CHAPTER

7

Properties of Convolution

A linear system's characteristics are completely specified by the system's impulse response, as governed by the mathematics of convolution. This is the basis of many signal processing techniques. For example: Digital filters are created by *designing* an appropriate impulse response. Enemy aircraft are detected with radar by *analyzing* a measured impulse response. Echo suppression in long distance telephone calls is accomplished by creating an impulse response that *counteracts* the impulse response of the reverberation. The list goes on and on. This chapter expands on the properties and usage of convolution in several areas. First, several common impulse responses are discussed. Second, methods are presented for dealing with cascade and parallel combinations of linear systems. Third, the technique of *correlation* is introduced. Fourth, a nasty problem with convolution is examined, the computation time can be unacceptably long using conventional algorithms and computers.

Common Impulse Responses

Delta Function

The simplest impulse response is nothing more that a delta function, as shown in Fig. 7-1a. That is, an impulse on the input produces an identical impulse on the output. This means that *all* signals are passed through the system *without change*. Convolving any signal with a delta function results in exactly the same signal. Mathematically, this is written:

EQUATION 7-1
The delta function is the identity for convolution. Any signal convolved with a delta function is left unchanged.

$$x[n] * \delta[n] = x[n]$$

This property makes the delta function the **identity** for convolution. This is analogous to *zero* being the identity for addition ($a + 0 = a$), and *one* being the identity for multiplication ($a \times 1 = a$). At first glance, this type of system

may seem trivial and uninteresting. Not so! Such systems are the ideal for data storage, communication and measurement. Much of DSP is concerned with passing information through systems without change or degradation.

Figure 7-1b shows a slight modification to the delta function impulse response. If the delta function is made larger or smaller in amplitude, the resulting system is an **amplifier** or **attenuator**, respectively. In equation form, amplification results if *k* is *greater than one*, and attenuation results if *k* is *less than one*:

EQUATION 7-2
A system that amplifies or attenuates has a scaled delta function for an impulse response. In this equation, *k* determines the amplification or attenuation.

$$x[n] * k\delta[n] = kx[n]$$

The impulse response in Fig. 7-1c is a delta function with a **shift**. This results in a system that introduces an identical shift between the input and output signals. This could be described as a signal **delay**, or a signal **advance**, depending on the direction of the shift. Letting the shift be represented by the parameter, *s*, this can be written as the equation:

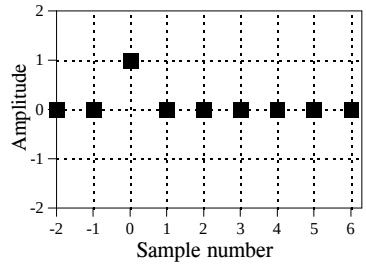
EQUATION 7-3
A relative shift between the input and output signals corresponds to an impulse response that is a shifted delta function. The variable, s, determines the amount of shift in this equation.

$$x[n] * \delta[n+s] = x[n+s]$$

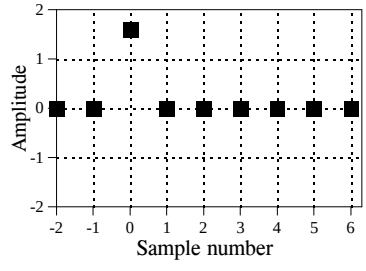
Science and engineering are filled with cases where one signal is a shifted version of another. For example, consider a radio signal transmitted from a remote space probe, and the corresponding signal received on the earth. The time it takes the radio wave to propagate over the distance causes a delay between the transmitted and received signals. In biology, the electrical signals in adjacent nerve cells are shifted versions of each other, as determined by the time it takes an action potential to cross the synaptic junction that connects the two.

Figure 7-1d shows an impulse response composed of a delta function plus a shifted and scaled delta function. By superposition, the output of this system is the input signal plus a delayed version of the input signal, i.e., an *echo*. Echoes are important in many DSP applications. The addition of echoes is a key part in making audio recordings sound natural and pleasant. Radar and sonar analyze echoes to detect aircraft and submarines. Geophysicists use echoes to find oil. Echoes are also very important in telephone networks, because you want to *avoid* them.

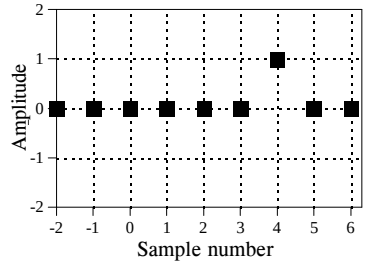
a. Identity
The delta function is the *identity* for convolution. Convolving a signal with the delta function leaves the signal unchanged. This is the goal of systems that transmit or store signals.



b. Amplification & Attenuation
Increasing or decreasing the amplitude of the delta function forms an impulse response that *amplifies* or *attenuates*, respectively. This impulse response will amplify the signal by 1.6.



c. Shift
Shifting the delta function produces a corresponding shift between the input and output signals. Depending on the direction, this can be called a *delay* or an *advance*. This impulse response delays the signal by four samples.



d. Echo
A delta function plus a shifted and scaled delta function results in an *echo* being added to the original signal. In this example, the echo is delayed by four samples and has an amplitude of 60% of the original signal.

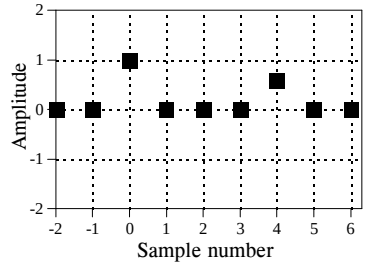


FIGURE 7-1
Simple impulse responses using shifted and scaled delta functions.

Calculus-like Operations

Convolution can change discrete signals in ways that resemble integration and differentiation. Since the terms "derivative" and "integral" specifically refer to operations on *continuous* signals, other names are given to their discrete counterparts. The discrete operation that mimics the *first derivative* is called the **first difference**. Likewise, the discrete form of the *integral* is called the

running sum. It is also common to hear these operations called the **discrete derivative** and the **discrete integral**, although mathematicians frown when they hear these informal terms used.

Figure 7-2 shows the impulse responses that implement the first difference and the running sum. Figure 7-3 shows an example using these operations. In 7-3a, the original signal is composed of several sections with varying slopes. Convolving this signal with the first difference impulse response produces the signal in Fig. 7-3b. Just as with the first derivative, the amplitude of each point in the first difference signal is equal to the *slope* at the corresponding location in the original signal. The running sum is the inverse operation of the first difference. That is, convolving the signal in (b), with the running sum's impulse response, produces the signal in (a).

