CHAPTER

5

Linear Systems

Most DSP techniques are based on a divide-and-conquer strategy called *superposition*. The signal being processed is broken into simple components, each component is processed individually, and the results reunited. This approach has the tremendous power of breaking a single complicated problem into many easy ones. Superposition can only be used with *linear systems*, a term meaning that certain mathematical rules apply. Fortunately, most of the applications encountered in science and engineering fall into this category. This chapter presents the foundation of DSP: what it means for a system to be linear, various ways for breaking signals into simpler components, and how superposition provides a variety of signal processing techniques.

Signals and Systems

A **signal** is a description of how one parameter varies with another parameter. For instance, voltage changing over time in an electronic circuit, or brightness varying with distance in an image. A **system** is any process that produces an *output signal* in response to an *input signal*. This is illustrated by the block diagram in Fig. 5-1. Continuous systems input and output continuous signals, such as in analog electronics. Discrete systems input and output discrete signals, such as computer programs that manipulate the values stored in arrays.

Several rules are used for naming signals. These aren't always followed in DSP, but they are very common and you should memorize them. The mathematics is difficult enough without a clear notation. First, *continuous* signals use parentheses, such as: $x(t)$ and $y(t)$, while *discrete* signals use brackets, as in: $x[n]$ and $y[n]$. Second, signals use lower case letters. Upper case letters are reserved for the frequency domain, discussed in later chapters. Third, the name given to a signal is usually descriptive of the parameters it represents. For example, a *voltage* depending on *time* might be called: $v(t)$, or a stock market *price* measured each *day* could be: $p[d]$.

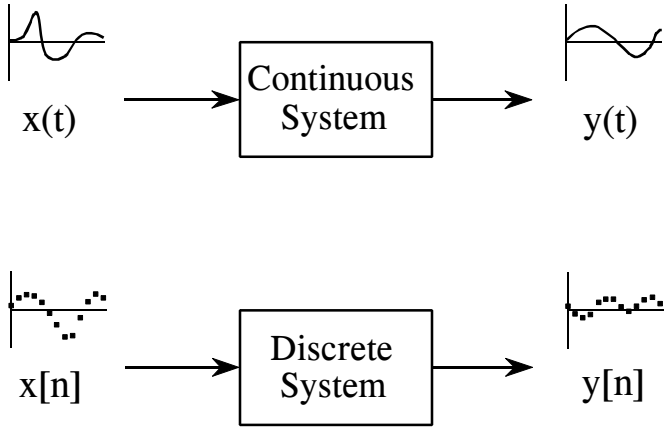


FIGURE 5-1
Terminology for signals and systems. A system is any process that generates an output signal in response to an input signal. Continuous signals are usually represented with parentheses, while discrete signals use brackets. All signals use lower case letters, reserving the upper case for the frequency domain (presented in later chapters). Unless there is a better name available, the input signal is called: $x(t)$ or $x[n]$, while the output is called: $y(t)$ or $y[n]$.

Signals and systems are frequently discussed without knowing the exact parameters being represented. This is the same as using x and y in algebra, without assigning a physical meaning to the variables. This brings in a fourth rule for naming signals. If a more descriptive name is not available, the input signal to a discrete system is usually called: $x[n]$, and the output signal: $y[n]$. For continuous systems, the signals: $x(t)$ and $y(t)$ are used.

There are many reasons for wanting to understand a *system*. For example, you may want to *design* a system to remove noise in an electrocardiogram, sharpen an out-of-focus image, or remove echoes in an audio recording. In other cases, the system might have a distortion or interfering effect that you need to characterize or measure. For instance, when you speak into a telephone, you expect the other person to hear something that resembles your voice. Unfortunately, the input signal to a transmission line is seldom identical to the output signal. If you understand how the transmission line (the system) is changing the signal, maybe you can compensate for its effect. In still other cases, the system may represent some physical process that you want to study or analyze. Radar and sonar are good examples of this. These methods operate by comparing the transmitted and reflected signals to find the characteristics of a remote object. In terms of system theory, the problem is to find the system that changes the transmitted signal into the received signal.

At first glance, it may seem an overwhelming task to understand all of the possible systems in the world. Fortunately, most useful systems fall into a category called **linear systems**. This fact is extremely important. *Without* the linear system concept, we would be forced to examine the individual

characteristics of many unrelated systems. *With* this approach, we can focus on the traits of the linear system category as a whole. Our first task is to identify what properties make a system linear, and how they fit into the everyday notion of electronics, software, and other signal processing systems.

Requirements for Linearity

A system is called *linear* if it has two mathematical properties: **homogeneity** (hōma-gen-ā-ity) and **additivity**. If you can show that a system has both properties, then you have proven that the system is linear. Likewise, if you can show that a system doesn't have one or both properties, you have proven that it isn't linear. A third property, **shift invariance**, is not a strict requirement for linearity, but it is a mandatory property for most DSP techniques. When you see the term *linear system* used in DSP, you should assume it includes *shift invariance* unless you have reason to believe otherwise. These three properties form the mathematics of how linear system theory is defined and used. Later in this chapter we will look at more intuitive ways of understanding linearity. For now, let's go through these formal mathematical properties.

As illustrated in Fig. 5-2, homogeneity means that a change in the input signal's amplitude results in a corresponding change in the output signal's amplitude. In mathematical terms, if an input signal of $x[n]$ results in an output signal of $y[n]$, an input of $kx[n]$ results in an output of $ky[n]$, for any input signal and constant, k.

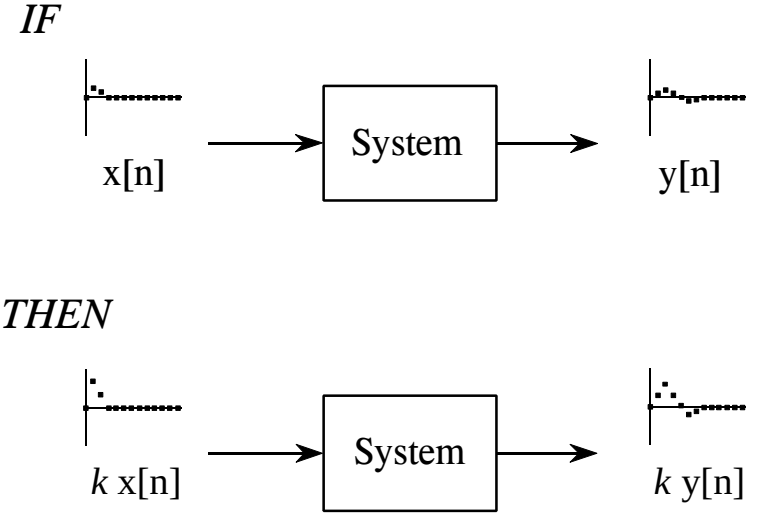


FIGURE 5-2
Definition of homogeneity. A system is said to be *homogeneous* if an amplitude change in the input results in an identical amplitude change in the output. That is, if $x[n]$ results in $y[n]$, then $kx[n]$ results in $ky[n]$, for any signal, $x[n]$, and any constant, k.

