bandpass or bandstop filtering before RMS computations to measure the RMS power in a specific band of frequencies. You also can use the Fast Fourier Transform (FFT) to pick out specific frequencies for RMS processing. Refer to Chapter 2, *Frequency Analysis*, for more information about the FFT.

The RMS level of a specific sine tone that is part of a complex or noisy signal can be extracted very accurately using frequency domain processing, leveraging the power of the FFT, and utilizing the benefits of windowing.

7

Distortion Measurements

This chapter defines harmonic distortion, THD, and SINAD, and explains when to use distortion measurements.

What Is Distortion?

When a pure single-frequency sine wave is applied to a perfectly linear system, it produces an output that has the same frequency as that of the input sine wave, but with possible changes in the amplitude and/or phase. This also is true when a composite signal consisting of several sine waves is applied at the input. The output signal consists of the same frequencies but with different amplitudes and/or phases.

Many real-world systems act as nonlinear systems when their input limits are exceeded, resulting in distorted output signals. If the input limits of a system are exceeded, the output consists of one or more frequencies that did not originally exist at the input. For example, if the input to a nonlinear system consists of two frequencies f_1 and f_2, the frequencies at the output could be f_1 and harmonics (integer multiples) of f_1, f_2 and harmonics of f_2, and sums and differences of f_1, f_2, and the harmonics of f_1 and f_2. The number of new frequencies at the output, their corresponding amplitudes, and their relationships with respect to the original frequencies vary depending on the transfer function. Use distortion measurements to quantify the degree of nonlinearity of a system. Some common distortion measurements include Total Harmonic Distortion (THD), Total Harmonic Distortion + Noise (THD + *N*), Signal Noise and Distortion (SINAD), and Intermodulation Distortion.

Application Areas

You can make distortion measurements for many devices, such as A/D and D/A converters, audio processing devices, such as preamplifiers, equalizers, and power amplifiers, analog tape recorders, cellular phones, radios, TVs, stereos, and loudspeakers.

Measurements of harmonics often provide a good indication of the cause of the nonlinearity. For example, nonlinearities that are not symmetrical around zero produce mainly even harmonics, whereas symmetrical

nonlinearities result in the production of mainly odd harmonics. You can use distortion measurements to diagnose faults such as bad solder joints, torn speaker cones, and components that have been incorrectly installed. Nonlinearities are not always undesirable, however. For example, many musical sounds are produced specifically by driving a device into its nonlinear region.

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 ($f_2 = 2f_1, f_3 = 3f_1, f_4 = 4f_1$, 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 $x(t)$, as shown in Figure 7-1.

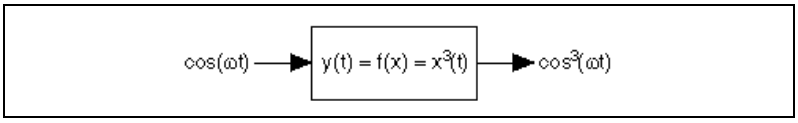


Figure 7-1. Example Nonlinear System

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), and so on, then the total harmonic distortion (THD) is given by

$$\mathrm{THD} = \frac{\sqrt{A_2^2 + A_3^2 + A_4^2 + \ldots}}{A_1}$$

The percentage total harmonic distortion (%THD) is given by the equation:

$$\%\mathrm{THD} = 100 \cdot \frac{\sqrt{A_2^2 + A_3^2 + A_4^2 + \ldots}}{A_1}$$

Thus, measurement of the total harmonic distortion requires measuring the amplitudes of the fundamental frequency and the amplitudes of the individual harmonics. A common cause of harmonic distortion is clipping that occurs when a system is driven beyond its capabilities. Symmetrical clipping results in odd harmonics, but asymmetrical clipping creates both even and odd harmonics.

Real-world signals are usually noisy. The system also can introduce additional noise into the signal. A useful measure of distortion, which also takes into account the amount of noise power, is total harmonic distortion + noise (THD + *N*) and is given by the equation:

$$\mathrm{THD} + N = \frac{\sqrt{A_2^2 + A_3^2 + \ldots + N^2}}{\sqrt{A_1^2 + A_2^2 + A_3^2 + \ldots N^2}}$$

where *N* is the noise power.

The percentage total harmonic distortion + noise (%THD + *N*) is given by the equation:

$$\%\mathrm{THD} + N = 100 \cdot \frac{\sqrt{A_2^2 + A_3^2 + \ldots + N^2}}{\sqrt{A_1^2 + A_2^2 + A_3^2 + \ldots N^2}}$$

Thus, measurement of THD + *N* requires measuring the amplitude of the fundamental frequency and the power present in the remaining signal after the fundamental frequency has been removed.

THD + *N* also includes the noise, a low measurement not only means that the system has a low amount of harmonic distortion, it also means that the contribution from the AC mains hum, wideband white noise, and other interfering signals is low. Measurements of THD or THD + *N* are usually specified in terms of the highest order harmonic that has been present in the measurement, for example, THD through the seventh harmonic or THD + *N* through the third harmonic.

SINAD

Another measurement that takes into account both harmonics and noise is SINAD. SINAD is given by the equation:

$$\text{SINAD} = \frac{\text{Fundamental} + \text{Noise} + \text{Distortion}}{\text{Noise} + \text{Distortion}}$$

SINAD is the reciprocal of THD + *N*. You can use SINAD to characterize the performance of FM receivers in terms of sensitivity, adjacent channel selectivity, and alternate channel selectivity.

8

Limit Testing

You can use limit testing to monitor a waveform and determine if it always satisfies a set of conditions, usually upper and lower limits. The region bounded by the specified limits is a mask. The result of a limit or mask test is generally a pass or fail.

Setting Up an Automated Test System

You can use the same method to create and control many different automated test systems. Complete the following basic steps to set up an automated test system for limit mask testing.

1. Configure the measurement by specifying arbitrary upper and lower limits. This defines your mask or region of interest.
2. Acquire data using a DAQ device.
3. Monitor the data to make sure it always falls within the specified mask.
4. Log the Pass/Fail results from step 3 to a file or visually inspect the input data and the points that fall outside the mask.

Repeat steps 2 through 4 to continue limit mask testing.

The following sections examine steps 1 and 3 in further detail. Assume that the signal to be monitored starts at $x = x_0$ and all the data points are evenly spaced. The spacing between each point is denoted by dx.

Specifying a Limit

Limits are classified into two types: continuous limits and segmented limits, as shown in Figure 8-1. The top graph in Figure 8-1 shows a continuous limit. A continuous limit is specified using a set of x and y points $\{\{x_1, x_2, x_3, \ldots\}, \{y_1, y_2, y_3, \ldots\}\}$. Completing step 1 creates a limit with the first point at x_0 and all other points at a uniform spacing of dx ($x_0 + dx$, $x_0 + 2dx$, …). This is done through a linear interpolation of the x and y values that define the limit. In Figure 8-1, black dots represent the points at which the limit is defined and the solid line represents the limit you create. Creating the limit in step 1 reduces test times in step 3. If the spacing

between the samples changes, you can repeat step 1. Notice that the limit is undefined in the region $x_0 < x < x_1$ and for $x > x_4$.

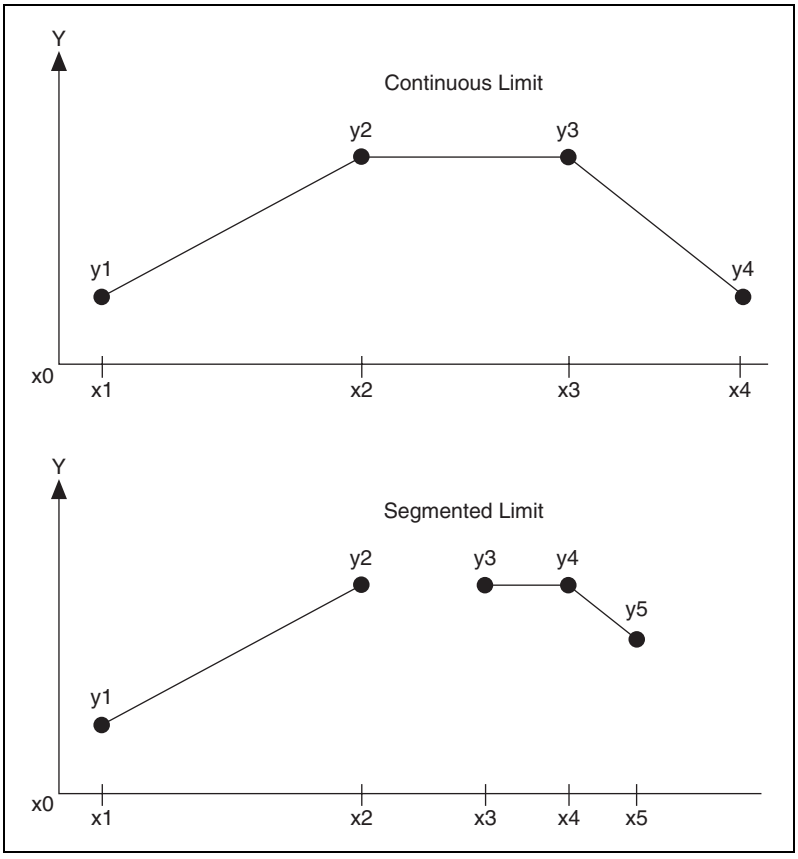


Figure 8-1. Continuous versus Segmented Limit Specification

The bottom graph of Figure 8-1 shows a segmented limit. The first segment is defined using a set of x and y points $\{\{x_1, x_2\}, \{y_1, y_2\}\}$. The second segment is defined using a set of points $\{x_3, x_4, x_5\}$ and $\{y_3, y_4, y_5\}$. You can define any number of such segments. As with continuous limits, step 1 uses linear interpolation to create a limit with the first point at x_0 and all other points with an uniform spacing of dx. Notice that the limit is undefined in the region $x_0 < x < x_1$ and in the region $x > x_5$. Also notice the limit is undefined in the region $x_2 < x < x_3$.

Specifying a Limit Using a Formula

You can specify limits using formulas. Such limits are best classified as segmented limits. Each segment is defined by start and end frequencies and a formula. For example, the ANSI T1-413 recommendation specifies the limits for the transmit and receive spectrum of an ADSL signal in terms of formula. Table 8-1, which only includes a part of the specification, shows the start and end frequencies and the upper limits of the spectrum for each segment.