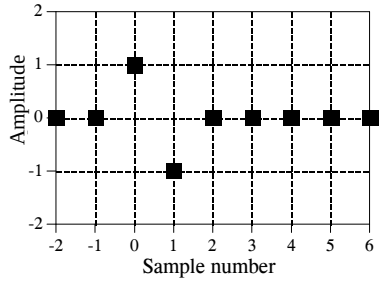
These impulse responses are simple enough that a full convolution program is usually not needed to implement them. Rather, think of them in the alternative mode: each sample in the output signal is a *sum of weighted samples from the input*. For instance, the first difference can be calculated:

EQUATION 7-4
Calculation of the first difference. In this relation, $x[n]$ is the original signal, and $y[n]$ is the first difference.

$$y[n] = x[n] - x[n-1]$$

That is, each sample in the output signal is equal to the difference between two adjacent samples in the input signal. For instance, $y[40] = x[40] - x[39]$. It should be mentioned that this is not the only way to define a *discrete derivative*. Another common method is to define the slope symmetrically around the point being examined, such as: $y[n] = (x[n+1] - x[n-1])/2$.

a. First Difference
This is the discrete version of the *first derivative*. Each sample in the output signal is equal to the *difference* between adjacent samples in the input signal. In other words, the output signal is the *slope* of the input signal.



b. Running Sum
The running sum is the discrete version of the *integral*. Each sample in the output signal is equal to the sum of all samples in the input signal to the *left*. Note that the impulse response extends to infinity, a rather nasty feature.

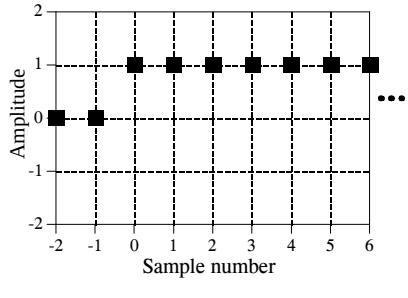


FIGURE 7-2
Impulse responses that mimic calculus operations.

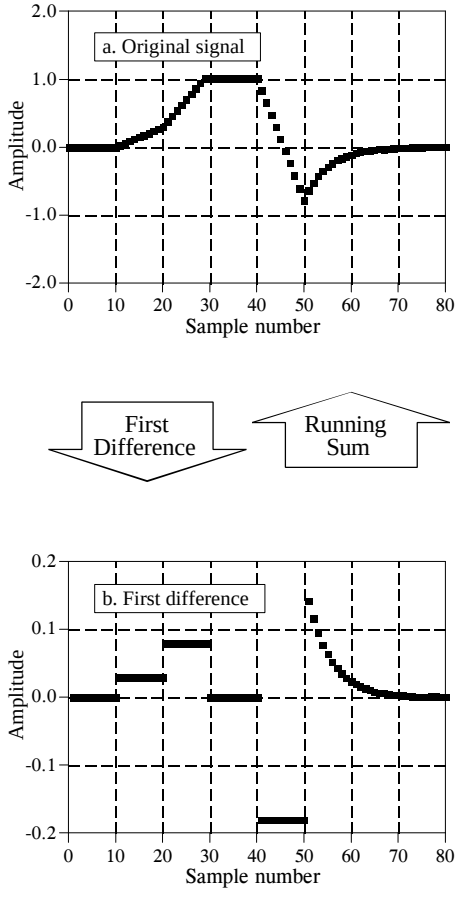


FIGURE 7-3
Example of calculus-like operations. The signal in (b) is the *first difference* of the signal in (a). Correspondingly, the signal is (a) is the *running sum* of the signal in (b). These processing methods are used with discrete signals the same as differentiation and integration are used with continuous signals.

Using this same approach, each sample in the running sum can be calculated by summing all points in the original signal to the *left* of the sample's location. For instance, if $y[n]$ is the running sum of $x[n]$, then sample $y[40]$ is found by adding samples $x[0]$ through $x[40]$. Likewise, sample $y[41]$ is found by adding samples $x[0]$ through $x[41]$. Of course, it would be very inefficient to calculate the running sum in this manner. For example, if $y[40]$ has already been calculated, $y[41]$ can be calculated with only a single addition: $y[41] = x[41] + y[40]$. In equation form:

EQUATION 7-5
Calculation of the running sum. In this relation, $x[n]$ is the original signal, and $y[n]$ is the running sum.

$$y[n] = x[n] + y[n-1]$$

Relations of this type are called **recursion equations** or **difference equations**. We will revisit them in Chapter 19. For now, the important idea to understand is that these relations are *identical* to convolution using the impulse responses of Fig. 7-2. Table 7-1 provides computer programs that implement these calculus-like operations.

```
100  'Calculation of the First Difference
110  Y[0] = 0
120  FOR I% = 1 TO N%-1
130    Y[I%] = X[I%] - X[I%-1]
140  NEXT I%
```

```
100  'Calculation of the running sum
110  Y[0] = X[0]
120  FOR I% = 1 TO N%-1
130    Y[I%] = Y[I%-1] + X[I%]
140  NEXT I%
```

Table 7-1
Programs for calculating the first difference and running sum. The original signal is held in X[ ], and the processed signal (the first difference or running sum) is held in Y[ ]. Both arrays run from 0 to N%-1.

Low-pass and High-pass Filters

The design of digital filters is covered in detail in later chapters. For now, be satisfied to understand the general shape of low-pass and high-pass *filter kernels* (another name for a filter's impulse response). Figure 7-4 shows several common low-pass filter kernels. In general, low-pass filter kernels are composed of a group of *adjacent positive points*. This results in each sample in the output signal being a weighted average of many adjacent points from the input signal. This averaging *smoothes* the signal, thereby removing high-frequency components. As shown by the sinc function in (c), some low-pass filter kernels include a few negative valued samples in the tails. Just as in analog electronics, digital low-pass filters are used for noise reduction, signal separation, wave shaping, etc.