A simple resistor provides a good example of both homogenous and non-homogeneous systems. If the input to the system is the voltage across the resistor, $v(t)$, and the output from the system is the current through the resistor, $i(t)$, the system is homogeneous. Ohm's law guarantees this; if the voltage is increased or decreased, there will be a corresponding increase or decrease in the current. Now, consider another system where the input signal is the voltage across the resistor, $v(t)$, but the output signal is the power being dissipated in the resistor, $p(t)$. Since power is proportional to the square of the voltage, if the input signal is increased by a factor of *two*, the output signal is increase by a factor of *four*. This system is not homogeneous and therefore cannot be linear.

The property of additivity is illustrated in Fig. 5-3. Consider a system where an input of $x_1[n]$ produces an output of $y_1[n]$. Further suppose that a different input, $x_2[n]$, produces another output, $y_2[n]$. The system is said to be *additive*, if an input of $x_1[n] + x_2[n]$ results in an output of $y_1[n] + y_2[n]$, for all possible input signals. In words, signals added at the input produce signals that are added at the output.

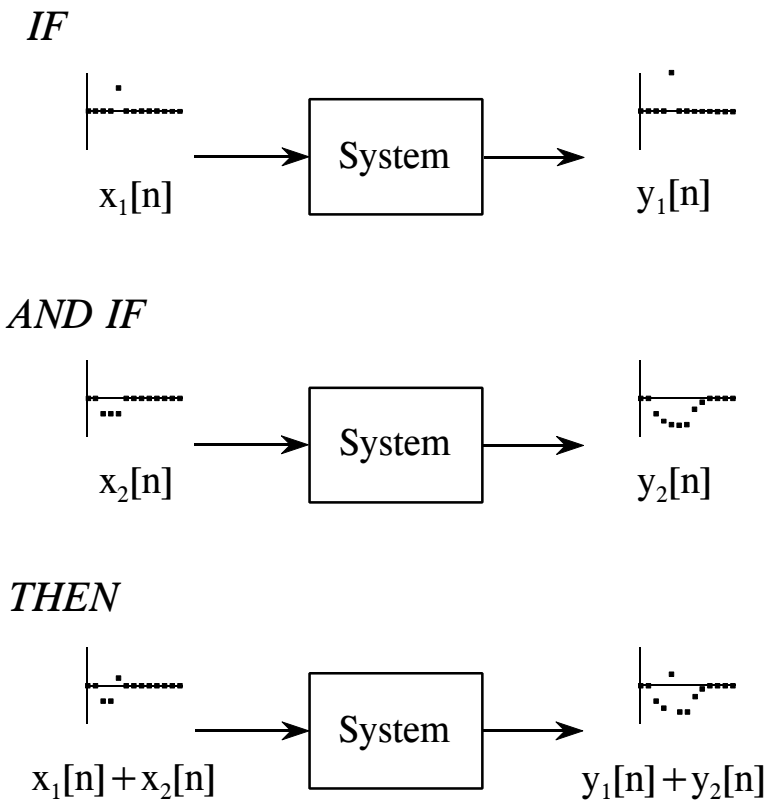


FIGURE 5-3
Definition of additivity. A system is said to be *additive* if added signals pass through it without interacting. Formally, if $x_1[n]$ results in $y_1[n]$, and if $x_2[n]$ results in $y_2[n]$, then $x_1[n]+x_2[n]$ results in $y_2[n]+y_2[n]$.

The important point is that *added* signals pass through the system without interacting. As an example, think about a telephone conversation with your Aunt Edna and Uncle Bernie. Aunt Edna begins a rather lengthy story about how well her radishes are doing this year. In the background, Uncle Bernie is yelling at the dog for having an accident in his favorite chair. The two voice signals are added and electronically transmitted through the telephone network. Since this system is additive, the sound you hear is the sum of the two voices as they would sound if transmitted individually. You hear *Edna* and *Bernie*, not the creature, *Ednabernie*.

A good example of a *nonadditive* circuit is the mixer stage in a radio transmitter. Two signals are present: an audio signal that contains the voice or music, and a carrier wave that can propagate through space when applied to an antenna. The two signals are added and applied to a nonlinearity, such as a pn junction diode. This results in the signals *merging* to form a third signal, a modulated radio wave capable of carrying the information over great distances.

As shown in Fig. 5-4, shift invariance means that a shift in the input signal will result in nothing more than an identical shift in the output signal. In more formal terms, if an input signal of $x[n]$ results in an output of $y[n]$, an input signal of $x[n+s]$ results in an output of $y[n+s]$, for any input signal and any constant, s. Pay particular notice to how the mathematics of this shift is written, it will be used in upcoming chapters. By adding a constant, s, to the independent variable, n, the waveform can be advanced or retarded in the horizontal direction. For example, when $s = 2$, the signal is shifted *left* by two samples; when $s = -2$, the signal is shifted *right* by two samples.

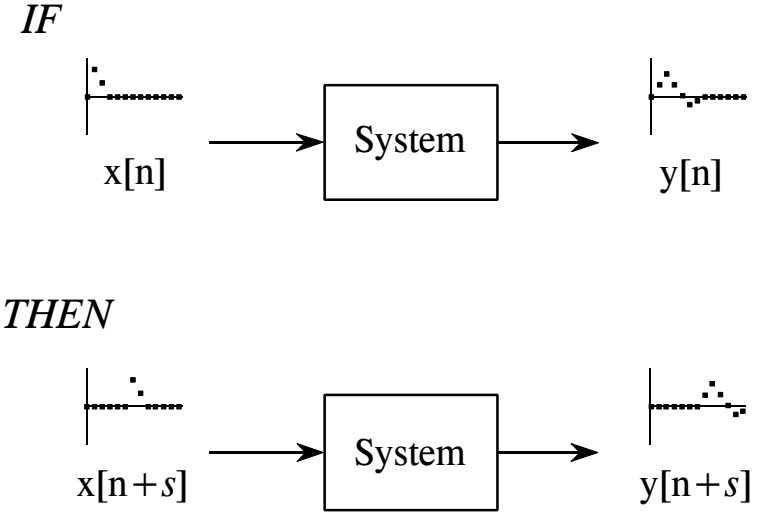


FIGURE 5-4
Definition of shift invariance. A system is said to be *shift invariant* if a shift in the input signal causes an identical shift in the output signal. In mathematical terms, if $x[n]$ produces $y[n]$, then $x[n+s]$ produces $y[n+s]$, for any signal, $x[n]$, and any constant, s.

Shift invariance is important because it means the characteristics of the system do not change with time (or whatever the independent variable happens to be). If a *blip* in the input causes a *blop* in the output, you can be assured that another *blip* will cause an identical *blop*. Most of the systems you encounter will be shift invariant. This is fortunate, because it is difficult to deal with systems that change their characteristics while in operation. For example, imagine that you have designed a digital filter to compensate for the degrading effects of a telephone transmission line. Your filter makes the voices sound more natural and easier to understand. Much to your surprise, along comes winter and you find the characteristics of the telephone line have changed with temperature. Your compensation filter is now mismatched and doesn't work especially well. This situation may require a more sophisticated algorithm that can *adapt* to changing conditions.

Why do homogeneity and additivity play a critical role in linearity, while shift invariance is something on the side? This is because linearity is a very broad concept, encompassing much more than just signals and systems. For example, consider a farmer selling oranges for $2 per crate and apples for $5 per crate. If the farmer sells only oranges, he will receive $20 for 10 crates, and $40 for 20 crates, making the exchange *homogenous*. If he sells 20 crates of oranges and 10 crates of apples, the farmer will receive: $20 \times \$2 + 10 \times \$5 = \$90$. This is the same amount as if the two had been sold individually, making the transaction *additive*. Being both homogenous and additive, this sale of goods is a linear process. However, since there are no signals involved, this is not a *system*, and *shift invariance* has no meaning. Shift invariance can be thought of as an additional aspect of linearity needed when signals and systems are involved.