Table 8-1. ADSL Signal Recommendations

Start kHz	End kHz	Maximum (Upper Limit) Value (dBm/Hz)
0.3	4.0	–97.5
4.0	25.9	$-92.5 + 21.5 \log_2(f/4{,}000)$
25.9	138.0	–34.5
138.0	307.0	$-34.5 - 48.0 \log_2(f/138{,}000)$
307.0	1221.0	–90

The limit is specified as an array of a set of *x* and *y* points, [{0.3, 4.0}{–97.5, –97.5}, {4.0, 25.9}{$-92.5 + 21.5 \log_2(f/4000)$, {$-92.5 + 21.5 \log_2(f/4000)$}, …, {307.0, 1221.0}{–90, –90}]. Each element of the array corresponds to a segment.

Figure 8-2 shows the segmented limit specified using formula as shown in Table 8-1. The *x*-axis is on a logarithmic scale.

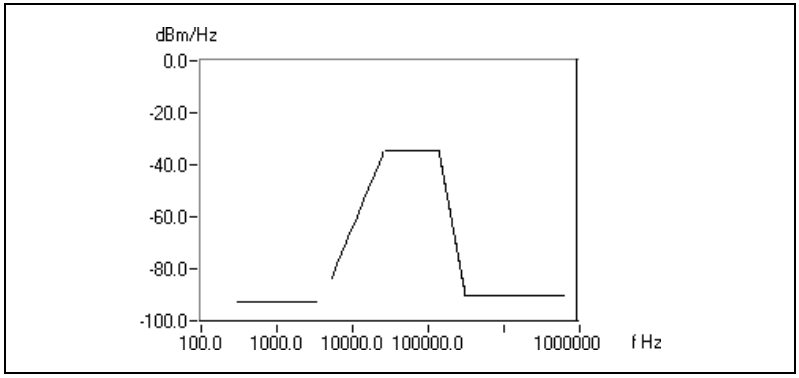


Figure 8-2. Segmented Limit Specified Using Formula

Limit Testing

After you define your mask, you acquire a signal using a DAQ device. The sample rate is set at 1/*dx* S/s. Compare the signal with the limit. In step 1, you create a limit value at each point where the signal is defined. In step 3, you compare the signal with the limit. For the upper limit, if the data point is less than or equal to the limit point, the test passes. If the data point is greater than the limit point, the test fails. For the lower limit, if the data point is greater than or equal to the limit point, the test passes. If the data point is smaller than the limit point, the test fails.

Figure 8-3 shows the result of limit testing in a continuous mask case. Here, the test signal falls within the mask at all the points it is sampled, other than points b and c. Thus the limit test fails. Point d is not tested because it falls outside the mask.

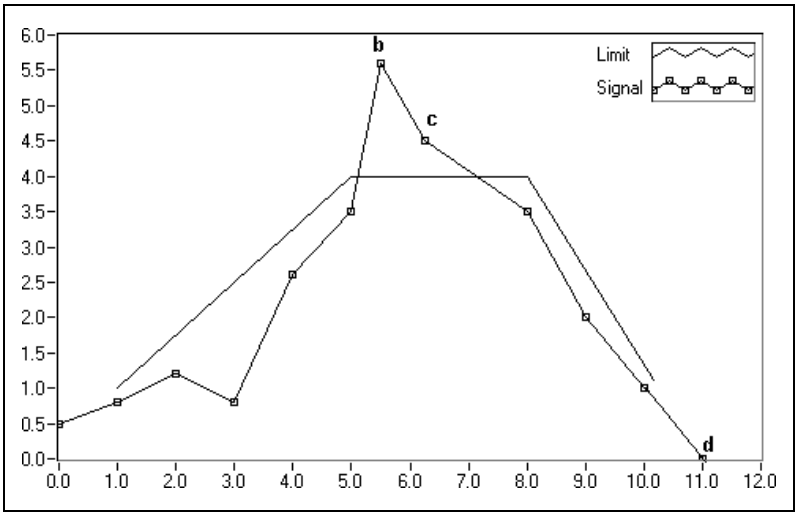


Figure 8-3. Result of Limit Testing with a Continuous Mask

Figure 8-4 shows the result of limit testing in a segmented mask case. Here, all the points fall within the mask. Points b and c are not tested because the mask is undefined at those points. Thus the limit test passes. Point d is not tested because it falls outside the mask.

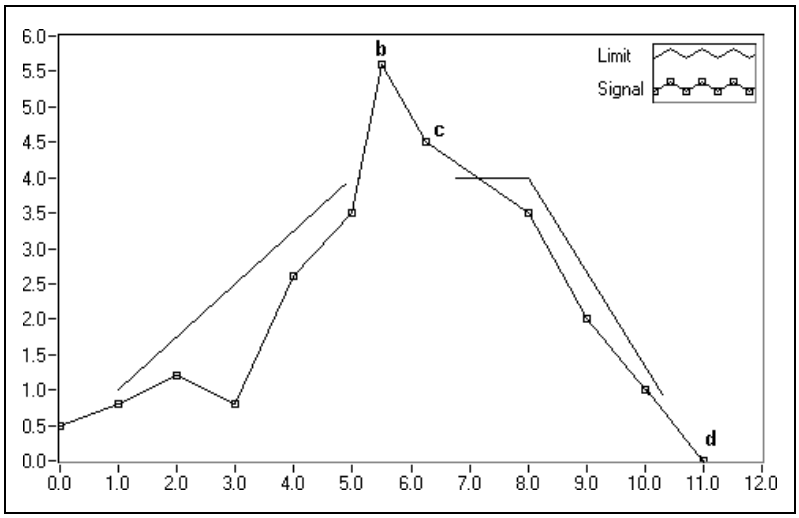


Figure 8-4. Result of Limit Testing with a Segmented Mask

Applications

You can use limit mask testing in a wide range of test and measurement applications. For example, you can use limit mask testing to determine that the power spectral density of ADSL signals meets the recommendations laid out in the ANSI T1-413 specification. Refer to the *Specifying a Limit Using a Formula* section of this chapter for more information about ADSL signal limits.

The following sections provide examples of when you can use limit mask testing. In all these examples, the specifications are recommended by standards-generating bodies, such as the CCITT, ITU-T, ANSI, and IEC, to ensure that all the test and measurement systems conform to a universally accepted standard. In some other cases, the limit testing specifications are proprietary and are strictly enforced by companies for quality control.

Modem Manufacturing Example

Limit testing is used in modem manufacturing to make sure the transmit spectrum of the line signal meets the V.34 modem specification as shown in Figure 8-5.

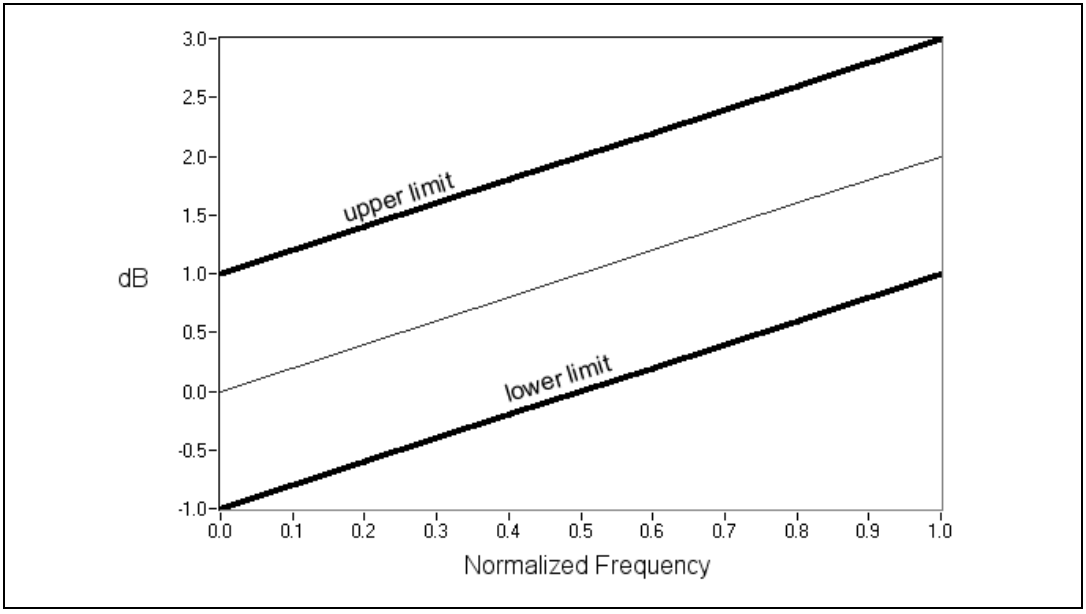


Figure 8-5. Upper and Lower Limit for V.34 Modem Transmitted Spectrum

The ITU-T V.34 recommendation contains specifications for a modem operating at data signaling rates up to 33,600 bits/s. It specifies that the spectrum for the line signal that transmits data conforms to the template shown in Figure 8-5. For example, for a normalized frequency of 1.0, the spectrum must always lie between 3 and 1 dB. All the modems must meet this specification. A modem manufacturer can set up an automated test system to monitor the transmit spectrum for the signals that the modem outputs. If the spectrum conforms to the specification, the modem passes the test and is ready for customer use. Recommendations such as the ITU-T V.34 are essential to ensure interoperability between modems from different manufacturers and to provide high-quality service to customers.

Digital Filter Design Example

You also can use limit mask testing in the area of digital filter design. You may want to design lowpass filters with a passband ripple of 10 dB and stopband attenuation of 60 dB. You can use limit testing to make sure the frequency response of the filter always meets these specifications. The first step in this process is to specify the limits. You can specify a lower limit of –10 dB in the passband region and an upper limit of –60 dB in the stopband region, as shown in Figure 8-6. After you specify these limits, you can run the actual test repeatedly to make sure that all the frequency responses of all the filters are designed meet these specifications.