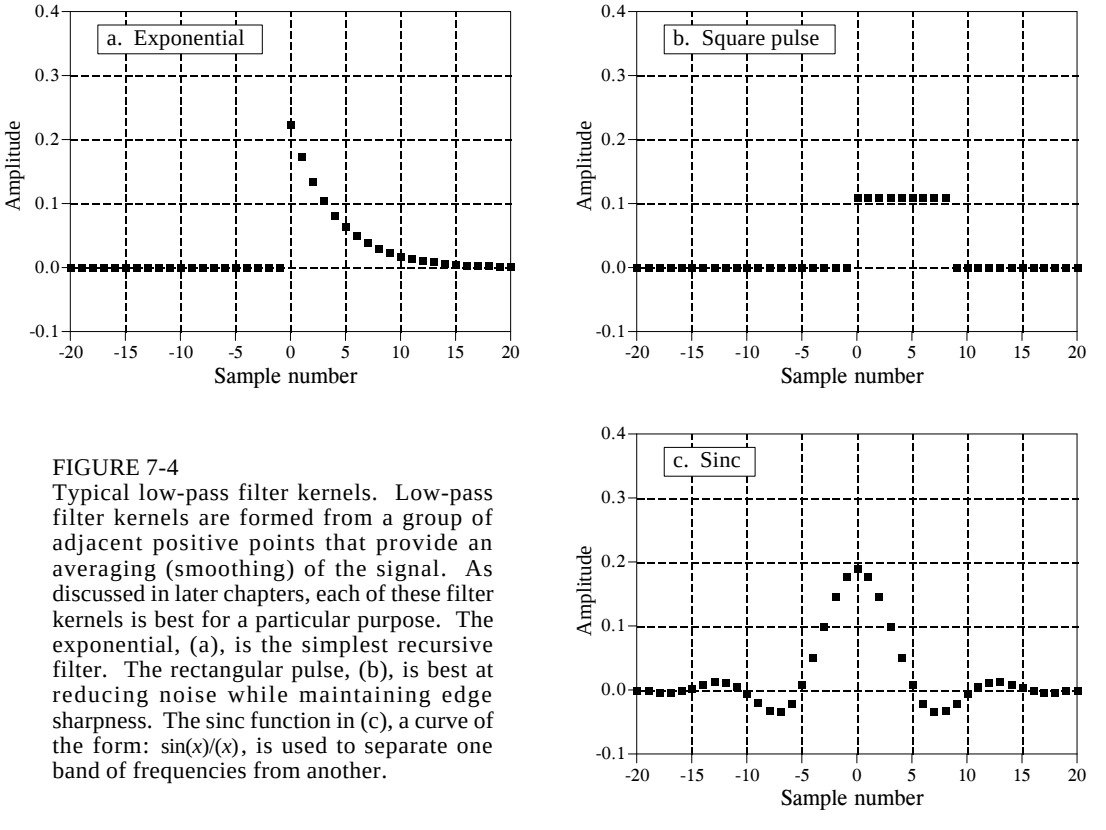


FIGURE 7-4
Typical low-pass filter kernels. Low-pass filter kernels are formed from a group of adjacent positive points that provide an averaging (smoothing) of the signal. As discussed in later chapters, each of these filter kernels is best for a particular purpose. The exponential, (a), is the simplest recursive filter. The rectangular pulse, (b), is best at reducing noise while maintaining edge sharpness. The sinc function in (c), a curve of the form: $\sin(x)/(x)$, is used to separate one band of frequencies from another.

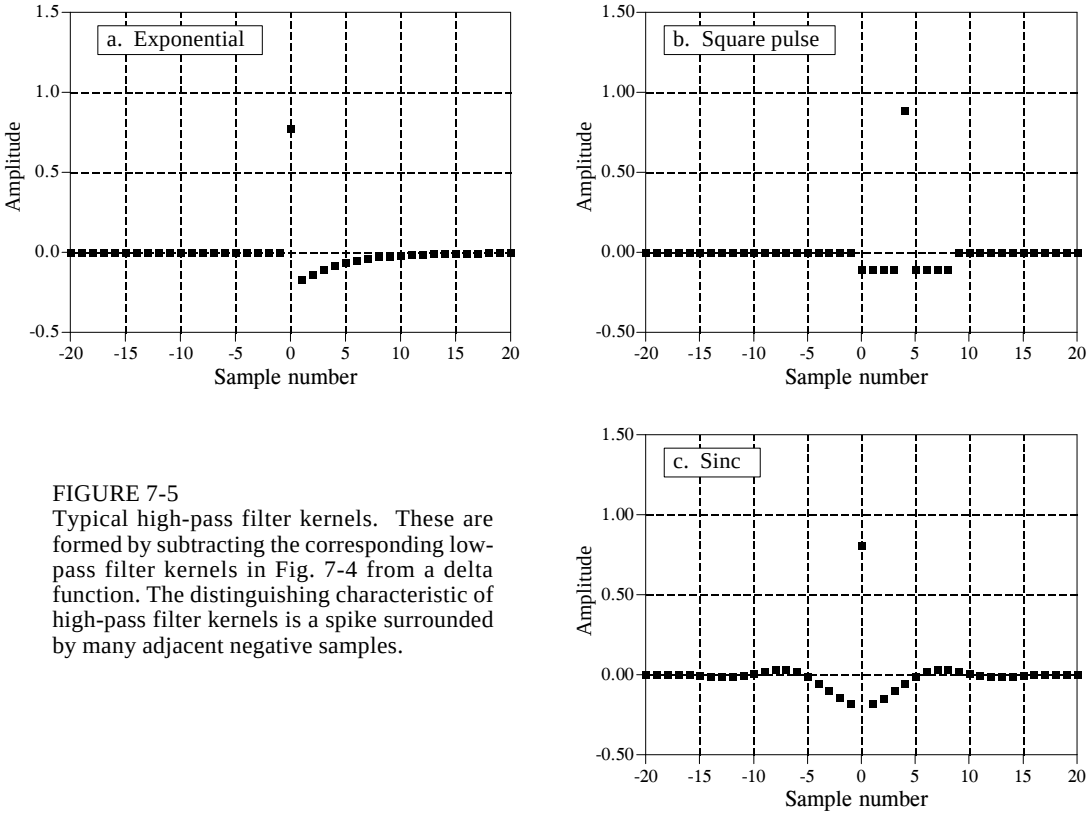


FIGURE 7-5
Typical high-pass filter kernels. These are formed by subtracting the corresponding low-pass filter kernels in Fig. 7-4 from a delta function. The distinguishing characteristic of high-pass filter kernels is a spike surrounded by many adjacent negative samples.

The cutoff frequency of the filter is changed by making filter kernel wider or narrower. If a low-pass filter has a gain of *one* at DC (zero frequency), then the sum of all of the points in the impulse response must be equal to *one*. As illustrated in (a) and (c), some filter kernels *theoretically* extend to infinity without dropping to a value of zero. In actual practice, the tails are truncated after a certain number of samples, allowing it to be represented by a finite number of points. How else could it be stored in a computer?

Figure 7-5 shows three common high-pass filter kernels, derived from the corresponding low-pass filter kernels in Fig. 7-4. This is a common strategy in filter design: first devise a low-pass filter and then transform it to what you need, high-pass, band-pass, band-reject, etc. To understand the low-pass to high-pass transform, remember that a delta function impulse response passes the entire signal, while a low-pass impulse response passes only the low-frequency components. By superposition, a filter kernel consisting of a delta function minus the low-pass filter kernel will pass the entire signal minus the low-frequency components. A high-pass filter is born! As shown in Fig. 7-5, the delta function is usually added at the center of symmetry, or sample zero if the filter kernel is not symmetrical. High-pass filters have zero gain at DC (zero frequency), achieved by making the sum of all the points in the filter kernel equal to *zero*.