Static Linearity and Sinusoidal Fidelity

Homogeneity, additivity, and shift invariance are important because they provide the mathematical basis for defining linear systems. Unfortunately, these properties alone don't provide most scientists and engineers with an intuitive feeling of what linear systems are about. The properties of **static linearity** and **sinusoidal fidelity** are often of help here. These are not especially important from a mathematical standpoint, but relate to how humans think about and understand linear systems. You should pay special attention to this section.

Static linearity defines how a linear system reacts when the signals aren't changing, i.e., when they are *DC* or *static*. The static response of a linear system is very simple: *the output is the input multiplied by a constant*. That is, a graph of the possible input values plotted against the corresponding output values is a straight line that passes through the origin. This is shown in Fig. 5-5 for two common linear systems: Ohm's law for resistors, and Hooke's law for springs. For comparison, Fig. 5-6 shows the static relationship for two nonlinear systems: a pn junction diode, and the magnetic properties of iron.

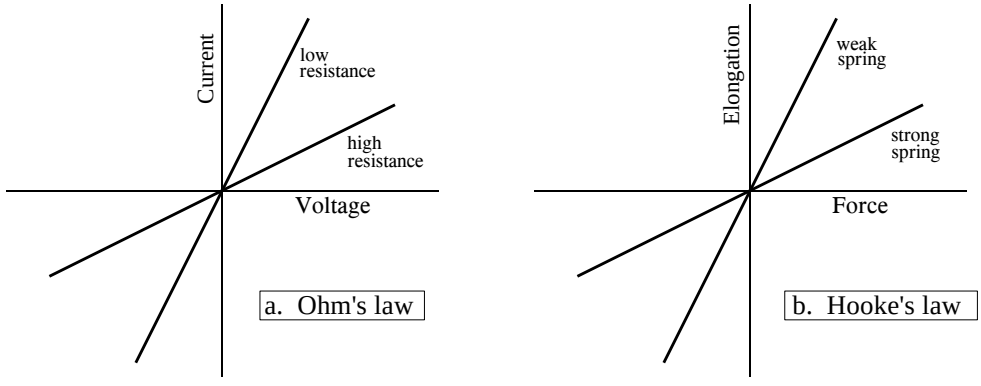


FIGURE 5-5
Two examples of static linearity. In (a), Ohm's law: the current through a resistor is equal to the voltage across the resistor divided by the resistance. In (b), Hooke's law: The elongation of a spring is equal to the applied force multiplied by the spring stiffness coefficient.

All linear systems have the property of *static linearity*. The opposite is usually true, but not always. There are systems that show static linearity, but are not linear with respect to changing signals. However, a very common class of systems can be completely understood with static linearity alone. In these systems it doesn't matter if the input signal is static or changing. These are called **memoryless** systems, because the output depends only on the present state of the input, and not on its history. For example, the instantaneous current in a resistor depends only on the instantaneous voltage across it, and not on how the signals came to be the value they are. If a system has static linearity, and is memoryless, then the system must be linear. This provides an important way to understand (and prove) the linearity of these simple systems.

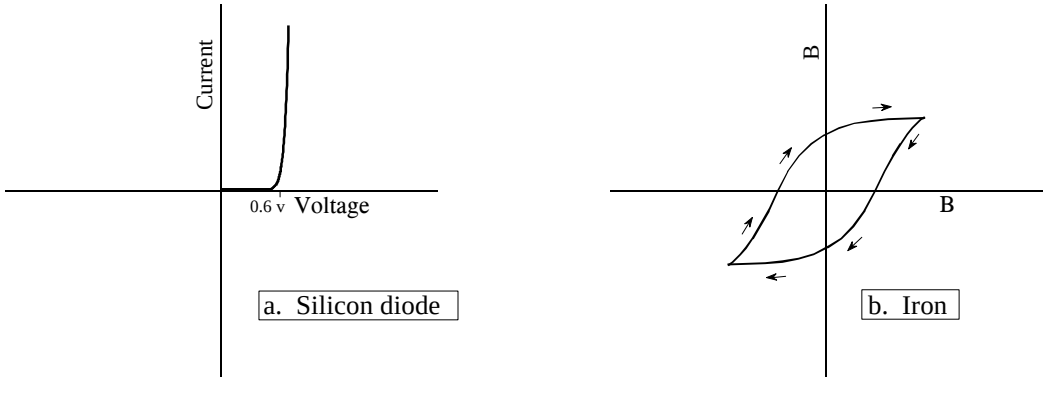


FIGURE 5-6
Two examples of DC nonlinearity. In (a), a silicon diode has an exponential relationship between voltage and current. In (b), the relationship between magnetic intensity, H, and flux density, B, in iron depends on the history of the sample, a behavior called *hysteresis*.

An important characteristic of linear systems is how they behave with sinusoids, a property we will call **sinusoidal fidelity**: *If the input to a linear system is a sinusoidal wave, the output will also be a sinusoidal wave, and at exactly the same frequency as the input.* Sinusoids are the only waveform that have this property. For instance, there is no reason to expect that a square wave entering a linear system will produce a square wave on the output. Although a sinusoid on the input guarantees a sinusoid on the output, the two may be different in *amplitude* and *phase*. This should be familiar from your knowledge of electronics: a circuit can be described by its *frequency response*, graphs of how the circuit's gain and phase vary with frequency.

Now for the reverse question: If a system always produces a sinusoidal output in response to a sinusoidal input, is the system guaranteed to be linear? The answer is no, but the exceptions are rare and usually obvious. For example, imagine an evil demon hiding inside a system, with the goal of trying to mislead you. The demon has an oscilloscope to observe the input signal, and a sine wave generator to produce an output signal. When you feed a sine wave into the input, the demon quickly measures the frequency and adjusts his signal generator to produce a corresponding output. Of course, this system is not linear, because it is not additive. To show this, place the sum of two sine waves into the system. The demon can only respond with a single sine wave for the output. This example is not as contrived as you might think; *phase lock loops* operate in much this way.

To get a better feeling for linearity, think about a technician trying to determine if an electronic device is linear. The technician would attach a sine wave generator to the input of the device, and an oscilloscope to the output. With a sine wave input, the technician would look to see if the output is also a sine wave. For example, the output cannot be clipped on the top or bottom, the top half cannot look different from the bottom half, there must be no distortion where the signal crosses zero, etc. Next, the technician would vary the amplitude of the input and observe the effect on the output signal. If the system is linear, the amplitude of the output must track the amplitude of the input. Lastly, the technician would vary the input signal's frequency, and verify that the output signal's frequency changes accordingly. As the frequency is changed, there will likely be amplitude and phase changes seen in the output, but these are perfectly permissible in a linear system. At some frequencies, the output may even be *zero*, that is, a sinusoid with zero amplitude. If the technician sees all these things, he will conclude that the system is linear. While this conclusion is not a rigorous mathematical proof, the level of confidence is justifiably high.