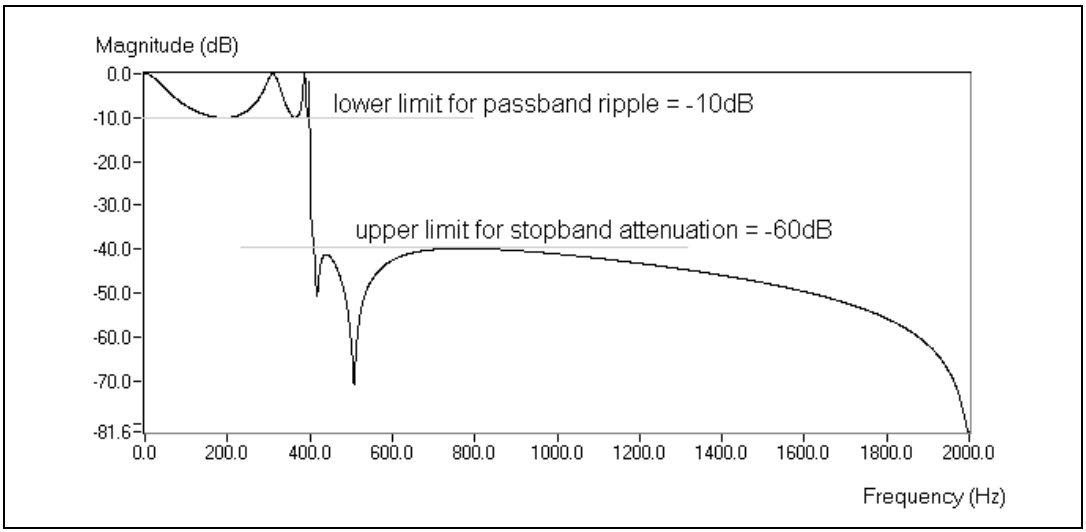


Figure 8-6. Limit Test of a Lowpass Filter Frequency Response

Pulse Mask Testing Example

The ITU-T G.703 recommendation specifies the pulse mask for signals with bit rates, $n \times 64$, where n is between 2 and 31. Figure 8-7 shows the pulse mask for interface at 1,544 kbits/s. Signals with this bit rate are also referred to as T1 signals. T1 signals must lie in the mask specified by the upper and lower limit. These limits are set to properly enable the interconnection of digital network components to form a digital path or connection.

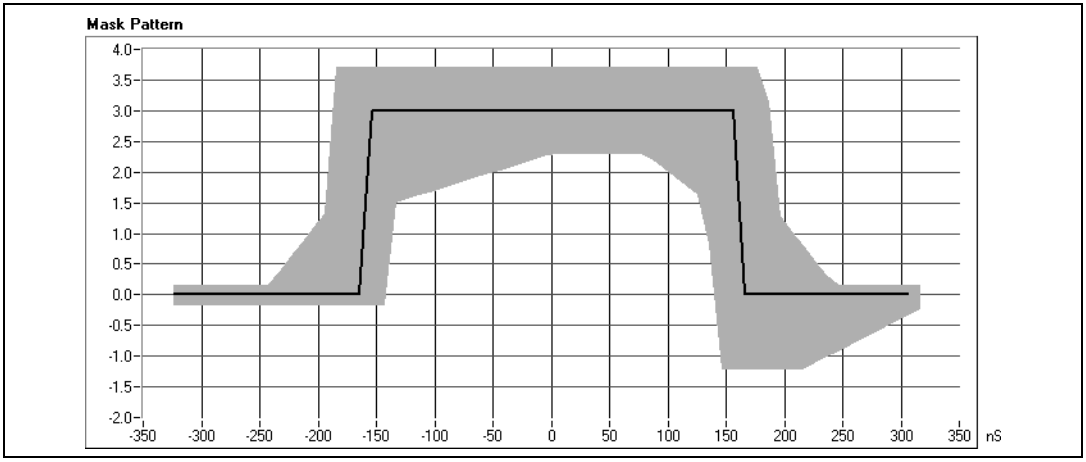


Figure 8-7. Pulse Mask Testing on T1/E1 Signals

9

Curve Fitting

This chapter describes how to extract information from a data set to obtain a functional description. Use the NI Example Finder, available by selecting **Help»Find Examples**, to find examples of how to use the regression VIs, available on the **Curve Fitting** palette.

Introduction to Curve Fitting

Curve fitting analysis is a technique for extracting a set of curve parameters or coefficients from the data set to obtain a functional description of the data set. The algorithm that fits a curve to a particular data set is known as the Least Squares Method and is discussed in most introductory textbooks in probability and statistics. The error is defined as

$$e(a) = [f(x,a) - y(x)]^2 \quad (9\text{-}1)$$

where $e(a)$ is the error, $y(x)$ is the observed data set, $f(x,a)$ is the functional description of the data set, and a is the set of curve coefficients which best describes the curve.

For example, let $a = \{a_0, a_1\}$. Then the functional description of a line is

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

The least squares algorithm finds a by solving the system

$$\frac{\partial}{\partial a} e(a) = 0 \quad (9\text{-}2)$$

To solve this system, you set up and solve the Jacobian system generated by expanding Equation 9-2. After you solve the system for a, you can obtain an estimate of the observed data set for any value of x using the functional description $f(x, a)$.

In LabVIEW, the curve fitting VIs automatically set up and solve the Jacobian system and return the set of coefficients that best describes your data set. You can concentrate on the functional description of your data and not worry about solving the system in Equation 9-2.

Two input sequences, *Y* Values and *X* Values, represent the data set $y(x)$. A sample or point in the data set is

$$(x_i, y_i)$$

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.

In general, for each predefined type of curve fit, there are two types of VIs, unless otherwise specified. One type returns only the coefficients, so that you can further manipulate the data. The other type returns the coefficients, the corresponding expected or fitted curve, and the mean squared error (MSE). Because it is a discrete system, the VI calculates the MSE, which is a relative measure of the residuals between the expected curve values and the actual observed values, using the formula

$$\text{MSE} = \frac{1}{n}\sum_{i=0}^{n-1}(f_i - y_i)^2 \qquad (9\text{-}3)$$

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 **Curve Fitting** palette offers both linear and nonlinear curve fitting algorithms. You can choose from the following types of curve fitting in LabVIEW:

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

$$y[i]=a_0 + a_1x[i]$$

- *Exponential Fit*—fits data to an exponential curve of the form $y = a^b$

$$y[i] = a_0^{a_1x[i]}$$

- *General Polynomial Fit*—fits data to a polynomial function of the form

$$y = a + bx + cx\text{^}2 + \ldots$$
$$y[i] = a_0 + a_1x[i]+a_2x[i]^2\ldots$$

- *General Linear Fit*—fits data to

$$y[i] = a_0 + a_1 f_1(x[i]) + a_2 f_2(x[i]) + \ldots$$

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

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

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.

Applications of Curve Fitting

The practical applications of curve fitting are numerous and include the following applications:

- 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).
- Obtaining the trajectory of an object based on discrete measurements of its velocity (first derivative) or acceleration (second derivative).

General LS Linear Fit Theory

The General LS Linear Fit Problem can be described as follows.

Given a set of observation data, find a set of coefficients that fit the linear "model."

$$y_i = b_o x_{i0} + \ldots + b_{k-1} x_{ik-1}$$

$$= \sum_{j=0}^{k-1} b_j x_{ij} \quad i = 0, 1, \ldots, n-1 \tag{9-4}$$

where B is the set of **Coefficients**, n is the number of elements in **Y Values** and the number of rows of **H**, and k is the number of **Coefficients**.

x_{ij} is your observation data, which is contained in **H**.

$$H = \begin{bmatrix} x_{00} & x_{01} \cdots & x_{0k-1} \\ x_{10} & x_{11} \cdots & x_{1k-1} \\ . & & \\ . & & \\ . & & \\ . & & \\ x_{n-10} & x_{n-12} \cdots & x_{n-1k-1} \end{bmatrix}$$

Equation 9-4 also can be written as $Y = HB$.

This is a multiple linear regression model, which uses several variables $x_{i0}, x_{i1}, \ldots, x_{ik-1}$, to predict one variable y_i. In contrast, the Linear Fit, Exponential Fit, and Polynomial Fit VIs are all based on a single predictor variable, which uses one variable to predict another variable.

In most cases, we have more observation data than coefficients. The equations in 9-4 may not have the solution. The fit problem becomes to find the coefficient *B* that minimizes the difference between the observed data, y_i and the predicted value:

$$z_i = \sum_{j=0}^{k-1} b_j x_{ij}$$

This VI uses the least chi-square plane method to obtain the coefficients in 9-4, that is, finding the solution, *B*, which minimizes the quantity:

$$\chi^2 = \sum_{i=0}^{n-1} \left(\frac{y_i - z_i}{\sigma_i} \right)^2 = \sum_{i=0}^{n-1} \left(\frac{y_i - \sum_{j=0}^{k-1} b_j x_{ij}}{\sigma_i} \right)^2 = |H_0 B - Y_0|^2 \qquad (9\text{-}5)$$

where

$$h_{oij} = \frac{x_{ij}}{\sigma_i}, \; y_{oi} = \frac{y_i}{\sigma_i}, \; i = 0, 1, \ldots, n-1; \; j = 0, 1, \ldots, k-1$$

In this equation, σ_i is the **Standard Deviation**. If the measurement errors are independent and normally distributed with constant standard deviation $\sigma_i = \sigma$, the preceding equation is also the least square estimation.

There are different ways to minimize χ^2. One way to minimize χ^2 is to set the partial derivatives of χ^2 to zero with respect to b_0, b_1, …, b_{k-1}.

$$\begin{cases} \frac{\partial \chi^2}{\partial b_0} = 0 \\ \frac{\partial \chi^2}{\partial b_1} = 0 \\ \vdots \\ \frac{\partial \chi^2}{\partial b_{k-1}} = 0 \end{cases}$$

The preceding equations can be derived to:

$$H_0^T H_0 B = H_0^T Y \tag{9-6}$$

Where H_0^T is the transpose of $\boldsymbol{H}_0$.

The equations in 9-6 are also called normal equations of the least-square problems. You can solve them using LU or Cholesky factorization algorithms, but the solution from the normal equations is susceptible to roundoff error.

An alternative, and preferred way to minimize χ^2 is to find the least-square solution of equations

$$H_0 B = Y_0$$

You can use QR or SVD factorization to find the solution, *B*. For QR factorization, you can choose Householder, Givens, and Givens2 (also called fast Givens).

Different algorithms can give you different precision, and in some cases, if one algorithm cannot solve the equation, perhaps another algorithm can. You can try different algorithms to find the best one based on your observation data.