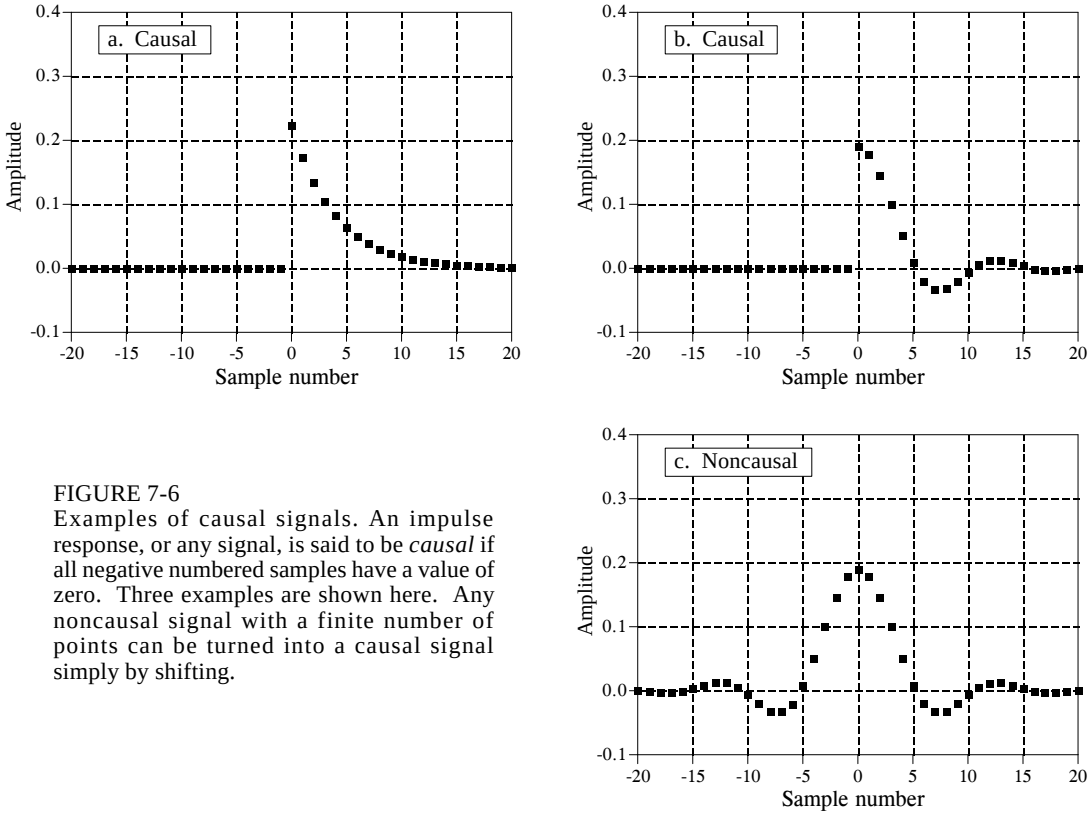


FIGURE 7-6
Examples of causal signals. An impulse response, or any signal, is said to be *causal* if all negative numbered samples have a value of zero. Three examples are shown here. Any noncausal signal with a finite number of points can be turned into a causal signal simply by shifting.

Causal and Noncausal Signals

Imagine a simple analog electronic circuit. If you apply a short pulse to the input, you will see a response on the output. This is the kind of cause and effect that our universe is based on. One thing we definitely know: *any effect must happen after the cause*. This is a basic characteristic of what we call *time*. Now compare this to a DSP system that changes an input signal into an output signal, both stored in arrays in a computer. If this mimics a real world system, it must follow the same principle of *causality* as the real world does. For example, the value at sample number eight in the input signal can only affect sample number eight or greater in the output signal. Systems that operate in this manner are said to be **causal.** Of course, digital processing doesn't necessarily have to function this way. Since both the input and output signals are arrays of numbers stored in a computer, any of the input signal values can affect any of the output signal values.

As shown by the examples in Fig. 7-6, the impulse response of a causal system must have a value of zero for all *negative numbered* samples. Think of this from the input side view of convolution. To be causal, an impulse in the input signal at sample number *n* must only affect those points in the output signal with a sample number of *n* or greater. In common usage, the term *causal* is applied to *any* signal where all the negative numbered samples have a value of zero, whether it is an impulse response or not.

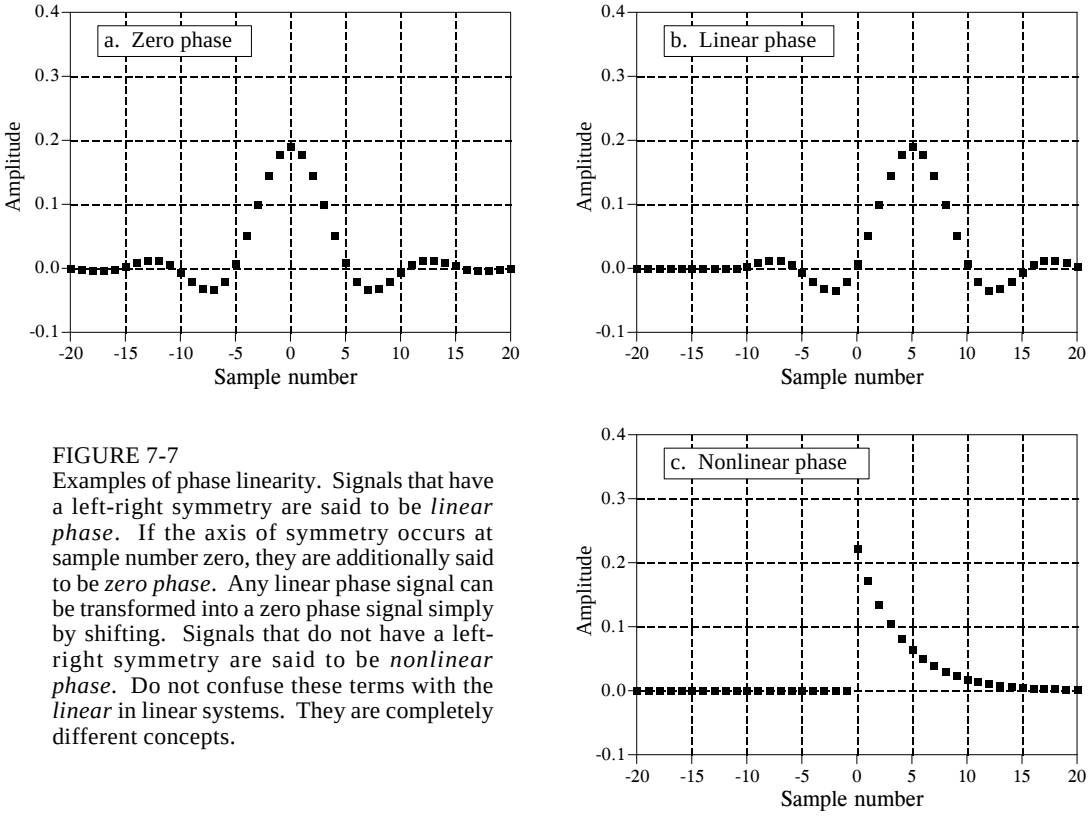


FIGURE 7-7
Examples of phase linearity. Signals that have a left-right symmetry are said to be *linear phase*. If the axis of symmetry occurs at sample number zero, they are additionally said to be *zero phase*. Any linear phase signal can be transformed into a zero phase signal simply by shifting. Signals that do not have a left-right symmetry are said to be *nonlinear phase*. Do not confuse these terms with the *linear* in linear systems. They are completely different concepts.

Zero Phase, Linear Phase, and Nonlinear Phase

As shown in Fig. 7-7, a signal is said to be **zero phase** if it has left-right symmetry around sample number zero. A signal is said to be **linear phase** if it has left-right symmetry, but around some point other than zero. This means that any linear phase signal can be changed into a zero phase signal simply by shifting left or right. Lastly, a signal is said to be **nonlinear phase** if it does not have left-right symmetry.