Examples of Linear and Nonlinear Systems

Table 5-1 provides examples of common linear and nonlinear systems. As you go through the lists, keep in mind the mathematician's view of linearity (*homogeneity*, *additivity*, and *shift invariance*), as well as the informal way most scientists and engineers use (*static linearity* and *sinusoidal fidelity*).

Examples of *Linear* Systems

Wave propagation such as sound and electromagnetic waves

Electrical circuits composed of resistors, capacitors, and inductors

Electronic circuits, such as amplifiers and filters

Mechanical motion from the interaction of masses, springs, and dashpots (dampeners)

Systems described by differential equations such as resistor-capacitor-inductor networks

Multiplication by a constant, that is, amplification or attenuation of the signal

Signal changes, such as echoes, resonances, and image blurring

The unity system where the output is always equal to the input

The null system where the output is always equal to the zero, regardless of the input

Differentiation and integration, and the analogous operations of *first difference* and *running sum* for discrete signals

Small perturbations in an otherwise nonlinear system, for instance, a small signal being amplified by a properly biased transistor

Convolution, a mathematical operation where each value in the output is expressed as the sum of values in the input multiplied by a set of weighing coefficients.

Recursion, a technique similar to convolution, except previously calculated values in the output are used in addition to values from the input

Examples of *Nonlinear* Systems

Systems that do not have static linearity, for instance, the voltage and power in a resistor: $P = V^2R$, the radiant energy emission of a hot object depending on its temperature: $R = kT^4$, the intensity of light transmitted through a thickness of translucent material: $I = e^{-\alpha T}$, etc.

Systems that do not have sinusoidal fidelity, such as electronics circuits for: peak detection, squaring, sine wave to square wave conversion, frequency doubling, etc.

Common electronic distortion, such as clipping, crossover distortion and slewing

Multiplication of one signal by another signal, such as in amplitude modulation and automatic gain controls

Hysteresis phenomena, such as magnetic flux density versus magnetic intensity in iron, or mechanical stress versus strain in vulcanized rubber

Saturation, such as electronic amplifiers and transformers driven too hard

Systems with a threshold, for example, digital logic gates, or seismic vibrations that are strong enough to pulverize the intervening rock

Table 5-1
Examples of linear and nonlinear systems. Formally, linear systems are defined by the properties of *homogeneity*, *additivity*, and *shift invariance*. Informally, most scientists and engineers think of linear systems in terms of *static linearity* and *sinusoidal fidelity*.

Special Properties of Linearity

Linearity is **commutative**, a property involving the combination of two or more systems. Figure 5-10 shows the general idea. Imagine two systems combined in a **cascade**, that is, the output of one system is the input to the next. If each system is linear, then the overall combination will also be linear. The commutative property states that the order of the systems in the cascade can be rearranged without affecting the characteristics of the overall combination. You probably have used this principle in electronic circuits. For example, imagine a circuit composed of two stages, one for amplification, and one for filtering. Which is best, amplify and then filter, or filter and then amplify? If both stages are linear, the order doesn't make any difference and the overall result is the same. Keep in mind that actual electronics has *nonlinear* effects that may make the order important, for instance: interference, DC offsets, internal noise, slew rate distortion, etc.

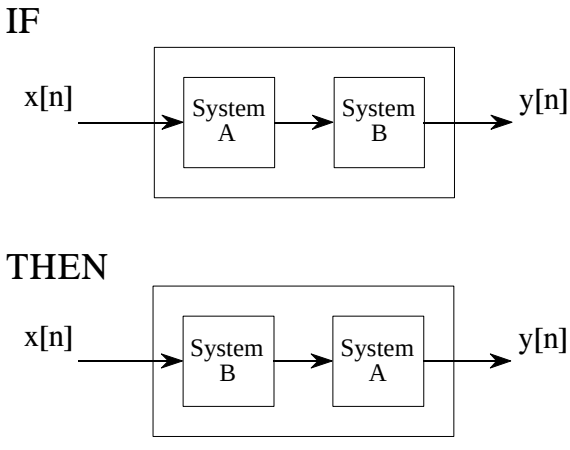


FIGURE 5-7
The commutative property for linear systems. When two or more linear systems are arranged in a cascade, the order of the systems does not affect the characteristics of the overall combination.

Figure 5-8 shows the next step in linear system theory: multiple inputs and outputs. A system with multiple inputs and/or outputs will be linear if it is composed of *linear subsystems* and *additions of signals*. The complexity does not matter, only that nothing *nonlinear* is allowed inside of the system.

To understand what linearity means for systems with multiple inputs and/or outputs, consider the following thought experiment. Start by placing a signal on one input while the other inputs are held at zero. This will cause the multiple outputs to respond with some pattern of signals. Next, repeat the procedure by placing another signal on a different input. Just as before, keep all of the other inputs at zero. This second input signal will result in another pattern of signals appearing on the multiple outputs. To finish the experiment, place both signals on their respective inputs simultaneously. The signals appearing on the outputs will simply be the *superposition* (sum) of the output signals produced when the input signals were applied separately.

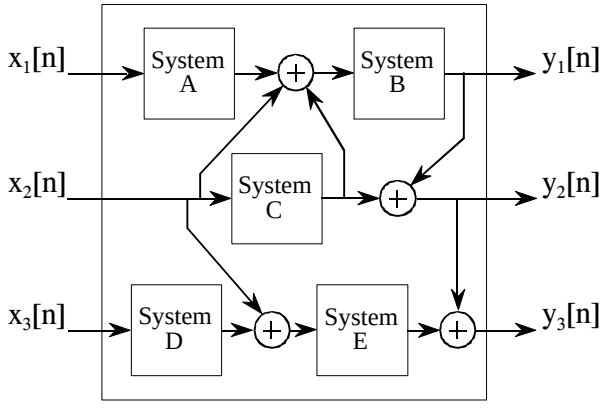
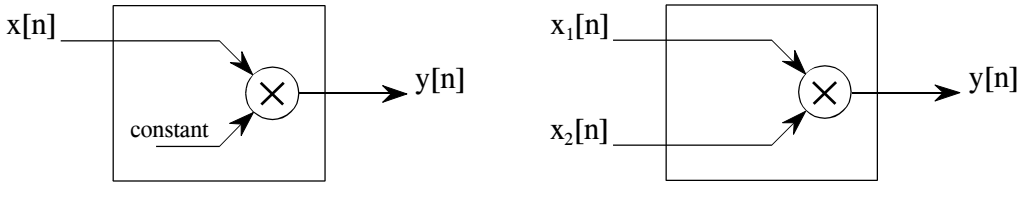


FIGURE 5-8
Any system with multiple inputs and/or outputs will be linear if it is composed of linear systems and signal additions.

The use of multiplication in linear systems is frequently misunderstood. This is because multiplication can be either linear or nonlinear, depending on what the signal is multiplied by. Figure 5-9 illustrates the two cases. A system that multiplies the input signal by a *constant*, is linear. This system is an amplifier or an attenuator, depending if the constant is greater or less than one, respectively. In contrast, multiplying a signal by *another signal* is nonlinear. Imagine a sinusoid multiplied by another sinusoid of a different frequency; the resulting waveform is clearly not sinusoidal.