The **Covariance** matrix *C* is computed as

$$C = (H_0^T H_0)^{-1}$$

The **Best Fit** Z is given by

$$z_i = \sum_{j=0}^{k-1} b_j x_{ij}$$

The mse is obtained using the following formula:

$$\boldsymbol{mse} = \frac{1}{n} \sum_{i=0}^{n-1} \left(\frac{y_i - z_i}{\sigma_i} \right)^2$$

The polynomial fit that has a single predictor variable can be thought of as a special case of multiple regression. If the observation data sets are $\{x_i, y_i\}$ where $i = 0, 1, \ldots, n-1$, the model for polynomial fit is

$$y_i = \sum_{j=0}^{k-1} b_j x_i^j = b_0 + b_1 x_i + b_2 x_i^2 + \ldots + b_{k-1} x_i^{k-1} \qquad (9\text{-}7)$$

$$i = 0, 1, 2, \ldots, n-1$$

Comparing equations 9-4 and 9-7 shows that $x_{ij} = x_i^j$. In other words,

$$\begin{aligned} x_{i0} &= x_i^0 \\ &= 1 \end{aligned}, \; x_{i1} = x_i, \; x_{i2} = x_i^2, \ldots \; x_{ik-1} = x_i^{k-1}$$

In this case, you can build $\boldsymbol{H}$ as follows:

$$H = \begin{Bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{k-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{k-1} \\ . & & & & \\ . & & & & \\ . & & & & \\ . & & & & \\ 1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{k-1} \end{Bmatrix}$$

Instead of using $x_{ij} = x_j^i$, you also can choose another function formula to fit the data sets $\{x_i, y_i\}$. In general, you can select $x_{ij} = f_j(x_i)$. Here, $f_j(x_i)$ is the function model that you choose to fit your observation data. In polynomial fit, $f_j(x_i) = x^j$.

In general, you can build $\boldsymbol{H}$ as follows:

$$H = \begin{Bmatrix} f_0(x_0) & f_1(x_0) & f_2(x_0) & \cdots & f_{k-1}(x_0) \\ f_0(x_1) & f_1(x_1) & f_2(x_1) & \cdots & f_{k-1}(x_1) \\ \cdot & & & & \\ \cdot & & & & \\ \cdot & & & & \\ \cdot & & & & \\ f_0(x_{n-1}) & f_1(x_{n-1}) & f_2(x_{n-1}) & \cdots & f_{k-1}(x_{n-1}) \end{Bmatrix}$$

Your fit model is:

$$y_i = b_0 f_0(x) + b_1 f_1(x) + \ldots + b_{k-1} f_{k-1}(x)$$

How to Use the General LS Linear Fit VI

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 a_1 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

$$y = a_0 + a_1 x^2 + a_2\cos(\omega x^2)$$

$$y = a_0 + a_1(3\sin(\omega x)) + a_2 x^3 + \frac{a_3}{x} + \ldots$$

In each case, notice 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 in Figure 9-1.

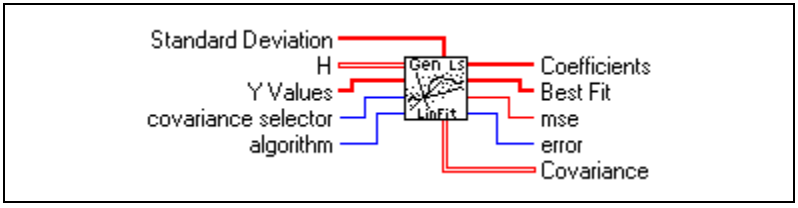


Figure 9-1. General LS Linear Fit VI

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. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for more information about 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 is a different function. For example, a_0 is multiplied by 1, a_1 is multiplied by $\sin(\omega x)$, a_2 is multiplied by $\cos(\omega x)$, 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:

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

If you have N data points and k coefficients ($a_0, a_1, \ldots a_{k-1}$) for which to solve, ***H*** will be an $N \times k$ 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**, use the General LS Linear Fit VI to build ***H***.

Nonlinear Lev-Mar Fit Theory

The Nonlinear Lev-Mar Fit VI determines the set of coefficients that minimize the chi-square quantity:

$$\chi^2 = \sum_{i=0}^{N-1} \left(\frac{y_i - f(x_i; a_1 \ldots a_M)}{\sigma_i} \right)^2 \quad (9\text{-}8)$$

In this equation, (x_i, y_i) are the input data points, and $f(x_i; a_1 \ldots a_M) = f(X, A)$ is the nonlinear function where $a_1 \ldots a_M$ are coefficients. If the measurement errors are independent and normally distributed with constant, standard deviation $\sigma_i = \sigma$, this is also the least-square estimation.

You must specify the nonlinear function $f = f(X, A)$ in the Formula Node on the block diagram of the Target Fnc & Deriv NonLin VI, which is a subVI of the Nonlinear Lev-Mar Fit VI.

This VI provides two ways to calculate the Jacobian (partial derivatives with respect to the coefficients) needed in the algorithm. These two methods follow:

- Numerical calculation—Uses a numerical approximation to compute the Jacobian.
- Formula calculation—Uses a formula to compute the Jacobian. You need to specify the Jacobian function $\partial f / \partial A$ in the Formula Node on the block diagram of the Target Fnc & Deriv NonLin VI, as well as the nonlinear function $f = f(X, A)$. This is a more efficient computation than the numerical calculation, because it does not require a numerical approximation to the Jacobian.

The input arrays **X** and **Y** define the set of input data points. The VI assumes that you have prior knowledge of the nonlinear relationship between the x and y coordinates. That is, $f = f(X, A)$, where the set of coefficients, A, is determined by the Levenberg-Marquardt algorithm.

Using this function successfully sometimes depends on how close your initial guess coefficients are to the solution. Therefore, it is always worth taking effort and time to obtain good initial guess coefficients to the solution from any available resources before using the function.

Using the Nonlinear Lev-Mar Fit VI

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 $\boldsymbol{A} = \{a_0, a_1, a_2, \ldots, a_k\}$ of the nonlinear relationship between $\boldsymbol{A}$ 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.

When using the Nonlinear Lev-Mar Fit VI, you also must 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 in Figure 9-2:

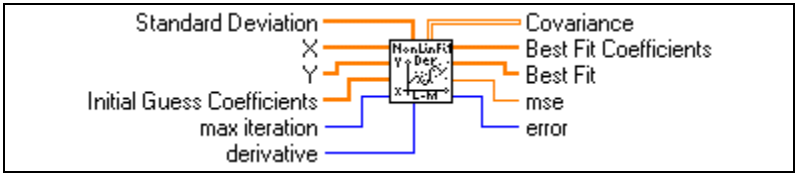


Figure 9-2. Nonlinear Lev-Mar Fit VI

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. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for more information about these inputs.

10

Linear Algebra

This chapter explains how to use the linear algebra VIs to perform matrix computation and analysis. Use the NI Example Finder, available by selecting **Help»Find Examples**, to find examples of how to use the linear algebra VIs, available on the **Linear Algebra** palette.

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 $(i,j)^{\text{th}}$ 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*.

Determinant of a Matrix

One of the most important attributes of a matrix is its *determinant*. In the simplest case, the determinant of a 2×2 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

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

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

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

$$2(-33) - 5(47) + 3(35) = -196$$

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*. Refer to the *Matrix Inverse and Solving Systems of Linear Equations* section of this chapter for more information about singularity and 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 A^T, then $b_{j,i} = a_{i,j}$. For the matrix $\boldsymbol{A}$ defined above,

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

In the case of complex matrices, we define complex conjugate transposition. If the matrix D represents the *complex conjugate transpose* (if $a = x + iy$, then complex conjugate $a^* = x - iy$) 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.

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*.

Linear Independence

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

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

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 $\alpha_1 = 0, \alpha_2 = 0, \ldots, \alpha_n = 0$, then the set of vectors $x_1, x_2, \ldots, x_n$ 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 $\alpha_1 = 0, \alpha_2 = 0, \ldots, \alpha_n = 0$. Therefore, to show the linear independence of the set, you must show that $\alpha_1 = 0, \alpha_2 = 0, \ldots, \alpha_n = 0$ 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. Let us now look at 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.

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 4×4 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

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

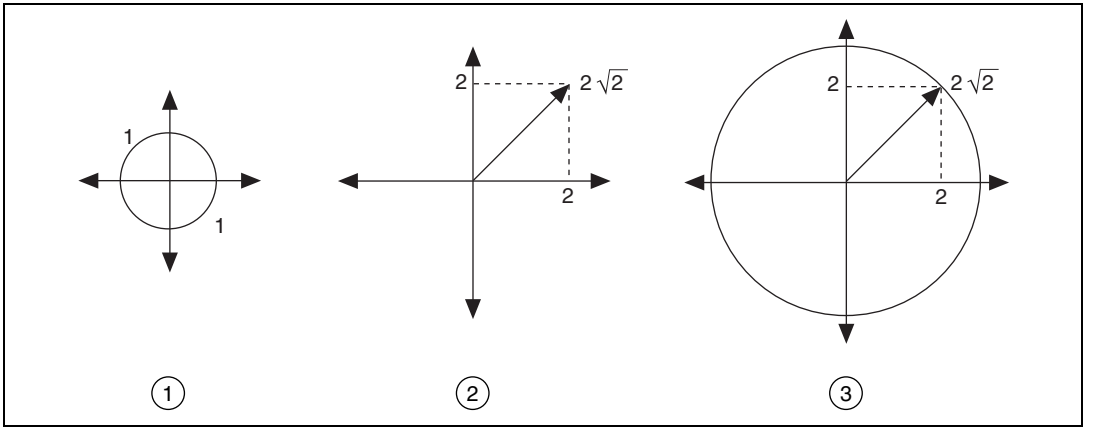
Hence, the rank of B is 3. A square matrix has full rank if and only if its determinant is different from zero. Matrix B is not a full-rank matrix.

"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 $x = [x1\ x2]$ and $y = [y1\ y2]$, 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 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 in order to encompass the given vector exactly. This is shown in Figure 10-1.



1 Unit ball of radius = 1 unit

2 Vector of length $\sqrt{2^2 + 2^2} = \sqrt{8} = 2\sqrt{2}$

3 Unit ball expanded by a factor of $2\sqrt{2}$

Figure 10-1. Euclidean Norm of a Vector

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.

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

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

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

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,

$$\|A\|_\infty = max(4, 6) = 6$$

The 2-norm is the most difficult to compute because it is given by the largest singular value of the matrix. Refer to the *Matrix Factorization* section of this chapter for more information about singular values.

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

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

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 of the matrix A, denoted by A^{-1}, and then multiply them together (the inverse of a square matrix A is a square matrix B such

that $AB = I$, where I is the identity matrix). As mentioned earlier, the 2-norm is difficult to calculate on paper. You can use the Matrix Norm VI to compute the 2-norm. For example,

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

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 47,168, 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 (a *permutation matrix* is an identity matrix with some rows and columns exchanged). 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.

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 $m \times n$ 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 $m \times p$ 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 i^{th} row and the j^{th} column of *C*, multiply the elements in the i^{th} row of *A* by the corresponding elements in the j^{th} column of *C*, and then add them all. This is shown pictorially in Figure 10-2.