You are probably thinking that these names don't seem to follow from their definitions. What does *phase* have to do with *symmetry*? The answer lies in the frequency spectrum, and will be discussed in more detail in later chapters. Briefly, the frequency spectrum of any signal is composed of two parts, the magnitude and the phase. The frequency spectrum of a signal that is symmetrical around zero has a phase that is zero. Likewise, the frequency spectrum of a signal that is symmetrical around some nonzero point has a phase that is a straight line, i.e., a linear phase. Lastly, the frequency spectrum of a signal that is not symmetrical has a phase that is not a straight line, i.e., it has a nonlinear phase.

A special note about the potentially confusing terms: *linear* and *nonlinear phase*. What does this have to do the concept of system linearity discussed in previous chapters? Absolutely nothing! System linearity is the broad concept

that nearly all of DSP is based on (superposition, homogeneity, additivity, etc). *Linear* and *nonlinear phase* mean that the phase is, or is not, a straight line. In fact, a system must be *linear* even to say that the phase is zero, linear, or nonlinear.

Mathematical Properties

Commutative Property

The commutative property for convolution is expressed in mathematical form:

EQUATION 7-6
The commutative property of convolution. This states that the order in which signals are convolved can be exchanged.

$$a[n] * b[n] = b[n] * a[n]$$

In words, the order in which two signals are convolved makes no difference; the results are identical. As shown in Fig. 7-8, this has a strange meaning for system theory. In any linear system, the input signal and the system's impulse response can be *exchanged* without changing the output signal. This is interesting, but usually doesn't have any physical meaning. The input signal and the impulse response are very different things. Just because the mathematics *allows* you to do something, doesn't mean that it makes sense to do it. For example, suppose you make: \$10/hour × 2,000 hours/year = \$20,000/year. The commutative property for multiplication provides that you can make the same annual salary by only working 10 hours/year at \$2000/hour. Let's see you convince your boss that this is meaningful! In spite of this, the commutative property sees great use in DSP for manipulating equations, just as in ordinary algebra.

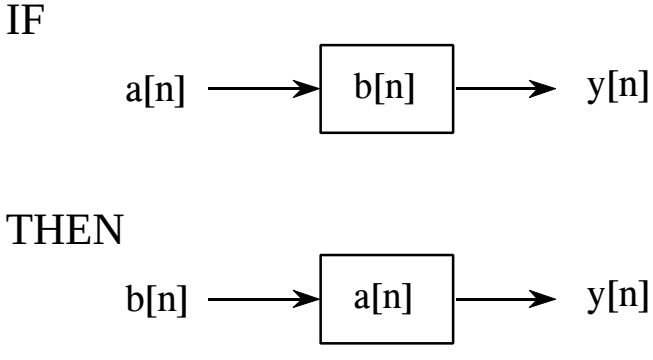


FIGURE 7-8
The commutative property in system theory. The commutative property of convolution allows the input signal and the impulse response of a system to be exchanged without changing the output. While interesting, this usually has no physical significance. (A signal appearing inside of a box, such as b[n] and a[n] in this figure, represent the *impulse response* of the system).

Associative Property

Is it possible to convolve three or more signals? The answer is yes, and the associative property describes how: convolve two of the signals to produce an intermediate signal, then convolve the intermediate signal with the third signal. The associative property provides that the order of the convolutions doesn't matter. As an equation:

EQUATION 7-7
The associative property of convolution describes how three or more signals are convolved.

$$(a[n] * b[n]) * c[n] = a[n] * (b[n] * c[n])$$

The associative property is used in system theory to describe how **cascaded systems** behave. As shown in Fig. 7-9, two or more systems are said to be in a *cascade* if the output of one system is used as the input for the next system. From the associative property, the order of the systems can be rearranged without changing the overall response of the cascade. Further, any number of cascaded systems can be replaced with a *single* system. The impulse response of the replacement system is found by convolving the impulse responses of all of the original systems.

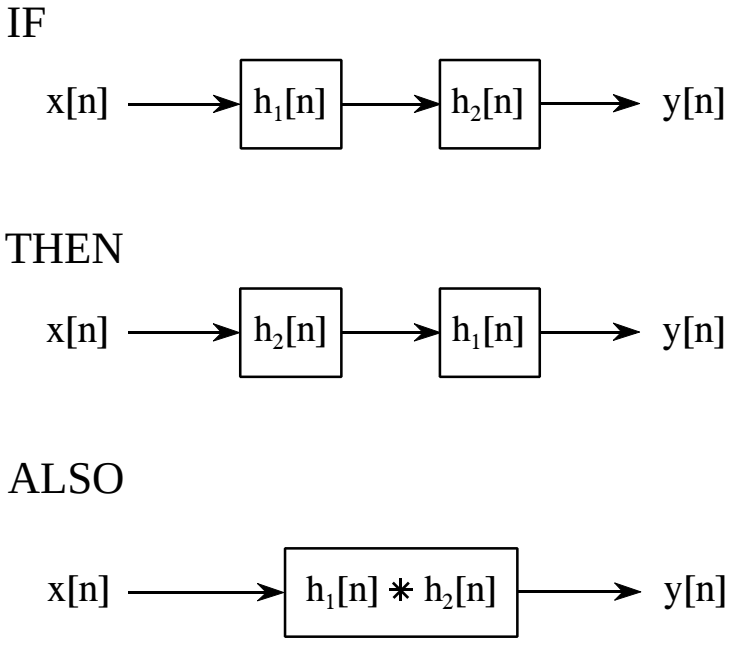


FIGURE 7-9
The associative property in system theory. The associative property provides two important characteristics of *cascaded* linear systems. First, the order of the systems can be rearranged without changing the overall operation of the cascade. Second, two or more systems in a cascade can be replaced by a single system. The impulse response of the replacement system is found by convolving the impulse responses of the stages being replaced.

Distributive Property
In equation form, the distributive property is written:

EQUATION 7-8
The distributive property of convolution describes how parallel systems are analyzed.

$$a[n] * b[n] + a[n] * c[n] = a[n] * (b[n] + c[n])$$

The distributive property describes the operation of **parallel systems with added outputs**. As shown in Fig. 7-10, two or more systems can share the same input, $x[n]$, and have their outputs added to produce $y[n]$. The distributive property allows this combination of systems to be replaced with a single system, having an impulse response equal to the *sum* of the impulse responses of the original systems.