Another commonly misunderstood situation relates to parasitic signals added in electronics, such as DC offsets and thermal noise. Is the addition of these extraneous signals linear or nonlinear? The answer depends on where the contaminating signals are viewed as originating. If they are viewed as coming from *within* the system, the process is nonlinear. This is because a sinusoidal input does not produce a pure sinusoidal output. Conversely, the extraneous signal can be viewed as *externally* entering the system on a separate input of a multiple input system. This makes the process linear, since only a signal addition is required within the system.



Linear
a. Multiplication by a constant

Nonlinear
b. Multiplication of two signals

FIGURE 5-9
Linearity of multiplication. Multiplying a signal by a constant is a linear operation. In contrast, the multiplication of two signals is nonlinear.

Superposition: the Foundation of DSP

When we are dealing with linear systems, the only way signals can be combined is by *scaling* (multiplication of the signals by constants) followed by *addition*. For instance, a signal cannot be multiplied by another signal. Figure 5-10 shows an example: three signals: $x_0[n]$, $x_1[n]$, and $x_2[n]$ are added to form a fourth signal, $x[n]$. This process of combining signals through scaling and addition is called **synthesis**.

Decomposition is the inverse operation of synthesis, where a single signal is broken into two or more additive components. This is more involved than synthesis, because there are infinite possible decompositions for any given signal. For example, the numbers 15 and 25 can only be synthesized (added) into the number 40. In comparison, the number 40 can be decomposed into: $1 + 39$ or $2 + 38$ or $-30.5 + 60 + 10.5$, etc.

Now we come to the heart of DSP: **superposition**, the overall strategy for understanding how signals and systems can be analyzed. Consider an input

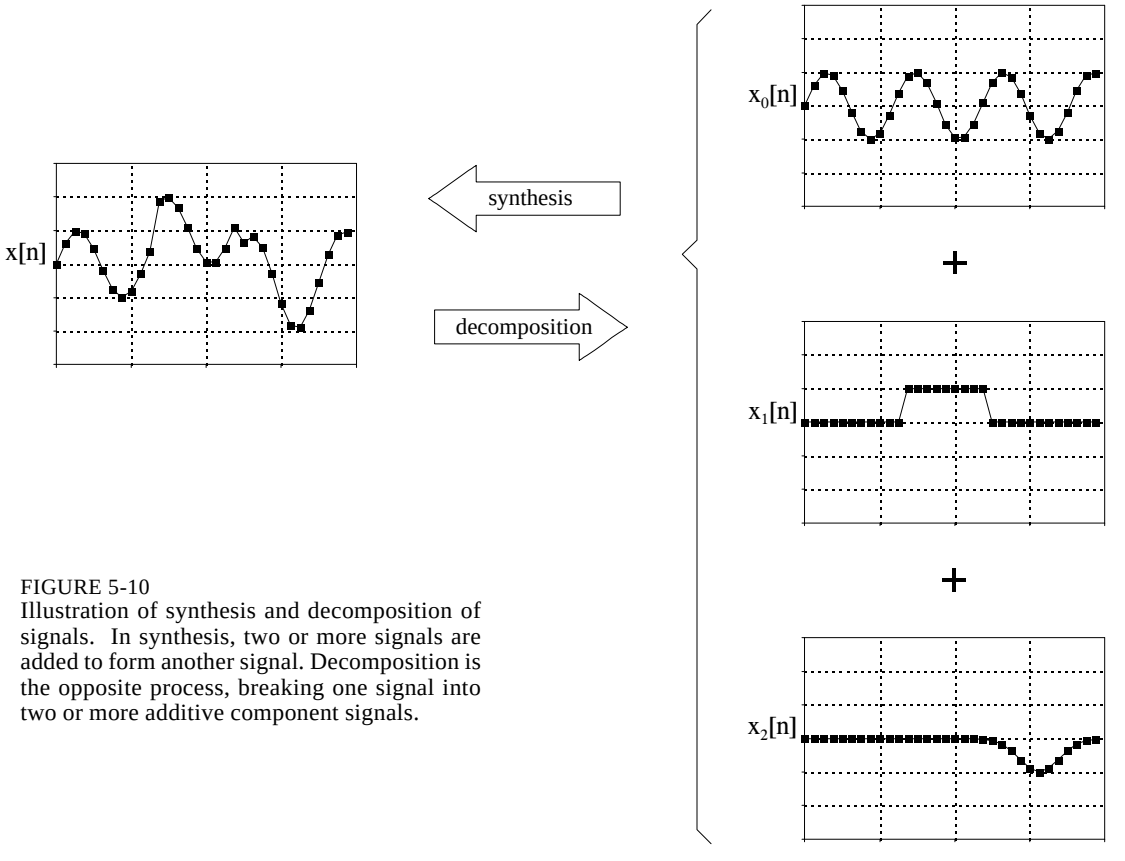


FIGURE 5-10
Illustration of synthesis and decomposition of signals. In synthesis, two or more signals are added to form another signal. Decomposition is the opposite process, breaking one signal into two or more additive component signals.