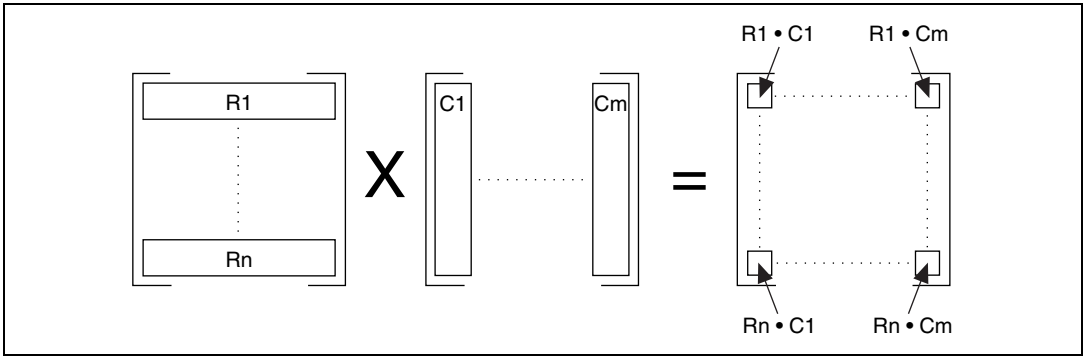


Figure 10-2. Matrix Multiplication

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*. Notice 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, illustrated in Figure 10-3.

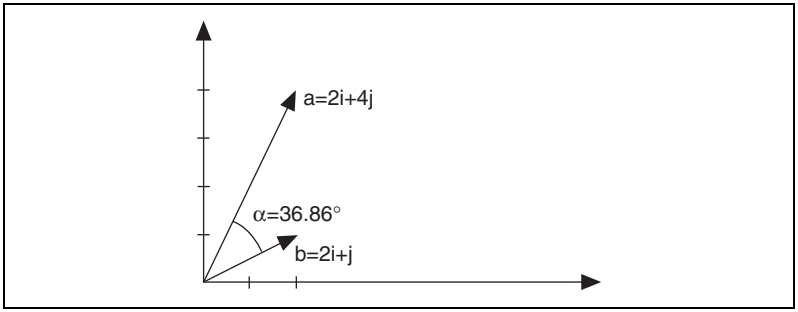


Figure 10-3. Vectors *a* and *b*

Then the dot product of these two vectors is given by

$$d = \begin{bmatrix} 2 \\ 4 \end{bmatrix} \bullet \begin{bmatrix} 2 \\ 1 \end{bmatrix} = (2 \times 2) + (4 \times 1) = 8$$

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, as shown in Figure 10-4. 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,

$$W = |a||d|\cos\alpha = a \bullet d$$

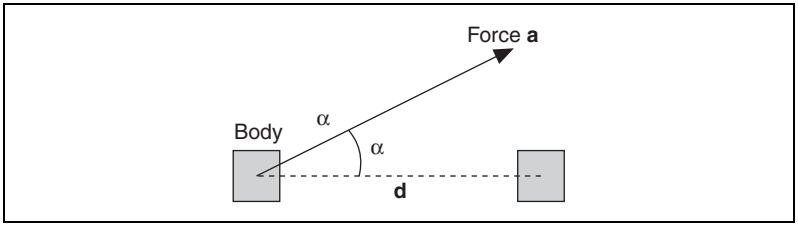


Figure 10-4. Force Vector

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 vector x. For any constant α, the vector αx is also an eigenvector with eigenvalue λ, because

$$A(\alpha x) = \alpha Ax = \lambda \alpha x$$

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.
- 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 EigenValues and Vectors VI is shown in Figure 10-5. The **Input Matrix** is an $N \times N$ real square matrix. **Matrix type** determines the type of the input matrix. **Matrix type** could be *0*, indicating a general matrix, or 1, indicating a symmetric matrix. A symmetric matrix always has real

eigenvalues and eigenvectors. A general matrix has no special property such as symmetry or triangular structure.

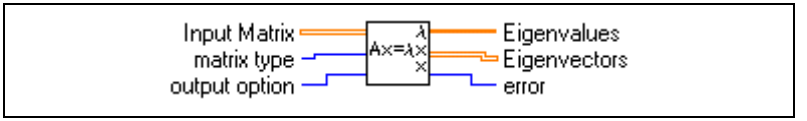


Figure 10-5. EigenValues and Vectors VI

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 nonsymmetric matrix. So, choose the matrix type control carefully.

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. Refer to the *Matrix Factorization* section of this chapter for more information about the pseudoinverse of a rectangular matrix.

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-hand-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-hand-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 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 $\boldsymbol{x}$ can be easily determined and then used to determine the other elements sequentially. Hence, the name back-substitution. So far, this chapter 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 **Linear Algebra** palette provides 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, and lower and upper triangular matrices. A real matrix is positive definite if and only if it is symmetric and the quadratic form for all nonzero vectors is X. 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 which minimizes the norm of $Ax - b$. The same holds true also for nonsquare matrices.

Matrix Factorization

The *Matrix Inverse and Solving Systems of Linear Equations* section discusses 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 $A = U^T U$, 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. It is easy to determine if a matrix is positive definite by using the Test Positive Definite VI.

A matrix Q is *orthogonal* if its columns are *orthonormal*. That is, if $Q^T Q = I$, 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 $A^T A$, and the columns of U and V, which are called left and right singular vectors, are orthonormal eigenvectors of AA^T and $A^T A$, respectively. SVD is useful for solving analysis problems such as computing the rank, norm, condition number, and pseudoinverse of matrices. The *Pseudoinverse* section discusses this last application.

Pseudoinverse

The pseudoinverse of a scalar σ is defined as $1/\sigma$ if $\sigma \neq 0$, and zero otherwise. In case of scalars, pseudoinverse is the same as the inverse. You can now define the pseudoinverse of a diagonal matrix by transposing the matrix and then taking the scalar pseudoinverse of each entry. Then the pseudoinverse of a general real $m \times n$ matrix A, denoted by $A^\dagger$, is given by

$$A^\dagger = VS^\dagger U^T$$

Notice that the pseudoinverse exists regardless of whether the matrix is square or rectangular. If A is square and nonsingular, then the

pseudoinverse is the same as the usual matrix inverse. The **Linear Algebra** palette includes a VI for computing the pseudoinverse of real and complex matrices.

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.

11

Probability and Statistics

This chapter explains some fundamental concepts on probability and statistics and shows how to use these concepts in solving real-world problems. Use the NI Example Finder, available by selecting **Help»Find Examples**, to find examples of how to use the probability and statistics VIs, available on the **Probability and Statistics** palette.

Probability and Statistics

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 has an average of 30 points a game this season," and so on are common. These statements give a lot of information, but we seldom think 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, or data, is generated. Consider a sports player. In a season, the player is in 51 games and scores a total of 1,568 points. The total of 1,568 points includes 45 points in Game A, 36 points in Game B, 51 points in Game C, 45 points in Game D, and 40 points in Game E. As the number of games increases, it becomes increasingly difficult to remember how many points the player scored in each individual game. The question is how to condense the 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. If you divide the total number of points that the player scored by the number of games played, you have a single number and can call it points per game *average*, as shown in the following equation.

$$\frac{1{,}568 \text{ points}}{51 \text{ games}} = 30.7 \text{ points per game average}$$

Another important aspect of statistics is *percentage*. For example, the officials of an American city are considering installing a traffic signal at a major intersection. The traffic signal is intended to protect motorists turning left from oncoming traffic. However, the city only has enough money to fund one traffic signal but has three intersections that potentially need the signal. Each of the three intersections is studied for a week. A count is kept of the total number of cars using the intersection, the number of cars travelling straight through the intersection, the number of cars making left-hand turns, and the number of cars making right-hand turns. The data collected for one of the intersections is shown in Table 11-1.

Table 11-1. Data for One Major Intersection

Day	Total Number of Cars Using the Intersection	Number of Cars Turning Left	Number of Cars Turning Right	Number of Cars Continuing Straight
1	1,258	528	330	400
2	1,306	549	340	417
3	1,355	569	352	434
4	1,227	515	319	393
5	1,334	560	346	428
6	694	291	180	223
7	416	174	108	134
Totals	7,590	3,186	1,975	2,429

Looking only at the raw data from each intersection might make determining which intersection needs the traffic signal difficult, because the raw numbers can vary widely. However, computing the percentage of cars turning at each intersection provides a common basis for comparison. To obtain the percentage of cars turning left, divide the number of cars turning left by the total number of cars using the intersection and multiply that result by 100. For the intersection whose data is shown in Table 11-1, the following equation gives the percentage of cars turning left.

$$\frac{3{,}186}{7{,}590} \times 100 = 42\%$$

For the intersection represented by the data in Table 11-1, 42% of the cars using that intersection turn left. Given the data for the other two intersections, the city officials can obtain the percentage of cars turning left at those two intersections. Thus, you can condense the information for the three intersections into single numbers representing the percentage of cars that turn left at each intersection. The city officials can compare the percentage of cars turning left at each intersection and rank the intersections in order of highest percentage of cars turning left to the lowest percentage of cars turning left. Ranking the intersections can help determine where the traffic signal is needed most. 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, comparing the percentage of cars turning left at the three different intersections can help city officials decide where to place a traffic signal. 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, it is *probable* that the remaining two intersections will need traffic signals in the future, you can use different concepts in the field of probability, as discussed later in this chapter, 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.

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, …, x_{n-1}. The mean value, or 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 in the sports example, the data set consisted of 51 samples. Each sample was equal to the number of points that the player scored in each game. The total of all these points was 1,568, 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

$$i = \frac{n-1}{2} \text{ and } k = \frac{n}{2}$$

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."

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.

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 Sample Variance VI computes sample variance, whereas the Variance VI computes the population variance. Statisticians and mathematicians prefer to use the latter, engineers the former. It really does not matter for large values of n, say $n \geq 30$.

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.

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*.

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. Notice that for $m = 2$, the moment about mean is equal to the population variance.

Histogram

So far, this chapter 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. 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.

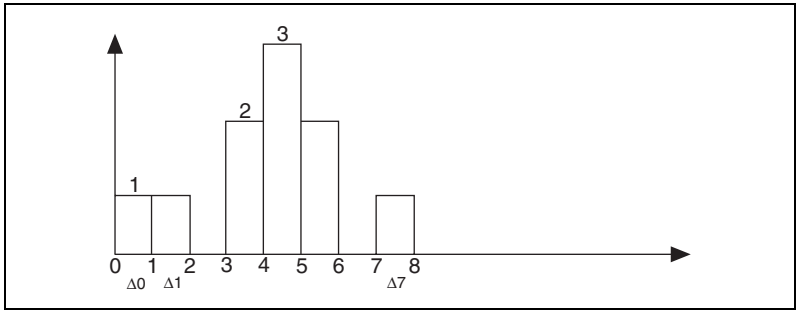


Figure 11-1. Histogram

Figure 11-1 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 histogram. Next you will see how it is done in the Histogram VI using the sequence *X*.