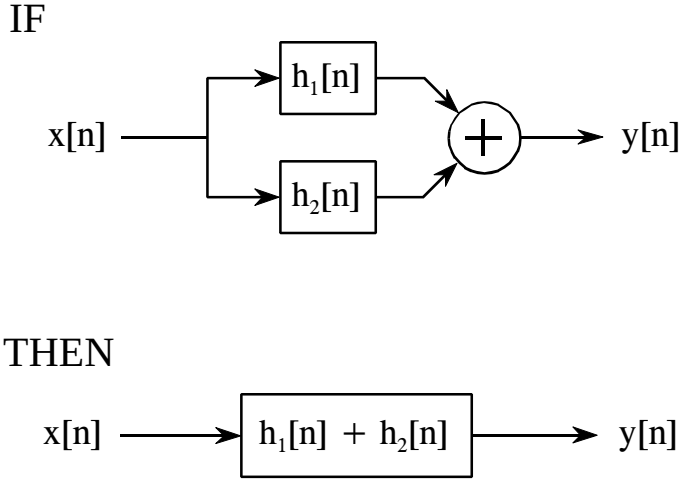


FIGURE 7-10
The distributive property in system theory. The distributive property shows that parallel systems with added outputs can be replaced with a single system. The impulse response of the replacement system is equal to the sum of the impulse responses of all the original systems.

Transference between the Input and Output
Rather than being a formal mathematical property, this is a way of thinking about a common situation in signal processing. As illustrated in Fig. 7-11, imagine a linear system receiving an input signal, $x[n]$, and generating an output signal, $y[n]$. Now suppose that the input signal is changed in some linear way, resulting in a new input signal, which we will call $x'[n]$. This results in a new output signal, $y'[n]$. The question is, how does the change in

the input signal relate to the change in the output signal? The answer is: *the output signal is changed in exactly the same linear way that the input signal was changed*. For example, if the input signal is amplified by a factor of two, the output signal will also be amplified by a factor of two. If the derivative is taken of the input signal, the derivative will also be taken of the output signal. If the input is filtered in some way, the output will be filtered in an identical manner. This can easily be proven by using the associative property.

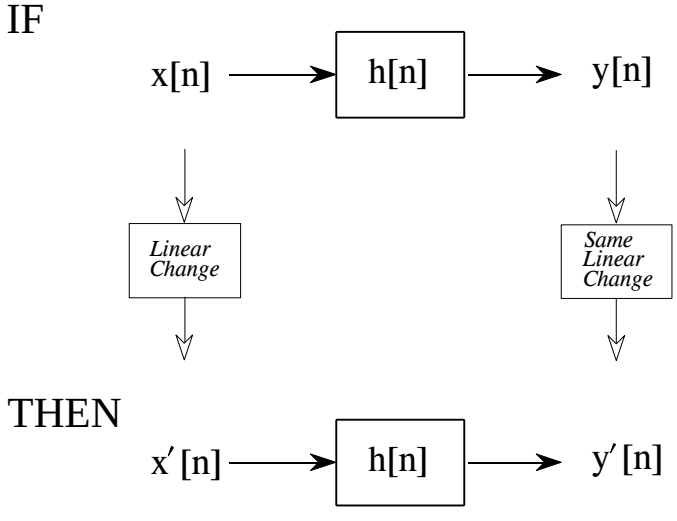


FIGURE 7-11
Tranference between the input and output. This is a way of thinking about a common situation in signal processing. A linear change made to the input signal results in the same linear change being made to the output signal.

The Central Limit Theorem

The Central Limit Theorem is an important tool in probability theory because it mathematically explains why the Gaussian probability distribution is observed so commonly in nature. For example: the amplitude of thermal noise in electronic circuits follows a Gaussian distribution; the cross-sectional intensity of a laser beam is Gaussian; even the pattern of holes around a dart board bull's eye is Gaussian. In its simplest form, the Central Limit Theorem states that a Gaussian distribution results when the observed variable is the sum of many random processes. Even if the component processes do not have a Gaussian distribution, the sum of them will.

The Central Limit Theorem has an interesting implication for convolution. If a pulse-like signal is convolved with *itself* many times, a Gaussian is produced. Figure 7-12 shows an example of this. The signal in (a) is an

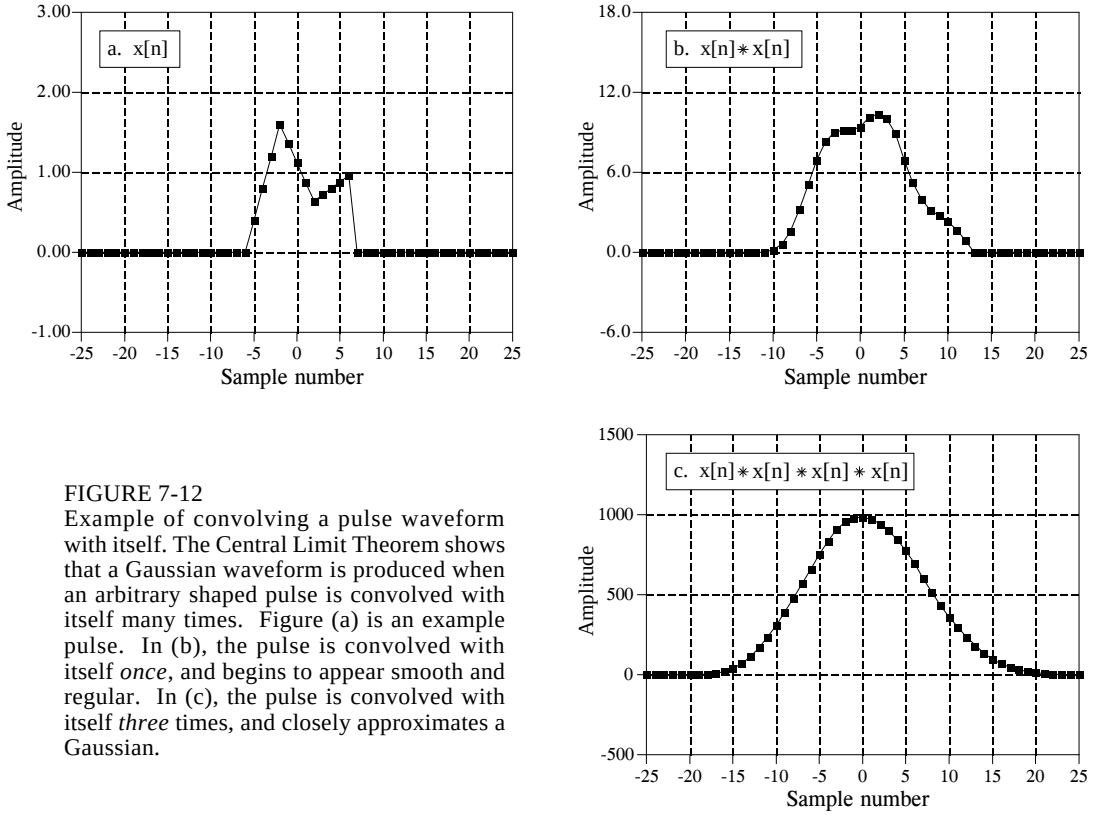


FIGURE 7-12
Example of convolving a pulse waveform with itself. The Central Limit Theorem shows that a Gaussian waveform is produced when an arbitrary shaped pulse is convolved with itself many times. Figure (a) is an example pulse. In (b), the pulse is convolved with itself *once*, and begins to appear smooth and regular. In (c), the pulse is convolved with itself *three* times, and closely approximates a Gaussian.

irregular pulse, purposely chosen to be very unlike a Gaussian. Figure (b) shows the result of convolving this signal with itself *one* time. Figure (c) shows the result of convolving this signal with itself *three* times. Even with only three convolutions, the waveform looks very much like a Gaussian. In mathematics jargon, the procedure *converges* to a Gaussian *very quickly*. The width of the resulting Gaussian (i.e., σ in Eq. 2-7 or 2-8) is equal to the width of the original pulse (expressed as σ in Eq. 2-7) multiplied by the square root of the number of convolutions.