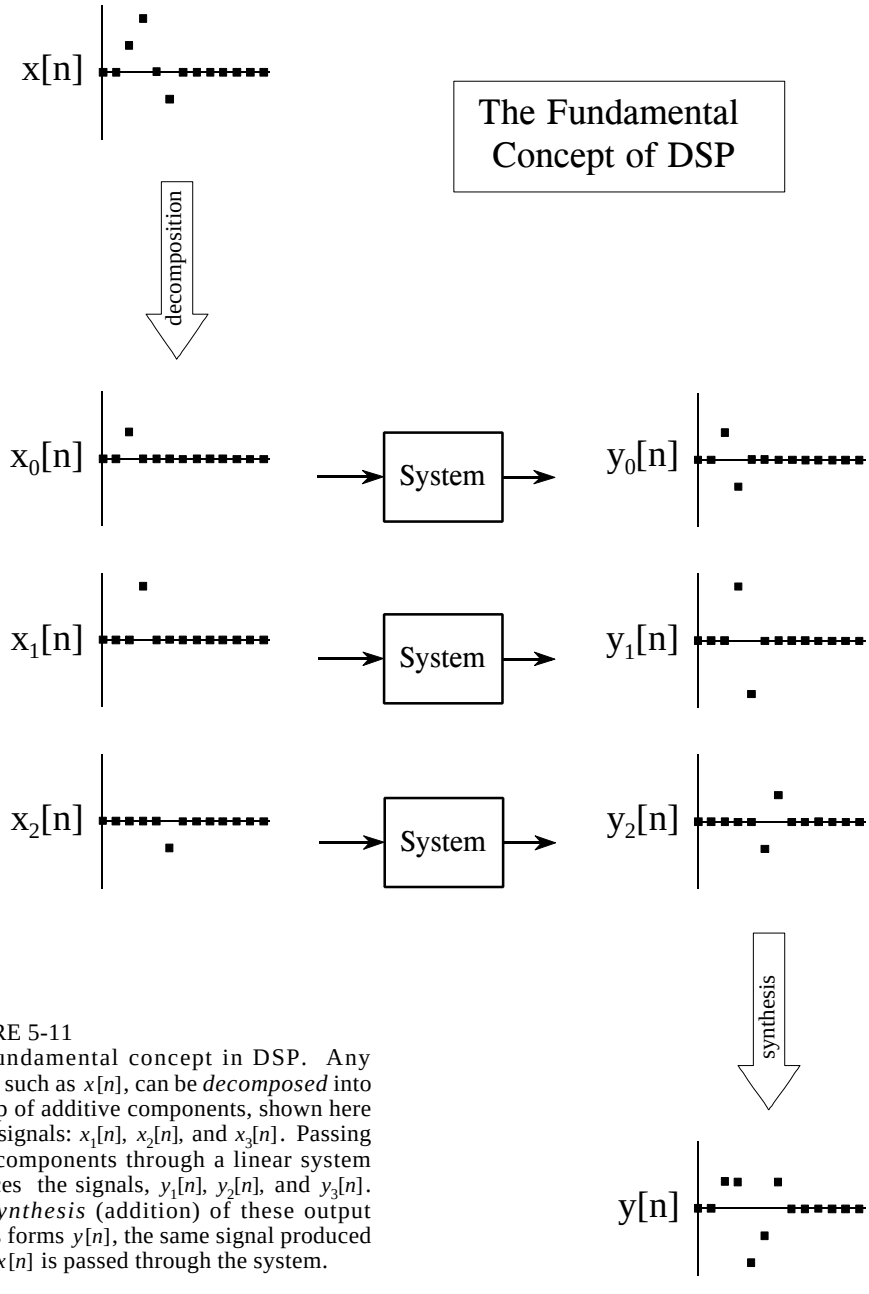


FIGURE 5-11
The fundamental concept in DSP. Any signal, such as $x[n]$, can be *decomposed* into a group of additive components, shown here by the signals: $x_1[n]$, $x_2[n]$, and $x_3[n]$. Passing these components through a linear system produces the signals, $y_1[n]$, $y_2[n]$, and $y_3[n]$. The *synthesis* (addition) of these output signals forms $y[n]$, the same signal produced when $x[n]$ is passed through the system.

signal, called $x[n]$, passing through a linear system, resulting in an output signal, $y[n]$. As illustrated in Fig. 5-11, the input signal can be decomposed into a group of simpler signals: $x_0[n]$, $x_1[n]$, $x_2[n]$, etc. We will call these the **input signal components.** Next, each input signal component is individually passed through the system, resulting in a set of **output signal components**: $y_0[n]$, $y_1[n]$, $y_2[n]$, etc. These output signal components are then synthesized into the output signal, $y[n]$.

Here is the important part: the output signal obtained by this method is *identical* to the one produced by directly passing the input signal through the system. This is a very powerful idea. Instead of trying to understanding how *complicated* signals are changed by a system, all we need to know is how *simple* signals are modified. In the jargon of signal processing, the input and output signals are viewed as a *superposition* (sum) of simpler waveforms. This is the basis of nearly all signal processing techniques.

As a simple example of how superposition is used, multiply the number 2041 by the number 4, in your head. How did you do it? You might have imagined 2041 match sticks floating in your mind, quadrupled the mental image, and started counting. Much more likely, you used superposition to simplify the problem. The number 2041 can be decomposed into: $2000 + 40 + 1$. Each of these components can be multiplied by 4 and then synthesized to find the final answer, i.e., $8000 + 160 + 4 = 8164$.

Common Decompositions

Keep in mind that the goal of this method is to replace a complicated problem with several easy ones. If the decomposition doesn't simplify the situation in some way, then nothing has been gained. There are two main ways to decompose signals in signal processing: *impulse decomposition* and *Fourier decomposition*. They are described in detail in the next several chapters. In addition, several minor decompositions are occasionally used. Here are brief descriptions of the two major decompositions, along with three of the minor ones.

Impulse Decomposition

As shown in Fig. 5-12, impulse decomposition breaks an N samples signal into N component signals, each containing N samples. Each of the component signals contains one point from the original signal, with the remainder of the values being zero. A single nonzero point in a string of zeros is called an *impulse*. Impulse decomposition is important because it allows signals to be examined one sample at a time. Similarly, systems are characterized by how they respond to impulses. By knowing how a system responds to an impulse, the system's output can be calculated for any given input. This approach is called *convolution*, and is the topic of the next two chapters.

Step Decomposition

Step decomposition, shown in Fig. 5-13, also breaks an N sample signal into N component signals, each composed of N samples. Each component signal is a *step*, that is, the first samples have a value of zero, while the last samples are some constant value. Consider the decomposition of an N point signal, $x[n]$, into the components: $x_0[n], x_1[n], x_2[n], \cdots x_{N-1}[n]$. The k^{th} component signal, $x_k[n]$, is composed of zeros for points 0 through $k-1$, while the remaining points have a value of: $x[k] - x[k-1]$. For example, the 5^{th} component signal, $x_5[n]$, is composed of zeros for points 0 through 4, while the remaining samples have a value of: $x[5] - x[4]$ (the difference between

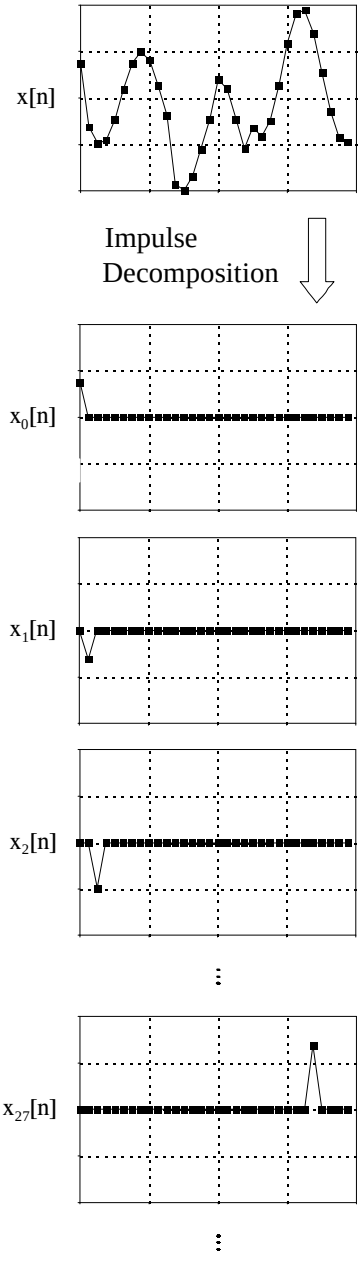


FIGURE 5-12
Example of impulse decomposition. An *N* point signal is broken into *N* components, each consisting of a single nonzero point.

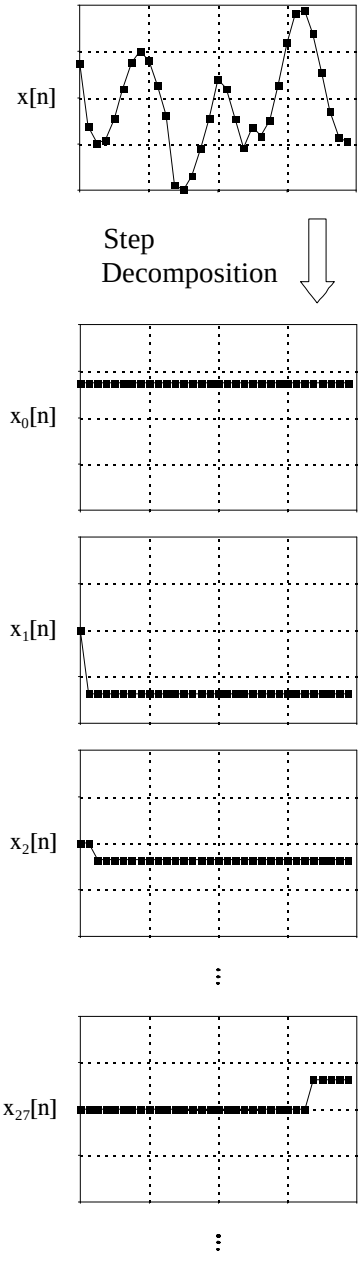


FIGURE 5-13
Example of step decomposition. An *N* point signal is broken into *N* signals, each consisting of a step function

sample 4 and 5 of the original signal). As a special case, $x_0[n]$ has all of its samples equal to $x[0]$. Just as impulse decomposition looks at signals one point at a time, step decomposition characterizes signals by the *difference* between adjacent samples. Likewise, systems are characterized by how they respond to a *change* in the input signal.