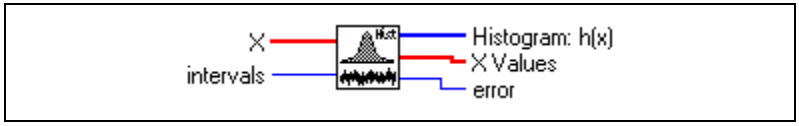


Figure 11-2. Histogram VI

As shown in Figure 11-2, 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

$$\chi_i = min + 0.5\Delta x + i\Delta x \quad \text{for } i = 0, 1, 2, \ldots, m - 1$$

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 $\chi_i + 0.5\Delta x$,

$$\Delta_i = [(\chi_i - 0.5\Delta x), (\chi_i + 0.5\Delta x)] \quad \text{for } i = 0, 1, 2, \ldots, m - 1$$

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

$$h_i = \sum_{j=0}^{n-1} y_i(x_j) \qquad \text{for } i = 0, 1, 2, \ldots, m-1$$

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, $h_0 = 1$, $h_4 = 3$, …, $h_7 = 1$.

The General Histogram VI is more advanced than the Histogram VI. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for more information about the probability and statistics VIs.

Mean Square Error (MSE)

If X and Y represent two input sequences, then 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, S_1. The output of this system is y_1. Now you acquire a new system, S_2, which is theoretically known to generate the same result as S_1 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 y_1 and y_2 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 y_1 and y_2. If the mse is smaller than an acceptable tolerance, then the system S_1 can be reliably replaced by the new system S_2.

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 $\frac{V_p}{\sqrt{2}}$.

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 following: 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.

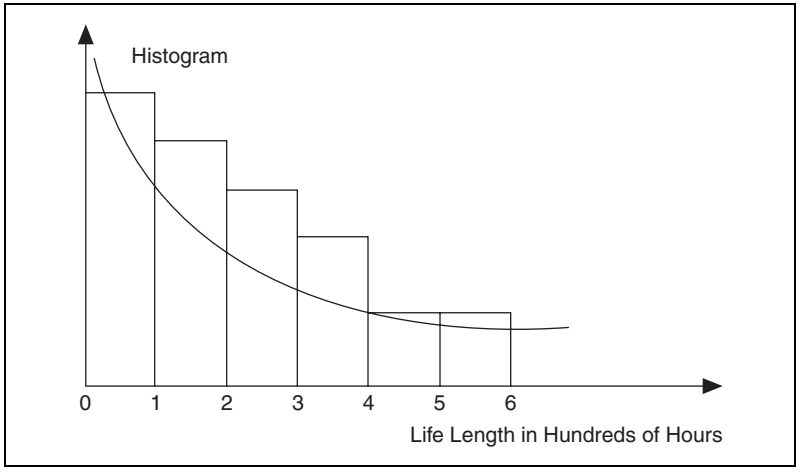


Figure 11-3. Life Lengths Histogram

Figure 11-3 shows that most of the life lengths are between zero and 100 hours, and the histogram values drop off smoothly when 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 four hundred 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 preceding information 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

$$X = \text{a}, P(X = a) = \int_{a}^{a} f(x)dx = 0$$

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.

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

$$f(x) = \frac{1}{\sqrt{2\pi}s} e^{-(x-\bar{x})^2/(2s^2)}$$

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*.

$$p = Prob(X \leq x)$$

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 1,000 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 1,000 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. 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. The **Probability and Statistics** palette has VIs that compute the one-sided probability for these different types of distributions. It also has VIs that perform the inverse operation.

12

Point-By-Point Analysis

This chapter discusses the concepts of point-by-point analysis, answers frequently asked questions about point-by-point analysis, and provides a case study designed to illustrate the use of the Point By Point VIs. Use the NI Example Finder, available by selecting **Help»Find Examples**, to find examples of how to use the Point By Point VIs.

Introduction to Point-By-Point Analysis

Point-by-point analysis is a method of continuous data analysis in which analysis occurs for each data point, point by point. Point-by-point analysis is ideally suited to real-time data acquisition. When your data acquisition system requires real-time, deterministic performance, you can build a program that uses point-by-point versions of array-based LabVIEW analysis VIs.

Real-time performance is a reality for data acquisition. With point-by-point analysis, data analysis also can utilize real-time performance. The discrete stages of array-based analysis, such as buffer preparation, analysis, and output, can make array-based analysis too slow for higher speed, deterministic, real-time systems.

Point-by-point analysis enables you to accomplish the following tasks:

- Track and respond to real-time events.
- Connect the analysis process directly to the signal for speed and minimal data loss.
- Perform programming tasks more easily, because you do not allocate arrays and you make fewer adjustments to sampling rates.
- Synchronize analysis with data acquisition automatically, because you work with a single signal instantaneously.

Using the Point By Point VIs

The Point By Point VIs correspond to each array-based analysis VI that is relevant to continuous data acquisition. However, you must account for programming differences. You usually have fewer programming tasks when you use the Point By Point VIs. Table 12-1 describes characteristic inputs and outputs of the Point By Point VI s.

Table 12-1. Characteristic Inputs and Outputs for Point By Point VIs

Parameter	Description
input data	Incoming data
output data	Outgoing, analyzed data
initialize	Routine that resets the internal state of a VI
sample length	Setting for your data acquisition system or computation system that best represents the area of interest in the data

Refer to the *Case Study of Point-By-Point Analysis* section of this chapter for an example of a point-by-point analysis system.

Initializing Point By Point VIs

This section describes when and how to use the point-by-point **initialize** parameter in many Point By Point VIs. This section also describes the LabVIEW First Call? function.

Purpose of Initialization in Point By Point VIs

Using the **initialize** parameter, you can reset the internal state of Point By Point VIs without interrupting the continuous flow of data or computation. You can reset a VI in response to events such as the following:

- A user changing the value of a parameter
- The application generating a specific event or reaching a threshold

For example, The Value Has Changed PtByPt VI can respond to change events such as the following:

- Receiving the input data
- Detecting the change
- Generating a Boolean TRUE value that triggers initialization in another VI
- Transferring the input data to another VI for processing

Figure 12-1 shows the Value Has Changed PtByPt VI triggering initialization in another VI and transferring data to that VI. In this case, the input data is a parameter value for the target VI.

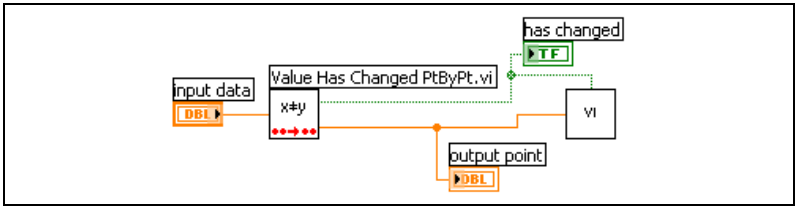


Figure 12-1. Typical Role of the Value Has Changed PtByPt VI

Many point-by-point applications do not require use of the **initialize** parameter because initialization occurs automatically whenever an operator quits an application and then starts again.

Using the First Call? Function

Where necessary, use the First Call? function to build point by point VIs. In a VI that includes the First Call? function, the internal state of the VI is reset once, the first time you call the VI. The value of the **initialize** parameter in the First Call? function is always TRUE for the first call to the VI. The value remains FALSE for the remainder of the time you run the VI. Figure 12-2 shows a typical use of the First Call? function with a While Loop.

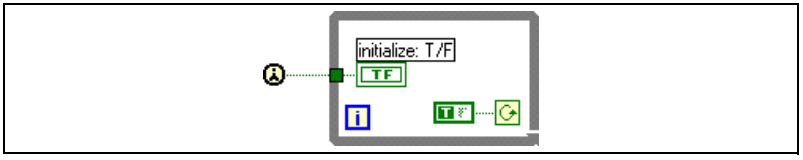


Figure 12-2. Using the First Call? Function with a While Loop

Error Checking and Initialization

The Point By Point VIs generate errors to help you identify flaws in the configuration of the applications you build. Several point-by-point error codes exist in addition to the standard LabVIEW error codes.

Error codes usually identify invalid parameters and settings. For higher-level error checking, configure your program to monitor and respond to irregularities in data acquisition or in computation. For example, you create a form of error checking when you range check your data.

A Point By Point VI generates an error code once at the initial call to the VI or at the first call to the VI after you initialize your application. Because Point By Point VIs generate error codes only once, they can perform optimally in a real-time, deterministic application.

The Point By Point VIs generate an error code to inform you of any invalid parameters or settings when they detect an error during the first call. In subsequent calls, the Point By Point VIs set the error code to zero and continue running, generating no error codes. You can program your application to take one of the following actions in response to the first error:

- Report the error and continue running.
- Report the error and stop.
- Ignore the error and continue running. This is the default behavior.

The following programming sequence describes how to use the Value Has Changed PtByPt VI to build a point-by-point error checking mechanism for Point By Point VIs that have an **error** parameter.

1. Choose a parameter that you want to monitor closely for errors.
2. Wire the parameter value as **input data** to the Value Has Changed PtByPt VI.
3. Transfer the **output data**, which is always the unchanged **input data** in Value Has Changed PtByPt VI, to the target VI.
4. Pass the TRUE event generated by the Value Has Changed PtByPt VI to the target VI to trigger initialization, as shown in Figure 12-1. The Value Has Changed PtByPt VI outputs a TRUE value whenever the input parameter value changes.

For the first call that follows initialization of the target VI, LabVIEW checks for errors. Initialization of the target VI and error checking occurs every time the input parameter changes.

Frequently Asked Questions

This section answers frequently asked questions about point-by-point analysis.

What Are the Differences between Point-By-Point Analysis and Array-Based Analysis in LabVIEW?

Tables 12-2 and 12-3 compare array-based LabVIEW analysis to point-by-point analysis from multiple perspectives. In Table 12-2, the differences between two automotive fuel delivery systems, carburation and fuel injection, demonstrate the differences between array-based data analysis and point-by-point analysis.