Correlation

The concept of correlation can best be presented with an example. Figure 7-13 shows the key elements of a radar system. A specially designed antenna transmits a short burst of radio wave energy in a selected direction. If the propagating wave strikes an object, such as the helicopter in this illustration, a small fraction of the energy is reflected back toward a radio receiver located near the transmitter. The transmitted pulse is a specific shape that we have selected, such as the triangle shown in this example. The received signal will consist of two parts: (1) a shifted and scaled version of the transmitted pulse, and (2) random noise, resulting from interfering radio waves, thermal noise in the electronics, etc. Since radio signals travel at a known rate, the speed of

light, the shift between the transmitted and received pulse is a direct measure of the distance to the object being detected. This is the problem: given a signal of some known shape, what is the best way to determine where (or if) the signal occurs in *another* signal. Correlation is the answer.

Correlation is a mathematical operation that is very similar to convolution. Just as with convolution, correlation uses two signals to produce a third signal. This third signal is called the **cross-correlation** of the two input signals. If a signal is correlated with *itself*, the resulting signal is instead called the **autocorrelation**. The convolution machine was presented in the last chapter to show how convolution is performed. Figure 7-14 is a similar

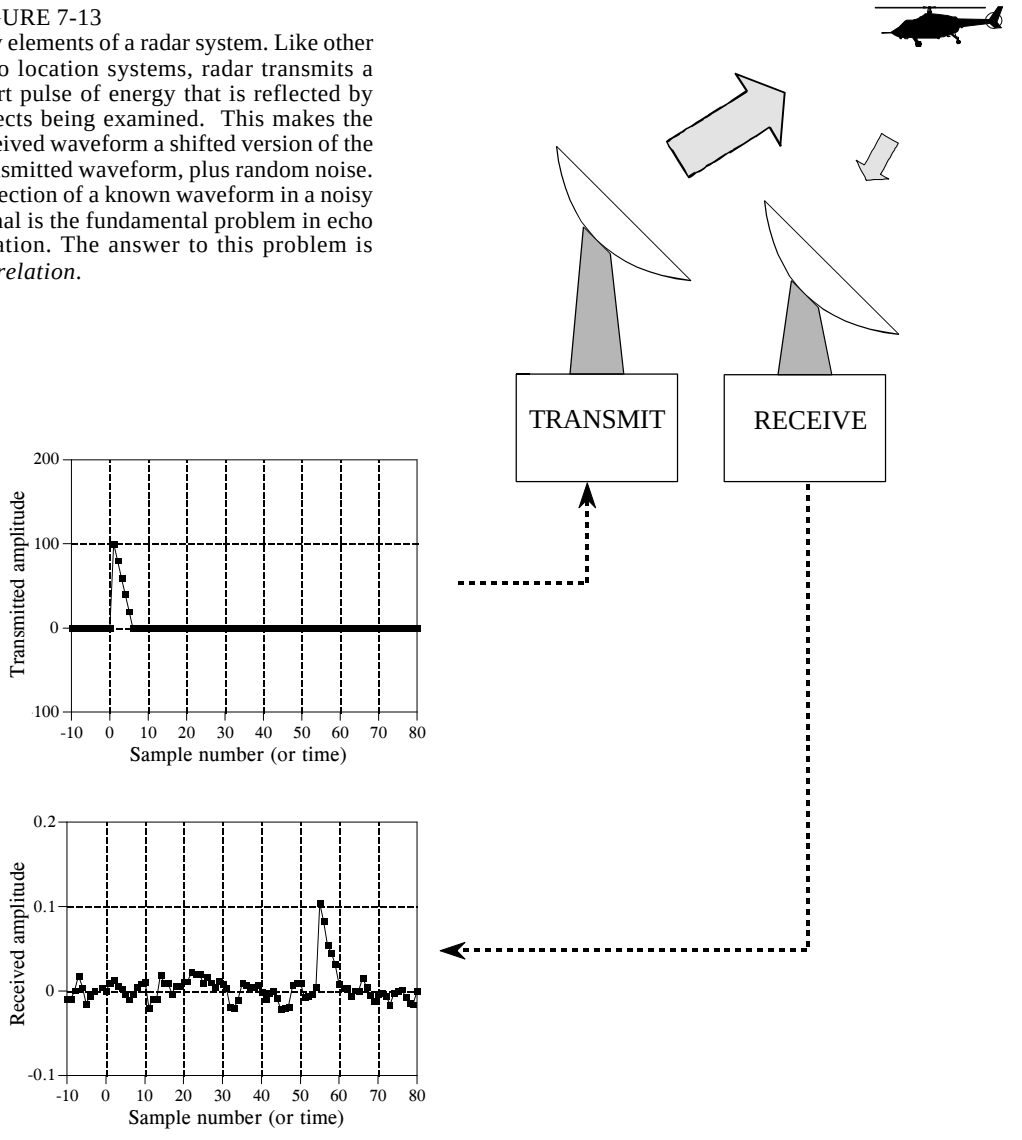


FIGURE 7-13
Key elements of a radar system. Like other echo location systems, radar transmits a short pulse of energy that is reflected by objects being examined. This makes the received waveform a shifted version of the transmitted waveform, plus random noise. Detection of a known waveform in a noisy signal is the fundamental problem in echo location. The answer to this problem is *correlation*.

illustration of a **correlation machine**. The received signal, $x[n]$, and the cross-correlation signal, $y[n]$, are fixed on the page. The waveform we are looking for, $t[n]$, commonly called the **target** signal, is contained *within* the correlation machine. Each sample in $y[n]$ is calculated by moving the correlation machine left or right until it points to the sample being worked on. Next, the indicated samples from the received signal fall into the correlation machine, and are multiplied by the corresponding points in the target signal. The sum of these products then moves into the proper sample in the cross-correlation signal.

The amplitude of each sample in the cross-correlation signal is a measure of how much the received signal *resembles* the target signal, *at that location*. This means that a peak will occur in the cross-correlation signal for every target signal that is present in the received signal. In other words, the value of the cross-correlation is maximized when the target signal is *aligned* with the same features in the received signal.

What if the target signal contains samples with a negative value? Nothing changes. Imagine that the correlation machine is positioned such that the target signal is perfectly aligned with the matching waveform in the received signal. As samples from the received signal fall into the correlation machine, they are multiplied by their matching samples in the target signal. Neglecting noise, a positive sample will be multiplied by itself, resulting in a positive number. Likewise, a negative sample will be multiplied by itself, also resulting in a positive number. Even if the target signal is completely negative, the peak in the cross-correlation will still be positive.