Even/Odd Decomposition

The even/odd decomposition, shown in Fig. 5-14, breaks a signal into two component signals, one having **even symmetry** and the other having **odd symmetry**. An N point signal is said to have even symmetry if it is a mirror image around point $N/2$. That is, sample $x[N/2 + 1]$ must equal $x[N/2 - 1]$, sample $x[N/2 + 2]$ must equal $x[N/2 - 2]$, etc. Similarly, odd symmetry occurs when the matching points have equal magnitudes but are opposite in sign, such as: $x[N/2 + 1] = -x[N/2 - 1]$, $x[N/2 + 2] = -x[N/2 - 2]$, etc. These definitions assume that the signal is composed of an even number of samples, and that the indexes run from 0 to $N-1$. The decomposition is calculated from the relations:

EQUATION 5-1
Equations for even/odd decomposition. These equations separate a signal, $x[n]$, into its even part, $x_E[n]$, and its odd part, $x_O[n]$. Since this decomposition is based on circularly symmetry, the zeroth samples in the even and odd signals are calculated: $x_E[0] = x[0]$, and $x_O[0] = 0$. All of the signals are N samples long, with indexes running from 0 to N-1

$$x_E[n] = \frac{x[n] + x[N-n]}{2}$$

$$x_O[n] = \frac{x[n] - x[N-n]}{2}$$

This may seem a strange definition of left-right symmetry, since $N/2 - ½$ (between two samples) is the exact center of the signal, not $N/2$. Likewise, this off-center symmetry means that sample zero needs special handling. What's this all about?

This decomposition is part of an important concept in DSP called **circular symmetry**. It is based on viewing the *end* of the signal as connected to the *beginning* of the signal. Just as point $x[4]$ is next to point $x[5]$, point $x[N-1]$ is next to point $x[0]$. Picture a snake biting its own tail. When even and odd signals are viewed in this circular manner, there are actually *two* lines of symmetry, one at point $x[N/2]$ and another at point $x[0]$. For example, in an even signal, this symmetry around $x[0]$ means that point $x[1]$ equals point $x[N-1]$, point $x[2]$ equals point $x[N-2]$, etc. In an odd signal, point 0 and point $N/2$ always have a value of zero. In an even signal, point 0 and point $N/2$ are equal to the corresponding points in the original signal.

What is the motivation for viewing the last sample in a signal as being next to the first sample? There is nothing in conventional data acquisition to support this circular notion. In fact, the first and last samples generally have less in common than any other two points in the sequence. It's common sense! The missing piece to this puzzle is a DSP technique called *Fourier analysis*. The mathematics of Fourier analysis inherently views the signal as being circular, although it usually has no physical meaning in terms of where the data came from. We will look at this in more detail in Chapter 10. For now, the important thing to understand is that Eq. 5-1 provides a valid decomposition, simply because the even and odd parts can be added together to reconstruct the original signal.

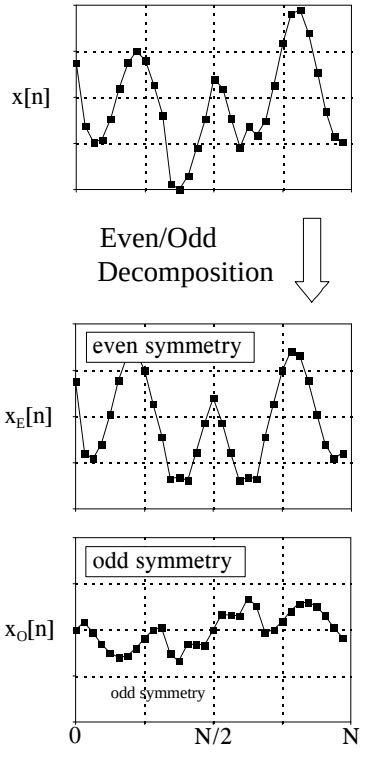


FIGURE 5-14
Example of even/odd decomposition. An *N* point signal is broken into two *N* point signals, one with even symmetry, and the other with odd symmetry.

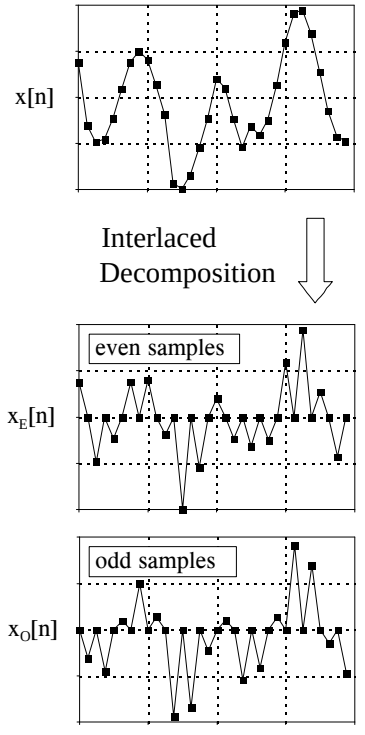


FIGURE 5-15
Example of interlaced decomposition. An *N* point signal is broken into two *N* point signals, one with the odd samples set to zero, the other with the even samples set to zero.

Interlaced Decomposition

As shown in Fig. 5-15, the interlaced decomposition breaks the signal into two component signals, the *even sample* signal and the *odd sample* signal (not to be confused with even and odd symmetry signals). To find the even sample signal, start with the original signal and set all of the odd numbered samples to zero. To find the odd sample signal, start with the original signal and set all of the even numbered samples to zero. It's that simple.

At first glance, this decomposition might seem trivial and uninteresting. This is ironic, because the interlaced decomposition is the basis for an extremely important algorithm in DSP, the Fast Fourier Transform (FFT). The procedure for calculating the Fourier decomposition has been known for several hundred years. Unfortunately, it is frustratingly slow, often requiring minutes or hours to execute on present day computers. The FFT is a family of algorithms developed in the 1960s to reduce this computation time. The strategy is an exquisite example of DSP: reduce the signal to elementary components by repeated use of the interlace transform; calculate the Fourier decomposition of the individual components; synthesize the results into the final answer. The

results are dramatic; it is common for the speed to be improved by a factor of *hundreds* or *thousands*.

Fourier Decomposition

Fourier decomposition is very mathematical and not at all obvious. Figure 5-16 shows an example of the technique. Any N point signal can be decomposed into $N+2$ signals, half of them sine waves and half of them cosine waves. The lowest frequency cosine wave (called $x_{C0}[n]$ in this illustration), makes *zero* complete cycles over the N samples, i.e., it is a DC signal. The next cosine components: $x_{C1}[n]$, $x_{C2}[n]$, and $x_{C3}[n]$, make 1, 2, and 3 complete cycles over the N samples, respectively. This pattern holds for the remainder of the cosine waves, as well as for the sine wave components. Since the frequency of each component is fixed, the only thing that changes for different signals being decomposed is the *amplitude* of each of the sine and cosine waves.