Table 12-2. Comparison of Traditional and Newer Paradigms

Traditional Paradigm	Newer Paradigm
Automotive Technology	
Carburation • Fuel accumulates in a float bowl. • Engine vacuum draws fuel through a single set of metering valves that serve all combustion chambers. • Somewhat efficient combustion occurs.	**Fuel Injection** • Fuel flows continuously from gas tank. • Fuel sprays directly into each combustion chamber at the moment of combustion. • Responsive, precise combustion occurs.
Data Analysis Technology	
Array-Based Analysis • Prepare a buffer unit of data. • Analyze data. • Produce a buffer of analyzed data. • Generate report.	**Point-By-Point Analysis** • Receive continuous stream of data. • Filter and analyze data continuously. • Generate real-time events and reports continuously.

Table 12-3 presents other comparisons between array-based and point-by-point analysis.

Table 12-3. Comparison of Array-Based and Point-By-Point Data Analysis

	Array-Based Analysis	Data Acquisition and Analysis with Point By Point VIs
Compatibility	Limited compatibility with real-time systems	Compatible with real-time systems; backward compatible with array-based systems
Data typing	Array-oriented	Scalar-oriented
Interruptions	Interruptions critical	Interruptions tolerated
Operation	You observe, offline	You control, online
Performance and programming	Compensate for start-up data loss (4–5 seconds) with complex “state machines”	Start-up data loss does not occur; initialize the data acquisition system once and run continuously
Point of view	Reflection of a process, like a mirror	Direct, natural flow of a process
Programming	Specify a buffer	No explicit buffers
Results	Output a report	Output a report and an event in real time
Run-time behavior	Delayed processing	Real time
Run-time behavior	Stop	Continue
Run-time behavior	Wait	Now
Work style	Asynchronous	Synchronous

Why Use Point-By-Point Analysis?

Point-by-point analysis works well with computer-based real-time data acquisition. In array-based analysis, the input-analysis-output process takes place for subsets of a larger data set. In point-by-point analysis, the input-analysis-output process takes place continuously, in real time.

What is New about Point-By-Point Analysis?

When you perform point-by-point analysis, keep in mind the following concepts:

- Initialization—You must initialize the point-by-point analysis application to prevent interference from settings you made in previous sessions of data analysis.
- Re-Entrant Execution—You must enable LabVIEW re-entrant execution for point-by-point analysis. Re-entrant execution allocates fixed memory to a single analysis process, guaranteeing that two processes that use the same analysis function never interfere with each other.

Note If you create custom VIs to use in your own point-by-point application, be sure to enable re-entrant execution. Re-entrant execution is enabled by default in almost all Point By Point VIs.

- Deterministic Performance—Point-by-point analysis is the natural companion to many deterministic systems, because it efficiently integrates with the flow of a real-time data signal.

What Is Familiar about Point-By-Point Analysis?

The approach used for most point-by-point analysis operations in LabVIEW remains the same as array-based analysis. You use filters, integration, mean value algorithms, and so on, in the same situations and for the same reasons that you use these operations in array-based data analysis. In contrast, the computation of zeroes in polynomial functions is not relevant to point-by-point analysis, and point-by-point versions of these array-based VIs are not necessary.

How is it Possible to Perform Analysis without Buffers of Data?

Analysis functions yield solutions that characterize the behavior of a data set. In array-based data acquisition and analysis, you might analyze a large set of data by dividing the data into 10 smaller buffers. Analyzing those 10 sets of data yields 10 solutions. You can further resolve those 10 solutions into one solution that characterizes the behavior of the entire data set.

In point-by-point analysis, you analyze an entire data set in real-time. A sample unit of a specific length replaces a buffer. The point-by-point sample unit can have a length that matches the length of a significant event in the data set that you are analyzing. For example, the application in

the *Case Study of Point-By-Point Analysis* section acquires a few thousand samples per second to detect defective train wheels. The input data for the train wheel application comes from the signal generated by a train that is moving at 60 to 70 km per hour. The sample length corresponds to the minimum distance between wheels.

A typical point-by-point analysis application analyzes a long series of sample units, but you are likely to have interest in only a few of those sample units. To identify those crucial samples of interest, the point-by-point application focuses on transitions, such as the end of the relevant signal.

The train wheel detection application in the *Case Study of Point-By-Point Analysis* section uses the end of a signal to identify crucial samples of interest. The instant the application identifies the transition point, it captures the maximum amplitude reading of the current sample unit. This particular amplitude reading corresponds to the complete signal for the wheel on the train whose signal has just ended. You can use this real-time amplitude reading to generate an event or a report about that wheel and that train.

Why is Point-By-Point Analysis Effective in Real-Time Applications?

In general, when you must process continuous, rapid data flow, point-by-point analysis can respond. For example, in industrial automation settings, control data flows continuously, and computers use a variety of analysis and transfer functions to control a real-world process. Point-by-point analysis can take place in real time for these engineering tasks.

Some real-time applications do not require high-speed data acquisition and analysis. Instead, they require simple, dependable programs. Point-by-point analysis offers simplicity and dependability, because you do not allocate arrays explicitly, and data analysis flows naturally and continuously.

Do I Need Point-By-Point Analysis?

As you increase the samples-per-seconds rate by factors of ten, the need for point-by-point analysis increases. The point-by-point approach simplifies the design, implementation, and testing process, because the flow of a point-by-point application closely matches the natural flow of the real-world processes you want to monitor and control.

You can continue to work without point-by-point analysis as long as you can control your processes without high-speed, deterministic, point-by-point data acquisition. However, if you dedicate resources in a real-time data acquisition application, use point-by-point analysis to achieve the full potential of your application.

What is the Long-Term Importance of Point-By-Point Analysis?

Real-time data acquisition and analysis continue to demand more streamlined and stable applications. Point-by-point analysis is streamlined and stable because it directly ties into the acquisition and analysis process. Streamlined and stable point-by-point analysis allows the acquisition and analysis process to move closer to the point of control in field programmable gate array (FPGA) chips, DSP chips, embedded controllers, dedicated CPUs, and ASICs.

Case Study of Point-By-Point Analysis

The case study in this section uses the Train Wheel PtByPt VI and shows a complete point-by-point analysis application built in LabVIEW with Point By Point VIs. The Train Wheel PtByPt VI is a real-time data acquisition application that detects defective train wheels and demonstrates the simplicity and flexibility of point-by-point data analysis. You can find the Train Wheel PtByPt VI in the `examples\ptbypt\PtByPt_No_HW.llb`.

Point-By-Point Analysis of Train Wheels

In this example, the maintenance staff of a train yard must detect defective wheels on a train. The current method of detection consists of a railroad worker striking a wheel with a hammer and listening for a different resonance that identifies a flaw. Automated surveillance must replace manual testing, because manual surveillance is too slow, too prone to error, and too crude to detect subtle defects. An automated solution also adds the power of dynamic testing, because the train wheels can be in service during the test, instead of standing still.

The automated solution to detect potentially defective train wheels needs to have the following characteristics:

- Detect even subtle signs of defects quickly and accurately.
- Gather data when a train travels during a normal trip.
- Collect and analyze data in real time to simplify programming and to increase speed and accuracy of results.

The Train Wheel PtByPt VI offers a solution for detecting defective train wheels. Figures 12-3 and 12-4 show the front panel and the block diagram, respectively, for the Train Wheel PtByPt VI.

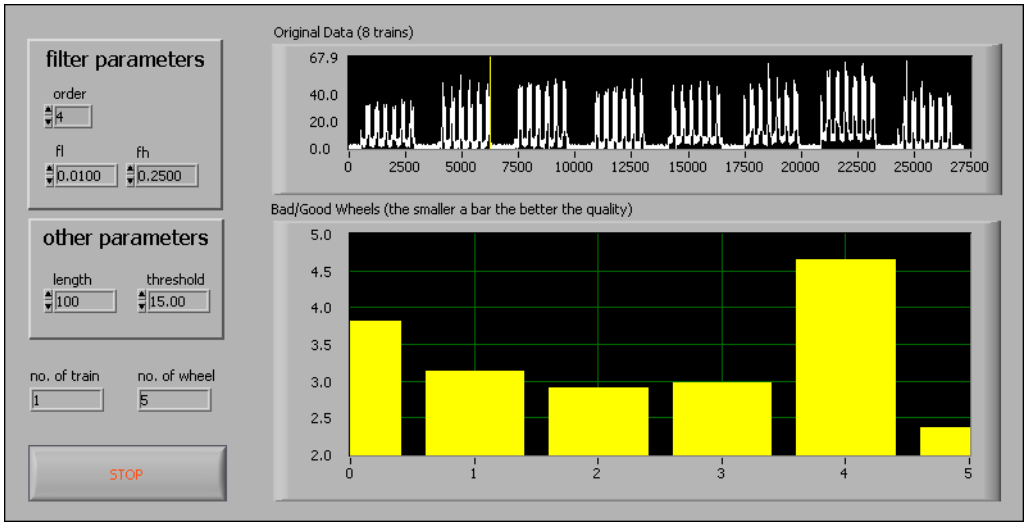


Figure 12-3. Front Panel of the Train Wheel PtByPt VI

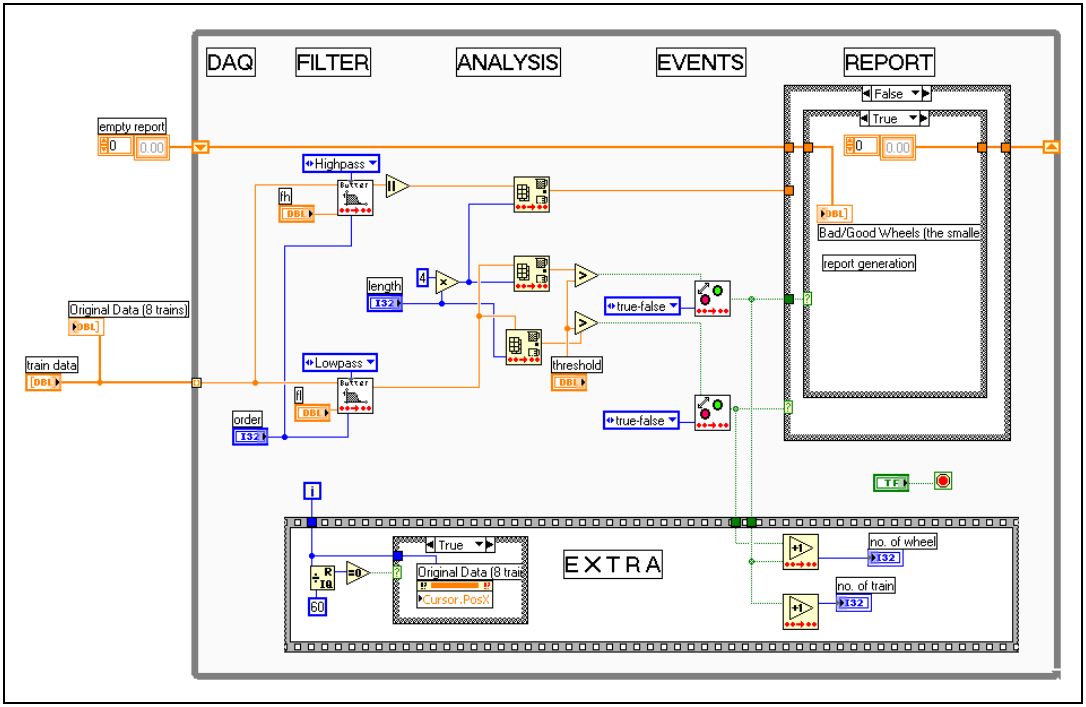


Figure 12-4. Block Diagram of the Train Wheel PtByPt VI

Note This example focuses on implementing a point-by-point analysis program in LabVIEW. The issues of ideal sampling periods and approaches to signal conditioning are beyond the scope of this example.