If there is noise on the received signal, there will also be noise on the cross-correlation signal. It is an unavoidable fact that random noise looks a certain amount like any target signal you can choose. The noise on the cross-correlation signal is simply measuring this similarity. Except for this noise, the peak generated in the cross-correlation signal is symmetrical between its left and right. This is true even if the target signal isn't symmetrical. In addition, the width of the peak is twice the width of the target signal. Remember, the cross-correlation is trying to *detect* the target signal, not *recreate* it. There is no reason to expect that the peak will even look like the target signal.

Correlation is the *optimal* technique for detecting a known waveform in random noise. That is, the peak is higher above the noise using correlation than can be produced by any other linear system. (To be perfectly correct, it is only optimal for *random white noise*). Using correlation to detect a known waveform is frequently called **matched filtering**. More on this in Chapter 17.

The correlation machine and convolution machine are identical, except for one small difference. As discussed in the last chapter, the signal inside of the convolution machine is *flipped* left-for-right. This means that samples numbers: 1, 2, 3··· run from the right to the left. In the correlation machine this flip doesn't take place, and the samples run in the normal direction.

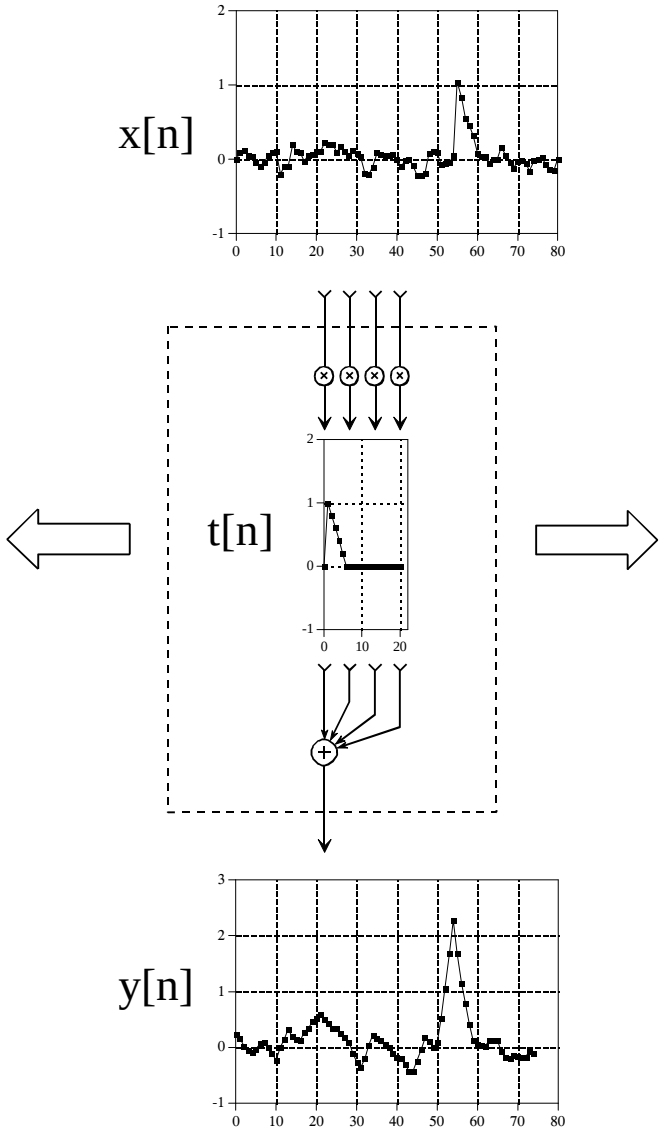


FIGURE 7-14
The correlation machine. This is a flowchart showing how the cross-correlation of two signals is calculated. In this example, $y[n]$ is the cross-correlation of $x[n]$ and $t[n]$. The dashed box is moved left or right so that its output points at the sample being calculated in $y[n]$. The indicated samples from $x[n]$ are multiplied by the corresponding samples in $t[n]$, and the products added. The correlation machine is identical to the convolution machine (Figs. 6-8 and 6-9), except that the signal inside of the dashed box is *not* reversed. In this illustration, the only samples calculated in $y[n]$ are where $t[n]$ is fully *immersed* in $x[n]$.

Since this signal reversal is the only difference between the two operations, it is possible to represent *correlation* using the same mathematics as *convolution*. This requires *preflipping* one of the two signals being correlated, so that the left-for-right flip inherent in convolution is canceled. For instance, when $a[n]$ and $b[n]$, are convolved to produce $c[n]$, the equation is written: $a[n] * b[n] = c[n]$. In comparison, the cross-correlation of $a[n]$ and $b[n]$ can

be written: $a[n] * b[-n] = c[n]$. That is, flipping $b[n]$ left-for-right is accomplished by reversing the sign of the index, i.e., $b[-n]$.

Don't let the mathematical similarity between convolution and correlation fool you; they represent very different DSP procedures. Convolution is the relationship between a system's input signal, output signal, and impulse response. Correlation is a way to detect a known waveform in a noisy background. The similar mathematics is only a convenient coincidence.

Speed

Writing a program to convolve one signal by another is a simple task, only requiring a few lines of code. Executing the program may be more painful. The problem is the large number of additions and multiplications required by the algorithm, resulting in long execution times. As shown by the programs in the last chapter, the time-consuming operation is composed of multiplying two numbers and adding the result to an accumulator. Other parts of the algorithm, such as indexing the arrays, are very quick. The multiply-accumulate is a basic building block in DSP, and we will see it repeated in several other important algorithms. In fact, the speed of DSP computers is often *specified* by how long it takes to preform a multiply-accumulate operation.

If a signal composed of N samples is convolved with a signal composed of M samples, $N \times M$ multiply-accumulations must be preformed. This can be seen from the programs of the last chapter. Personal computers of the mid 1990's requires about one microsecond per multiply-accumulation (100 MHz Pentium using single precision floating point, see Table 4-6). Therefore, convolving a 10,000 sample signal with a 100 sample signal requires about one second. To process a one million point signal with a 3000 point impulse response requires nearly an hour. A decade earlier (80286 at 12 MHz), this calculation would have required three days!

The problem of excessive execution time is commonly handled in one of three ways. First, simply keep the signals as short as possible and use integers instead of floating point. If you only need to run the convolution a few times, this will probably be the best trade-off between execution time and programming effort. Second, use a computer designed for DSP. DSP microprocessors are available with multiply-accumulate times of only a few tens of nanoseconds. This is the route to go if you plan to perform the convolution many times, such as in the design of commercial products.

The third solution is to use a better algorithm for implementing the convolution. Chapter 17 describes a very sophisticated algorithm called *FFT convolution*. FFT convolution produces exactly the same result as the convolution algorithms presented in the last chapter; however, the execution time is dramatically reduced. For signals with thousands of samples, FFT convolution can be hundreds of times faster. The disadvantage is program complexity. Even if you are familiar with the technique, expect to spend several hours getting the program to run.