Fourier decomposition is important for three reasons. First, a wide variety of signals are inherently created from superimposed sinusoids. Audio signals are a good example of this. Fourier decomposition provides a direct analysis of the information contained in these types of signals. Second, linear systems respond to sinusoids in a unique way: a sinusoidal input always results in a sinusoidal output. In this approach, systems are characterized by how they change the amplitude and phase of sinusoids passing through them. Since an input signal can be decomposed into sinusoids, knowing how a system will react to sinusoids allows the output of the system to be found. Third, the Fourier decomposition is the basis for a broad and powerful area of mathematics called *Fourier analysis*, and the even more advanced *Laplace* and *z-transforms*. Most cutting-edge DSP algorithms are based on some aspect of these techniques.

Why is it even possible to decompose an arbitrary signal into sine and cosine waves? How are the amplitudes of these sinusoids determined for a particular signal? What kinds of systems can be designed with this technique? These are the questions to be answered in later chapters. The details of the Fourier decomposition are too involved to be presented in this brief overview. For now, the important idea to understand is that when all of the component sinusoids are added together, the original signal is exactly reconstructed. Much more on this in Chapter 8.

Alternatives to Linearity

To appreciate the importance of linear systems, consider that there is only *one* major strategy for analyzing systems that are nonlinear. That strategy is to make the nonlinear system *resemble* a linear system. There are three common ways of doing this:

First, ignore the nonlinearity. If the nonlinearity is small enough, the system can be approximated as linear. Errors resulting from the original assumption are tolerated as noise or simply ignored.

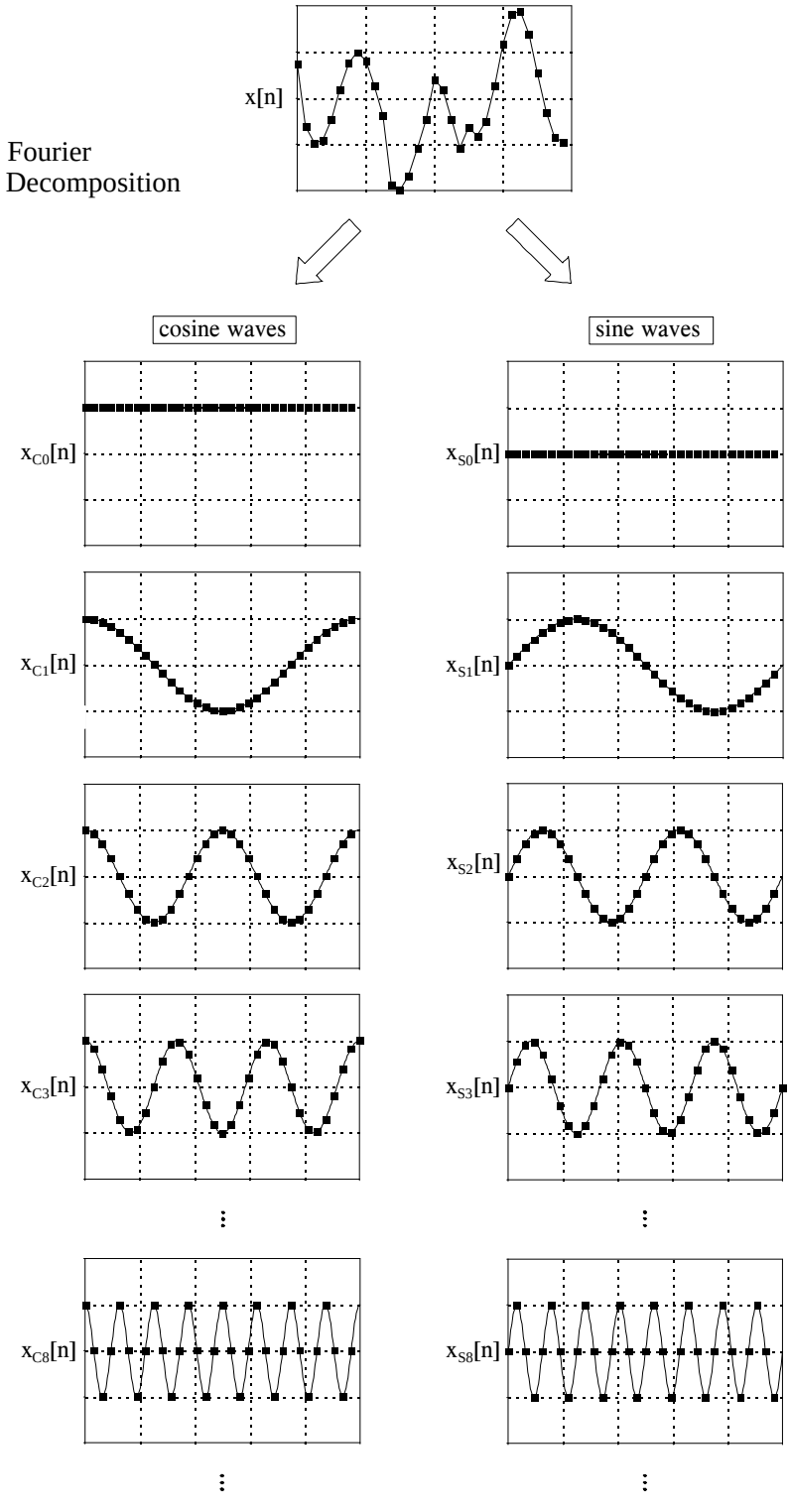


FIGURE 5-16
Illustration of Fourier decomposition. An N point signal is decomposed into N+2 signals, each having N points. Half of these signals are cosine waves, and half are sine waves. The frequencies of the sinusoids are fixed; only the amplitudes can change.

Second, keep the signals very small. Many nonlinear systems appear linear if the signals have a very small amplitude. For instance, transistors are very nonlinear over their full range of operation, but provide accurate linear amplification when the signals are kept under a few millivolts. Operational amplifiers take this idea to the extreme. By using very high open-loop gain together with negative feedback, the input signal to the op amp (i.e., the difference between the inverting and noninverting inputs) is kept to only a few microvolts. This minuscule input signal results in excellent linearity from an otherwise ghastly nonlinear circuit.

Third, apply a linearizing transform. For example, consider two signals being multiplied to make a third: $a[n] = b[n] \times c[n]$. Taking the logarithm of the signals changes the nonlinear process of multiplication into the linear process of addition: $\log(a[n]) = \log(b[n]) + \log(c[n])$. The fancy name for this approach is *homomorphic* signal processing. For example, a visual image can be modeled as the reflectivity of the scene (a two-dimensional signal) being multiplied by the ambient illumination (another two-dimensional signal). Homomorphic techniques enable the illumination signal to be made more uniform, thereby improving the image.

In the next chapters we examine the two main techniques of signal processing: *convolution* and *Fourier analysis*. Both are based on the strategy presented in this chapter: (1) decompose signals into simple additive components, (2) process the components in some useful manner, and (3) synthesize the components into a final result. This is DSP.