Overview of the LabVIEW Point-By-Point Solution

As well as Point By Point VIs, the Train Wheel PtByPt VI requires standard LabVIEW programming objects, such as Case structures, While Loops, numeric controls, and numeric operators.

The data the Train Wheel PtByPt VI acquires flows continuously through a While Loop. The process carried out by the Train Wheel PtByPt VI inside the While Loop consists of five analysis stages that occur sequentially. The following list reflects the order in which the five analysis stages occur, briefly describes what occurs in each stage, and corresponds to the labeled portions of the block diagram in Figure 12-4:

1. In the data acquisition stage (DAQ), waveform data flows into the While Loop.
2. In the Filter stage, separation of low- and high-frequency components of the waveform occurs.
3. In the Analysis stage, detection of the train, wheel, and energy level of the waveform for each wheel occurs.
4. In the Events stage, responses to signal transitions of trains and wheels occurs.
5. In the Report stage, the logging of trains, wheels, and trains that might have defective wheels occurs.

Characteristics of a Train Wheel Waveform

The characteristic waveform that train wheels emit determines how you analyze and filter the waveform signal point-by-point. A train wheel in motion emits a signal that contains low- and high-frequency components. If you mount a strain gauge in a railroad track, you detect a noisy signal similar to a bell curve. Figure 12-5 shows the low- and high-frequency components of this curve.

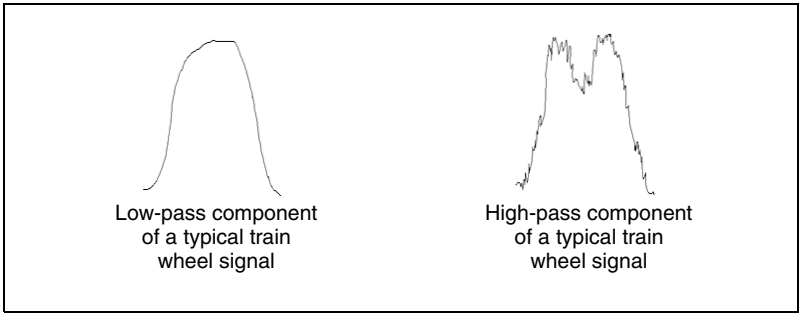


Figure 12-5. Low- and High-Frequency Components of a Train Wheel Signal

The low-frequency component of train wheel movement represents the normal noise of operation. Defective and normal wheels generate the same low-frequency component in the signal. The peak of the curve represents the moment when the wheel moves directly above the strain gauge. The lowest points of the bell curve represent the beginning and end of the wheel, respectively, as the wheel passes over the strain gauge.

The signal for a train wheel also contains a high-frequency component that reflects the quality of the wheel. In operation, a defective train wheel generates more energy than a normal train wheel. In other words, the high frequency component for a defective wheel has greater amplitude.

Analysis Stages of the Train Wheel PtByPt VI

The waveform of all train wheels, including defective ones, falls within predictable ranges. This predictable behavior allows you to choose the appropriate analysis parameters. These parameters apply to the five stages described in the *Overview of the LabVIEW Point-By-Point Solution* section. This section discusses each of the five analysis stages and the parameters use in each analysis stage.

Note You must adjust parameters for any implementation of the Train Wheel PtByPt VI, because the characteristics of each data acquisition system differ.

DAQ Stage

Data moves into the Point By Point VIs through the **input data** parameter. The point-by-point detection application operates on the continuous stream of waveform data that comes from the wheels of a moving train. For a train moving at 60 to 70 kilometers per hour, a few hundred to a few thousand samples per second are likely to give you sufficient information to detect a defective wheel.

Filter Stage

The Train Wheel PtByPt VI must filter low- and high-frequency components of the train wheel waveform. Two Butterworth Filter PtByPt VIs perform the following tasks:

- Extract the low-frequency components of the waveform.
- Extract the high-frequency components of the waveform.

In the Train Wheel PtByPt VI, the Butterworth Filter PtByPt VIs use the following parameters:

- **order** specifies the amount of the waveform data that the VI filters at a given time and is the filter resolution. `2` is acceptable for the Train Wheel PtByPt.
- **fl** specifies the low cut-off frequency, which is the minimum signal strength that identifies the departure of a train wheel from the strain gauge. `0.01` is acceptable for the Train Wheel PtByPt.
- **fh** specifies the high cut-off frequency, which is the minimum signal strength that identifies the end of high frequency waveform information. `0.25` is acceptable for the Train Wheel PtByPt.

Analysis Stage

The point-by-point detection application must analyze the low- and high-frequency components separately. The Array Max & Min PtByPt VI extracts waveform data that reveals the level of energy in the waveform for each wheel, the end of each train, and the end of each wheel.

Three separate Array Max & Min PtByPt VIs perform the following discrete tasks:

- Identify the maximum high-frequency value for each wheel.
- Identify the end of each train.
- Identify the end of each wheel.

Note The name Array Max & Min PtByPt VI contains the word array only to match the name of the array-based form of this VI. You do not need to allocate arrays for the Array Max & Min PtByPt VI.

In the Train Wheel PtByPt VI, the Array Max & Min PtByPt VIs use the following parameters and functions:

- **sample length** specifies the size of the portion of the waveform that the Train Wheel PtByPt VI analyzes. To calculate the ideal **sample length**, consider the speed of the train, the minimum distance between wheels, and the number of samples you receive per second. `100` is acceptable for the Train Wheel PtByPt VI. The Train Wheel PtByPt VI uses **sample length** to calculate values for all three Array Max & Min PtByPt VIs.
- The Multiply function sets a longer portion of the waveform to analyze. When this longer portion fails to display signal activity for

train wheels, the Array Max & Min PtByPt VIs identify the end of the train. 4 is acceptable for the Train Wheel PtByPt VI.

- **threshold** provides a comparison point to identify when no train wheel signals exist in the signal that you are acquiring. **threshold** is wired to the Greater? function. 3 is an acceptable setting for **threshold** in the Train Wheel PtByPt VI.

Events Stage

After the Analysis stage identifies maximum and minimum values, the Events stage detects when these values cross a threshold setting.

The Train Wheel PtByPt VI logs every wheel and every train that it detects. Two Boolean Crossing PtByPt VIs perform the following tasks:

- Generate an event each time the Array Max & Min PtByPt VIs detect the transition point in the signal that indicates the end of a wheel.
- Generate an event every time the Array Max & Min PtByPt VIs detect the transition point in the signal that indicates the end of a train.

The Boolean Crossing PtByPt VIs respond to transitions. When the amplitude of a single wheel waveform falls below the **threshold** setting, the end of the wheel has arrived at the strain gauge. For the Train Wheel PtByPt VI, 3 is a good **threshold** setting to identify the end of a wheel. When the signal strength falls below the **threshold** setting, the Boolean Crossing PtByPt VIs recognize a transition event and pass that event to a report.

Analysis of the high-frequency signal identifies which wheels, if any, might be defective. When the Train Wheel PtByPt VI encounters a potentially defective wheel, the VI passes the information directly to the report at the moment the end-of-wheel event is detected.

In the Train Wheel PtByPt VI, the Boolean Crossing PtByPt VIs use the following parameters:

- **initialize** resets the VI for a new session of continuous data acquisition.
- **direction** specifies the kind of Boolean crossing.

Report Stage

The Train Wheel PtByPt VI reports on all wheels for all trains that pass through the data acquisition system. The Train Wheel PtByPt VI also reports any potentially defective wheels.

Every time a wheel passes the strain gauge, the Train Wheel PtByPt VI captures its waveform, analyzes it, and reports the event. Table 12-4 describes the components of a report on a single train wheel.

Table 12-4. Example Report on a Single Train Wheel

Information Source	Meaning of Results
Counter mechanism for waveform events	Stage One: Wheel number four has passed the strain gauge.
Analysis of highpass filter data	Stage Two: Wheel number four has passed the strain gauge and the wheel might be defective.
Counter mechanism for end-of-train events	Stage Three: Wheel number four in train number eight has passed the strain gauge, and the wheel might be defective.

The Train Wheel PtByPt VI uses point-by-point analysis to generate a report, not to control an industrial process. However, the Train Wheel PtByPt VI acquires data in real time, and you can modify the application to generate real-time control responses, such as stopping the train when the Train Wheel PtByPt VI encounters a potentially defective wheel.

Conclusion

When acquiring data with real-time performance, point-by-point analysis helps you analyze data in real time. Point-by-point analysis occurs continuously and instantaneously. While you acquire data, you filter and analyze it, point by point, to extract the information you need and to make an appropriate response. This case study demonstrates the effectiveness of the point-by-point approach for generation of both events and reports in real time.

A

Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Online technical support resources include the following:
  - **Self-Help Resources**—For immediate answers and solutions, visit our extensive library of technical support resources available in English, Japanese, and Spanish at ni.com/support. These resources are available for most products at no cost to registered users and include software drivers and updates, a KnowledgeBase, product manuals, step-by-step troubleshooting wizards, conformity documentation, example code, tutorials and application notes, instrument drivers, discussion forums, a measurement glossary, and so on.
  - **Assisted Support Options**—Contact NI engineers and other measurement and automation professionals by visiting ni.com/support. Our online system helps you define your question and connects you to the experts by phone, discussion forum, or email.
- **Training**—Visit ni.com/custed for self-paced tutorials, videos, and interactive CDs. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, NI Alliance Program members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.